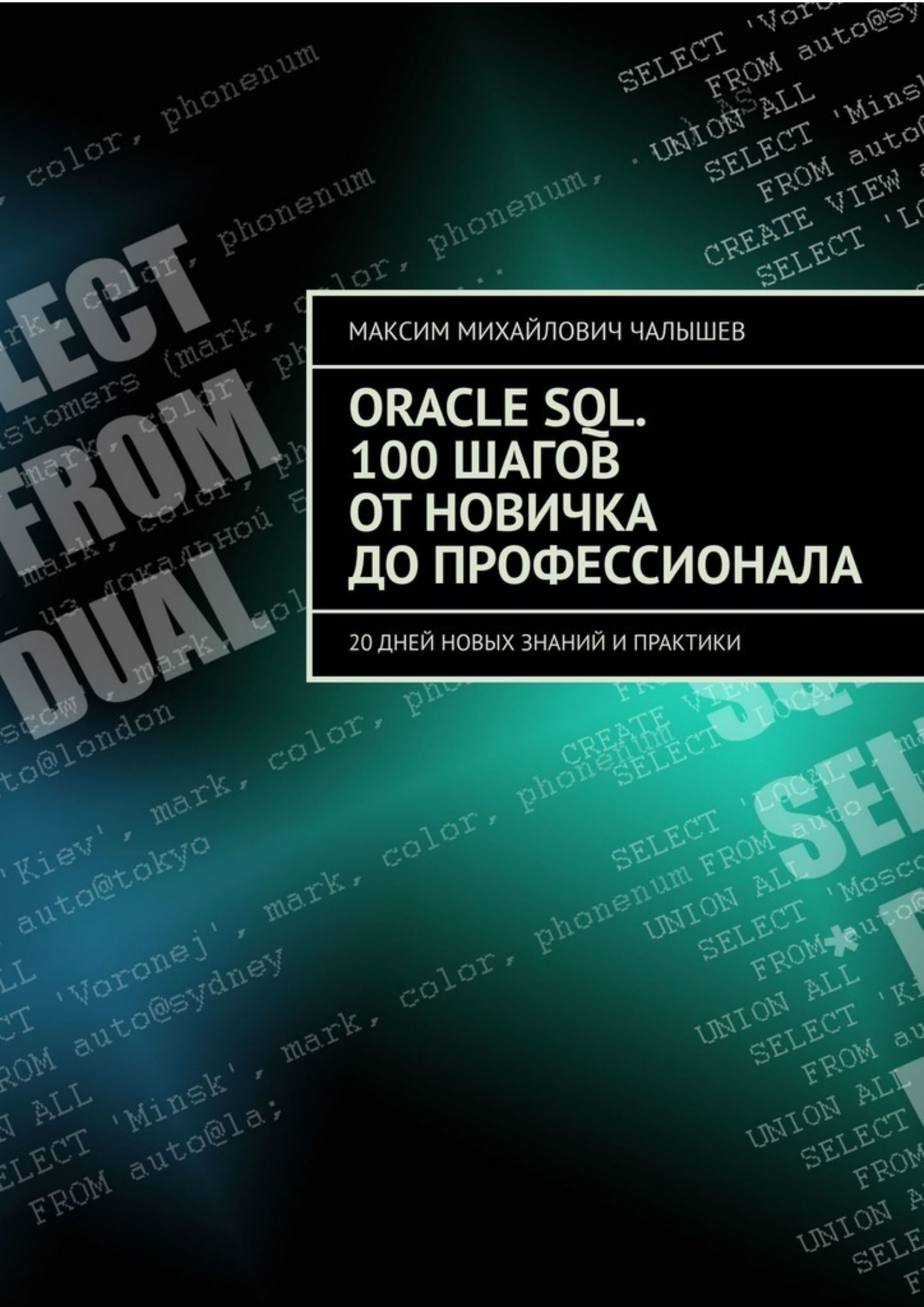


МАКСИМ МИХАЙЛОВИЧ ЧАЛЫШЕВ

ORACLE SQL. 100 ШАГОВ ОТ НОВИЧКА ДО ПРОФЕССИОНАЛА

20 ДНЕЙ НОВЫХ ЗНАНИЙ И ПРАКТИКИ

Максим Чалышев

Oracle SQL. 100 шагов от новичка до профессионала. 20 дней новых знаний и практики

«Издательские решения»

Чалышев М. М.

Oracle SQL. 100 шагов от новичка до профессионала. 20 дней новых знаний и практики / М. М. Чалышев — «Издательские решения»,

ISBN 978-5-00-500618-9

Более 300 ответов на вопросы.Более 500 практических заданий.Более 1000 разобранных примеров.Учебник справочник по языку SQL.

ISBN 978-5-00-500618-9

Содержание

Oracle SQL. 100 шагов от новичка до профессионала 20 дней новых знаний и практики

Максим Михайлович Чалышев

SQL.100 шагов от новичка до профессионала

ISBN 978-5-0050-0618-9
Создано в интеллектуальной издательской системе Ridero

20 дней новых знаний и практики.
Более 300 ответов на вопросы.
Более 500 практических заданий.
Более 1000 разобранных примеров.

Данную книгу я посвящаю своим друзьям:
Кузнецову Алексею – профессионалу управления в сфере ИТ,
Коршакову Артему – будущему высококлассному IT-специалисту.

Чалышев Максим

Введение

Приветствую. Сначала как автор этой книги расскажу немного о своем профессиональном опыте. На данный момент я работаю в сфере информационных технологий уже почти 20 лет.

Основной моей специализацией в ИТ были и остаются базы данных и, прежде всего, СУБД ORACLE.

В первый раз я познакомился с данной СУБД в институте, один из моих преподавателей проходил стажировку в США. Он рассказывал студентам о базе данных ORACLE, о применении на производстве, в финансовых организациях, в крупных государственных учреждениях.

Во время изучения я был очарован четкой и понятной организацией структур информации в СУБД ORACLE, обширностью средств работы с данными, ее мощностью и вместе с тем неповторимой гибкостью, а также уникальными возможностями данной СУБД.

Далее, после института и защиты диплома, в течение нескольких лет я работал на крупном производственном предприятии, информационная система которого была построена преимущественно на использовании СУБД ORACLE.

Потоки данных представлялись грандиозными для того времени, каждый день сервер ORACLE рассчитывал сотни тысяч производственных компонентов, технологических маршрутов, спецификаций изделий.

Стоит понимать, что на дворе был конец XX века и объемы, которые тогда казались нам огромными, сейчас вызывают улыбку, технологические мощности ушли далеко вперед, и сейчас даже средний ноутбук сопоставим по быстродействию с многопроцессорными серверами тех лет.

Поле этого я перешел в сектор телекоммуникаций, и именно здесь в своей работе я столкнулся с быстрыми транзакциями, а также обработкой огромных объемов данных в сотни и сотни миллионов записей.

Далее была длительная работа в процессинге крупного банка, где также использовалась СУБД ORACLE, интернет-сатрапе средней компании-разработчика ПО, на этом этапе я познакомился с СУБД других производителей, таких как MS SQL PostgreSQL, MySQL; также я работал на большом DWH-проекте, где объемы в миллиарды записей передавались за считаные минуты.

И каждый раз, сталкиваясь с новой для себя сферой, я открывал дополнительные возможности и новые уникальные приемы разработки, которыми и хотел бы поделиться в этой книге.

Именно после многих лет работы начинаешь осознавать, что практика и теория отличаются. Те примеры, которые описаны в документации, могут работать по-разному в разных условиях, а также иметь множество нюансов использования в конкретной ситуации в конкретной сфере.

Основой моей работы была именно СУБД ORACLE, хотя я также неизбежно сталкивался вплотную и с другими технологиями, такими как Java, SAS, Python, веб-разработка JavaScript, Node JS.

Важно осознавать, что ORACLE сейчас представляет собой целый конгломерат производственных решений, куда входит, например, ORACLE Siebel CRM, JaVA, ORACLE Service Bus, но все же основным продуктом данной корпорации была и остается именно СУБД ORACLE.

Что же касается настоящего момента, то сейчас я провожу специальные курсы по SQL в базовой и расширенной версиях, на которые каждый из вас может записаться.

Адрес курсов www.sqladv.ru

Вопросы учеников, часть практических примеров взяты непосредственно с этих курсов.

День первый

Шаг 1. Что такое SQL, назначение языка

Приветствую вас, уважаемый читатель. Позволю написать несколько слов о себе.

На текущий момент вот уже более 20 лет я работаю IT-специалистом. Я занимал должности архитектора, администратора баз данных, разработчика баз данных.

У меня также есть своя IT-школа sqladv.ru, где один из курсов, который я веду сам, посвящен разработке баз данных и языку SQL.

Эта книга представляет собой также своеобразный курс обучения: с помощью данной книги, упорно занимаясь, вы освоите язык SQL от начального уровня до уровня ведущего разработчика.

В данной книге рассмотрен диалект языка ORACLE SQL как один из наиболее распространенных на сегодняшний день. Также специалисты ORACLE обычно имеют более высокую зарплату по сравнению с другими разработчиками.

Когда я преподавал в своей IT-школе sqladv.ru, то убедился, что программирование – это, прежде всего, практика, поэтому в каждой главе данной книги разбираются актуальные примеры и всегда присутствует несколько практических заданий, обязательных для выполнения.

В каждой главе книги вы сможете найти наиболее интересные и актуальные вопросы моих учеников, разумеется, с моими ответами.

Итак, перейдем непосредственно к теоретической части.

Что такое базы данных, назначение баз данных

Трудно себе представить, что раньше вся информация размещалась на бумажных носителях и архивы документов занимали подчас целые здания.

Сейчас же пришел новый век, новая информационная эпоха, и теперь для хранения и обработки информации используются в основном электронные системы.

Огромное количество таких систем работает под управлением баз данных. Спектр применения систем управления базами данных на сегодняшний день практически необъятен – базы данных используются в интернете, в производстве, в промышленности, в маркетинге, в мобильных устройствах, в финансовой и банковской сферах, на телевидении, в телекоммуникациях и рекламе.

Какова применимость баз данных, то есть где используются базы данных?
Вот лишь некоторые области, где базы данных нашли применение.

Веб-технологии

Веб-технологии проникли в нашу жизнь, и без них уже сложно представить современный мир. Социальные сети, почта, поисковые системы, онлайн-сервисы погоды и навигации – всем этим большинство из нас пользуется ежедневно.

Почти все онлайн-ресурсы работают с базами данных. Практически каждый сайт, поисковая система, социальная сеть построены на основе баз данных и используют язык SQL.

Поэтому, если ваш бизнес или ваша профессия каким-либо образом связаны с интернет-проектами, если вы веб-дизайнер или веб-программист (и не важно, на каком языке вы специализируетесь), знание баз данных и языка SQL вам обязательно пригодится в работе.

Мобильные устройства

Большинство мобильных приложений, приложений для планшетов также использует базы данных в своей работе. Система управления базами данных для мобильных устройств называется SQLite. SQLite имеет ряд особенностей, связанных с характеристиками мобильных устройств, но в целом использует такой же синтаксис SQL, как и другие базы данных.

Игры

Современные компьютерные игры также невозможны без использования баз данных. Игры – это множество объектов, игроков, карт, вооружений, стратегий, юнитов… С этой информацией надо активно работать. Без систем управления базами данных тут не обойтись. И если вы планируете связать свою жизнь с интереснейшей профессией игрового дизайнера или программиста компьютерных игр, знание баз данных будет для вас очень и очень желательным.

Крупные корпорации

Во всех без исключения крупных компаниях используются базы данных. Системы управления базами данных управляют банкоматами и пунктами выдачи наличных, на них реализованы бухгалтерия, учет и управление кадрами. Системы управления базами данных считают телефонные звонки по тарифам, вычисляют стоимость интернет-подключений и трафика.

Если вы работаете в крупной компании или собираетесь связать свою карьеру с работой на корпорацию, вам необходимо понимать, что такое база данных, принципы ее устройства, знать и понимать язык для работы с базами – SQL.

Назначение языка SQL, необходимость изучения этого языка

Structured Query Language (SQL) – язык структурированных запросов.

Язык запросов SQL – универсальный язык для работы с данными базы. Язык запросов SQL используется для управления массивами данных в БД, множествами.

Язык SQL предоставляет возможность для вывода структурированной заданной информации из базы. SQL также применяется для изменения данных, добавления данных из базы.

Язык SQL относится к функциональным языкам программирования. Он отличается от алгоритмических языков. Основу языка составляет не алгоритм как таковой, а совокупность команд, определяющих взаимоотношения информационных множеств и подмножеств.

Следует отметить, что системы управления базами данных – СУБД – имеют различные реализации, такие как ORACLE, MS SQL, MY SQL.

Язык SQL в разных СУБД имеет небольшие отличия, например в детальном синтаксисе описания операторов.

Такие отличия присутствуют в специальных функциях, относящихся к той или иной СУБД, но все же в основном язык – это общий синтаксис, практически идентичный для любой СУБД.

В данном курсе мы будем рассматривать общепринятый синтаксис SQL ORACLE.

Данная книга, как я ранее писал, обучает диалекту ORACLE SQL как наиболее востребованному и сложному, но на страницах книги вы также сможете найти некоторые примеры на других диалектах.

Вопросы учеников

Я программирую на PHP, пригодятся ли мне знания из данной книги?

Да, язык PHP используется для доступа к данным команды SQL, поэтому если вы намерены повышать свой профессиональный уровень, вам необходимо изучить материалы этой книги.

Какой уровень знаний у меня будет после прочтения данной книги и выполнения всех практических заданий?

Вы будете знать SQL на профессиональном уровне, вполне достаточном для разработки сложных баз данных.

В данный момент получили широкое распространение NoSQL базы данных, какой язык используется для работы с такими СУБД?

Для каждой NoSQL СУБД используется свой язык программирования, отличный от SQL, разумеется.

С каким уровнем знаний можно приступить к чтению этой книги?

С начальным уровнем знаний. Вполне достаточно небольшого уровня компьютерной грамотности.

Шаг 2. Теория и практика. Учебная схема данных. Организация работы

Научиться языку SQL на профессиональном уровне по данной книге вполне реально, более того – я не вижу причин, по которым это может не получиться. Разумеется, многое зависит от вашего личного упорства, методичности обучения, системности выполнения практических заданий. Выделите каждый день по три—четыре часа вашего времени, и уже через три недели вы сможете писать SQL-запросы любой сложности.

Книга называется «100 шагов», и это соответствует действительности: это 100 шагов, которые вам необходимо пройти, для того чтобы овладеть SQL и базами данных на профессиональном уровне. Каждый шаг представляет собой отдельную главу книги. Глава книги – обзор определенного вопроса или темы по предмету: базы данных или язык SQL.

Каждая глава поделена, в свою очередь, на следующие разделы:

1. **Введение** – в этом разделе рассказывается, собственно, о предмете или теме, которой посвящается данная глава книги.
2. **Теория и практика** – читателю даются теоретические обоснования темы, разбирается синтаксис рассматриваемых в главе операторов. Приводятся понятные и доступные примеры.
3. **Важные замечания** – в любой теме есть свои особенности, свои нюансы, эти нюансы и обозреваются в разделе.
4. **Вопросы учеников** – здесь я отвечаю на наиболее популярные и интересные вопросы, которые задавали мне слушатели моих курсов.
5. **Контрольные вопросы и задания для самостоятельного выполнения** – вопросы по уроку, вопросы и задания, которые вам необходимо решить самим.

Общая схема процесса обучения, или Как читать данную книгу

Книгу следует рассматривать как учебное пособие, и я не стану скрывать, что в книге используются материалы моих курсов по обучению базам данных и языку SQL школы sqladv.ru.

Внимательно изучите теорию, запомните, проанализируйте синтаксис команд.

Все примеры необходимо прорешивать, самостоятельно писать запросы, создавать схемы, размышлять над теоретическим обоснованием примера, то есть стараться понять, почему получается так, а не иначе.

Практические задания также необходимо решать самостоятельно, и при решении данных заданий следует обратиться к материалам из теории.

Особое внимание следует уделить вопросам, рассматриваемым в разделах «Важные замечания» и «Вопросы учеников». Именно в этих разделах, как правило, содержится наиболее важная и полезная информация.

Несколько терминов

Иногда в тексте книги могут встретиться некоторые сокращения и специальные термины; расшифруем их значение.

СУБД – система управления базами данных – совокупность программных средств, обеспечивающих управление созданием и использованием баз данных. Наиболее распространенные: ORACLE, MS SQL, mySQL, PostgreeSQL.

БД – база данных – совокупность данных, хранимых в соответствии со схемой данных, манипулирование которыми выполняют в соответствии с правилами средств моделирования данных.

Учебная схема

Для занятий нам понадобится учебная схема данных.

И есть хорошая новость.

Многие примеры начального курса могут быть выполнены онлайн, для этого вам потребуется просто перейти по заданной ссылке и заполнить некоторые поля.

Итак, большинство примеров и практических заданий вы сможете выполнить на онлайн-сервисах.

Первый сервис предоставляет компания ORACLE для разработки, я уже зарегистрировал там аккаунт, и сейчас там уже есть все таблицы, которые нам потребуются в учебе.

Также там есть все необходимые заполненные данные для выполнения учебных задач и запросов.

Вам достаточно пройти по ссылке

https://apex.oracle.com/pls/apex/

и ЗАПОЛНИТЬ регистрационную информацию.

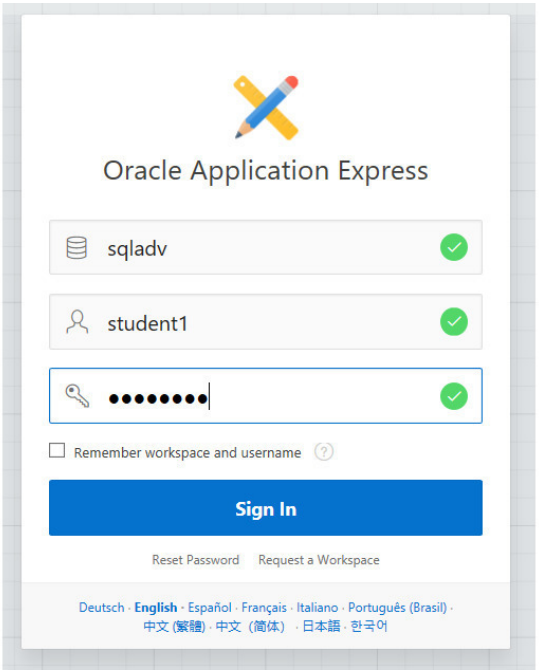


Рисунок 1. Форма авторизации в APPEX

Первое поле сверху мы заполняем SQLADV, во второе поле мы вносим имя пользователя student1 и заполняем пароль – также student1.

Также будут работать учетные записи: student2/, student2, student3/, student3… student11/, student11).

Перед вами откроется среда разработки.

Выберите пункт меню SQL Workshop, а дальше SQL ComMANd.

Пред вами откроется среда выполнения SQL-запросов.

Напишите следующий учебный запрос:

SELECT * FROM AUTO;

Нажмите кнопку RUN SQL, и далее в нижней части экрана должен появиться результат выполнения запроса.

Это и будет тот сервис, которым вы в основном сможете пользоваться для выполнения практических заданий.

Данный ресурс предоставлен компанией ORACLE в рекламных маркетинговых целях.

Если не устраивает данный сервис или почему-то этот сервис у вас не работает, тогда существует второй способ.

Вы можете воспользоваться сервисом SQL-Фидель.

Учебная схема, по которой необходимо заниматься, состоит из нескольких таблиц, которые заполнены соответствующими данными.

Введите в поле браузера ссылку
http://sqlfIDdle.com/.
Выберите тип базы данных ORACLE 11 g r2.

Далее вам потребуется загрузить учебную схему, по которой мы будем заниматься. Скрипт учебной схемы находится по следующей ссылке в текстовом документе:
http://sqladv.ru/dev/sql.txt.

Скопируйте содержимое скрипта в поле в левой части экрана и нажмите кнопку BuildSchema.

После чего уже в правой части экрана SQLFIDlle вы сможете писать необходимые запросы и выполнять учебные задания.

Учебные запросы в SQLFIDdle пишутся в текстовом поле в правой части экрана.

После создания схемы напишите следующий учебный запрос:

SELECT * FROM MAN

Нажмите RUN SQL, в нижней части экрана должен появиться результат выполнения запроса.

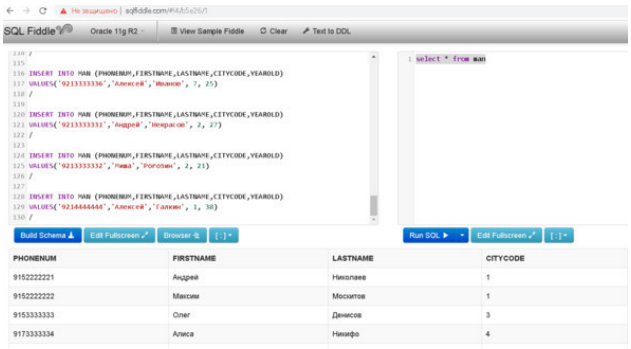


Рисунок 2. Пробный запрос SQLFIDlle

Если все получилось, то вы можете приступать к учебе.

Вопросы учеников

Так все-таки в каком из сервисов лучше выполнять практические задания?

В любом, лично мне больше нравится сервис APEX.

Я пытаюсь залогиниться в сервис APEX, но появляется сообщение об ошибке. В чем может быть дело?

Проверьте правильность ввода имени пользователя и пароля. Самое верхнее поле должно содержать значение SQLADV.

Во время работы с сервисом SQLFIDlle при создании схемы возникает ошибка.

Проверьте, пожалуйста, правильно ли установлен переключатель СУБД – значение выпадающего списка должно быть ORACLE 11g. Также проверьте, скопирован ли скрипт при создании схемы.

Шаг 3. Подготовка к работе. Процесс обучения. Описание интерфейса ORACLE APEX

Подготовка к работе

До шага 51 все занятия и практические упражнения можно выполнить с использованием онлайн-сервисов, в шаге 51 подробно описано, какое программное обеспечение следует установить дополнительно и как это сделать.

Процесс обучения

Как я уже говорил, самое важное в процессе обучения языку SQL – это именно практика, при этом не важно, как много вы знаете. Главное – научиться использовать свои знания в работе.

Тысяча практических задач из данной книги, прорешенных вами, даст гораздо больший эффект, чем изучение одной лишь теории.

Данная книга ориентирована исключительно на практическую работу. Теория здесь поясняется практическими примерами.

Изучив теорию и синтаксис выражения, сразу приступайте к практике, чтобы понять, как работает данный синтаксис, данное выражение.

Практические задания находятся сразу после пояснения теоретических положений. Примерный текст практического задания выглядит следующим образом:

> Выберите из таблицы автомобилей (AUTO) машины синего и зеленого цветов.

Попробуйте сначала самостоятельно написать запрос.

Сравните свое решение с решением, приведенным в книге.

Если сразу не получается, тогда внимательно изучите решение, приведенное в книге, и постарайтесь понять сам принцип, как решается данная задача, что является главным в решении.

После решения каждого из заданий переходите к заданиям для самостоятельного выполнения.

В данной книге рассматривается более тысячи практических примеров и упражнений по SQL.

Кроме того, в книге есть не только практические задачи – после примеров в некоторых шагах встречаются контрольные вопросы по теоретическим материалам.

Такие вопросы необходимы для более четкого понимания изучаемой темы.

Внимательно изучите вопрос и постарайтесь дать на каждый вопрос развернутый и подробный ответ.

Некоторые названия таблиц заменены их смысловыми обозначениями. Вам следует самостоятельно понять, исходя из смысла слова, в какой таблице находятся соответствующие данные.

На самом деле основных таблиц всего три, так что это будет несложно. Например, если речь идет об авто, тогда это таблица AUTO, если речь идет о людях, покупателях, тогда имеется в виду таблица MAN, если о городах – CITY.

Вот несколько практических советов, как улучшить процесс обучения SQL.

Составьте карточки

Наиболее сложные, трудно запоминаемые теоретические вопросы и ответы лучше записывать на карточки, чтобы в конце занятий повторять сложные темы.

Также рекомендуется повторять эти карточки через каждые 10 глав книги.

Данная карточка может выглядеть следующим образом. С одной стороны пишется ключевой вопрос: «*Какой оператор в SQL-запросах отвечает за группировку данных?*»

С другой стороны ответ: «*Оператор GROUP BY*».

И далее несколько примеров запросов с использованием данного оператора.

Составляйте свой список вопросов

Для себя составьте дополнительные вопросы и ответьте на них.

Например, мы изучили агрегатные функции SUM, MAX, MIN, а как работает агрегатная функция COUNT? Не бойтесь задавать себе сложные вопросы и изучать новую информацию.

Создайте свое задание

Придумайте свои практические задания и порешайте их.

В дополнение к практическим заданиям из книги продумайте свои собственные и попробуйте их решить.

Например, у вас есть практическое задание: выбрать из таблицы AUTO все автомобили марки BMW синего цвета.

Выбрать из таблицы AUTO все автомобили марки BMW синего и зеленого цветов.

Интерфейс ORACLE APEX

Онлайн-сервис ORACLE APEX обладает рядом дополнительных возможностей, которые будут нам помогать в процессе обучения. Поэтому, если вы используете для выполнения практических задач APEX, рекомендую ознакомиться с данным материалом.

К дополнительным возможностям сервиса APEX относится просмотр объектов схемы данных.

После входа в сервис APEX следует воспользоваться пунктом меню SQL Workshop, выберите подпункт меню OBJECT BROWSER.

В левой части экрана располагается список объектов, где через выпадающий список указывается, объекты какого типа отражаются в списке.

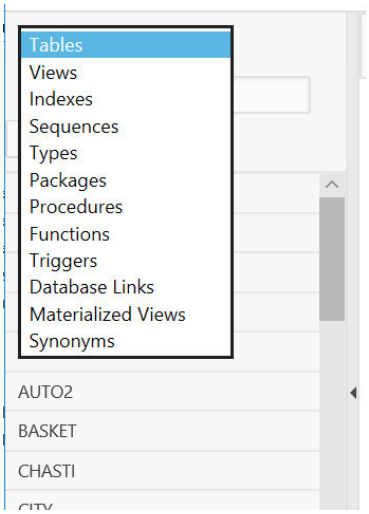


Рисунок 4. Список объектов

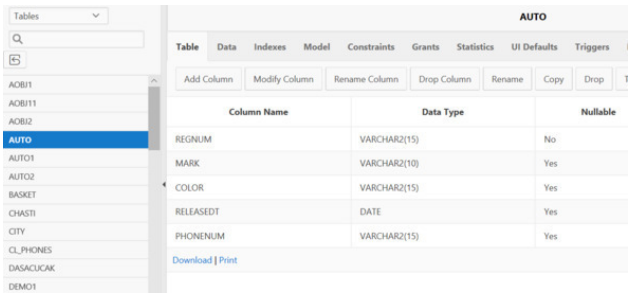
Выбор соответствующего типа объекта покажет список объектов заданного типа. Щелчок по заданному объекту позволяет отобразить структуру и свойства заданного объекта.

Просмотр структуры таблиц

Выберите в выпадающем списке в левой верхней части формы TABLEs. Список отразит все таблицы, которые вам доступны.

Выберите любую из таблиц.

В правой части страницы отразится структура выбранной таблицы.



Также доступны следующие вкладки, которые показывают сведения о таблице:

- TABLE – структура избранной таблицы;
- Data – данные избранной таблицы;
- INdexes – сведения об индексах заданной таблицы;
- ConstraINts – ограничения заданной таблицы;
- Grants – права базы данных по заданной таблице;
- SQL – SQL-код таблицы. Если вам необходимо посмотреть SQL-код таблицы, тогда следует обратиться к этой вкладке.

Просмотр кода процедур

В некоторых шагах мы обращаемся к исходному коду процедур и функций.

Выберите в выпадающем списке одно из следующих наименований: SEQUENCEs, Function, Procedures, Packages.

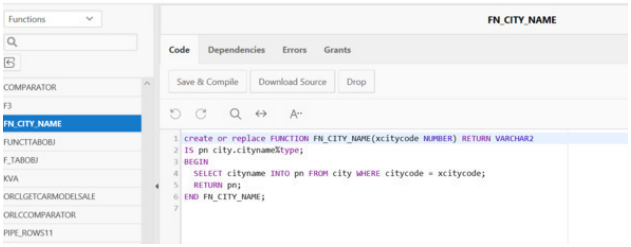


Рисунок 5. Исходный код функции Fn_CITY_NAME

В правой части веб-страницы будет отображен исходный код выбранного объекта.

Вопросы учеников

Сколько времени следует уделять занятиям?

Рекомендую уделять занятиям не менее четырех часов в день, тогда за 20 дней вы успеете пройти все шаги.

Если я все же не могу найти ответ на вопрос или не могу решить задание, что мне делать?

На сайте sqladv.ru есть ссылка на нашу группу в «Фейсбуке», там вы наверняка найдете ответ и решение задачи, с которой испытываете трудности.

В SQLFIDele есть такие же возможности по просмотру и редактированию таблиц, как в ORACLE APEX?

Нет, *SQLFIDele* – это менее сложный инструмент, тем не менее его возможностей достаточно, чтобы выполнить большинство практических заданий из этой книги.

Сколько примерно времени в пропорции уделять теории, а сколько посвятить практике?

Лучше всего из четырех часов рабочего времени следует один час уделить теории, а три часа – практике. Таким образом, примерно 80 процентов вашего учебного времени должно занимать выполнение практических задач.

Контрольные вопросы и задания для самостоятельного выполнения

1. Сколько рекомендуется тратить времени на занятия?
2. Как сделать карточки с наиболее сложными для понимания вопросами?
3. Как в APEX посмотреть все таблицы схемы?
4. Как в APEX посмотреть структуру заданной таблицы?

Шаг 4. Таблицы в базе данных

Теория

База данных – это, прежде всего, таблицы. Таблицы базы данных можно представить как таблицы в WORD или EXCEL, где в каждой ячейке содержатся определенные данные, но также есть и некоторые отличия.

Дело в том, что таблицы в базах данных создаются по некоторым правилам, и вот основные правила для таблиц в базе данных.

Так как мы изучаем SQL-диалект ORACLE СУБД, то данные правила справедливы именно для СУБД ORACLE:

• любая таблица в базе имеет уникальное наименование в рамках схемы данных;

• у каждой таблицы всегда есть заданное количество колонок: больше нуля и меньше 1024;

• каждая колонка также должна иметь уникальное наименование, но уже в рамках данной таблицы;

• в таблице в базе данных может быть практически неограниченное количество строк, здесь ограничения касаются только объема диска базы данных;

• для данных в таблице можно создавать ограничения. Ограничения касаются всех данных в колонке, на которую установлено ограничение;

• имена таблиц, имена колонок имеют ограничения по количеству символов и не могут называться зарезервированным словом, например командой из языка SQL или PL SQL. Также наименование колонки таблицы не должно начинаться с цифр;

• имя колонки в рамках таблицы также должно быть уникальным.

Создание таблиц по указанным правилам – это первый шаг в разработке базы данных.

Таблицы в базе, состав колонок таблицы должны производиться в соответствии со стандартами проектирования реляционной базы данных.

Работа со структурой таблиц, данными в таблицах осуществляется с помощью языка запросов SQL.

Одна или несколько колонок в таблице могут быть обозначены как первичный ключ.

Первичным ключом обозначаются колонки таблицы, содержащие набор уникальных значений, по которым мы можем однозначно идентифицировать строчку в рамках этой таблицы. Первичный ключ не может содержать пустые значения, так как всегда имеет ограничение NOT NULL.

Вторичный ключ – так обозначается колонка таблицы, в которой есть данные, используемые для связи с другой таблицей.

Таблицы нашей учебной схемы

Наша учебная схема очень проста и состоит всего лишь из четырех таблиц.

Первая таблица MAN содержит сведения о людях, которые приобрели машины.

Колонки таблицы MAN:

• PHONEnum – уникальный телефонный номер человека, первичный ключ для таблицы MAN, содержит текстовые данные;

• CITYCode – код города, вторичный ключ для связи с таблицей CITY;

• FirstName – имя человека (текстовые данные);

• LAStName – фамилия человека (текстовые данные);

• YearOld – возраст человека (числовые данные).

Таблица CITY – справочник городов, состоит из трех колонок:

• CITYCODE – уникальный код города, ключевое поле для таблицы CITY (числовые данные);

• CITYNAME – наименование города (текстовые данные);

• PEOPLES – население города, количество человек, которые проживают в городе (числовые данные).

Таблица AUTO – сведения об автомобилях автосалона.

Колонки таблицы AUTO:

• REGnum – уникальный регистрационный номер автомобиля (содержит текстовые данные);

• PHONEnum – телефонный номер покупателя, вторичный ключ для связи с таблицей MAN;

• MARK – марка авто (текстовые данные);

• COLOR – цвет авто (текстовые данные);

• ReleASeDT – дата создания авто, дата/время (специальный тип данных).

Таблица AUTO1 является копией таблицы AUTO и имеет те же колонки, что и таблица AUTO, и достаточно похожие данные, эта таблица используется в нескольких учебных заданиях (так же, как CITY1, MAN1).

Следующее изображение показывает основные таблицы в учебной базе данных в виде схемы:

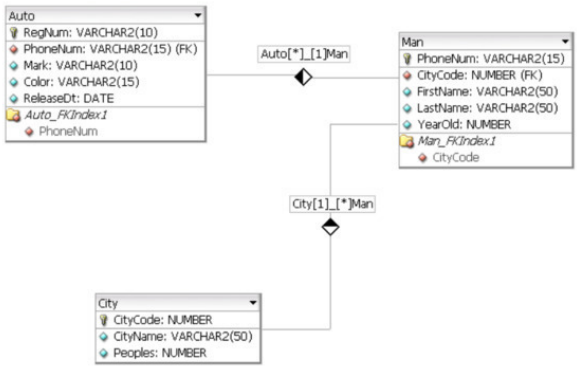


Рисунок 6. Учебная схема базы данных

Вопросы учеников

Вы сказали, что наименования таблиц должны быть уникальны в рамках одной схемы. Что такое схема?

В СУБД есть понятие схемы – это особая логическая область, ассоциированная с заданной учетной записью, которая объединяет несколько объектов базы данных.

Почему телефонный номер покупателя PHONEnum – текстовое поле, разве оно не должно быть числовым?

Иногда телефонный номер заполняют со скобками. Чтобы разрешить это противоречие, я использовал текстовый (VARCHAR2) тип данных для этой колонки; кроме того, так сделать правильно, так как это упрощает поиск нужных нам номеров по префиксу.

Какие команды SQL позволяют изменять структуру таблицы, добавлять новые колонки, например?

Это команда ALTER TABLE, с которой мы познакомимся чуть позже.

Контрольные вопросы и задания для самостоятельного выполнения

1. Могут ли колонки разных таблиц называться одинаково?
2. Что такое первичный ключ, какие колонки (первичные ключи) есть в нашей учебной схеме?
3. Какие данные находятся в учебной таблице CITY?
4. Какая колонка в таблице MAN нашей учебной схемы содержит данные о возрасте человека?

Шаг 5. Типы данных

Для удобства в SQL все данные разделены на различные типы: например, есть строковый тип, к которому относятся только строки и текст; есть целочисленный тип, к нему можно отнести только целые числа; определен специальный тип данных для чисел с плавающей точкой.

Каждой колонке в таблице назначается свой определенный тип данных, то есть каждая колонка таблицы может сохранять данные строго заданного типа и никаких данных другого типа, отличного от этого. Например, в одной из колонок могут находиться только строки и текст, а в другой – только числа.

Типов данных в SQL ORACLE-диалекте множество, мы же рассмотрим самые основные из них.

Теория и практика

Ниже приведена таблица основных типов данных, используемая в SQL ORACLE. В таблице колонка-размер означает, какой максимальный объем информации сможет вместить этот тип данных. Например, тип данных VARCHAR2 может вместить в строку длиной не более 4000 символов.

Типы данных	Размер	Описание
char(размер)	Максимальный размер 2000 байт.	Для текстовых данных не превышающих 2000 символов. Статический выделение памяти под текст. Если сохраняемое значение короче, строка дополняется пробелами до 2000 символов. В скобках указывается количество символов фиксированной длины
nvarchar2(размер)	Максимальный размер 4000 байт.	Специальный тип для сохранения текстовых данных в формате UNICODE. В скобках указывается – количество сохраняемых символов в кодировке Unicode переменной длины.
varchar2(размер)	Максимальный размер 4000 байт.	Для строк и текста не превышающих 4000 символов. В скобках указывается количество сохраняемых символов переменной длины.
CLOB	Максимальный размер 2GB.	Для сохранения больших тестов. Символьные данные переменной длины.
raw	Максимальный размер 2000 байт.	Для двоичных данные переменной длины
number(точность,масштаб)	Точность может быть в диапазоне от 1 до 38. Масштаб может быть в диапазоне от -84 до 127.	Например, number (14,5) представляет собой число, которое имеет 9 знаков до запятой и 5 знаков после запятой.
numeric(точность,масштаб)	Точность может быть в диапазоне от 1 до 38.	Например, numeric(14,5) представляет собой число, которое имеет 9 знаков до запятой и 5 знаков после запятой.
decimal(точность,масштаб)	Точность может быть в диапазоне от 1	Например, decimal (5,2) — это число, которое имеет 3 знака перед запятой

таблица. Типы данных

	до 38.	и 2 знака после .
date	date может принимать значения от 1 января 4712 года до н.э. до 31 декабря 9999 года нашей эры.	Используется для хранения информации дата время.

Таблица. Типы данных

Важные замечания

Для удобства (во избежание излишней путаницы) в учебных примерах для этой книги рассматриваются в основном только три типа данных, однако нам достаточно этих типов для решения большинства учебных задач и понимания учебного материала.

Основные типы данных, используемые в книге в практических задачах:

• VARCHAR2 (n) – тип для хранения текстовой информации, в скобках указывается максимальное количество символов в строке. Данный тип используется при работе со строковыми данными разной длины, память под такие данные выделяется динамически;

• NUMBER – тип данных для хранения числовой информации, причем можно использовать как для целых чисел, так и для чисел с плавающей точкой;

• DATE – специальный тип данных для сохранения специальной информации – дата-время, например дата и время создания записи, дата и время электронной подписи документа, дата и время заключения сделки.

Эти типы данных достаточно часто используются на практике и применяются в работе.

Вопросы учеников

Вы рассказали про тип данных VARCHAR2 для хранения строк, но в виде строки можно записать и числа, и даты тоже. Зачем так много разных типов, может, они не нужны?

Да, вы можете сохранять числа как строки, но в дальнейшем вам будет сложно работать с этими данными, обработка будет затруднена. Например, со строками нельзя производить математических вычислений, и вам придется применять операции преобразования, что негативно скажется на качестве и скорости работы вашей программы.

Типы данных в других SQL СУБД, отличных от ORACLE, также различаются?

Есть незначительные различия, например в MS SQL используется VARCHAR, а не VARCHAR2, или вместо CLOB используется тип TEXT. Необходимо обратиться к соответствующему разделу документации выбранной СУБД, чтобы понять, какие именно типы данных различаются.

Почему в ORACLE SQL используется именно VARCHAR2, а не просто VARCHAR, как в MS SQL-сервер?

В ORACLE SQL тоже существует тип VARCHAR, но исторически сложилось так, что в ORACLE SQL между этими двумя типами существует разница:

• VARCHAR может хранить до 2000 символов, а VARCHAR2 может хранить до 4000 символов;

• если мы объявим тип данных как VARCHAR, то будет зарезервировано место для пустых NULL VALUES.

Поэтому чаще всего на практике в ORACLE используется VARCHAR2.

Контрольные вопросы и задания для самостоятельного выполнения

1. Для чего используется тип DATE?
2. Нам необходимо в одной из колонок таблиц сохранять целые числа, какой тип следует использовать?
3. Какой тип правильно использовать для хранения данных о цвете автомобиля?
4. Какой тип данных необходимо использовать для хранения сведений о росте человека?

День второй

Шаг 6. Создание таблиц

Как мы уже поняли, основная информация, с которой нам предстоит работать в базе данных, находится в таблицах. Из шага 4 (таблицы в базе данных) про таблицы мы узнали следующие важные сведения:

- каждая таблица обладает уникальным наименованием;
- у таблицы обязательно должны быть колонки, каждая из которых обладает уникальным наименованием в рамках этой таблицы;
- для каждой колонки таблицы задается свой тип данных (про типы данных подробно рассказывается в предыдущем шаге).

Важные замечания

1. Имена таблиц, имена колонок в SQL имеют ограничение по количеству символов и не могут называться зарезервированным словом, таким как команда из языка SQL.

Например, нельзя назвать таблицу или колонку GROUP, это зарезервированное слово, часть команды GROUP BY, или же недопустимо называть таблицу или колонку таблицы FROM, SELECT, INSERT, KEY.

2. Имена таблиц и имена колонок не могут начинаться с цифр; также принято использовать при именовании таблиц и колонок латинские буквы, хотя, впрочем, вполне возможно назвать таблицы и колонки таблиц на русском, китайском и даже хинди (я видел и такое), но подобные выкрутасы не приветствуются.

3. Очень желательно, чтобы наименования таблиц, а также наименования колонок таблиц отражали смыл относительно данных, которые содержатся в этих таблицах, например: MANS, CARS, STAFF – люди, машины, персонал, GOODS – товары, ITEMS – элементы.

Теория и практика

Теперь, когда мы разобрались с теоретической частью, самое время заняться практическими упражнениями.

Разберемся на практике, как создавать таблицы в базе, используя язык SQL.

Для создания таблиц используется специальная команда SQL CREATE TABLE.

Синтаксис упрощенный.

```
CREATE TABLE имя таблицы (
Column_NAME1 column_type (NUMBER, или VARCHAR2 (n) или DATE) primary KEY,
Column_NAME 2 column_type (NUMBER, или VARCHAR2 (n) или DATE),
Column_NAMEn column_type (NUMBER или VARCHAR2 (n) или DATE)
);
```

Для простоты на начальном этапе в наших практических примерах мы будем использовать три основных типа данных.

(NUMBER, VARCHAR2 (n), DATE), соответственно, для хранения чисел, строковых данных и данных календарного типа (дата/время).

Для текстового типа VARCHAR2 (n) после VARCHAR2 в скобках указывается количество символов для данной колонки.

Итак, сначала идет команда создания таблицы CREATE TABLE, далее – наименование таблицы: MANS, GOODS, ITEMS или любое другое.

Далее в скобках через запятую перечисляются наименования колонок и тип колонок.

Вот несколько примеров, как создавать таблицы в языке SQL:

1. Создать таблицу «Мебель»:

- артикул;
- наименование;
- количество;
- номер партии.

```
CREATE TABLE furnit (artikl VARCHAR2 (50) PRIMARY KEY, NAME VARCHAR2 (50),partCOUNT NUMBER, partnum NUMBER);
```

2. Создать таблицу «Корзина для веб-магазина»:

- артикул;
- наименование товара;
- имя покупателя;
- количество;
- дата покупки.

```
CREATE TABLE shopINgcart (
article VARCHAR2 (50) PRIMARY KEY
,itemNAME VARCHAR2 (50)
,buyerNAME VARCHAR2 (50)
,itemCOUNT NUMBER
,dtbuy DATE
);
```

Создать таблицу «Предприятие»:
• название бригады;
• номер бригады;
• количество человек;
• дата создания;
• направление деятельности.

```
CREATE TABLE plant
(
NAMEteam VARCHAR2 (15),
numteam NUMBER PRIMARY KEY,
MANCOUNT NUMBER,
crDATE DATE,
dirToDo VARCHAR2 (30)
);
```

Вопросы учеников

Можно ли использовать заглавные буквы в языке SQL и когда это допустимо?

Язык SQL не зависит от регистра, то есть при составлении команд можно писать и заглавными, и строчными буквами.

Примеры

Create TABLE Tab1 (TABno INteger PRIMARY KEY, NAME VARCHAR2 (10));
Create TABLE Tab1 (TABno INteger PRIMARY KEY, NAME VARCHAR2 (10));
CREATE TABLE Tab1 (TABNo INTEGER PRIMARY KEY, NAME VARCHAR2 (10));

Как переносить команды SQL на другую строку, если в одну строчку не помещается, существуют ли какие-то специальные правила?

Язык SQL допускает достаточно вольный перенос строк, главное, не разделять этим переносом осмысленные команды, а также соблюдать последовательность команд.

Примеры

Можно написать так:

```
CREATE TABLE TAB1 (TABno INteger PRIMARY KEY, NAME
VARCHAR2 (10));
```

А можно и так:

```
CREATE TABLE
TAB1 (
TABno INteger PRIMARY KEY,
NAME VARCHAR2 (10));
```

А вот такая запись уже неверна:

```
CREATE TABLE TAB1 (TABno INteger PRIMARY
KEY, NAME VARCHAR2
(10));
```

Еще один пример неверной записи:

```
CREATE TABLE
PRIMARY KEY
TAB1 (TABno INteger,
NAME VARCHAR2 (10));
```

Контрольные вопросы и задания для самостоятельного выполнения

1. Найдите ошибку в скрипте создания таблицы.

CREATE TABLE ORACLE1 (81NAME VARCHAR2 (20), ITEMS NUMBER);

2. Найдите ошибку в другом скрипте создания таблицы.

CREATE TABLE DELTA (SELECT VARCHAR2 (20), COUNT NUMBER);

3. Можно ли при наименовании таблицы использовать строчные и заглавные символы?

4. Создайте самостоятельно таблицу «Запчасти», задайте имена колонок и название таблицы сами, правильно определите типы данных.

Таблица «Запчасти»:
- номер запчасти;
- марка авто;
- название запчасти;
- количество данных запчастей;
- стоимость запчасти.

Создайте самостоятельно таблицу «Фото», задайте имена колонок и название таблицы сами, правильно определите типы данных.

Таблица «Фото»:
- название фото;
- размеры;
- подпись;
- дата создания.

Создайте самостоятельно таблицу «Уроки» («Занятия»), задайте имена колонок и название таблицы сами:

- название занятия;
- день недели;
- дата начала занятия;
- дата окончания занятия.

Шаг 7. Структура таблицы

Введение

Мы научились создавать таблицы на предыдущем шаге. Таблицы и колонки таблиц, их названия, расположение, последовательность колонок, типы данных колонок называются структурой таблицы.

Структуру таблицы можно менять, то есть добавлять новые колонки в таблицу, удалять колонки из таблицы, менять типы данных у заданной колонки. Также, если таблица нам больше не нужна или просто надоела, существует возможность такую таблицу удалить.

Теория и практика

Существует несколько команд для изменения структуры таблицы, добавления, удаления или изменения типа данных колонки таблицы.

Все эти команды объединяет то, что они начинаются с ключевой команды ALTER TABLE.

Добавление колонки.

Добавляем новую колонку к нашей таблице.

Синтаксис:

ALTER TABLE TABLE_NAME ADD (column_NAME column_type);

TABLE_NAME – наименование таблицы.

Column_NAME – наименование колонки.

Column_type – тип данных колонки (VARCHAR (n) или NUMBER или DATE).

Примеры:

Пусть у нас есть таблица GOODS, необходимо добавить колонку itemprice типа NUMBER, цена изделия.

ALTER TABLE GOODS ADD (itemprice NUMBER);

Пусть у нас есть таблица MANS, необходимо добавить колонку DATEreg типа DATE, дата регистрации, и колонку patronymic – отчество VARCHAR2 (50).

ALTER TABLE MANS ADD (DATEreg DATE);
ALTER TABLE MANS ADD (patronymic VARCHAR2 (50));

Удаление колонки

Также мы можем удалить колонку из заданной таблицы с помощью специальной SQL-команды DROP COLUMN.

Синтаксис:

ALTER TABLE TABLE_NAME DROP COLUMN column_NAME;

Примеры:
Пусть у нас есть таблица GOODS, необходимо удалить колонку COLOR.

ALTER TABLE GOODS DROP COLUMN COLOR;

Пусть у нас есть таблица MANS, необходимо удалить колонку YEAROLD.

ALTER TABLE MANS DROP COLUMN YEAROLD;

Меняем тип данных для колонки таблицы.

Синтаксис изменения типа колонки:

ALTER TABLE TABLE_NAME MODIFY (column_NAME DATA_type);

Column_NAME – наименование колонки.
Data_type – тип данных колонки (VARCHAR (n) или NUMBER или DATE).

Примеры:
– заменить в таблице MANS тип поля NAME на VARCHAR2 (90);

ALTER TABLE MANS MODIFY (NAME VARCHAR2 (90));

– заменить в таблице GOODS тип поля INSERT_DATE на DATE;

ALTER TABLE GOODS MODIFY (INSERT_DATE DATE);

Удаляем таблицу из базы данных.

Синтаксис команды SQL для удаления таблицы:

DROP TABLE TABLE_NAME;

Здесь TABLE_NAME – наименование таблицы.

Примеры:
– удалить таблицу DOC;

DROP TABLE doc;

– удалить таблицу ITEMS;

DROP TABLE ITEMS;

– удалить таблицу BILLING_PERIOD со связанными данными в таблице PERIODS.

DROP TABLE BILLING_PERIODS CASCADE;

Важные замечания

1. При выполнении действий по изменению структуры таблицы следует быть особенно осторожным, следует тщательно взвешивать свои действия: восстановление таблицы в прежнем виде может быть затруднительно или невозможно.

2. Если вы используете команды изменения типов данных и встречаетесь с ошибкой **ORA-01439, модифицируемый столбец при смене типа данных должен быть пуст**. Сохраните данные в столбце и используйте специальные преобразования, о которых будет рассказано в следующих шагах.

3. В некоторых случаях удаление таблицы или колонки таблицы будет запрещено, поскольку могут быть еще таблицы со связанными данными. Требуется сначала удалить данные в связанных таблицах, а уже затем удалять таблицу либо колонку. Или же воспользоваться специальной командой DROP CASCADE.

Вопросы учеников

Можно ли переименовать таблицу?

Да, вполне, и для этого есть две команды:

ALTER TABLE TABLE_NAME RENAME TO new_TABLE_NAME;

или же

RENAME <old_TABLE> TO <new_TABLE>

Универсальный же синтаксис предполагает использование ALTER TABLE.

Примеры:

Переименуем таблицу с названием STAFF в EMP:

ALTER TABLE STAFF RENAME TO emp;

Переименуем таблицу с названием TRADES в TRADE:

ALTER TABLE trades RENAME TO trade;

Можно ли переименовать столбец в таблице?

ALTER TABLE TABLE_NAME RENAME COLUMN old_column_NAME to new_column_NAME;

Пример:

Переименовать колонку с наименованием NAME в таблице STAFF в колонку LASTNAME:

ALTER TABLE STAFF RENAME COLUMN NAME TO LASTNAME;

Контрольные вопросы и задания для самостоятельного выполнения

1. С помощью какой команды можно добавить колонку к заданной таблице?
2. Повторите команды изменения структуры таблицы.
3. К таблице «Запчасти» добавить колонку «Вес запчастей» – напишите SQL-команду.
4. Удалите из таблицы FOTO колонку Fotosize – напишите SQL-команду.
5. У нас есть таблица «Предметы» («Уроки в школе»), необходимо добавить колонку «Преподаватель» – напишите SQL-команду.
6. Удалите из базы данных таблицу FOTO.

Шаг 8. Первичные ключи. Вторичные ключи

Введение

Первичный ключ – это сочетание значений колонок таблицы, уникально определяющее каждое значение таблицы. Такие колонки называются первичным ключом. Первичные ключи таблицы необходимы для поддержания целостности базы данных.

Любая колонка в таблице может быть обозначена как первичный ключ, это уникальные колонки, в которых только уникальные значения, по которым мы можем однозначно идентифицировать строчку в рамках этой таблицы.

Колонку для связи таблицы с другой таблицей называют вторичным ключом, то есть если есть две таблицы связаны по одной или нескольким колонкам, такая колонка во второй связанной таблице называется вторичным ключом.

Вторичный ключ также называют внешним ключом таблицы.

Теория и практика

В нашем примере есть две таблицы (таблица CITY, MAN по колонке CITYCODE), в таблице CITY CITYCODE является первичным ключом.

В таблице MAN CITYCODE будет вторичным ключом.

Синтаксис создания первичного ключа:

```
CREATE TABLE TABLE_NAME
(
column1 DATAtype NULL/NOT NULL,
column2 DATAtype NULL/NOT NULL,
…
CONSTRAINT constraINt_NAME PRIMARY KEY (column1, column2, … column_n)
);
```

Также возможно создание первичного ключа с помощью конструкции ALTER TABLE:

```
ALTER TABLE TABLE_NAME ADD CONSTRAINT constraINt_NAME PRIMARY KEY (column1, column2, … column_n);
```

Синтаксис создания вторичного ключа:

```
CREATE TABLE TABLE_NAME (
column1 DATAtype NULL/NOT NULL,
…
CONSTRAINT fk_column
FOREIGN KEY (column1, column2, … column_n)
REFERENCES parent_TABLE (column1, column2, … column_n));
```

Добавление вторичного ключа с помощью конструкции ALTER TABLE:

```
ALTER TABLE TABLE_NAME
ADD CONSTRAINT constraINt_NAME
FOREIGN KEY (column1, column2, … column_n)
REFERENCES parent_TABLE (column1, column2, … column_n);
```

Важные замечания

Первичный ключ может состоять из одной или нескольких колонок.
Пример:

```
ALTER TABLE TABLE_NAME ADD CONSTRAINT constraINt_NAME PRIMARY KEY (column1);
```

или же

```
ALTER TABLE TABLE_NAME ADD CONSTRAINT constraINt_NAME PRIMARY KEY (column1, columnN);
```

Вопросы учеников

Обязательно ли обозначать внешний ключ? Почему это не будет работать просто так?

Будет работать, но для поддержания ссылочной целостности необходимо использование конструкций SQL для первичных и вторичных ключей.

Контрольные вопросы и задания для самостоятельного выполнения

1. Повторите материалы данного шага.
2. Поясните, в чем отличие первичного и вторичного ключей.

Шаг 9. Ограничения

Введение

Для колонок таблицы в базе данных можно создавать ограничения.

Ограничения – это специальные синтаксические конструкции уровня колонок таблицы, которые предназначены для поддержания ссылочной целостности данных или для вставки правильных данных согласно бизнес-логике приложения.

То есть ограничения допускают вставку в ячейку таблицы только определенных данных, ограниченных заданными правилами.

Теория и практика

Все ограничения, которые используются в ORACLE SQL, можно разделить на следующие группы:

1. Ограничения на вставку пустых значений NOT NULL

Подобный вид ограничений создается, чтобы огранить вставку пустых значений в базу данных.

Снять данное ограничение можно с помощью команды изменения поля таблицы MODIFY.

Синтаксис

Добавлять ограничения на вставку пустых значений можно при создании таблицы.

```
CREATE TABLE TABLENAME (
Column1 NOT NULL, ColumnN NOT NULL
);
```

Это стандартный синтаксис создания таблицы, к имени колонки добавляется синтаксис ограничения NOT NULL.

Или изменять значения для уже готовой таблицы с помощью команды ALTER TABLE.

```
ALTER TABLE TABLENAME MODIFY ColumnName NOT NULL
```

Примеры

Создание таблицы «Корзина» с ограничением на вставку пустых значений в колонки itemNAME, itemCOUNT.

```
CREATE TABLE shopINgcart (
article VARCHAR2 (50) PRIMARY KEY
,itemNAME VARCHAR2 (50) NOT NULL
,itemCOUNT NUMBER NOT NULL
);
```

Запрет добавления пустого значения в FirstName в таблицу MAN.

```
ALTER TABLE MAN MODIFY FirstName NOT NULL
```

Запрет добавления пустого значения в LAStName.

```
ALTER TABLE MAN MODIFY LAStName NOT NULL
```

То есть значение в колонке LAStName MAN обязательно должно быть заполнено.

Снимаем ограничение на вставку пустых значений:

```
ALTER TABLE MAN MODIFY LAStName NULL
```

После выполнения команды значение в колонке LAStName MAN обязательно должно быть заполнено.

Ограничения на уникальность

Данный вид ограничений позволяет указать, что в заданные колонки необходимо добавлять только уникальные неповторяющиеся значения.

Синтаксис

ALTTER TABLE TABLENAME ADD CONSTRAINT cINs_NAME UNIQUE (columnNAME);

Пример:

ALTTER TABLE CITY ADD CONSTRAINT CITY_uniq UNIQUE (CITYNAME);

Названия городов, только уникальные значения.

Ограничения на вторичный ключ

Подобный вид ограничений мы уже рассматривали, когда изучали первичные и вторичные ключи. Смысл данного ограничения в том, что в колонке некоторой таблицы (вторичный ключ) могут находится только значения, которые есть в другой, основной таблице в колонке первичного ключа.

Синтаксис

```
ALTER TABLE for_TABLE
ADD CONSTRAINT fk_const_NAME
FOREIGN KEY (fk_column)
REFERENCES primary_talbe (pk_column);
```

Здесь for_TABLE, fk_column – таблица, колонка, куда устанавливается ограничение. Проверка значений происходит в таблице primary_talbe и колонке pk_column.

Пример

Здесь для таблицы MAN колонки CITYCODE устанавливается ссылочное ограничение по колонке CITYCODE с таблицей CITY, где CITY является главной таблицей.

```
ALTER TABLE MAN
ADD CONSTRAINT fk_MAN_CITY_CODE
FOREIGN KEY (CITYCODE)
REFERENCES CITY (CITYCODE);
```

Ограничение CHECK на вставку и изменение данных

– вычитание;

Синтаксис

```
ALTER TABLE TABLENAME ADD CONSTRAINT CHECK_NAME
CHECK (condition);
```

Здесь condition – условие, CHECK_NAME – наименование ограничения, TABLENAME – имя таблицы.

Пример

Ограничение в таблице MAN на возраст (YEAROLD) больше 16 лет:

```
ALTER TABLE MAN ADD CONSTRAINT
CHECK_YEAROLD_MAN
CHECK (YEAROLD> 16);
```

Важные замечания

Ограничение уникальности можно также создавать для нескольких колонок таблицы, это делается следующим образом:

```
ALTTER TABLE CITY ADD CONSTRAINT CITY_uniq
UNIQUE (CITYNAME, CITYCODE);
```

При этом отдельно необходимо контролировать вставку пустых значений для соответствующих полей таблицы.

Существуют дополнительные опции для создания ограничений ссылочной целостности:
- On delete cAScade – автоматическое удаление связанных строк по внешнему ключу;
- On delete NULL – значение внешнего ключа устанавливается в NULL.

При создании множества ограничений CHECK необходимо, чтобы между ними не было конфликтов, то есть чтобы правила не противоречили друг другу.

Условие в ограничении CHECK может ссылаться на любой столбец таблицы, но не может ссылаться на столбцы другой таблицы.

Вопросы учеников

Чем ограничение на уникальность UNIQUE отличается от первичного ключа (Primary KEY)?

Первичный ключ в таблице может быть только один, ограничений на уникальность может быть много. В таблице в поле с ограничением уникальности допускается вставка пустых значений. Также для первичного ключа создается специальный индекс (pk).

Если колонка в таблице содержит пустые значения, а мы добавляем к этой колонке ограничение NOT NULL, что произойдет?

Вы получите сообщение об ошибке, и ограничение не будет добавлено.

Можно ли создавать ограничения для колонок BLOB, CLOB?

Нет, некоторые виды ограничений можно создавать только для простых типов полей.

Контрольные вопросы и задания для самостоятельного выполнения

1. Добавьте ограничение для уникальных значений на колонки PHONENUM таблицы AUTO.

2. Добавьте ограничение на вставку пустых значений для колонки YEAROLD в таблице MAN.

3. Добавьте ограничение CHECK для колонки YEAROLD в таблице MAN, чтобы YEAROLD было меньше 100.

4. Добавьте ограничение ссылочной целостности для двух таблиц – AUTO и MAN.

Шаг 10. Индексы

Введение

Индексы – это специальные ссылочные массивы в базах данных. Назначение индексов – ускорение поиска данных, процессов сортировки данных. Обычно индексы увеличивают производительность запросов к базе данных.

Теория и практика

Индексы работают по принципу b-tree, то есть сбалансированной древовидной структуры.

> Для примера: у нас в одной из колонок таблицы есть уникальные значения от 1 до 500 000.
>
> Нам необходимо найти значение 255 000. По ссылочному индексу мы определяем, больше или меньше искомое значение 250 000, то есть половины всех значений, далее больше или меньше искомое значение 260 000, далее мы переходим к нашему значению 255 000.
>
> Мы не просматривали каждую запись таблицы, а нашли наше значение за несколько итераций.

Индексы создаются для определенной колонки (колонок) таблицы.

Процесс пересоздания индексов может занимать значительное время, это необходимо учитывать при операциях вставки и обновления, удаления данных.

Синтаксис

CREATE INDEX IDx_NAME ON TABLE_NAME (column_NAME);

Idx_NAME – наименование индекса;
TABLE_NAME – наименование таблицы, где создается индекс;
column_NAME – наименование колонки, для которой создается индекс.
Пример создания индекса для колонки MARK таблицы AUTO:

CREATE INDEX IDx_AUTO_MARK ON AUTO (MARK);

Реверсивный индекс

Если нам необходимо более часто читать записи, отсортированные в обратном порядке, тогда имеет смысл использовать реверсивные индексы. Например, есть таблица валют, в своих расчетах мы чаще используем данные с более поздней датой курса валют, в этом случае действительно лучше использовать реверсивный индекс для даты курса валют.

Синтаксис

CREATE INDEX IDx_NAME ON TABLE_NAME (column_NAME) REVERSE;

Пример: создание реверсивного индекса для колонки MARK таблицы AUTO.

CREATE INDEX reg_DATE ON AUTO (reg_num) REVERSE;

Удаление индекса

Для удаления индекса используется команда

DROP INDEX IDx_NAME;

Индексы создаются для определенной колонки таблицы.

Процесс пересоздания индексов может занимать значительное время, это необходимо учитывать при операциях вставки и обновления, удаления данных.

Важные замечания

Обычно использование индексов улучшает производительность базы данных, но в таблицах, где предполагается большое количество операций вставок, обновлений, много индексов использовать не рекомендуется. В этом случае производительность базы данных может существенно снизиться.

Индексы рекомендуется создавать на колонках, которые используются в операциях объединения.

Индекс автоматически создается для столбцов первичных ключей и для столбцов, на которых есть ограничение уникальности.

При наименовании индексов следует придерживаться следующего правила: IDx_имя таблицы_имена_колонок.

Вопросы учеников

Если таблица небольшая, в ней не более 200 записей например, нужен ли в такой таблице индекс?

Нет, индексы для такой таблицы, скорее всего, не понадобятся.

Какие типы индексов существуют в различных СУБД?

Существует множество разных типов индексов, но более подробно мы разберем эту тему в следующих шагах.

В моей базе данных фильтр (WHERE) данных, поиск данных всегда осуществляется одновременно по определенному набору колонок. Какие индексы следует использовать в этом случае?

В этом случае создается композитный индекс.
Синтаксис:

CREATE INDEX IDx_NAME ON TABLE_NAME (column_NAME1, column_NAMEn) REVERSE;

Контрольные вопросы и задания для самостоятельного выполнения

1. В таблице есть ограничение уникальности, имеет ли смысл создавать на этой колонке индекс?
2. В некоторой таблице постоянно обновляются записи, следует ли использовать индексы в этой таблице?
3. Создайте индекс на колонку COLOR в таблице AUTO.
4. Создайте реверсивный индекс для колонки YEAROLD в таблице MAN.

День третий

Шаг 11. Простые запросы

Введение

А сейчас отвлечемся на некоторое время от структуры таблиц и поговорим о том, как извлекать данные из базы.

Логично предположить, что если у нас есть данные, нам необходимо их выбирать из базы, обрабатывать и выводить на экран в удобном, понятном, читаемом виде.

В нашей рабочей схеме уже есть три таблицы с данными – это таблицы AUTO, CITY, MAN.

Напомню, что в таблице MAN хранятся сведения о покупателях: их имена, их возраст, CITY – это данные о городах, а таблица AUTO содержит сведения об автомобилях некоторого автосалона.

Если человек приобретает автомобиль, то в таблице AUTO в колонке PHONEnum выставляется номер телефона человека, который приобрел машину.

Для извлечения данных из базы и вывода этих данных на экран используются команды, называемые запросами к базе данных, специальная команда SQL – SELECT. Эта команда является наиболее часто используемой командой в языке SQL и постоянно применяется на практике.

Теория и практика

Начнем с самого легкого запроса, рассмотрим синтаксис самого простого оператора SELECT:

SELECT * FROM TABLE_NAME

Здесь TABLE_NAME – имя таблицы, из которой мы запрашиваем данные.
Символ * означает, что мы выводим на экран данные из всех колонок.
Откройте тестовую среду и выполните запрос

SELECT * FROM CITY;

CITYCODE	CITYNAME	PEOPLES
1	Москва	10000000
7	Котлас	110000
8	Мурманск	400000
2	Владимир	500000
3	Орел	300000
4	Курск	200000
10	Казань	2010000

Рисунок 7. Запрос из таблицы CITY

На экран выведены названия колонок в первой строке, а также данные в каждой колонке из таблицы CITY.

А теперь другой вариант синтаксиса такого же простого запроса SQL:

SELECT column_NAME1, column_NAME1, column_NAMEn FROM TABLE_NAME

В этом варианте вместо звездочки используются наименования колонок и на экран будут выведены только перечисленные колонки из заданной в поле FROM таблицы.

Пример (выполните в нашей тестовой среде):

SELECT CITYCODE, CITYNAME FROM CITY;

CITYCODE	CITYNAME
1	Москва
7	Котлас
8	Мурманск
2	Владимир
3	Орел

Рисунок 8. Запрос к таблице CITY по колонкам CITYCODE, CITYNAME

Результат запроса – на экран выведены только те две колонки, которые мы указали после оператора SELECT.

Существует и другой вариант синтаксиса для SQL-запросов:

SELECT TABLE_NAME.* FROM TABLE_NAME

или

SELECT TABLE_NAME. column_NAME1, TABLE_NAME. column_NAME1, TABLE_NAME. column_NAMEn FROM TABLE_NAME

Прорешаем задания, которые мы уже выполнили, с использованием такого синтаксиса:

SELECT CITY.* FROM CITY

При выполнении этого запроса SELECT получается результат, совершенно аналогичный тому, что и в примерах выше.

Небольшой лайфхак.

Как я составляю запросы? Сначала пишу SELECT *, затем FROM, имя таблицы, выполняю запрос, а уже после перечисляю колонки, которые необходимо вывести на экран, и далее выполняю запрос повторно.

Фильтр строк WHERE в запросе SELECT

Итак, мы научились выводить на экран все данные из заданной таблицы, но как же поступить, если нам необходимо вывести на экран только избранные строки? Допустим, что в заданной таблице миллион строк, а нам необходимо посмотреть из них лишь 10.

К счастью, язык SQL позволяет это сделать. Для этого в языке SQL и в частности в команде SELECT предусмотрен специальный оператор – **WHERE**.

Рассмотрим синтаксис команды **SELECT** с оператором **WHERE**:

SELECT * или перечень колонок FROM TABLE_NAME WHERE условие отбора строк

Примеры:

Выберем названия городов, где население 300 000 человек.

SELECT * FROM CITY WHERE PEOPLES = 300000

Альтернативная форма записи:

SELECT CITY.* FROM CITY WHERE CITY.PEOPLES = 300000

Выражение в WHERE формируется с помощью математических операндов сравнения, рассмотрим этот момент подробнее.

Операнды сравнения

> больше
<меньше
= строгое равенство
или неравенство**! =**

Примеры

Выберем все колонки (*) из таблицы городов, где население больше 300 000 человек.

SELECT * FROM CITY WHERE PEOPLES> 300000

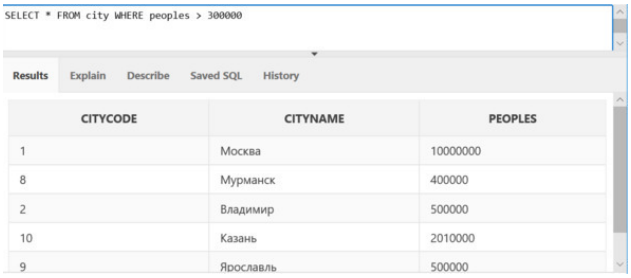


Рисунок 9. Запрос таблице CITY с условием

Выберем название города с кодом города, равным 2, из CIty

```
SELECT citycode,cityname FROM city WHERE citycode = 2
```

Альтернативная форма записи:

```
SELECT city.* FROM city WHERE city.citycode = 2
```

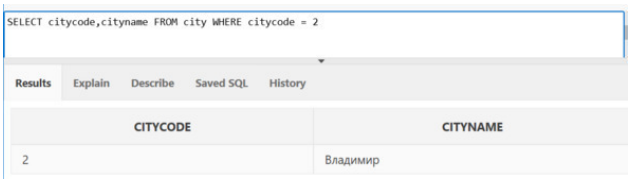


Рисунок 10. Запрос к таблице CITY по заданному CITYCODE

Выберем все имена и фамилии из таблицы MAN:

```
SELECT firstname, lastname FROM man
```

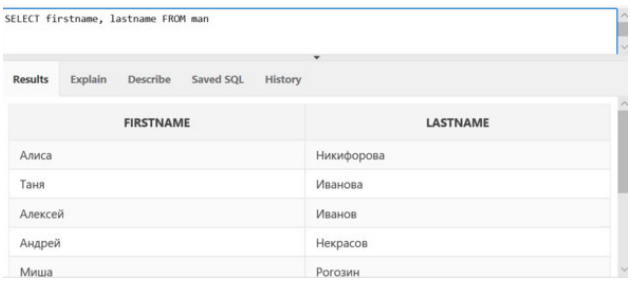


Рисунок 11. Запрос двух колонок к таблице MAN

Все колонки (*) возраст больше 27 лет из таблицы MAN:

```
SELECT * FROM man WHERE yearold > 27
```

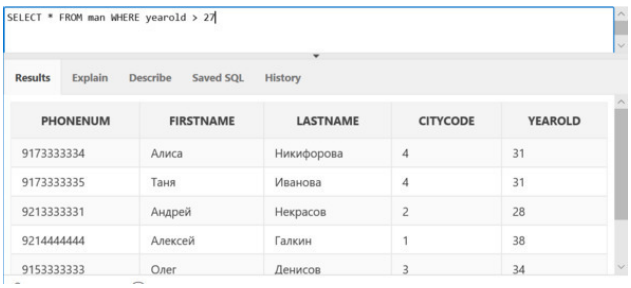


Рисунок 12. Запрос к таблице MAN, где возраст больше 27 лет

Из таблицы AUTO выберем номера автомобилей, выпущенных после 1 февраля 2005 года.

```
SELECT regnum, mark FROM auto WHERE releasedt > date'2005-01-01'
```

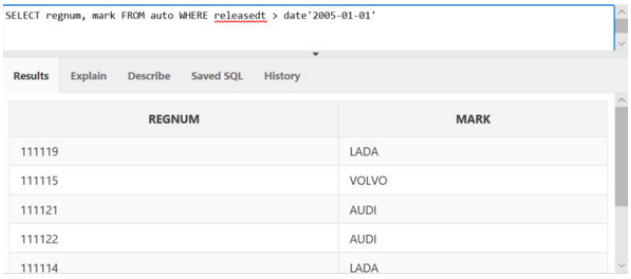


Рисунок 13. Запрос к таблице AUTO с ограничением по дате

Из таблицы AUTO выберем только зеленые автомобили.

```
SELECT * FROM auto WHERE color = 'ЗЕЛЕНЫЙ'
```

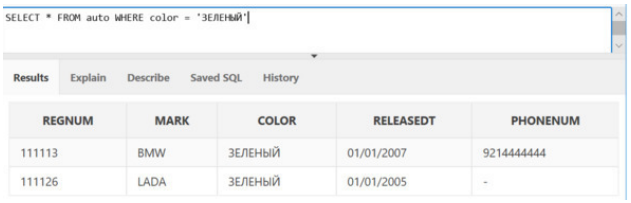


Рисунок 14. Запрос к таблице AUTO, где цвет авто зеленый

Из таблицы MAN выберем только людей с именем Миша.

```
SELECT * FROM man WHERE firstname = 'Миша'
```

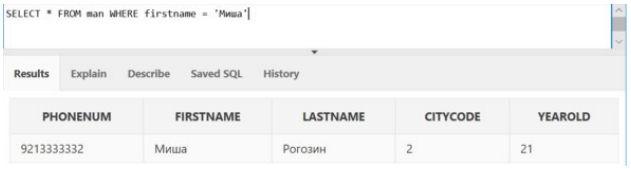


Рисунок 15. Запрос к таблице MAN: выбираем людей с именем Миша

Если осуществляется сравнение строковых данных, то есть тип данных в колонке сравнения VARCHAR, VARCHAR2, то строка сравнения заключается в одинарные кавычки.

Примеры

Выбрать из таблицы MAN все колонки (*), где имя Миша (равно Миша).

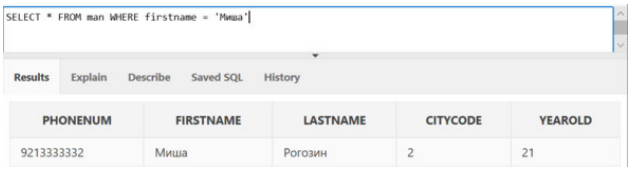


Рисунок 16. Запрос к MAN, где имя равно Миша

Выбрать из таблицы MAN все колонки (*), где имя не Олег (не равно Олег).

```
SELECT * FROM man WHERE firstname != 'Олег'
```

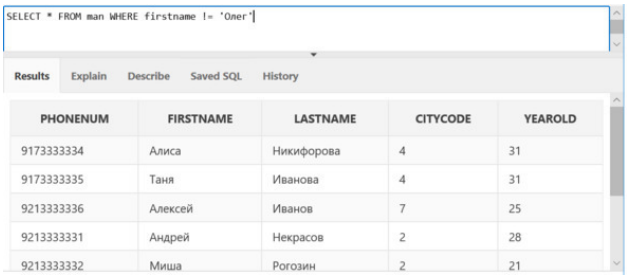


Рисунок 17. Запрос к MAN, где имя не равно Олег

Важные замечания

Несмотря на то что в SQL можно сочетать большие и маленькие буквы, в выражении в одинарных кавычках, при отборе и фильтрации текстовых данных регистр должен соблюдаться, иначе запрос отработает некорректно.

Выражение DATE'YYYY-MM-DD» работает только в СУБД ORACLE, в MS SQL SERVER и POSTGREESQL работа с данными типа «дата» осуществляется по-другому (смотрите подробности документации к этим СУБД).

Следует учитывать, что в некоторых типах баз данных для неравенства можно использовать <> или знак!=, подобную информацию необходимо уточнять в документации к СУБД.

Вопросы учеников

Какой способ написания команды SELECT наиболее часто используется?

Вы можете использовать любой способ записи, но наиболее удобным, с точки зрения синтаксиса и читаемости запроса, я считаю способ с указанием имени таблицы после оператора SELECT.

Так все-таки какой смысл в этой звездочке вместо перечисления колонок?

SELECT * выведет информацию о всех колонках в заданной таблице, и это можно использовать, чтобы посмотреть, какие именно колонки присутствуют и как они называются.

Мы можем использовать форму записи с именем таблицы в фильтре WHERE?

Да, и вот пример. SELECT * FROM MAN WHERE MAN.FIRSTNAME= «Олег».

Контрольные вопросы и задания для самостоятельного выполнения

1. Чем отличаются разные формы записи SQL-запроса SELECT?
2. Выбрать из таблицы MAN (*) людей, где возраст (YEAROLD) человека больше 30 лет.
3. Выбрать из таблицы городов все колонки (*), где город (CITYNAME) называется Москва.
4. Выбрать названия (CITYNAME) городов CITY с населением (PEOPLES) больше миллиона человек.
5. Выбрать телефоны людей из MAN, чья фамилия (LASTNAME) не Денисов.
6. Выбрать информацию о машинах car (*) синего цвета (COLOR).

Шаг 12. Более сложные условия. Знакомимся с логикой выбора строк

Введение

Язык SQL позволяет задавать и более сложные фильтры отбора строк с помощью оператора WHERE. Для этого в языке SQL применяются **логические операнды**, позволяющие комбинировать несколько условий, создавать тем самым сложные логические выражения.

Теория и практика

Итак, **логические операнды** позволяют объединять несколько условий, чтобы создать более сложные критерии выбора строк в операторе WHERE. Разберемся поподробнее, как это работает.

усл1 AND усл2 – логическое И, позволяет объединить несколько условных выражений, так что запрос вернет строку таблицы, если каждое из условий будет верным.

усл1 OR усл2 – логическое Или, позволяет выбрать строки, если одно из заданных условий верно.

NOT усл – логическое отрицание, выбирает строки, если выражение полностью неверно.

AND OR и NOT – как указано выше, можно гармонично сочетать в запросе.

Синтаксис

```
SELECT перечень колонок таблицы через запятую или * FROM
Tablename
WHERE условие1 AND условие2 OR  условие3 NOT условиеn
```

Где TABLENAME – имя таблицы, а условие1…условиеn – различные условия (WHERE) в SQL-запросе.

Последовательность логических операндов может комбинироваться.

Примеры

Разберем действие данных логических операндов на примерах:

Выбрать из таблицы AUTO машины (*) BMW синего цвета (COLOR).

```
SELECT * FROM auto WHERE color = 'СИНИЙ' and mark='BMW'
```

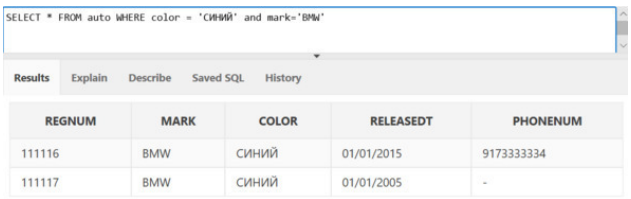


Рисунок 18. Запрос на синие авто BMW

Выбрать из таблицы MAN имена (FIRSTNAME) и фамилии (LASTNAME) людей, которым больше 27 лет и меньше 41 года (YEAROLD).

```
SELECT firstname, lastname FROM man WHERE yearold > 27 and yearold <41
```

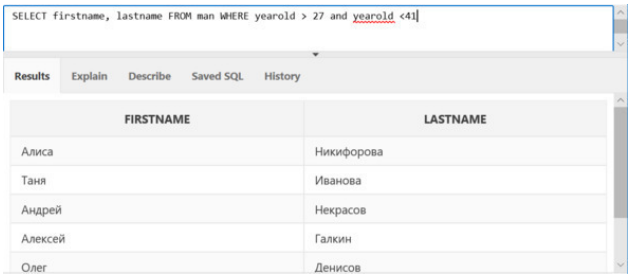


Рисунок 19. Запрос к MAN, где возраст больше 27 и меньше 41

Выбрать из таблицы MAN имена и фамилии людей с именем (FIRSTNAME) Андрей, которым больше 27 лет (YEAROLD).

```
SELECT firstname, lastname FROM man WHERE firstname='Андрей' and yearold > 27
```

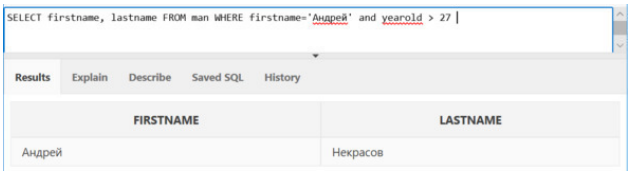


Рисунок 20. Запрос к таблице MAN: Андрей, возраст больше 27 лет

Выбрать из таблицы MAN имена (FIRSTNAME) и фамилии (LASTNAME) людей, которым не больше 27 лет (YEAROLD).

```
SELECT firstname, lastname FROM man WHERE firstname='Андрей' and not  yearold > 27
```

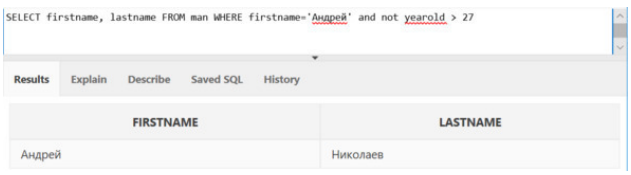


Рисунок 21. Запрос к таблице MAN: Андрей, возраст больше 27 лет

Выбрать из таблицы MAN имена (FIRSTNAME) и фамилии (LASTNAME) людей с именем Андрей или Алексей.

```
SELECT firstname, lastname FROM man WHERE firstname='Андрей' or firstname= 'Алексей'
```

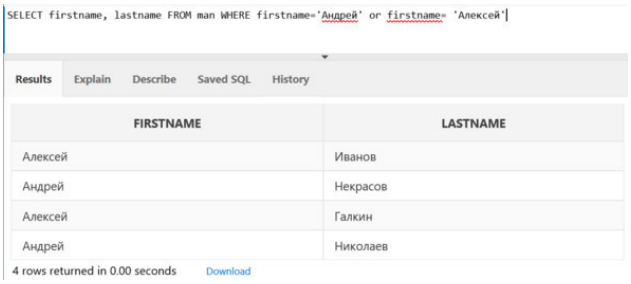


Рисунок 22. Запрос к таблице MAN: Андрей и Алексей

Выбрать из таблицы CITY города (*) с населением (PEOPLES) 400, 500 тысяч жителей.

```
SELECT * FROM city WHERE peoples=300000 or peoples=400000
```

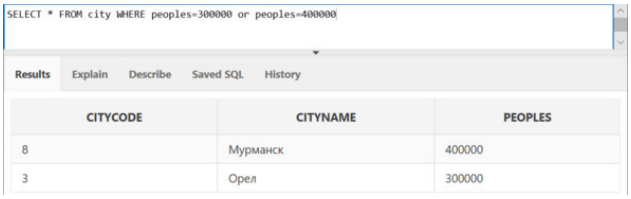


Рисунок 23. Запрос: города с населением 300 и 400 тысяч

Выбрать из таблицы CITY города (*) с населением (PEOPLES) не 400, 500 тысяч жителей.

```
SELECT * FROM city WHERE not( peoples=300000 or peoples=400000)
```

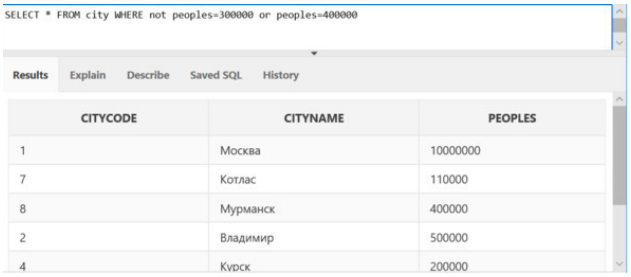


Рисунок 24. Запрос: города с населением не 300 и не 400 тысяч

Важные замечания

Несколько условий можно объединять скобками, например из таблицы MAN нам необходимо вывести те строки, где людям 25 или 28 лет с именем Иван. Это можно сделать с помощью следующего запроса:

```
SELECT * FROM man WHERE (yearsold = 25 or yearsold = 28) and firstname = 'Иван'
```

Очень важно понимать отличие AND от OR, например **выведите авто с марками LADA и BMW** – в этом запросе необходимо использовать инструкцию OR и ни в коем случае не AND.

```
SELECT * FROM auto WHERE mark = 'LADA' or mark = 'BMW'
```

Вопросы учеников

Можно ли неравенство заменить на инструкцию NOT?

Да, в большинстве запросов можно так сделать.
Например, запросы

```
SELECT * FROM man WHERE firstname <> 'Иван'
и
SELECT * FROM man WHERE not firstname = 'Иван'
```

идентичны.

Как поменять несколько условий, перечисленных после WHERE в SQL-запросе, на обратные?

Вопрос не очень понятен, но предположим, у нас есть запрос, который возвращает все строки из таблицы AUTO с марками BMW и LADA.

```
SELECT * FROM AUTO WHERE mark = 'BMW' or mark='LADA'
```

Если необходимо посмотреть авто не BMW и LADA, то запрос обретет следующий вид:

```
SELECT * FROM AUTO WHERE not(mark = 'BMW' or mark='LADA')
```

Контрольные вопросы и задания для самостоятельного выполнения

1. Сколько строк вернет запрос?

```
SELECT * FROM city WHERE cityname = 'Москва' and cityname = 'Воронеж'
```

2. Выберите из таблицы CITY (*) город, где 200 тысяч жителей (PEOPLES); город с наименованием Москва (CITYNAME).

3. Выберите из таблицы CITY города с населением (PEOPLES) не 500 тысяч жителей.

4. Выберите из таблицы AUTO (*) все синие AUDI (COLOR, MARK).

5. Выберите из таблицы AUTO номера (regnum) и марки (MARK) всех VOLVO и BMW.

6. Выберите из таблицы MAN имена, фамилии (FIRSTNAME, LASTNAME) и возраст людей (YEAROLD), которым больше 29 и меньше 35 лет.

Шаг 13. Сортировка результатов запросов

Введение

Данные, выводимые с помощью запроса **SELECT**, часто необходимо сортировать по одному или нескольким выводимым колонкам.

Для этого в языке SQL есть специальный оператор **ORDER BY**, который используется в команде **SELECT** и пишется в конце запроса. Для того чтобы изменить порядок сортировки на обратный, используется ключевое слово **DESC**.

Теория и практика

ORDER BY используется для сортировки выводимых данных по заданным колонкам: от большего к меньшему, и наоборот.

Текстовые данные будут отсортированы от А до Я или же от Я до А – при использовании ключевого операнда **DESC**.

Числовые данные сортируются от меньшего числа к большему числу и от большего числа к меньшему при использовании ключевого операнда **DESC**.

Данные DATE также: от меньшей даты к большей, и наоборот.

Синтаксис

```
SELECT перечень колонок через , или * FROM  таблица
WHERE условия выборки
ORDER BY  колонка сортировки1, колонка сортировки2, колонка сортировкиN
```

Где таблица – наименование таблицы, условия выборки которой могут быть объединены логическими операндами.

Примеры запросов

Выведите все записи из таблицы AUTO, отсортируйте автомобили по марке (MARK).

```
SELECT * FROM auto ORDER BY mark
```

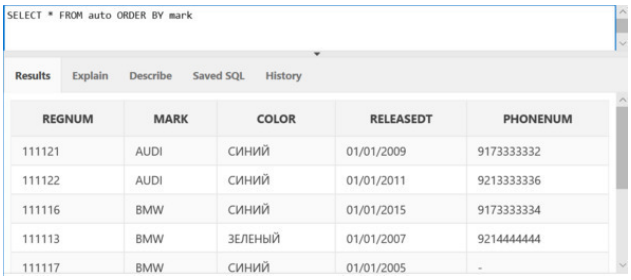


Рисунок 25. Выбрать из таблицы AUTO, отсортировать по марке

Выведите все записи из таблицы AUTO, отсортируйте автомобили по дате создания и цвету в обратном порядке (RELEASED, MARK).

```
SELECT * FROM auto ORDER BY releasedt desc, mark desc
```

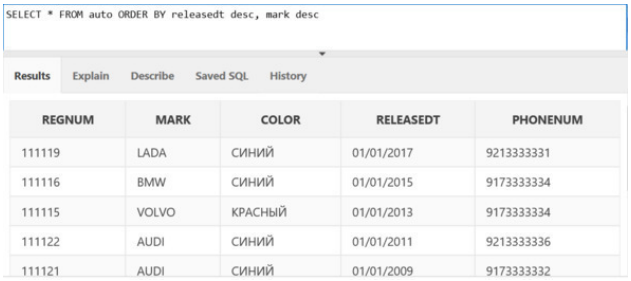


Рисунок 26. Запрос к таблице AUTO: сортировка по дате и цвету авто

Сортировка по двум колонкам RELEASED от меньшей даты к большей, MARK от A—Z.

Выберите только те записи из таблицы MAN, где возраст (YEAROLD) человека больше 30 лет; отсортируйте выбранные записи по возрасту, сортировку произведите в обратном порядке.

```
SELECT * FROM man WHERE yearold>30 ORDER BY yearold DESC
```

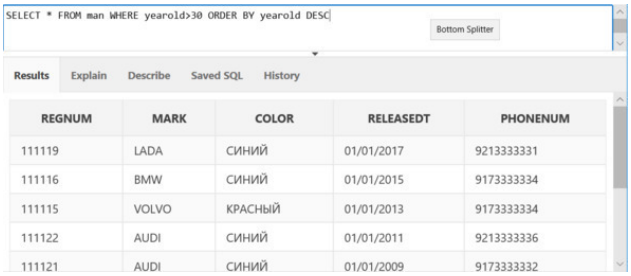


Рисунок 27. Выбор из таблицы MAN, где возраст больше 30 лет; сортировка по году выпуска

Пример использования ORDER BY и WHERE

Выберите наименования городов (CITYNAME) и население (PEOPLES) из таблицы CITY, отсортируйте выбранные данные в обратном порядке по наименованию города и количеству населения (PEOPLES).

```
SELECT cityname, peoples from CITY order by cityname desc, peoples desc
```

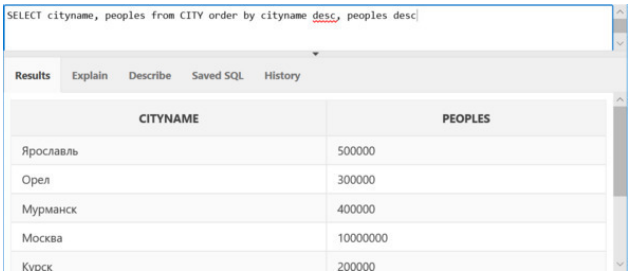


Рисунок 28. Запрос к CITY: сортировка по названию и населению в обратном порядке

Важные замечания

Следует пояснить, как работает сортировка по нескольким колонкам.

Сначала данные сортируются по первой колонке в инструкции **ORDER BY**, затем уже в рамках этой сортировки данные сортируются по второму признаку, по второй колонке из инструкции **ORDER BY**, затем третьей и так далее.

Команда сортировки **ORDER BY** выполняется в запросе последней, сортируется итоговое выражение запроса.

Это важная информация, и она пригодится нам в дальнейшем, при создании сложных SQL-запросов.

Вопросы учеников

Как сделать, если я хочу одну колонку отсортировать по возрастанию, а две другие по убыванию?

Давайте рассмотрим пример.

Выберите наименования городов (CITYNAME) и население (PEOPLES) из таблицы CITY, отсортируйте выбранные данные по коду города (CITYCODE), в обратном порядке по наименованию города и количеству населения.

```
SELECT citycode, cityname, peoples from CITY ORDER BY citycode , cityname desc, peoples
desc
```

Я знаю, что для инструкции ORDER BY существует альтернативный синтаксис. Расскажите о нем.

Перепишем этот запрос с использованием альтернативного синтаксиса:

```
SELECT citycode ,cityname, peoples FROM city ORDER BY 1 , 2 desc, 3 desc
```

То есть вместо названий колонок таблиц в инструкции ORDER BY мы используем порядковый номер колонки в нашем запросе – у CITYCODE он равен 1, у CITYNAME 2, у PEOPLES 3.

Можно ли отсортировать запрос по полю, которого нет в запросе, но которое есть в таблице после инструкции FROM?

Да, синтаксис SQL это допускает, и вот пример:

```
SELECT cityname, peoples FROM city ORDER BY citycode
```

Контрольные вопросы и задания для самостоятельного выполнения

1. Выберите записи из таблицы CITY, где в названии города (CITYNAME) есть слог «ем», отсортируйте запрос по названию города (CITYNAME) и по популяции (PEOPLES) в обратном порядке.

2. Выберите все записи из таблицы AUTO, отсортируйте записи по цвету (COLOR) и по марке (MARK) автомобиля в обратном порядке.

Шаг 14. Ограничение на количество выбранных строк ROWNUM, TOP (n)

Введение

Иногда запросы строятся таким образом, что на экран сразу выводится множество строк.

А что если нам необходимо ограничить количество строк выводимой информации, то есть из десятков тысяч строк нам достаточно нескольких строк для анализа информации?

Теория и практика

Для решения этой задачи в разных диалектах языка SQL используются разные синтаксические конструкции: в MS SQL это конструкция TOP, в ORACLE есть специальный предикат ROWNUM, в PostgreSql, MYSQL для этого существует конструкция LIMIT.

Разберем диалект SQL ORACLE.

Конструкция ROWNUM позволит ограничить количество выводимых строк на заданную величину.

Синтаксис

```
SELECT перечень полей или * FROM таблица
WHERE rownum<колстрок +1 and or not  доп условия
```

Примеры
Вывести первые 5 строк из таблицы AUTO.

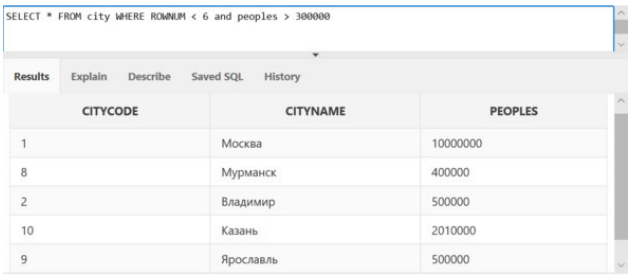
```
SELECT rownum, auto.* FROM auto WHERE ROWNUM < 6
```

ROWNUM	REGNUM	MARK	COLOR	RELEASEDT	PHONENUM
1	111119	LADA	СИНИЙ	01/01/2017	9213333331
2	111115	VOLVO	КРАСНЫЙ	01/01/2013	9173333334
3	111121	AUDI	СИНИЙ	01/01/2009	9173333332
4	111122	AUDI	СИНИЙ	01/01/2011	9213333336
5	111114	LADA	КРАСНЫЙ	01/01/2008	9152222221

Рисунок 29. Запрос с ограничением строк (первые 5)

Вывести первые 5 строк из таблицы CITY, где население (PEOPLES) городов больше 300 000.

```
SELECT * FROM City WHERE ROWNUM < 6 and peoples > 300000
```



CITYCODE	CITYNAME	PEOPLES
1	Москва	10000000
8	Мурманск	400000
2	Владимир	500000
10	Казань	2010000
9	Ярославль	500000

Рисунок 30. Выбрать первые 5 строк, запрос города: население больше 300 000

Важные замечания

Если использовать ROWNUM совместно с сортировкой, то необходимо прибегнуть к специальному приему, иначе ROWNUM не будет работать как нужно и запрос вернет неверные данные.

Правильно следует написать так:

```
SELECT * FROM (SELECT поле1, поле2 или * FROM имя таблицы
WHERE условия ORDER BY поля сортировки)
```

Вывести первые 5 автомобилей, отсортированных по дате производства (RELEASEDT).
Ошибочный запрос:

```
SELECT * FROM auto WHERE ROWNUM < 6 ORDER BY releasedt
```

Правильный запрос:

```
SELECT * FROM (SELECT * FROM Auto ORDER BY releasedt) WHERE ROWNUM<кол строк +1
```

Вопросы учеников

Покажите, как ограничивать вывод строк в MYSQL, MS SQL и PostgreSQL.

Разберемся на примере.

Вывести первые 5 строк из таблицы CITY, где население (PEOPLES) городов больше 3000.

ORACLE SQL

```
SELECT * FROM City WHERE ROWNUM < 6 and peoples > 3000
```

MY SQL

```
SELECT * FROM City WHERE peoples > 30000 limit 5
```

MS SQL

```
SELECT top 5 * FROM City WHERE peoples > 30000
```

PostgreSQL

```
SELECT * FROM City WHERE peoples>3000 limit 5
```

Контрольные вопросы и задания для самостоятельного выполнения

1. Поясните, как использовать ROWNUM с сортировкой в SQL-запросе.
2. Выведите на экран первые 3 строчки из таблицы CITY.
3. Выведите на экран первые 4 строчки из таблицы CITY, отсортированные по количеству населения (PEOPLES).
4. Выведите на экран данные из таблицы MAN.

Шаг 15. Вставка данных в таблицу – INSERT

Введение

Ранее мы с вами изучили, как создавать таблицы и изменять структуру таблиц, а также как извлекать данные из таблиц на экран с помощью команды SELECT. Мы выбирали данные, которые уже были в таблицах. Настало время узнать о том, как добавить необходимые данные в таблицу.

Теория и практика

Для добавления данных в SQL используется команда INSERT.
Команда INSERT существует в SQL в двух вариантах.
Во-первых, для добавления заданных значений – VALUES.
Синтаксис

```
INSERT INTO  имя таблицы (колонки через запятую)
VALUES (значения через запятую);
```

Важно помнить, что количество колонок в скобках и количество добавляемых значений должны соответствовать друг другу.

Также очень важно, чтобы колонки и значения соответствовали по типу данных.

Примеры
Добавим новые сведения о человеке в таблицу MAN:

```
INSERT INTO man(phonenum, firstname, lastname, citycode, yearold)
VALUES('120120120','Максим','Леонидов',2,25);
Commit;
```

В таблицу MAN добавлена строка о человеке с номером телефона «120120120», именем «Максим», фамилией «Леонидов», кодом города 2, ему 25 лет (PHONENUM, FIRSTNAME, LASTNAME, CITYCODE, YEAROLD).

Добавим сведения о новой машине в таблицу AUTO:

```
INSERT INTO auto(regnum, mark, color, released, phonenum)
VALUES('128877655','LADA','КРАСНЫЙ',date'2001-01-01','123114444');
Commit;
```

В таблицу AUTO добавлена строка о авто с номером «128877655» марки «LADA», цвет «КРАСНЫЙ», дата выпуска «2001-01-01», «123114444».

Второй вариант, для добавления из запроса SELECT

Здесь источником данных являются не одиночные значения, а настоящий запрос SELECT.

Синтаксис:

```
INSERT INTO   имя таблицы (колонки через запятую)
SELECT FROM имя таблицы WHERE условия
```

Примеры

В таблицу CITY1 добавить все строки из CITY, где население больше миллиона человек.

```
INSERT INTO city1(citycode, cityname, peoples)
SELECT citycode, cityname, peoples WHERE peoples>1000000
```

В таблицу MAN1 добавить все строки из MAN.

```
INSERT INTO man1(phonenum, firstname, lastname, citycode, yearold)
SELECT phonenum, firstname, lastname, citycode, yearold FROM man;
```

Важные замечания

Команда INSERT является командой модификации данных, поэтому ее выполнение необходимо завершить одной из следующих команд:

COMMIT – фиксация изменений;

ROLLBACK – откат изменений.

Только после выполнения фиксации изменений данные появятся в базе.

Обратите внимание: при добавлении данных типа дата (**DATE**) мы использовали следующую конструкцию: **DATE'YYYY-MM-DD»**, YYYY – текущий год, MM – месяц, DD – день.

При использовании **INSERT** с запросом **SELECT** также необходимо, чтобы последовательность и типы колонок, перечисленных после команды **INSERT**, совпадали с последовательностью и типами колонок в запросе **SELECT**.

Вопросы учеников

Все же зачем применять COMMIT или ROLLBACK и что будет, если эти команды не выполнять?

Во многих СУБД применяется транзакционная модель, что это такое – узнаем чуть позже, но сейчас необходимо понимать, что при запуске операций изменения, вставки, удаления данных эти изменения появятся в базе только после выполнения команды COMMIT.

Я использовал только команду INSERT и не применял ни COMMIT, ни ROLLBACK, но данные все равно появились в базе. Почему так вышло?

Некоторые редакторы поддерживают режим автофиксации изменений, то есть выполняют команду COMMIT за вас.

Если я перечислю не все колонки, какие есть в таблице, куда мы добавляем данные, будет ошибка?

Необязательно. В колонки, которые вы не перечислили, будет добавлено пустое (NULL) значение, ошибка возникнет только если на колонках этой таблицы есть ограничение NOT NULL или первичный ключ.

Контрольные вопросы и задания для самостоятельного выполнения

1. Повторите материалы записей из данного шага.
2. Добавьте в таблицу городов новый город; код города и название придумайте сами.
3. В таблицу CITY1 добавьте все строки из CITY, где популяция меньше полумиллиона человек.
4. В таблицу AUTO добавьте новую машину 999999 ЛАДА, цвет зеленый, 1999 года выпуска.

День четвертый

Шаг 16. Обновление данных – UPDATE

Введение

Мы научились добавлять данные в таблицу, но очень часто возникают ситуации, когда нам необходимо обновить, пересчитать данные в одной или нескольких колонках.

Теория и практика

Обновление данных в строчках и ячейках таблицы осуществляется с помощью команды UPDATE.

Синтаксис команды

```
UPDATE table_name SET column1 = знач1, column 2 = знач2 , column n = значn WHERE
условия отбора строк для обновления
```

Здесь TABLE_NAME – имя таблицы, где обновляются данные.

Здесь через запятую перечислены имена колонок = значения, а после WHERE описываются условия для отбора обновляемых строк.

Примеры

Добавить к наименованию (CITYNAME) города CITY с кодом (CITYCODE) меньше 2 символ #.

```
UPDATE city SET cityname = cityname| |'#' WHERE citycode<2
```

К населению города с населением (PEOPLES) больше 1 000 000 добавить 10.

```
UPDATE city SET peoples = peoples+10 WHERE peoples > 1000 000
```

У всех у людей MAN с именем (FIRSTNAME) Алексей поменять имя на Максим.

```
UPDATE man SET firstname = 'Максим' WHERE firstname = 'Алексей'
```

У всех у людей с телефоном (PHONENUM), заканчивающихся на 3, поменять имя (FIRSTNAME) на Александр.

```
UPDATE man SET firstname = 'Александр' WHERE phonenum like '%3'
```

Применение обновления UPDATE возможно также без использования предиката WHERE, в этом случае обновятся все строки указанной таблицы в заданных колонках.

Пример, который не нужно выполнять, но обязательно следует изучить.

Обнулить колонку PEOPLES в таблице CITY.

```
UPDATE city SET proples = 0
```

Во всей таблице значение колонки PEOPLES будет равно 0.

Важные замечания

Команда обновления данных UPDATE тоже должна завершаться выполнением **COMMIT –** фиксацией изменений, либо **ROLLBACK –** откатом изменений.

При выполнении обновления данных следует быть предельно аккуратным и внимательным: последующее восстановление измененных данных, возврат таблицы к состоянию до выполнения UPDATE очень часто бывает затруднителен.

Команда UPDATE TABLE set column1 = val1, columnn = valn без инструкции WHERE обновит значения во всех строчках таблицы TABLE колонок column1, column, будьте особо осторожны при выполнении таких команд.

Вопросы учеников

Если мы обновляем текстовое поле, необходимо ли значение писать в одинарных кавычках?

Да, при обновлении текстовых данных необходимо заключать значения в одинарные кавычки.

Приведите пример обновления данных типа DATE.

Пример: обновите даты выпуска автомобилей LADA на 01.01.2010.

```
UPDATE auto SET releasedate = date'2010-01-01' WHERE mark = 'BMW'
```

Я выполнил команду UPDATE, попытался выполнить ее повторно из другого окна, но ничего не получилось и программа зависла, в чем причина?

Причина в том, что вы забыли зафиксировать изменения с помощью команды COMMIT. При выполнении первого UPDATE на строки были установлены блокировки, повторное обновление не выполнилось, так как изменения не были вами зафиксированы. Об этом будет более подробная информация в книге в дальнейшем.

Контрольные вопросы и задания для самостоятельного выполнения

1. Сколько строк обновит команда UPDATE AUTO SET MARK = «BMW» WHERE 1=0?
2. Добавить 1 к возрасту людей (MAN) с именем Андрей.
3. Добавить 2 к возрасту людей (MAN), чей возраст больше 19 лет.
4. К имени людей (MAN), чей телефон содержит 915, добавить префикс 915.
5. У всех у людей (MAN) с телефоном, чей возраст больше 22 лет, поменять имя на Роман.

Шаг 17. Удаление данных – DELETE

Введение

Иногда некоторые данные нам необходимо удалять из таблиц. Для этого в языке SQL есть специальная команда – DELETE.

Теория и практика

Команда DELETE.

Синтаксис

```
DELETE имя таблицы WHERE условие для удаления строк
```

Здесь Имя таблицы – имя таблицы, из которой мы удаляем строки, WHERE – условие для отбора строк, которые мы удаляем.

Примеры

Удалить записи из таблицы CITY1 с кодами городов CITYCODE 7, 9.

```
DELETE city1 WHERE citycode = 7 or citycode =9
```

Удалить из таблицы AUTO1 все автомобили BMW (MARK).

```
DELETE auto1 WHERE mark = 'BMW'
```

Удалить из таблицы AUTO1 все автомобили с датой выпуска больше 01.01.2017.

```
DELETE auto1 WHERE releasedate > date'2017-01-01'
```

Завершите выполнение командой COMMIT для фиксации изменений.

Важные замечания

Данная команда может удалить записи только в одной заданной таблице.

Команда DELETE также должна завершаться инструкцией COMMIT для фиксации изменений в базе.

Следует помнить, что язык SQL в ORACLE поддерживает механизм транзакций, поэтому модификации данных, в том числе и операции удаления, чаще всего необходимо завершать командой **COMMIT** или **ROLLBACK**.

COMMIT – для фиксации изменений;
ROLLBACK – для отката изменений.

Команда DELETE без инструкции WHERE очищает все строки в таблице, поэтому при использовании команды DELETE надо соблюдать предельную осторожность.

Вопросы учеников

Чем отличается команда DELETE от команды DROP?

Это совершенно разные команды. Я предлагаю повторить материал на эту тему и ответить на данный вопрос самостоятельно.

Как еще можно очистить таблицу?

Для этого есть специальный оператор TRUNCATE TABLE TABLENAME. При его выполнении происходит быстрая очистка всех записей таблицы. Мы еще будем изучать эту тему в следующих шагах.

Я пытаюсь удалить записи из некоторой таблицы, но возникает ошибка.

Вероятно, есть связанные записи в другой таблице по внешнему ключу. Сначала необходимо удалить записи в связанной таблице.

Контрольные вопросы и задания для самостоятельного выполнения

1. Чем команда DELETE TABLE_NAME отличается от команды DROP TABLE TABLE_NAME?
2. Напишите команду для удаления из таблицы AUTO таких записей, где дата выпуска авто больше 2018 года.
3. Повторите материалы данного шага.

Шаг 18. Псевдонимы

Введение

В языке SQL есть такая синтаксическая конструкция, как псевдонимы.

С помощью псевдонимов мы можем большим или сложным наименованиям таблиц или колонок таблиц в запросе SQL присвоить более короткие, удобные и понятные нам псевдонимы (ALIAS).

Теория и практика

Псевдонимы для колонок, выводимых в запросе, задаются с помощью инструкции AS, псевдонимы же для таблиц указываются сразу же после имени таблицы.

Примеры

Вывести из таблицы MAN колонки «имя», «фамилия» и «возраст».

Для таблицы задать псевдоним m для колонки имени (FIRSTNAME), для колонки фамилии (LASTNAME) – Fml.

```
SELECT firstname as Im, lastname as Fml FROM man m
```

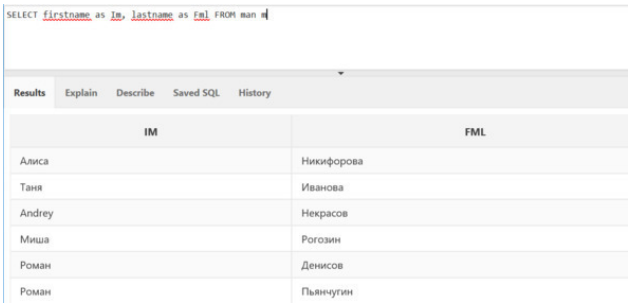


Рисунок 31. Демонстрация работы псевдонимов: запрос

Вывести из таблицы AUTO марку и цвет автомобиля.

- Для таблицы AUTO задать псевдоним AV;
- для колонки «марка» задать псевдоним Mr;
- для колонки «цвет» задать псевдоним CV.

Вывести только автомобили с годом выпуска больше 01.01.2001.

```
SELECT mark as Mr, color as CV FROM auto Av WHERE av.releasedt > date'2001-01-01'
```

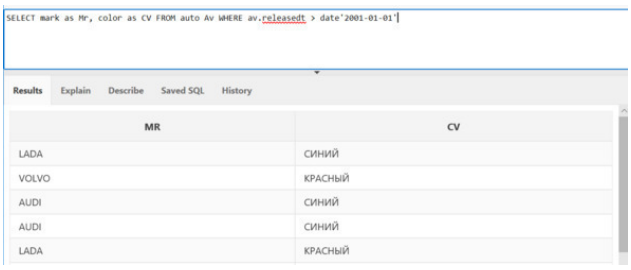


Рисунок 32. Запрос к AUTO: псевдонимы

Обратите внимание, как формируется условие для при обращении к колонке типа DATE.

Важные замечания

Псевдонимы не могут повторяться в рамках одного запроса и подзапроса, то есть их имена должны быть уникальными.

Если мы задали псевдоним для таблицы, из которой SQL-запрос выбирает данные, то и в условии WHERE мы также должны использовать заданный псевдоним.

Вопросы учеников

Если мы используем псевдоним для таблицы и псевдонимы для колонок, должны ли мы обращаться к колонкам в инструкции WHERE по псевдонимам колонок?

Как ни странно, так делать нельзя. Вы должны указать именно настоящее имя колонки в этом случае.

Пример

```
SELECT MARK as mr, COLOR  as cl FROM AUTO WHERE color='СИНИЙ'
```

Запись вида

```
SELECT MARK as mr, COLOR  as cl FROM AUTO WHERE cl='СИНИЙ'
```

будет неверной.

Можно ли нескольким выводимым колонкам запроса SQL задать псевдоним, а другим не задавать?

Конечно; вот пример такого запроса:

```
SELECT firstname as fn, lastname  FROM man.
```

Контрольные вопросы и задания для самостоятельного выполнения

1. Найдите ошибку в следующем запросе:

```
SELECT citycode as cc, cityname as cn  FROM city cc
```

2. Найдите ошибку в еще одном запросе:

```
SELECT citycode as cc, cityname as cn  FROM city c WHERE city.citycode = 1;
```

3. Выведите с помощью запроса SQL наименование города из таблицы CITY. Для таблицы задайте псевдоним GR, для колонки «наименования» задайте псевдоним NM.

4. Выведите количество жителей в городе Москва, для колонки PEOPLES задайте псевдоним CLZ, для таблицы задать псевдоним MS.

Шаг 19. BETWEEN

Введение

В языке SQL есть специальная конструкция, которая позволяет работать с интервалами – своего рода фильтр, позволяющий выбирать данные, соответствующие заданному интервалу значений. Этот оператор называется BETWEEN и может использоваться как в выборке SELECT, так и в операциях модификации и удаления данных (UPDATE, DELETE).

Теория и практика

Рассмотрим применение BETWEEN в команде SELECT.
Синтаксис:

```
SELECT перечень полей или * FROM TAB1
WHERE поле1 BETWEEN нижняя граница интервала AND верхняя граница интервала
Из  TAB1 Будут выбраны строки,  соответствующее интервалу ЗАДАННОМУ В BETWEEN
```

Примеры
Выбираем из таблицы MAN (*) всех людей, чей возраст (YEAROLD) от 25 до 32 лет.

```
SELECT * FROM man WHERE yearold BETWEEN 25 and 32
```

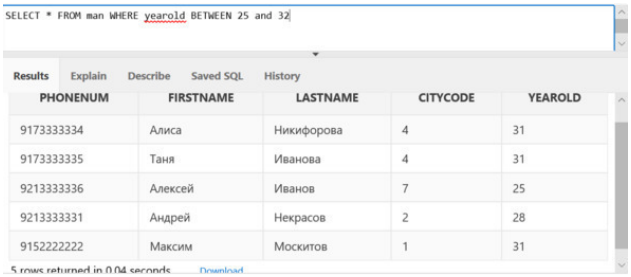


Рисунок 33. Запрос к MAN: возраст от 25 до 32

Выбираем города CITY * с населением (PEOPLES) от 300 000 до 1 000 000.

```
SELECT * FROM city WHERE peoples BETWEEN 300000 and 1000000
```

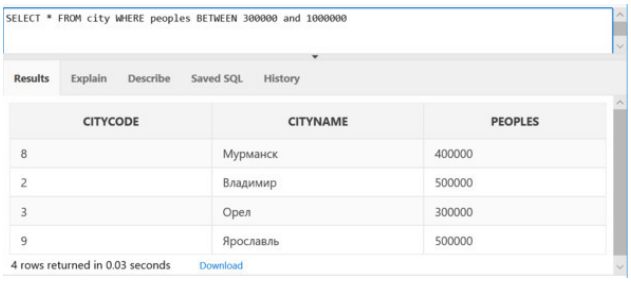


Рисунок 34. Запрос к таблице CITY: население от 300 000 до 1 000 000

Выбираем людей * с именем Андрей (FIRSTNAME) и возрастом (YEAROLD) от 25 до 28 лет.

```
SELECT * FROM man WHERE yearold BETWEEN 25 and 28 and firstname = 'Андрей'
```

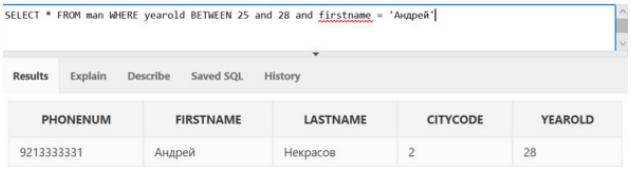


Рисунок 35. Запрос к таблице MAN: население от 25 до 28 лет, имя Андрей

Выбираем людей * с возрастом не в интервале от 25 до 28 лет.

```
SELECT * FROM man WHERE NOT yearold BETWEEN 25 and 28
```

Выбрать машины * с годом выпуска (RELEASEDT) от 2001 до 2014.

```
SELECT * FROM auto WHERE Releasedt BETWEEN date'2001-01-01' and date'2014-01-01'
```

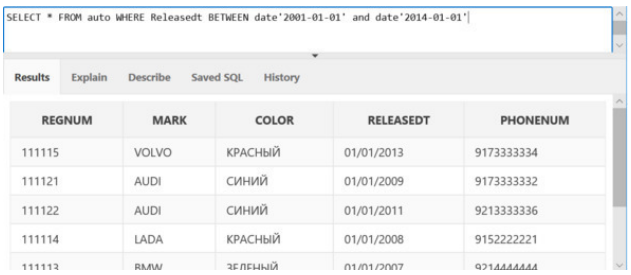


Рисунок 36. Запрос всех записей из AUTO

Важные замечания

Следует отметить, что использование BETWEEN разумно только для данных числовых типов и для данных типа DATE, что естественно, так как мы можем найти числа в заданном интервале и даты в заданном интервале, а вот со строкам и текстом это сделать затруднительно.

Также важное замечание: обратите внимание, что BETWEEN позволяет выбрать данные, принадлежащие и равные нижней и верхней границам заданного интервала.

Вопросы учеников

Оператор BETWEEN на что-то похож, можно ли обойтись без него?

Оператор BETWEEN создан для удобства, и, разумеется, его можно заменить несколькими логическими выражениям, и вот пример.

Пример:

Выбираем города CITY * с населением (PEOPLES) от 300 000 до 1 000 000.

```
SELECT * FROM city WHERE peoples BETWEEN 300000 and 1000000
```

Не будем использовать BEETWEEN.

```
SELECT * FROM city WHERE peoples >=300000 and peoples <=1000000
```

Можно ли в запросе использовать несколько BETWEEN, объединенных логическими операндами?

Да, разумеется, такой запрос можно написать. Выберем из таблицы MAN людей с возрастом в интервалах от 20 до 30 и от 35 до 39 лет.

```
SELECT * FROM man WHERE yearold BETWEEN 20 and 30 or yearold BETWEEN 35 and 39
```

Вы говорили, что BETWEEN также можно использовать и в операторах модификации данных?

Да, как и все другие операторы, используемые для выбора строк в WHERE.

Примеры

```
UPDATE auto SET Releasedt= date'2014-01-01'   WHERE Releasedt BETWEEN date'2011-01-01'
and date'2014-01-01'
```

Обновим дату создания авто на 01.01.2014 для автомобилей с датой создания в интервале от 01.01.2011 до 01.01.2014.

DELETE MAN WHERE YEAROLD BETWEEN 7 AND 16

Удаление всей информации о людях, которым от 7 до 16 лет. Обратите внимание: возраст 7 и 16 входит в заданный интервал и эта информация также будет удалена.

Контрольные вопросы и задания для самостоятельного выполнения

1. Как можно заменить оператор BETWEEN и обойтись без него, чтобы запрос возвращал правильные данные?

2. Выберите города с населением от 40 000 до 2 000 000 человек, напишите SQL-запрос.

3. Выберите города с населением НЕ в интервале 40 000 до 2 000 000 человек, напишите SQL-запрос.

4. Выберите людей * с возрастом НЕ в интервале 25 до 28 лет, используйте NOT, напишите SQL-запрос.

5. Выберите машины с годом выпуска от 2007 до 2011, напишите SQL-запрос.

Шаг 20. DISTINCT, дубликаты значений

Введение

В определенных запросах SQL как неприятный побочный результат выводится множество одинаковых, повторяющихся записей. Иногда нам необходимо уйти от данных повторений, убрать дубли из результатов запроса.

Теория и практика

В SQL существует специальная команда DISTINCT, которая предназначена для выбора в запросе только уникальных значений, уникальных строк, то есть исключает из вывода повторения и дублирования строк.

Синтаксис

```
SELECT DISTINCT перечень полей или * FROM таблица WHERE условия
```

Примеры

Вывести из таблицы AUTO марки автомобилей (MARK), исключить повторения – дубли.
Запрос без DISTINCT:

```
SELECT mark FROM auto
```

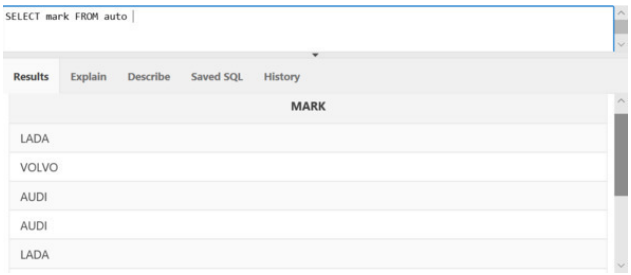


Рисунок 37. Запрос: дубли марок

Есть дублирующиеся марки авто AUDI, LADA в результате вывода.
Используем DISTINCT:

SELECT DISTINCT MARK FROM AUTO

```
SELECT DISTINCT mark FROM auto
```

Дубли строк не выводятся.

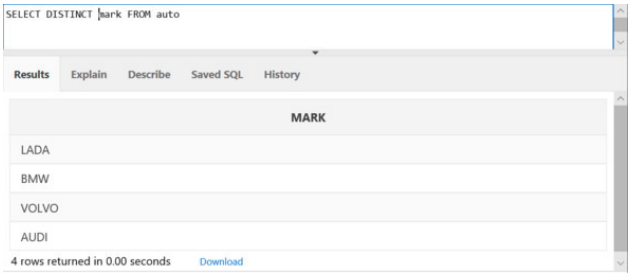


Рисунок 38. Запрос: марки авто

Вывести из таблицы AUTO марки и цвета автомобилей (MARK, COLOR), исключить повторения.

```
SELECT mark, color FROM auto
```

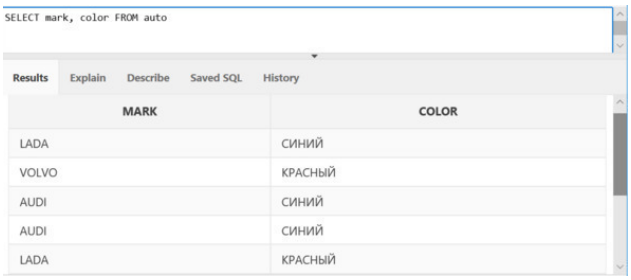


Рисунок 39. Запрос: марки авто и цвета

Несколько синих AUDI, используем DISTINCT.

```
SELECT DISTINCT mark, color FROM auto
```

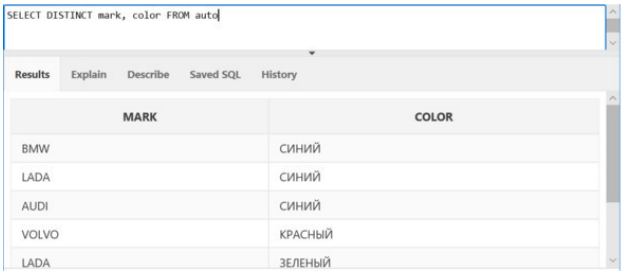


Рисунок 40. Запрос: марки, цвета авто, только уникальные записи

Важные замечания

Важно отметить, что дубли исключаются DISTINCT только из колонок, перечисленных в SELECT; других колонок DISTINCT не касается.

С помощью DISTINCT очень удобно просмотреть, какие вообще значения есть в заданной колонке, например автомобили каких цветов присутствуют в таблице AUTO:

```
SELECT DISTINCT COLOR FROM AUTO
```

Вопросы учеников

Есть ли еще какой-либо способ исключить дубли из запроса?

Да, подобный способ называется группировка записей, и мы изучим его позже.

Можно ли использовать DISTINCT с ROWNUM?

Да, но тогда оператор DISTINCT бесполезен, его использование потеряет смысл.

Можно ли использовать DISTINCT с WHERE?

Конечно, это возможно, нет никаких ограничений, и вот пример такого запроса. Выведите из таблицы AUTO автомобили марки LADA уникальных цветов.

```
SELECT DISTINCT mark, color FROM auto WHERE mark = 'LADA'
```

Контрольные вопросы и задания для самостоятельного выполнения

1. Повторите материалы данного шага.

2. Выведите из таблицы AUTO цвета автомобилей, исключите повторения, напишите SQL-запрос.

3. Выведите из таблицы MAN имена людей, исключите повторения, напишите SQL-запрос.

День пятый

Шаг 21. Математика в запросах

Введение

Мы уже использовали математику в SQL-командах ранее: вспомните, в одном из примеров использования UPDATE мы добавляли год к возрасту человека из MAN.

Так вот, в запросах SQL мы можем использовать результаты математических вычислений, причем мы можем как выводить результаты математических вычислений в колонках выбора SELECT, так и использовать математические выражения при формировании условий отбора строк WHERE.

Теория и практика

Для создания математических выражений в языке SQL используются следующие операции:

+ сложение;
– вычитание;
/ деление;
* умножение.

А также знакомые нам со школы функции:
Sqrt – квадратный корень;
MOD – остаток от деления;
trunc – округление до целого;
sIN – синус;
cos – косинус
(на самом деле этих функций больше, мы рассматриваем основные).

Все математические операции выполняются только для числовых значений, числовых колонок с типами NUMBER или производными от NUMBER (INt, float) – более подробную информацию можно посмотреть в документации к СУБД. То есть мы можем использовать в математических выражениях значения соответствующих колонок при выводе на экран и в фильтре WHERE.

Запомним, что математические операции используются также в критериях отбора строк WHERE.

Посмотрим, как это делается.
Вывести из таблицы MAN имя, фамилию и возраст (FIRSTNAME, LASTNAME, YEAROLD) человека, разделенный на 10.

```
SELECT firstname,lastname,yearold/10 as y10 FROM man
```

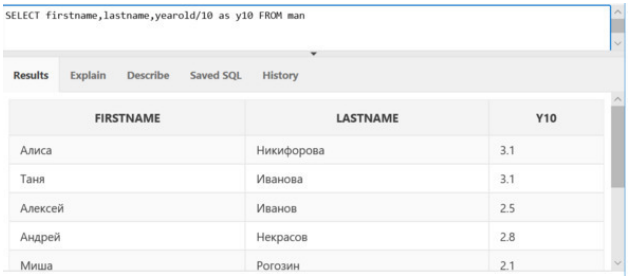


Рисунок 41. Запрос к MAN: результат математического вычисления

В колонке с псевдонимом y10 выводится результат выражения YEAROLD/10.

Вывести из таблицы MAN имя, фамилию и возраст человека (FIRSTNAME, LASTNAME, YEAROLD), умноженный на sIN (1), округлить до целого.

```
SELECT firstname,lastname,trunc(yearold*sin(1)) as ysin FROM man
```

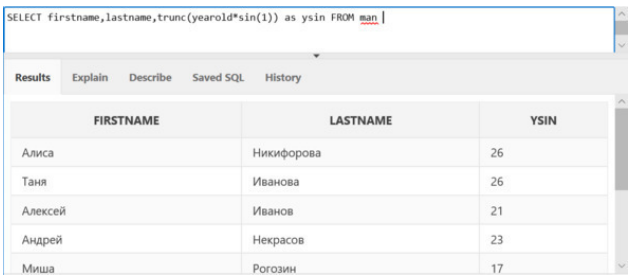


Рисунок 42. Запрос к MAN: результат вычисления sIN

Вывести из таблицы CITY записи (*), где квадратный корень от количества населения города больше 1000 (PEOPLES).

```
SELECT * FROM city WHERE sqrt(city.peoples)>1000
```

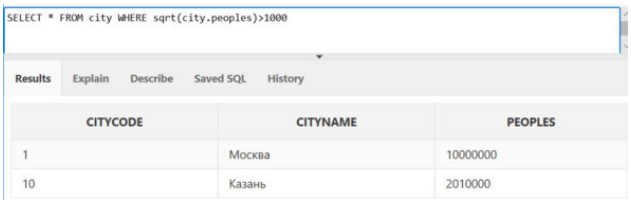


Рисунок 43. Запрос к CITY: квадратный корень больше 1000

Пример демонстрирует использование математического выражения sqrt(CITY.PEOPLES) при фильтрации строк в WHERE.

Вывести из таблицы CITY название города (CITYNAME), квадратный корень от количества населения (PEOPLES), где значение кода города (CITYCODE) делится нацело на 3.

```
SELECT cityname, sqrt(peoples) as spep FROM city WHERE mod(citycode,3) = 0
```

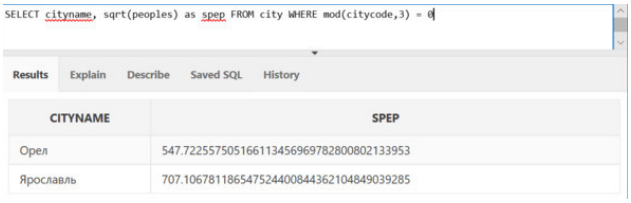


Рисунок 44. Запрос к CITY с математическим выражением

Вывести из таблицы CITY название города (CITYNAME), код города, разделенный на 3 (CITYCODE), где значение населения (PEOPLES), разделенное на 100, не больше 1 000 000.

```
SELECT cityname, citycode/3 as ccode FROM city WHERE Not peoples / 100 > 1000000
```

Рисунок 45. Запрос к CITY: математическое выражение WHERE и SELECTLISt

Важные замечания

При выполнении математических выражений иногда возникают ошибочные исключительные ситуации, например при делении некоторого значения на ноль: в этом случае запрос выдает некорректный результат или вообще не выполняется, но при этом некоторые среды разработки подавляют сообщения об ошибке.

Об этом следует помнить, когда вы пишете запросы, использующие математические выражения.

Вопросы учеников

Зачем использовать псевдонимы для колонок, которые рассчитываются как математические выражения?

Для удобного, понятного отображения названия колонки.

Где найти полный перечень математических функций, поддерживаемых СУБД?

В документации по вашей СУБД.

Контрольные вопросы и задания для самостоятельного выполнения

1. Вывести из таблицы MAN имя, фамилию и квадратный корень 10.
2. Вывести из таблицы MAN имя, фамилию и возраст человека, умноженный на cos (5).
3. Вывести из таблицы CITY записи (*), где популяция делится без остатка на 10 000.
4. Вывести из таблицы CITY название города, квадратный корень от популяции, умноженный на 10, где значение кода города делится нацело на 3.

Продолжением данного шага является следующий шаг, посвященный таблице DUAL.

Шаг 22. Запрос к результату выражения и специальная таблица DUAL

Введение

В SQL ORACLE диалекта есть специальная таблица DUAL. Если в других СУБД существуют альтернативные варианты синтаксиса, то для SQL ORACLE необходимо использовать таблицу DUAL.

Теория и практика

Данная таблица выполняет следующие задачи:

• делает удобным вывод в запросе «результаты вычислений математического выражения»;

• помогает при вычислении функции, которая возвращает одно значение.

При этом для результата вычисления данных выражений нам не требуется участие колонок из каких-либо таблиц в нашей базе данных.

Разберем на примерах:

Вычислить значение математического выражения 1000* (123—11) /100 с помощью SELECT.

```
SELECT 1000*(123-11)/100 as v FROM dual
```

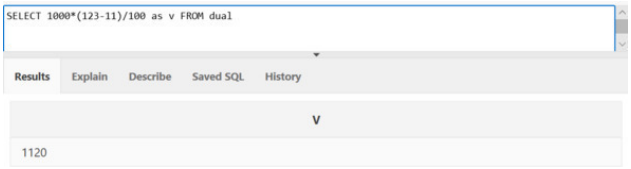


Рисунок 46. Вычисление выражения

Вычислить значение функции sIN (10) /100 с помощью SELECT.

```
SELECT sin(10)/100 as v FROM dual
```

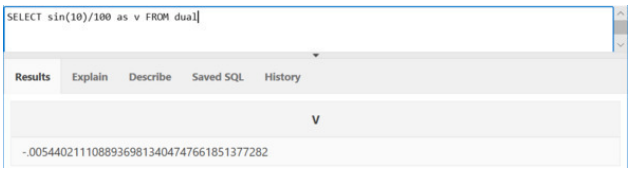


Рисунок 47. Вычисление выражения, таблица DUAL

Вычислить значение текущей даты с помощью SELECT.

```
SELECT sysdate  as dt FROM dual
```

– 21.11.1999 11:22

Вычислить значение sIN (1) *cos (1) с помощью SELECT.

```
SELECT sin(1)*cos(1) as n1 FROM dual
```

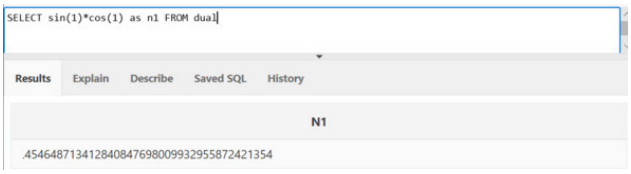


Рисунок 48. Вычисление выражения и таблица sIN (1) * cos (1)

Важные замечания

При обращении к таблице **DUAL** запрос возвращает всего одну строку, при попытке добавить запись в таблицу **DUAL** могут произойти непредсказуемые последствия, не рекомендую это делать.

Вопросы учеников

Вы говорили, что в других СУБД, например MS SQL, можно обойтись без таблицы DUAL?

Да в MS SQL, например, будет работать такой запрос: SELECT sIN (1), здесь нет FROM, ORACLE в этом случае более соответствует стандарту SQL ANSI 92.

Контрольные вопросы и задания для самостоятельного выполнения

1. Вычислить значение площади квадрата со стороной 5 с помощью SELECT и DUAL.
2. Вычислить значение площади круга с радиусом 7 помощью SELECT и DUAL.
3. Вычислить значение выражения 100*sIN (1) *cos (3) с помощью SELECT и DUAL.

Шаг 23. Пустые значения в базе. NULL, NOT NULL, NVL

Введение

В базах данных присутствует тип NULL для обозначения пустых значений, то есть если данные в ячейке таблицы не заполнены, это значит, что там NULL-значение.

Теория и практика

Для работы с типом NULL в языке SQL используется специальный синтаксис.

Синтаксис

```
SELECT * FROM table1 WHERE колонка1 IS NULL
```

Выполняется проверка на наличие пустых значений в колонке1 таблицы TABLE1.

Синтаксис
Проверка на отсутствие пустых значений в колонке1 таблицы TABLE1.

```
SELECT * FROM table1  WHERE колонка1 IS NOT NULL
```

Примеры для изучения

Вывести из таблицы AUTO все строки, где значения поля PHONENUM пустые.

```
SELECT * FROM auto WHERE phonenum IS NULL
```

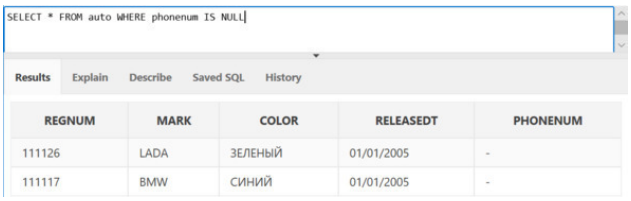


Рисунок 49. Запрос к AUTO с незаполненным номером телефона

Вывести из таблицы AUTO все строки, где значения поля PHONENUM не пустые.

```
SELECT * FROM auto WHERE phonenum IS NOT null
```

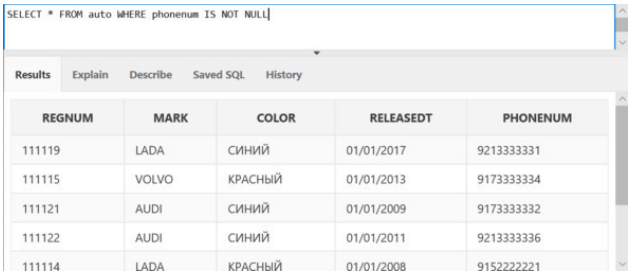


Рисунок 50. Запрос к AUTO, где PHONENUM заполнено

Так же в ORACLE SQL применяется специальная встроенная функция NVL, которая преобразует пустое значение в другое, заданное значение.

Синтаксис

```
SELECT перечень полей или *, NVL(поле где Null, новое значение) FROM имя таблицы
WHERE условия выбора строк
SELECT перечень полей или * новое значение) FROM имя таблицы WHERE NVL(поле где
Null, значение) = условие сравнения + другие условия выбора строк
```

Примеры использования NVL:

Вывести на экран все строки из AUTO, а в поле PHONENUM вместо пустого значения выводим 0.

```
SELECT auto.*, NVL(auto.phonenum, 0) as nn FROM auto
```

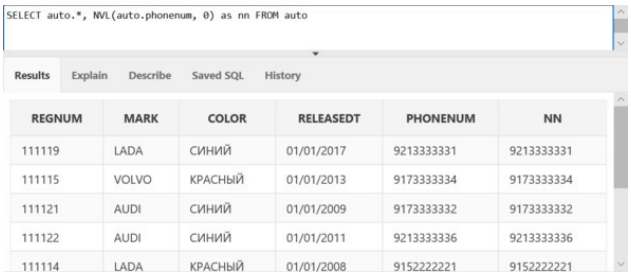


Рисунок 51. Использование NVL

Вывести из таблицы AUTO все строки, где значения поля PHONENUM не пустые, используя NVL.

```
SELECT auto.* FROM auto WHERE NVL(auto.phonenum, 0) != 0
```

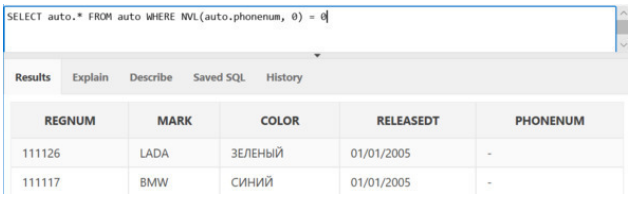


Рисунок 52. Запрос: использование NVL

Важные замечания

Для работы с пустыми значениями в СУБД ORACLE и других СУБД необходимо использовать конструкцию **IS NOT NULL** и **IS NULL** для проверки пустых значений или использовать **NVL**.

Запрос вида

```
SELECT * FROM таблица WHERE поле = NULL или SELECT таблица WHERE поле!= NULL
```

почти всегда приведет к некорректному результату.

Вопросы учеников

Есть ли возможность в языке SQL, если в поле находится пустое значение, вывести один результат, а если не пустое, то другой?

Да, для этого в ORACLE SQL существует специальная функция NVL2, об особенностях ее работы вы можете прочитать в документации.

Можно ли в одном запросе использовать несколько проверок, проверить несколько колонок на пустые значения?

Да, разумеется; и вот пример такого запроса:

```
SELECT * FROM auto WHERE phone is not null or MARK is not null.
```

Как запретить добавлять в таблицу пустые значения?

Этот вопрос подробно рассмотрен в шаге 9.

Контрольные вопросы и задания для самостоятельного выполнения

1. Найдите ошибку в запросе:

```
SELECT * FROM auto WHERE phone=null
```

2. Повторите запросы из примеров.
3. Выберите телефоны из таблицы AUTO. Вместо пустых телефонов выведите: NO PHONE, нет.

Шаг 24. Оператор LIKE

Введение

Для более удобной работы с текстовыми данными в запросах SQL существует специальный оператор – LIKE, который устанавливает соответствие значения текстовой ячейки некоторому заданному шаблону. LIKE – пожалуй, наиболее часто используемый оператор для поиска, фильтра текстовых данных в таблице.

Теория и практика

В зависимости от того, соответствует ли значение в заданной ячейке таблицы шаблону LIKE, строка будет выведена в запросе или нет; естественно, если значение ячейки не соответствует шаблону, строка не выводится в запросе SELECT.

Синтаксис

```
SELECT перечень полей или * FROM имя таблицы
WHERE колонка LIKE 'шаблон'
```

Шаблон LIKE несложный и состоит из следующих специальных символов:
символ подчеркивание _ заменяет любой символ;
символ процент % заменяет собой любое количество различных символов.

Примеры шаблонов LIKE

«%им» – выведет все записи, где значение в заданной колонке заканчивается «им», то есть любое количество символов (%), затем «им»;

«_осква» – выведет все записи, где значение в заданной колонке заканчивается «осква»;

«%лек%' – выведет все записи, где значение в заданной колонке содержит слог «лег»;

«Москв%' – выведет все записи начинающиеся Москв – то есть Москв, а затем любые символы.

Следует помнить, что LIKE применяется только для текстовых данных в ячейке таблицы, а также то, что шаблон указывается в одинарных кавычках.

Кроме LIKE допустимо использовать конструкцию NOT LIKE.

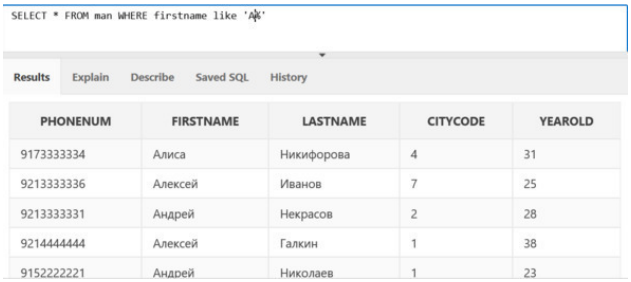
Синтаксис

```
SELECT перечень полей или * FROM имя таблицы
WHERE NOT поле LIKE 'шаблон' в одинарных кавычках
```

Примеры

Выбрать из таблицы Man людей *, имена (FIRSTNAME) которых начинаются с буквы A.

```
SELECT * FROM man WHERE firstname like 'A%'
```

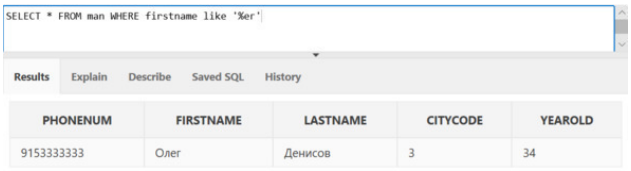


PHONENUM	FIRSTNAME	LASTNAME	CITYCODE	YEAROLD
9173333334	Алиса	Никифорова	4	31
9213333336	Алексей	Иванов	7	25
9213333331	Андрей	Некрасов	2	28
9214444444	Алексей	Галкин	1	38
9152222221	Андрей	Николаев	1	23

Рисунок 53. Запрос к MAN с LIKE

Выбрать из таблицы Man людей *, имена (FIRSTNAME) которых заканчиваются на «ег».

```
SELECT * FROM man WHERE firstname like '%er'
```

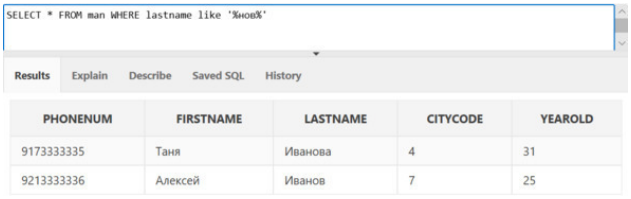


PHONENUM	FIRSTNAME	LASTNAME	CITYCODE	YEAROLD
9153333333	Олег	Денисов	3	34

Рисунок 54. Запрос к MAN с LIKE

Выбрать из таблицы Man имена и фамилии (FIRSTNAME, LASTNAME) людей, фамилии (LASTNAME) которых содержат слог «нов».

```
SELECT * FROM man WHERE lastname like '%нов%'
```



PHONENUM	FIRSTNAME	LASTNAME	CITYCODE	YEAROLD
9173333335	Таня	Иванова	4	31
9213333336	Алексей	Иванов	7	25

Рисунок 55. Запрос к MAN с LIKE

Выбрать из таблицы CITY города *, в названии (CITYNAME) которых вторая буква «л».

```
SELECT * FROM city WHERE cityname LIKE '_л%'
```

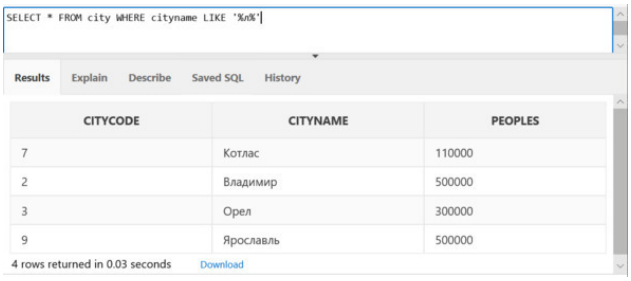


Рисунок 56. Запрос к CITY с LIKE

Выбрать из таблицы CITY города *, названия которых (CITYNAME) не начинаются с «К».

```
SELECT * FROM city WHERE NOT cityname LIKE 'К%'
```

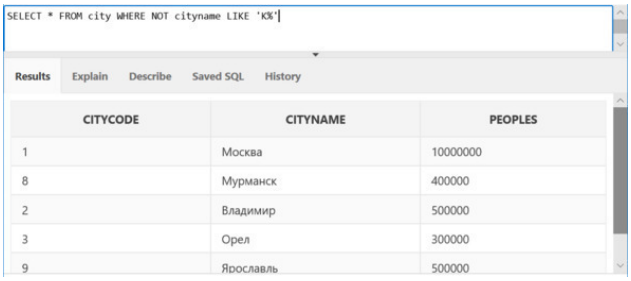


Рисунок 57. Запрос к CITY с NOT LIKE

Важные замечания

При использовании шаблонов с оператором LIKE в шаблоне необходимо соблюдать заглавные или строчные буквы, то есть регистр букв имеет значение при использовании LIKE.

Вопросы учеников

Приведите пример использования символа подчеркивания в шаблоне.

Да, вот пример:

```
SELECT * FROM AUTO WHERE color LIKE '_ИНИЙ'
```

Будут выбраны строки, начинающиеся с любого символа и заканчивающиеся на ИНИЙ.

Контрольные вопросы и задания для самостоятельного выполнения

1. Выберите из таблицы MAN всех людей *, чьи фамилии не начинаются со слога «Ив», напишите SQL-запрос.
2. Выберите из таблицы MAN всех людей *, в чьих именах есть русская буква «е».
3. Выберите из таблицы CITY города, где третья буква в наименовании «а».

Шаг 25. Работаем с датами

Введение

Для работы с календарным типом «дата-время» в языке SQL предусмотрено множество сервисных функций. Одни отвечают за преобразование типа «дата-время» в строку, другие позволяют выполнять операции с этим типом. Сейчас мы рассмотрим основные функции, наиболее часто применяемые на практике.

Теория и практика

Напоминаю: тип DATE содержит дату и время, это может быть, например, время заполнения накладной, дата и время создания записи в таблице.

Функция SYSDATE возвращает текущие дату и время.

```
SELECT sysdate FROM DUAL; -- выведет на экран текущее число, и текущее время.
```

Если мы попытаемся найти предыдущий день, тогда
SYSDATE-1 – предыдущий день, то же самое время;
SYSDATE-1/24*3 текущее время – 3 часа назад, то есть единица в типе данных «дата-время» (DATE) означает сутки, а 1/24 (1 час).
Функция TRUNC обрезает время из даты, оставляя только дату.

```
SELECT sysdate FROM dual; -- выведет дату и время текущее
SELECT trunc(sysdate) FROM dual; -- Только текущее число месяц год
```

Рассмотрим еще несколько примеров.

```
SELECT TRUNC(sysdate, 'MONTH') "FIRSTDAYOFMONTH" FROM DUAL;
```

– 01.09.2012
Первый день месяца, другая запись:

```
SELECT TRUNC(sysdate, 'MM')  "FIRSTDAYOFMONTH" FROM DUAL;
```

– 01.09.2012
Первый день года:

```
SELECT TRUNC(sysdate, 'YEAR')  "FIRSTDAYOFYEAR" FROM DUAL;
```

– 01.01.2012
Первый день года, другая запись:

```
SELECT TRUNC(sysdate, 'YY') "FIRSTDAYOFYEAR" FROM DUAL;
```

Функция EXTRACT предназначена для того, чтобы извлечь месяц, год, день из типа «дата время» (DATE).

Примеры работы функции EXTRACT
Извлечь месяц из даты.

```
SELECT EXTRACT(MONTH FROM sysdate) FROM dual;
```

Извлечь день из текущей даты.

```
SELECT EXTRACT(day FROM sysdate) FROM dual;
```

Извлечь год из текущей даты.

```
SELECT EXTRACT(YEAR FROM sysdate) FROM dual;
```

Преобразование даты в строку и строки в дату
Преобразование даты в строку делается с использованием функции TO_CHAR.

Синтаксис TO_CHAR (dt, формат даты):
формат даты

- YYYY – год;
- MM – месяц;
- DD – день;
- DN – день недели;
- hh24 – час;
- mi – минута;

- ss – секунда.

Преобразование строки в тип даты.
Синтаксис TO_DATE (строка, формат даты)

Примеры
Преобразовать текущую дату к строке в формате DD.MM.YYYY hh24:mi.

```
SELECT to_char(sysdate,'DD.MM.YYYY hh24:mi') as cdt from dual
```

Преобразовать вчерашнюю дату к строке DD.MM.YYYY hh24:mi.

```
SELECT to_char(sysdate-1,'DD.MM.YYYY hh24:mi') as cdt from dual
```

Преобразовать следующую строку к дате ’01.11.2011 21:11».

```
select to_date('01.11.2011 21:11', 'DD.MM.YYYY hh24:mi') from dual
```

Преобразовать к строке в запросе SELECT даты создания всех автомобилей BMW в формате DD.MM.YYYY hh24:mi из таблицы AUTO к строке:

```
select mark, to_char(releasedt, 'DD.MM.YYYY hh24:mi') as dt from AUTO where mark='BMW'
```

Важные замечания

Универсальный формат даты

В SQL ORACLE диалекта существует универсальная конструкция для работы с данными типа DATA, «дата-время» – DATE'YYYY-MM-DD», где YYYY – текущий год, MM – месяц, DD – день.

Примеры использования универсальной конструкции

Из таблицы AUTO вывести все автомобили, выпущенные после 01.05.2011.

```
select * from auto where releasedt > date'2011-05-01'
```

Из таблицы AUTO вывести все автомобили, выпущенные до 01.10.2014.

```
select * from auto where releasedt < date'2015-10-01'
```

Из таблицы AUTO вывести все автомобили, выпущенные после 01.01.2010 и до 01.10.2014, используя BETWEEN.

```
select * from auto where releasedt between date'2015-10-01' and date'2010-01-01' and
date'2014-10-01
```

Формат преобразования даты

Формат преобразования даты при преобразовании строки в дату должен соответствовать формату преобразуемой даты, иначе вы получите ошибку соответствия форматов.

Вопросы учеников

Как найти разницу между двумя датами в днях, часах, минутах?

В MS SQL для этого используется специальная функция DATEdiff, а в ORACLE все более сложно. Воспользуемся функцией NUMTODSINTERVAL и напишем запрос.

```
SELECT trunc(sysdate-trunc(sysdate,'MM')) || ' дней(я) ' as dtd ,
 EXTRACT(HOUR FROM NUMTODSINTERVAL((sysdate-trunc(sysdate,'MM')),'HOUR')) || ' часа
' as dth ,
 EXTRACT(MINUTE FROM NUMTODSINTERVAL((sysdate-trunc( sysdate,
'MM'))*1440,'MINUTE')) || ' мин ' as dtm ,
 TRUNC(EXTRACT(SECOND FROM NUMTODSINTERVAL((sysdate-trunc(sysdate,
'MM'))*86400,'SECOND'))) || ' сек ' as dts
from dual
```

Как найти дату-время, которая будет на 50 секунд позже текущего момента, как написать такой запрос?

Пример такого запроса: мы знаем, что один день равен 1, соответственно час 1/24, минута 1/ (24*60) и секунда 1/ (24*60*60) = 86400. Соответственно, запрос:

```
SELECT sysdate + 50*1/86400 from DUAL вернет нам дату время через 50 секунд от
настоящего момента.
```

А если нам нужны миллисекунды, что делать?

Для этого существует специальный тип данных **TIMESTAMP**, но с особенностями работы с этим типом данных мы познакомимся чуть позже.

Контрольные вопросы и задания для самостоятельного выполнения

1. Вывести год, месяц, день производства авто из таблицы AUTO.
2. Вывести дату 30 дней назад.
3. Вывести последний день месяца 80 дней назад.
4. Преобразовать следующую строку к дате 01.11.2011 21:11.
5. Преобразовать дату 5 дней назад к строке DD.MM.YYYY hh24:mi.
6. Вывести на экран дату и время 4 часа назад от текущего момента.

День шестой

Шаг 26. Функции и операторы для работы со строками и текстом

Введение

Обзор основных функций работы с текстом в SQL ORACLE диалекта и их применения.

Тестовые данные окружают нас повсюду, и без специальных сервисных функций работы со строками и текстом обойтись было бы сложно. SQL ORACLE диалекта предлагает множество функций для работы с текстовыми данными, основные из которых мы сейчас рассмотрим.

Теория и практика

Мы уже познакомились с оператором LIKE, позволяющим выбирать из базы данных строки с текстом, соответствующие определенному шаблону.

В SQL ORACLE также есть множество полезных сервисных функций для модификации строк, работы с подстроками, объединения строк.

Объединение строк – по-правильному конкатенация.

Для объединения строк в языке SQL диалекта ORACLE используется специальная синтаксическая конструкция ||.

```
select 'теле'||'визор' as q from DUAL
    -   телевизор
select 'LA'||'DA' as q from DUAL
- LADA
select firstname||' '||lastname as q from man
```

– телевизор.

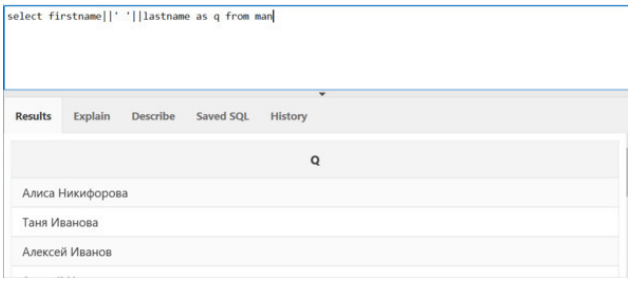


Рисунок 58. Запрос: пример объединения строк

INstr – поиск позиции подстроки в строке.

INSTR (STR1, STR2, POSn, DIRECTION) – возвращает позицию STR2 в строке STR1, где поиск осуществляется в позиции POSn

в направлении DIRECTION 1 – от начала строки, 0 – от окончания строки, то есть откуда мы начинаем поиск – от начала строки или с конца строки.

Примеры

```
SELECT INSTR('AAABAAAAABA','AB',1) FROM DUAL
-- 3
SELECT INSTR('AAABAAAAABA','AB',5) FROM DUAL
-- 9
SELECT INSTR('AAABAAAAABA','AB',1,1) FROM DUAL
-- 3
```

Найти первое вхождение буквы «а» в название городов в таблице CITY, вывести на экран и вхождение символа «а».

```
select cityname, instr(cityname, 'a', 1) iname from city
```

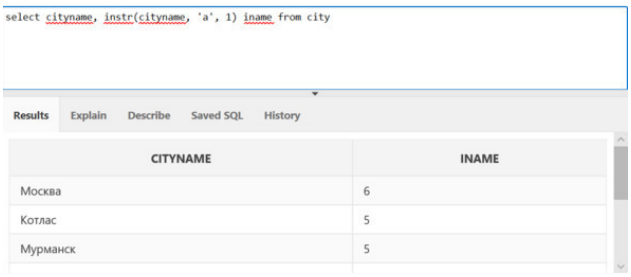


Рисунок 59. Запрос: пример INSTR

Найти **последнее вхождение буквы «а»** в название городов в таблице CITY, вывести на экран и номер последнего вхождения символа «а».

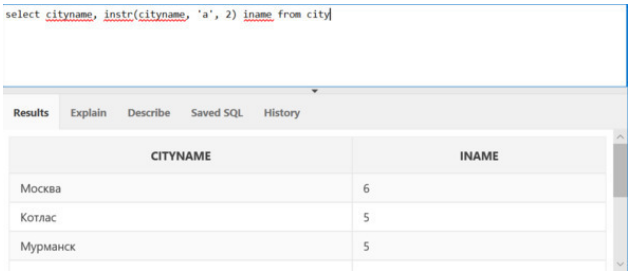


Рисунок 60. Запрос к CITY: пример INSTR

Length – длина строки в символах.
LENGTH (str1) возвращает длину строки str1 в символах.
Примеры
Длина строки «AAA»

```
select length('AAA') as ln from dual
```

– 3.
Длина строки «AAA1234»

```
select length('AAA1234') as ln from dual
```

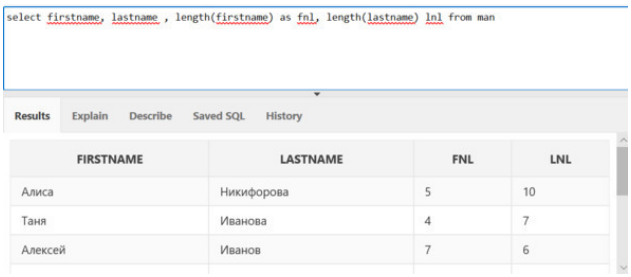
– 7.
Длина строки «привет мир»

```
select length('привет мир') as ln from dual
```

– 10.

Вывести из таблицы MAN имя, фамилию, длину имени и фамилии в символах.

```
select firstname, lastname , length(firstname) as fnl, length(lastname) lnl from man
```



Вывести из таблицы MAN записи *, где длина имени> 5.

```
select * from man where length(firstname) > 5
```

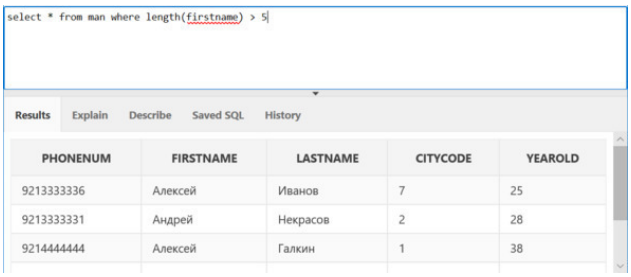


Рисунок 62. Запрос к MAN: пример LENGTH

Выбор подстроки из строки SUBSTR

SUBSTR (STR1, POS, LEN) выбирает LEN символов в строке str1, начиная с позиции POS.
STR1 – оригинальная строка.
POS – позиция, с которой начинается выделение.
NEWSUB – подстрока, на которую заменяем по умолчанию.

Примеры

Рассмотрим работу данной функции на примерах.

```
SELECT SUBSTR('ABCDEF',2,3) FROM DUAL
```

– BCD.

Выбрать все имена из MAN, которые начинаются с «Ан».

```
select firstname from man where substr(firstname,1,2)='Ан'
```

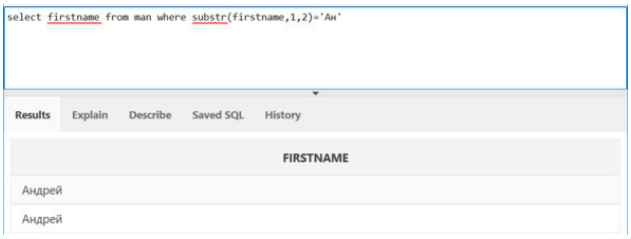


Рисунок 63. Пример работы SUBSTR: запрос

Замена подстроки в строке REPLACE

REPLACE (SRCSTR, OLDSUB, NEWSUB) – функция, которая возвращает преобразованную строку SRCSTR, где подстрока OLDSUB из строки SRCSTR заменяется на подстроку NEWSUBю

SRCSTR – оригинальная строка.

OLDSUB – заменяемая подстрока.

NEWSUB – подстрока, на которую заменяем, по умолчанию NULL.

Примеры

Заменить в имени в таблице MAN все буквы а на #.

```
select firstname, replace(firstname,'a','#') as fn from man
```

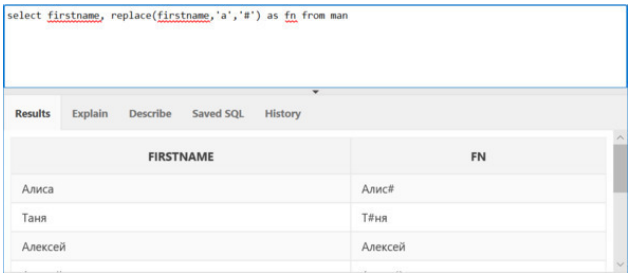


Рисунок 64. Запрос: пример REPLACE

Заменить в имени в таблице MAN слог «Ма» на #*.

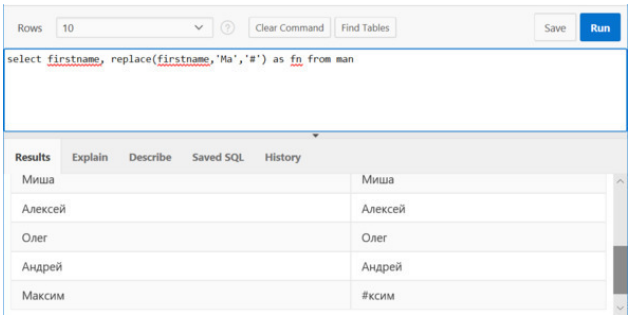


Рисунок 65. Запрос к MAN: пример REPLACE

Еще несколько примеров:

```
SELECT REPLACE('AAABB','AA','$$') FROM DUAL
```

– $$ABB.

```
SELECT REPLACE('AAABAAAAABA','AB',',ИНФО') FROM DUAL
```

– AA, ИНФОAAAA, ИНФОA.

Важные замечания

Функцию SUBSTR можно использовать с неполным набором параметров.

Например, в данном случае параметр LEN не задан, поэтому выбираются все символы в строке начиная с Pos.

```
SELECT SUBSTR('ABCDEF',3) FROM DUAL
```

Если мы укажем в качестве второго параметра отрицательное значение, то будет выбрано указанное количество символов с конца строки.

```
select substr('ABCDEF',-3) from dual;
```

– DEF.

То есть для того чтобы выбрать последние символы в подстроке, достаточно использовать отрицательный параметр pos.

Функция REPLACE поможет убрать ненужные символы из строки.

```
select replace('молоко','о') from dual
```

– млк.

Вопросы учеников

В других СУБД используются такие же функции для работы со строками?

Нет, функции для работы с текстовыми данными и строками несколько различаются в разных СУБД. Но принципиальные различия небольшие. Функции для работы со строками подробно описаны в документации к каждой СУБД.

Можно ли использовать регулярные выражения при выборе данных?

Да, разумеется, и для этого существуют специальные функции. Работу с регулярными выражениями мы рассмотрим далее, в следующих шагах.

Контрольные вопросы и задания для самостоятельного выполнения

1. Найти записи из таблицы **MAN**, начинающиеся на Ан, использовать SUBSTR.
2. Найти записи из таблицы **MAN**, где количество символов в фамилии человека больше 10.
3. Вывести из таблицы **MAN** имя, фамилию, количество символов в фамилии, где количество символов в фамилии человека больше 5.
4. Заменить буквы «о» в слове «молоко» на @, вывести результат, использовать **REPLACE** и DUAL.
5. Подсчитать количество букв «о» в слове «молоко», вывести результат, использовать **REPLACE**, **DUAL** и математику.
6. Найти первое вхождение буквы **«о**» в городе с кодом 1 из таблицы **CITY**.
7. Найти первое и последнее вхождения буквы «**и**» в городе с кодом 2 из таблицы **CITY**.

Шаг 27. Математика и пустые значения в запросах. Случайность – RANDOM

Введение

В запросах SQL мы можем использовать результаты математических вычислений.

Причем мы можем как выводить результаты математических вычислений, так и использовать математические выражения в условиях отбора строк WHERE.

Теория и практика

Повторим вычисление математических выражений в SQL.
Для математических выражений используются следующие операции:

+ сложение,
– вычитание,
/ деление,
* умножение.

А также знакомые нам со школы функции:

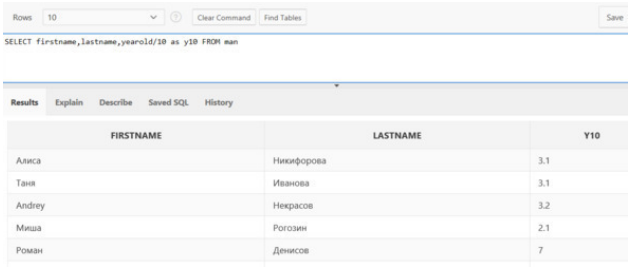
SQrt – квадратный корень,
MOD – остаток от деления,
trunc – округление до целого,
sIN – синус
cos – косинус.

Все математические операции выполняются только для числовых значений, числовых колонок с типами NUMBER или производными от NUMBER.
Более подробную информацию можно посмотреть в документации.

Примеры

Вывести из таблицы MAN имя, фамилию и возраст (FIRSTNAME, LASTNAME, YEAROLD) человека, разделенный на 10.

```
SELECT firstname,lastname,yearold/10 as y10 FROM man
```

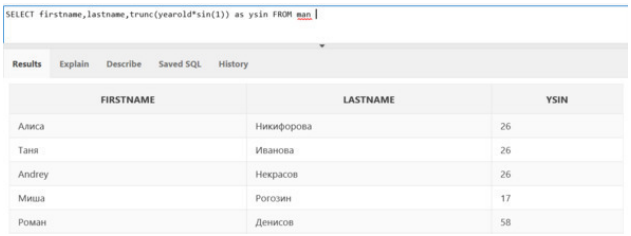


FIRSTNAME	LASTNAME	Y10
Алиса	Никифорова	3.1
Таня	Иванова	3.1
Andrey	Некрасов	3.2
Миша	Рогозин	2.1
Роман	Денисов	7

Рисунок 66. Запрос из MAN

Вывести из таблицы MAN имя, фамилию и возраст человека (FIRSTNAME, LASTNAME, YEAROLD), умноженный на sIN (1), округлить до целого.

```
SELECT firstname,lastname,trunc(yearold*sin(1)) as ysin FROM man
```



SELECT firstname,lastname,trunc(yearold*sin(1)) as ysin FROM man

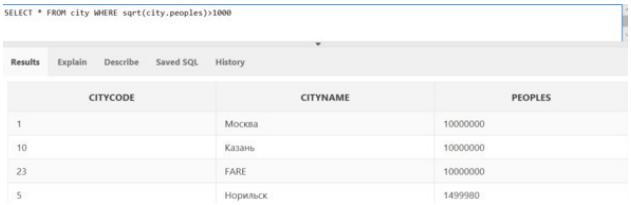
Results | Explain | Describe | Saved SQL | History

FIRSTNAME	LASTNAME	YSIN
Алиса	Никифорова	26
Таня	Иванова	26
Andrey	Некрасов	26
Миша	Рогозин	17
Роман	Денисов	58

Рисунок 67. Запрос к таблице MAN с вычислением

Вывести из таблицы CITY записи (*), где квадратный корень от населения города больше 1000 (PEOPLES).

```
SELECT * FROM city WHERE sqrt(city.peoples)>1000
```



SELECT * FROM city WHERE sqrt(city.peoples)>1000

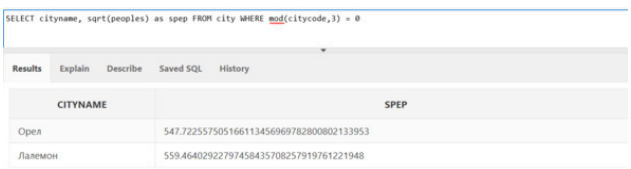
Results | Explain | Describe | Saved SQL | History

CITYCODE	CITYNAME	PEOPLES
1	Москва	10000000
10	Казань	10000000
23	FARE	10000000
5	Норильск	1499980

Рисунок 68. Запрос к CITY

Вывести из таблицы CITY название города (CITYNAME), квадратный корень от количества населения (PEOPLES), где значение кода города (CITYCODE) делится нацело на 3.

```
SELECT cityname, sqrt(peoples) as spep FROM city WHERE mod(citycode,3) = 0
```



SELECT cityname, sqrt(peoples) as spep FROM city WHERE mod(citycode,3) = 0

Results | Explain | Describe | Saved SQL | History

CITYNAME	SPEP
Орел	547.722557505166113456969782800802133953
Лалемон	559.464029227974584357082579197612219 48

Рисунок 69. Запрос к CITY: функция MOD

Вывести из таблицы CITY название города (CITYNAME), код города, разделенный на 3 (CITYCODE), где значение населения (PEOPLES), разделенное на 100, не больше 1 000 000.

```
SELECT cityname, citycode/3 as ccode FROM city WHERE Not peoples / 100 > 1000000
```

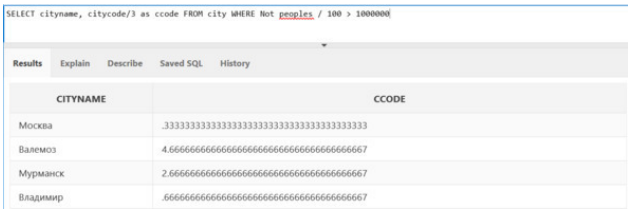


Рисунок 70. Запрос к CITY: деление

Вывести на экран результат выражения sIN (3) *10+10.

```
SELECT sin(3)*10+10 as v FROM dual
-- 11.411200080598672221007448028081102798 7
```

Вывести на экран результат выражения 1/10+10/1.

```
SELECT 1/10+10/1 as v FROM dual
```

– 10,1

Математика и пустые значения NULL

Если в математическом выражении используется пустое значение NULL, тогда значение любого математического выражения также будет NULL.

Например

– 10+NULL = NULL;

– 11*NULL+52+sIN (1) = NULL.

Эту особенность следует учитывать при построении запросов.

Генерация случайных чисел

SQL ORCALE диалекта также позволяет генерировать случайные значения, для этого используется специальный встроенный пакет (набор функций и процедур) dbms_rANDom.

Для генерации случайного числа используется специальная функция Value.

Функция VALUE возвращает случайное число, большее или равное 0 и меньшее 1, с 38 цифрами справа от десятичной части (38 знаков после запятой). Кроме того, вы можете получить случайное число x, где x больше или равно LOW и менее HIGH.

Синтаксис

DBMS_RANDOM.VALUE RETURN NUMBER

```
DBMS_RANDOM.VALUE( low IN NUMBER, high IN NUMBER) RETURN NUMBER;
```

Параметры:

LOW – наименьшее количество в диапазоне для генерации случайного числа. Номер, который генерируется, может быть равен LOW;

HIGH – наибольшее число для генерации случайного числа. Номер, который генерируется, будет меньше, чем HIGH.

Возвращаемое значение – NUMBER.

Примеры

```
select dbms_random.value from dual
```

– 0,777585712081073.

```
select dbms_random.value(10,15) from dual
```

– 11,3383710413575.

```
select dbms_random.value(3,5) from dual
```

– 3,67901998206503.

Важные замечания

Существует еще один способ генерации случайных текстовых данных с использованием пакета dbms_rANDom.

DBMS_RANDOM.STRING opt IN CHAR,
len IN NUMBER) RETURN VARCHAR2;
Параметры

«U», «U» – результат прописные буквы;
«L», «L» – результат строчные буквы;
«A», «A» – результат смешанные буквы, дело;
«X», «X» – результат верхний регистр букв и цифр;
«P», «P» – результат любых печатных символов;
len – длина возвращаемой строки.

Примеры

```
select DBMS_RANDOM.STRING('u', 4) from dual;
-- NXBM
  select DBMS_RANDOM.STRING('i', 5) from dual;
-- TTULB
  select DBMS_RANDOM.STRING('a', 6) from dual;
-- drpGPp
  select DBMS_RANDOM.STRING('x', 7) from dual;
-- RXP5CGQ
```

Вопросы учеников

Как будет вести себя запрос с математическими выражениями и NULL?

Очень важно понимать, что если в вашем запросе встречаются математические выражения и значения **NULL**, тогда результаты запроса могут быть некорректными, особенно если это касается агрегатных функций, SUM, avg, MIN или MAX.

Если в результате выполнения запроса возникнет математическая ошибка, например деление на 0?

Будет сообщение об исключении, и запрос выдаст пустой результат. Подобную ситуацию следует отслеживать в сложных запросах с множеством математических вычислений.

Контрольные вопросы и задания для самостоятельного выполнения

1. Вывести из таблицы MAN имя, фамилию и квадратный корень из 133.
2. Вывести из таблицы MAN имя, фамилию и возраст человека, умноженный на cos (5).
3. Вывести из таблицы CITY записи (*), где популяция делится без остатка на 10 000.
4. Вывести из таблицы CITY название города, квадратный корень от популяции, умноженный на 10, где значение кода города делится нацело на 5.

Шаг 28. Оператор IN

Введение

Предположим, нам необходимо сделать выборку данных по определенному перечню заданных значений, например выбрать автомобили десяти заданных цветов. Можно, конечно, написать большой сложный запрос с несколькими условиями, перечисленными в операторе OR.

Но сделать такой запрос достаточно утомительно, поэтому, к счастью, язык SQL предоставляет нам более удобную возможность – сделать выбор данных по заданному списку значений.

Теория и практика

Для удобной фильтрации выборки по списку значений в SQL существует специальный оператор IN.

Он позволяет сравнить значение заданного поля со списком значений и выбирать данные по результатам сравнения.

Синтаксис

```
SELECT перечень полей или * FROM таблица WHERE
поле IN (значение1, значение2, значение )
```

Примеры

Выбрать из таблицы MAN имена и фамилии людей (FIRSTNAME, LASTNAME), которым 22, 31, 34, 27 лет (YEAROLD).

```
SELECT firstname, lastname FROM man WHERE yearold in (22,31,34,27)
```

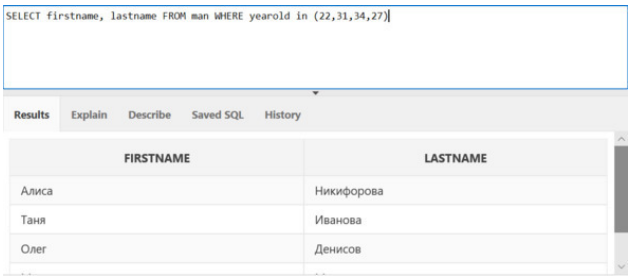


Рисунок 71. Запрос к таблице MAN: демонстрация IN

Выбрать из таблицы AUTO автомобили, марки которых VOLVO, BMW, AUDI.

```
SELECT * FROM auto WHERE mark in ('VOLVO', 'BMW', 'AUDI')
```

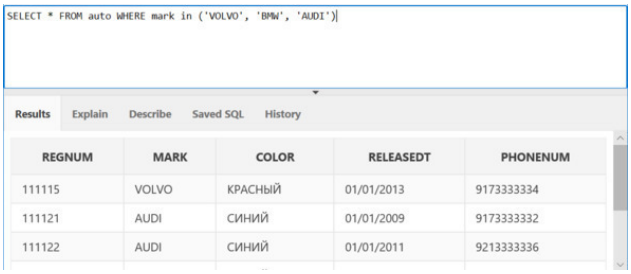


Рисунок 72. Запрос к таблице AUTO: оператор IN

Выбрать из таблицы AUTO * автомобили красного, синего и зеленого цветов (COLOR).

```
SELECT * FROM auto WHERE color in ('СИНИЙ','КРАСНЫЙ','ЗЕЛЕНЫЙ')
```

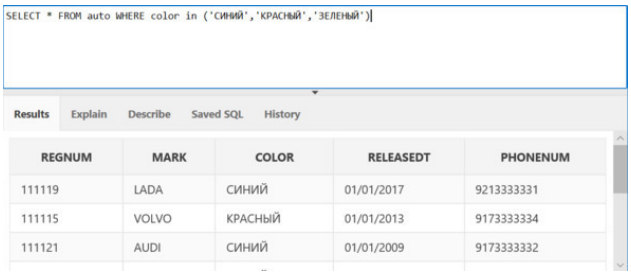


Рисунок 73. Запрос к таблице AUTO: COLOR IN

Выбрать из таблицы CITY * города с кодами 3, 5, 7 (CITYCODE), где население (PEOPLES) больше 100 000 человек.

```
SELECT * FROM city WHERE citycode in (3,5,7) and peoples > 100000
```

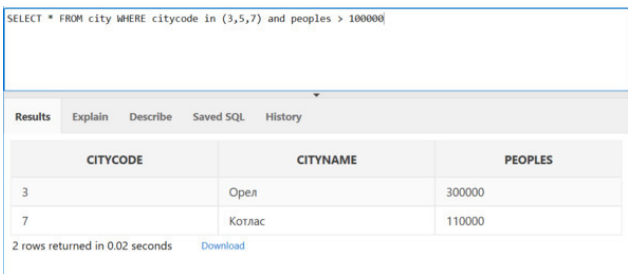


Рисунок 74. Запрос к таблице CITY: работаем с IN

Важные замечания

Следует помнить, что при работе со строковыми данными выражения в списках **IN** следует заключать в одинарные кавычки. Также в этом случае необходимо указывать заглавные и строчные буквы.

Вопросы учеников

С какими типами данных можно использовать конструкцию IN в языке SQL?

С любыми перечислимыми типами данных; исключения, наверное, составляют CLOB, BLOB.

Если в списке IN будет некоторое значение, а в таблице его не будет, что произойдет в этом случае?

Будут выбраны только те значения, которые есть в списке IN.

Контрольные вопросы и задания для самостоятельного выполнения

1. Выбрать из таблицы AUTO, где REGNUM – 111114, 111115, 111116.
2. Выбрать из таблицы MAN * людей с именами (FIRSTNAME) Андрей, Максим, Алиса.
3. Выбрать из таблицы CITY * города (CITYNAME) Москва, Владимир, Казань.
4. Выбрать из таблицы CITY * города с кодами (CITYCODE) 1, 3, 5, 7.

Шаг 29. Объединение нескольких таблиц в запросе

Введение

Важным достоинством языка SQL является возможность вывести на экран данные в одном запросе сразу из нескольких таблиц.

Теория и практика

Давайте рассмотрим записи из таблицы города CITY, записи из таблицы MAN нашей схемы.

Мы видим, что и в одной, и в другой таблице есть колонка «код города» (CITYCODE).

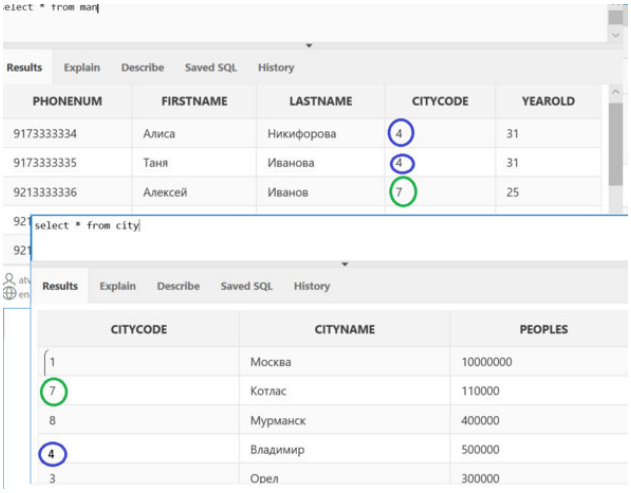


Рисунок 75. Связи между таблицами

Если посмотреть значения этой колонки (CITYCODE) и в той, и в другой таблице, то мы увидим, что числа, значения в этих колонках совпадают.

Например, в MAN есть записи с CITYCODE = 1 и в CITY есть записи CITYCODE = 1, то есть эта колонка является колонкой связи для таблиц CITY и MAN. По этим значениям мы можем выбрать данные из указанных таблиц, поэтому, используя эту колонку, мы можем извлечь данные из обоих таблиц.

Делается это следующим образом.

Первый вариант синтаксиса

```
select перечень полей или * from табл1, табл2
where табл1.kod = табл2.kod
```

Запомните, как объединяются таблицы в нашей схеме:
AUTO – > MAN = PHONENUM
CITY – > MAN = CITYCODE

Примеры

Объединим MAN и CITY по колонке CITYCODE, выведем значения всех колонок из этих таблиц *.

```
SELECT * FROM man m, city c WHERE c.citycode = m.citycode
```

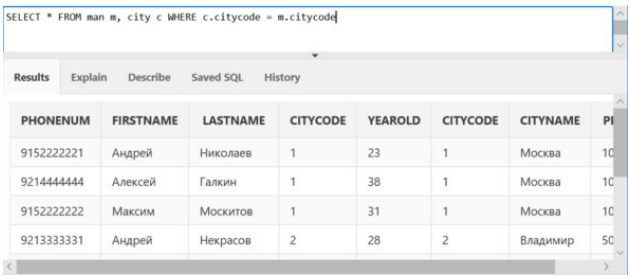


Рисунок 76. Запрос к таблицам CITY и MAN: объединение таблиц

Объединим MAN и CITY по колонке CITYCODE, выведем наименование и население (CITYNAME, PEOPLES) из таблицы CITY и имя и фамилию из таблицы MAN (FIRSTNAME, LASTNAME).

```
SELECT m.firstname, m.lastname, c.cityname,c.peoples FROM man m, city c WHERE c.citycode
= m.citycode
```

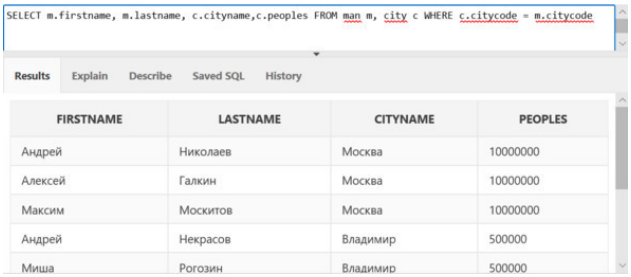


Рисунок 77. Объединение таблиц MAN, CITY

Этот синтаксис объединения двух таблиц является правильным, но существует еще и другая, более классическая форма записи для объединения двух таблиц – это объединение двух таблиц посредством конструкции JOIN.

Вариант синтаксиса с JOIN:

```
select перечень полей или * from табл1 inner join табл2
on табл1.kod = табл2.kod
```

Пример

Объединим MAN и CITY по колонке CITYCODE, выведем все колонки из этих таблиц * с использованием JOIN.

```
SELECT * FROM man m INNER JOIN city c ON c.citycode = m.citycode
```

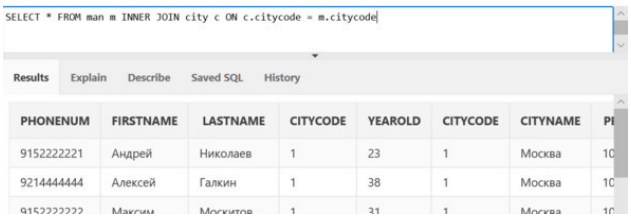
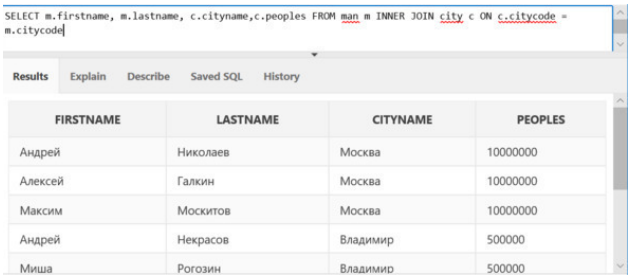


Рисунок 78. Объединение таблиц MAN и CITY: запрос

Объединим MAN и CITY по колонке CITYCODE, выведем код, наименование и население из таблицы CITY и имя и фамилию из таблицы MAN, используем JOIN.

```
SELECT m.firstname, m.lastname, c.cityname,c.peoples FROM man m INNER JOIN city c ON
c.citycode = m.citycode
```



Важные замечания

При объединении нескольких в таблиц в запросе необходимо обязательно учитывать наличие пустых (NULL) значений в объединяемых колонках, а также возможность выводить записи, значения которых есть в одной из таблиц в ключевых полях, а в другой нет. Такие объединения мы рассмотрим в следующем шаге.

Вопросы учеников

Как объединить сразу три таблицы?

Несложно, вот пример.
Вывести данные из таблиц AUTO, CITY, MAN.

```
select * from city, man, auto where auto.phonenum = man.phonenum and city.citycode =
man.citycode
```

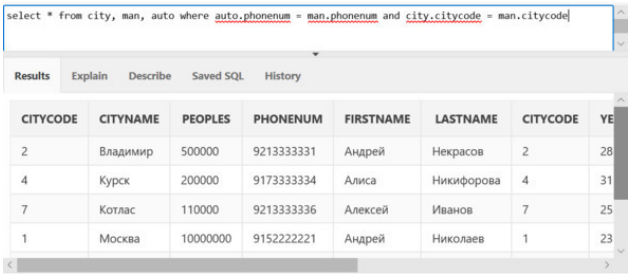


Рисунок 80. Объединение таблиц с JOIN

Или с JOIN:

```
select * from auto join man on auto.phonenum = man.phonenum join city on city.citycode =
man.citycode
```

Необходимы ли первичные ключи и вторичные ключи для объединения JOIN?

Желательны, но это не является обязательным условием.

Контрольные вопросы и задания для самостоятельного выполнения

1. Можно ли сделать объединение таблиц по нескольким полям?

2. Вывести все значения из таблицы MAN, из таблицы AUTO, объединить эти две таблицы посредством поля «номер телефона» (PHONENUM).

3. Вывести все значения из таблицы MAN, из таблицы CITY, объединить таблицы по коду города.

4. Вывести имя и фамилию из таблицы MAN, а также марку из таблицы AUTO, объединить две таблицы по номеру телефона (PHONENUM).

Шаг 30. Правое и левое объединение таблиц

Введение

В языке SQL существуют более сложные типы объединения таблиц. В этом шаге рассмотрим правое и левое объединение.

Теория и практика

Если внимательно присмотреться, то можно заметить, что в таблице CITY есть коды городов, которых нет в таблице MAN.

Точно так же и в таблице MAN есть номера телефонов, которых нет в таблице AUTO.

А что если нам необходимо выбрать из таблицы CITY все записи, а из таблицы MAN только те записи, которые совпадают с таблицей CITY по коду города (CITYCODE)?

Разумеется, в запросе, который объединяет обе эти таблицы.

Для этого в SQL ORACLE диалекта предусмотрен синтаксис правого и левого объединения таблиц, или RiGHT JOIN и LEFT JOIN.

Синтаксис LEFT JOIN

SELECT – перечень полей или * FROM – таблица, из которой мы извлекаем все записи; LEFT JOIN – таблица, где мы извлекаем только совпадающие записи; on – условие объединения ON (т1.код=т2.код).

Синтаксис RIGHT JOIN

SELECT перечень полей или * FROM – мы извлекаем только совпадающие записи; RIGHT JOIN – таблица, из которой мы извлекаем все записи ON (т1.код=т2.код).

Итак, если мы используем правое объединение RIGHT JOIN, из правой таблицы от конструкции JOIN будут выбраны все записи, а из левой таблицы только совпадающие записи.

Если мы используем левое соединение LEFT JOIN, из левой таблицы от конструкции JOIN будут выбраны все записи, а из правой таблицы будут выбраны совпадающие записи.

Примеры правого и левого объединения таблиц

Выбрать все записи из MAN и только совпадающие из AUTO.

```
SELECT * FROM man m LEFT JOIN auto a on m.phonenum = a.phonenum
```

В данном примере таблица MAN находится слева, в запросе от JOIN из нее будут выбраны все записи. Из таблицы AUTO будут выбраны только совпадающие записи.

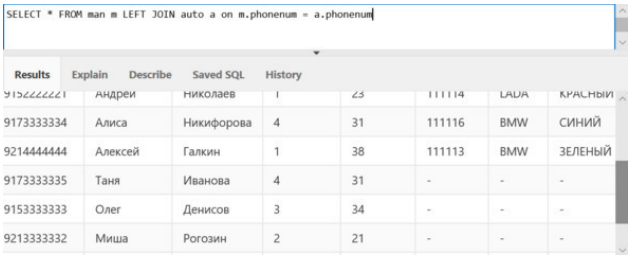


Рисунок 81. Объединение таблиц LEFT JOIN

Выбрать все записи из CITY и только совпадающие из MAN.

```
SELECT * FROM man m RIGHT JOIN city c on m.citycode = c.citycode
```

В данном примере таблица CITY находится справа от конструкции JOIN, из нее будут выбраны все записи, из таблицы MAN будут выбраны только совпадающие записи.

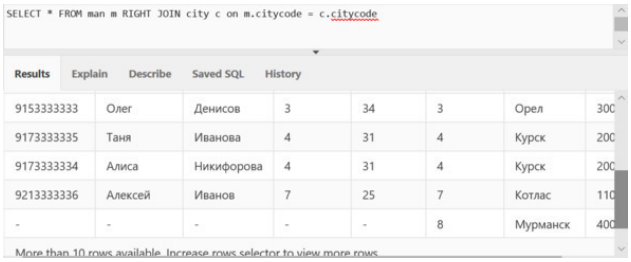


Рисунок 82. Объединение таблиц

Важные замечания

Конструкцию JOIN можно также применять с оператором WHERE.
Пример такого применения: вывести все авто и автовладельцев старше 30 лет.

```
SELECT * FROM man m RIGHT JOIN auto a on m.phonenum = a.phonenum where m.yearold >
30 or m.yearold is null
```

С помощью конструкции LEFT JOIN можно применить специальный фильтр, чтобы отсечь ненужные записи с помощью конструкции IS NOT NULL, IS NULL.

Обратите внимание, как используются псевдонимы в запросах с JOIN. Для конструкций, где связываются несколько таблиц, использование псевдонимов становится необходимым.

Вопросы учеников

Какие еще типы объединений существуют в языке SQL?

Существуют объединения CROSS JOIN, FULL JOIN, они используются на практике достаточно редко, вы можете прочитать об этих типах объединений в документации.

Контрольные вопросы и задания для самостоятельного выполнения

1. Повторите самостоятельно примеры из данного шага.
2. Используя RIGHT JOIN, выберите все записи из CITY и только совпадающие из MAN.
3. Используя LEFT JOIN, выберите все записи из CITY и только совпадающие из MAN.

День седьмой

Шаг 31. Объединение нескольких таблиц, дополнительные условия и сортировка результатов

Введение

Для запросов с объединением нескольких таблиц также можно применить сортировку.

Рассмотрим результаты сортировки при объединении нескольких таблиц. Приведем несколько примеров подобных запросов.

Теория и практика

При объединении таблиц сортировка данных также осуществляется с помощью операции ORDER BY.

Приведем несколько примеров использования сортировок.

Выведем на экран сведения: имя и фамилию человека (MAN) и название города (CITY), где проживает этот человек; отсортировать данные необходимо по количеству населения, в обратном порядке.

```
SELECT firstname, lastname, cityname FROM man INNER JOIN city ON man.citycode =
city.citycode ORDER BY city.peoples DESC
```

FIRSTNAME	LASTNAME	CITYNAME
915915%	Николаев	Москва
Hayk	Hovha	Москва
915915%	Москитов	Москва
Таня	Пьянчугин	Владимир
Andrey	Maksimov	Владимир
Max	Леонтьев	Владимир

Рисунок 83. Результаты запроса с сортировкой

Вывести из запроса сведения о автомобиле из таблицы AUTO, из таблицы MAN: выводим имя, фамилию, марку, цвет автомобиля – отсортируем результаты запроса по цвету автомобиля.

```
SELECT firstname, lastname, mark , color FROM man INNER JOIN auto ON man.phonenum =
auto.phonenum ORDER BY auto.color
```

FIRSTNAME	LASTNAME	MARK	COLOR
Hayk	Hovha	BMWE60	black
Алиса	Никифорова	VOLVO	КРАСНЫЙ
915915%	Николаев	LADA	КРАСНЫЙ
Алиса	Никифорова	BMW	СИНИЙ
Andrey	Некрасов	LADA	СИНИЙ

Рисунок 84. Результаты запроса с сортировкой

Важные замечания

Сортировка в запросе с объединением осуществляется по общему набору объединенных данных.

При сортировке данных с правым или левым объединением необходимо учитывать пустые значения колонок, по которым осуществляется сортировка.

Пример

```
SELECT firstname, lastname, mark , color FROM man LEFT JOIN auto ON man.phonenum =
auto.phonenum ORDER BY auto.mark DESC
```

Или обратная сортировка:

```
SELECT firstname, lastname, mark , color FROM man LEFT JOIN auto ON man.phonenum =
auto.phonenum ORDER BY auto.mark
```

Вопросы учеников

Можно ли сочетать сортировку в нескольких таблицах с фильтром WHERE?

Да, привожу пример запроса.

```
SELECT firstname, lastname, mark , color FROM man LEFT JOIN auto ON man.phonenum =
auto.phonenum
 WHERE man.yearold>25 ORDER BY auto.mark DESC
```

Контрольные вопросы и задания для самостоятельного выполнения

1. Повторите материалы данного шага.

2. Объедините таблицы AUTO, CITY, MAN; выведите следующие данные: марка автомобиля, цвет, имя владельца и город, где проживает владелец; отсортируйте результаты по колонке «название города».

3. Объедините таблицы AUTO, MAN; выведите следующие данные: марка автомобиля, цвет, имя владельца; отсортируйте результаты по колонке «марка автомобиля», «цвет автомобиля» в обратном порядке.

4. Объедините таблицы CITY, MAN; выведите следующие данные: название города, имя; отсортируйте результаты по колонке «название города», используя левое объединение LEFT JOIN.

Шаг 32. Группировка данных и агрегатные функции

Введение

Группировки являются одной из самых сложных для понимания тем, поэтому я прошу вас отнестись к этому шагу с максимальным вниманием.

Язык SQL позволяет делить данные, полученные в результате выборки, на группы, объединенные по набору колонок – признаков группы.

Например, мы видим, что в запросе по таблице AUTO есть автомобили КРАСНОГО, ЗЕЛЕНОГО и СИНЕГО цветов.

То есть в данной таблице можно определить группы КРАСНЫХ, ЗЕЛЕНЫХ, СИНИХ автомобилей.

Мы сгруппировали автомобили по признаку цвета – колонке таблицы COLOR, и мы также можем сгруппировать авто по марке LADA или BMW, колонка MARK.

Теория и практика

Для группировки данных, работы с группами в языке SQL используется специальный оператор GROUP BY.

Синтаксис

```
SELECT перечень колонок FROM имя таблицы WHERE условие отбора строк GROUP BY
колонка группировки 1 , колонка группировки n
```

Здесь группировки 1—n, колонка группировки – это названия колонок таблицы, по которой мы группируем данные.

Примеры

Группируем автомобили AUTO по марке (MARK).

```
SELECT mark FROM auto GROUP BY mark
```

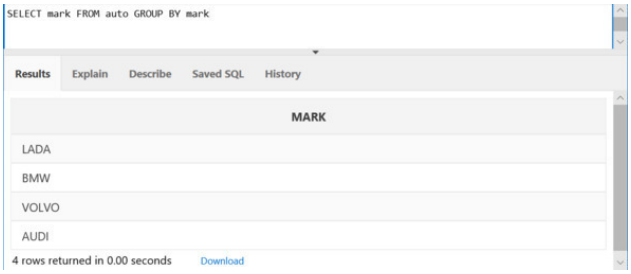


Рисунок 85. Демонстрация работы GROUP BY: таблица AUTO MARK

Группируем автомобили AUTO по цвету (COLOR).

```
SELECT color FROM auto GROUP BY color
```

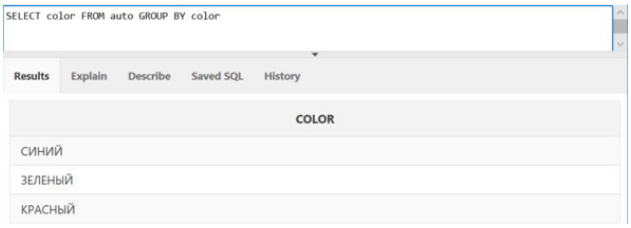


Рисунок 86. Группировка по цвету: запрос

Какие цвета автомобилей COLOR есть в таблице AUTO?

Группируем имена MAN по первой букве имени (FIRSTNAME).

```
SELECT SUBSTR(firstname,1,1) as firstlett FROM man GROUP BY SUBSTR(firstname,1,1)
```

В этом примере данные группируются не по колонке, а по результату выражения.

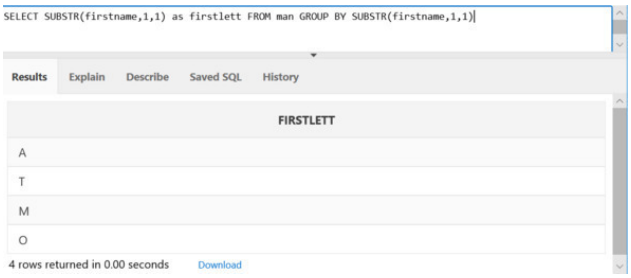


Рисунок 87. Группировка по первой букве имени

Агрегатные функции

Определение группировки было бы неполным без понимания агрегатных функций.

Агрегатная функция позволяет нам собрать статистическую информацию по заданной группе, количество элементов, сумму, среднее значение элементов в группе.

Основные агрегатные функции:

- COUNT (колонка) – возвращает количество элементов в группе;
- SUM () – сумма по заданным значениям, только для числовых данных;
- avg () – среднее по заданным значениям в группе, только для числовых данных;
- MAX () – максимальное значение в группе, только для числовых данных и дат;
- MIN () – максимальное значение в группе, только для числовых данных и дат.

Примеры

Вывести наименование города (CITYNAME), сумму и среднее население (PEOPLES) в группах городов CITY.

```
SELECT cityname, SUM(peoples) as sm, AVG(peoples) as av FROM city GROUP BY cityname,
citycode
```

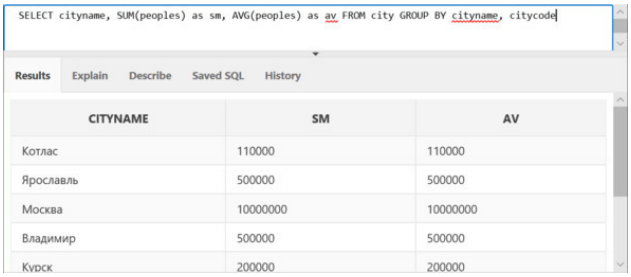


Рисунок 88. Группировка CITY по CITYNAME, CITYCODE

Один самых часто используемых на практике примеров – подсчет количества элементов в группе.

Вывести количество авто (AUTO), сгруппированных по цвету (COLOR, то есть сколько синих, сколько красных…).

```
SELECT color , COUNT(1) FROM auto GROUP BY color
```

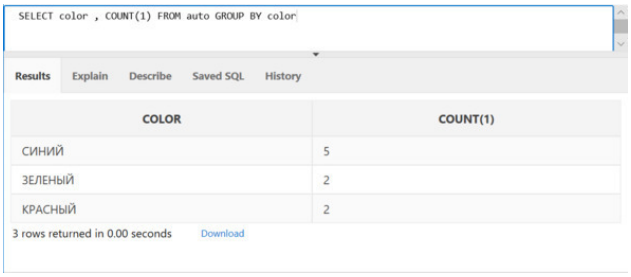


Рисунок 89. Группировка по цвету: запрос

Вывести максимальный и минимальный возраст (YEAROLD) людей из таблицы MAN, сгруппированных по имени.

```
SELECT firstname, MAX(yearold) as my, MIN(yearold) as miny FROM man GROUP BY firstname
```

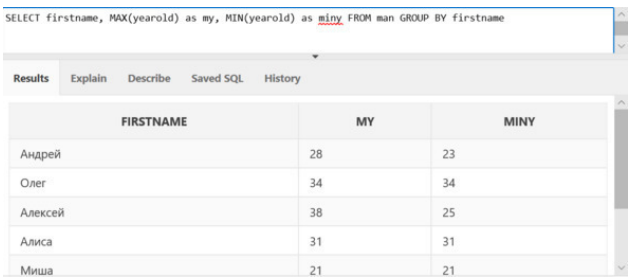


Рисунок 90. Запрос: группировка по FIRSTNAME MAN

Вывести средний возраст (YEAROLD) людей (MAN) и количество человек в группе, сгруппированных по имени.

```
SELECT firstname, AVG(yearold) as avy, COUNT(1) as cnt FROM man GROUP BY firstname
```

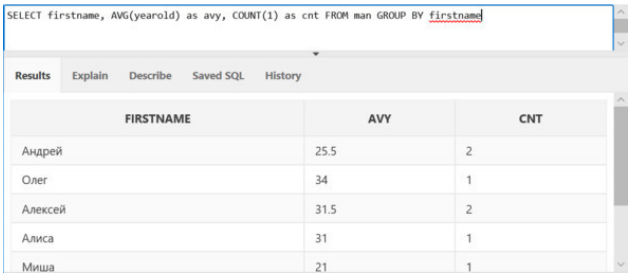


Рисунок 91. Таблица MAN: запрос группировки по FIRSTNAME

Вывести количество авто (AUTO) марки BMW, сгруппированных по цвету (COLOR, то есть сколько синих, сколько красных…).

```
SELECT color , COUNT(1) FROM auto WHERE mark = 'BMW'  GROUP BY color
```

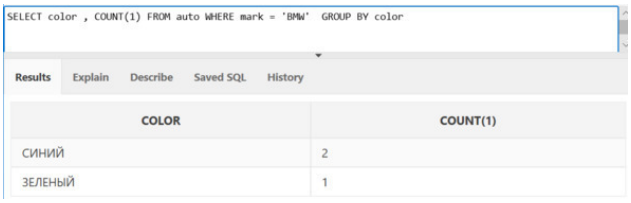


Рисунок 92. Группировка по цвету: запрос к AUTO

Важные замечания

Важное условие группировки: перечень колонок в запросе после команды SELECT должен входить в группировку GROUP BY или быть частью агрегатной функции.

Ошибочный запрос:

```
SELECT count(yearold) as cnt, firstname, lastname FROM Man GROUP BY  firstname
```

Правильный запрос:

```
SELECT count(yearold), firstname, lastname FROM Man GROUP BY  firstname, lastname
```

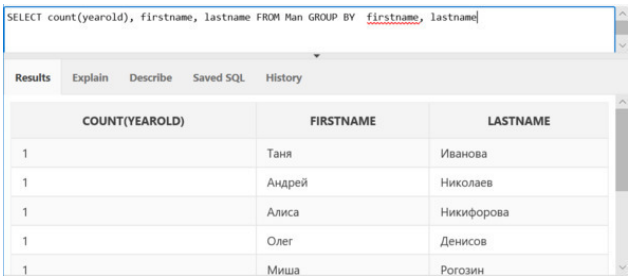


Рисунок 93. Агрегатная функция COUNT: запрос к таблице MAN

Агрегатные функции можно также применять и без использования GROUP BY. В этом случае результат будет считаться не для конкретных групп, а для всех записей запроса.

Примеры

```
SELECT COUNT(1) as cnt FROM auto
```

– результат 9.

Количество красных автомобилей (COLOR) в таблице AUTO:

```
SELECT COUNT(1) as cnt FROM auto WHERE color = 'КРАСНЫЙ'
```

– результат 2.

Количество людей (MAN), которым больше 27 лет (YEAROLD):

```
SELECT COUNT(1) as cnt FROM man WHERE yearold > 27
```

– результат 6.

Максимальный и минимальный возраст людей, которым больше 27 лет:

```
SELECT MAX(yearold ) as mxt, MIN(yearold ) mnt FROM man WHERE yearold > 27
```

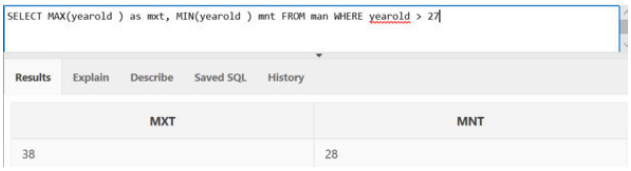


Рисунок 94. Агрегатные функции MIN, MAX: запрос к MAN

Максимальная и минимальная дата выпуска (RELEASEDT) машин BMW:

```
SELECT MAX(releasedt) as mxt, MIN(releasedt) mnt FROM auto
```

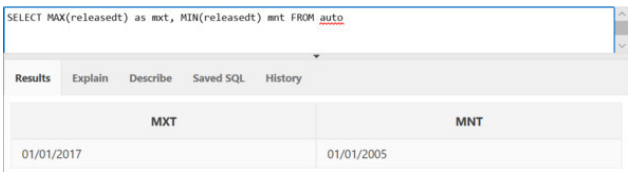


Рисунок 95. Агрегатные функции MIN, MAX: запрос к AUTO

Сумма населения (PEOPLES) во всех городах таблицы CITY:

```
SELECT SUM(peoples) as sumpeoples FROM city
```

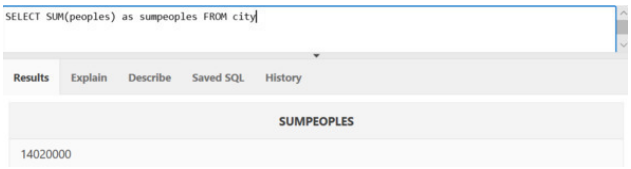


Рисунок 96. Агрегатная функция SUM: запрос к CITY

Сумма возраста (YEAROLD) людей MAN, которым больше 30 лет:

```
SELECT SUM(yearold ) as sxt FROM man WHERE yearold > 30
```

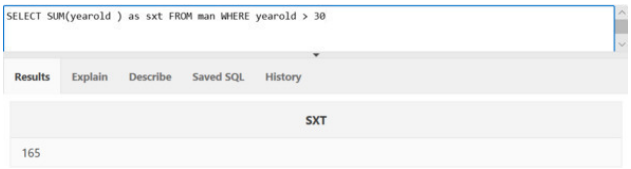


Рисунок 97. Агрегатная функция SUM: запрос к MAN

Агрегатные функции и пустые значения

Особенно аккуратно необходимо относиться к наличию пустых значений (NULL) в ячейках таблицы при применении агрегатных функций.

Рассмотрим два запроса:

```
SELECT count(phonenum) as cpn FROM auto
```

– результат 7.

```
SELECT count(*) as cpn FROM auto
```

– результат 9.

Так произошло, потому что агрегатная функция COUNT подсчитает количество элементов в заданной колонке без учета пустых значений в ячейках этой колонки, без учета NULL-значений.

Агрегатные функции SUM, MAX, MIN по пустым значениям NULL вернут нам пустое значение в качестве результата – NULL, но сумма, или максимальное, или минимальное значение будут правильными.

Вопросы учеников

Я видел выражение COUNT (1) в одном из примеров. Что значит COUNT (1) и почему именно 1?

Оператор COUNT подсчитывает количество элементов в группе, вместо элемента мы задаем единичку; таким образом, каждой строчке в группе будет соответствовать единица, количество строчек будет считаться как сумма единичек, строчек.

Как использовать группировки совместно с функциями для работы со строками и другими встроенными функциями языка SQL? Приведите несколько примеров, пожалуйста.

У нас уже был подобный пример, но для лучшего понимания посмотрим еще несколько примеров.

Группируем имена MAN по первой букве имени (FIRSTNAME); найдем, сколько имен начинаются с заданной буквы в таблице MAN.

```
SELECT SUBSTR(firstname,1,1) as firstlett, count(1) cnt FROM man GROUP BY
SUBSTR(firstname,1,1)
```

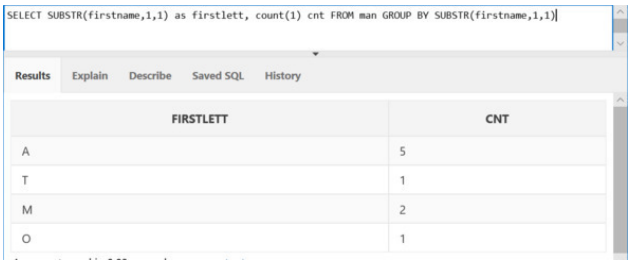


Рисунок 98. Группировка по SUBSTR: таблица MAN

Сгруппируем названия городов по последним двум буквам и найдем сумму количества населения в каждой группе.

```
SELECT substr(cityname,-2) as sbn , count(1) cnt, sum(peoples) as sp FROM city GROUP by
substr(cityname,-2)
```

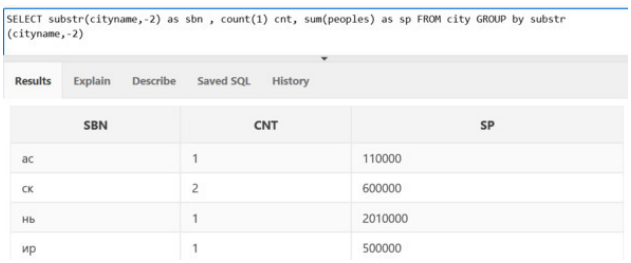


Рисунок 99. Группировка по SUBSTR: таблица MAN

GROUP BY можно использовать вместо DISTINCT?

Во множестве случаев GROUP BY действительно заменяет DISTINCT.
Например, вывести уникальные цвета и марки автомобилей из таблицы AUTO.

```
SELECT color , mark FROM auto GROUP BY color, mark
```

Результат одинаков при использовании GROUP BY и DISTINCT.

```
SELECT DISTINCT color , mark FROM auto
```

COLOR	MARK
КРАСНЫЙ	VOLVO
ЗЕЛЕНЫЙ	BMW
ЗЕЛЕНЫЙ	LADA
СИНИЙ	LADA
СИНИЙ	AUDI

Рисунок 100. Запрос к AUTO: демонстрация DISTINCT

Контрольные вопросы и задания для самостоятельного выполнения

1. Найдите ошибку в запросе:

```
SELECT mark, color FROM auto GROUP BY color
```

2. Выведите количество авто из AUTO, сгруппированных по марке (MARK) (то есть сколько BMW, сколько LADA…).

3. Выведите количество синих (COLOR) автомобилей в таблице AUTO.

4. Выведите максимальную и минимальную численность населения (PEOPLES) во всех городах (CITY).

5. Выведите средний возраст людей (MAN) и количество человек в группе, сгруппированных по ПЕРВОЙ БУКВЕ имени (FIRSTNAME).

Шаг 33. Сложные группировки с объединениями, сортировка результатов

Введение

Язык SQL позволяет гармонично сочетать операции группировки данных GROUP BY с операторами объединения JOIN.

Теория и практика

Конструкция GROUP BY оперирует с результатами запроса, поэтому прекрасно работает с запросами на объединение данных JOIN.

Приведем несколько примеров.

Вывести сведения: список марок автомобиля, а также сколько людей приобрели авто указанной марки; объединить таблицы MAN и AUTO по колонке PHONENUM.

```
select mark, count(man.phonenum) cnt from auto inner join man on man.phonenum =
auto.phonenum group by mark
```

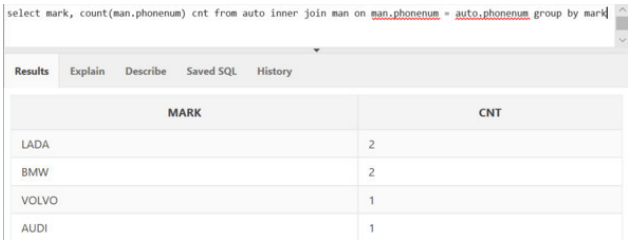


Рисунок 101. Группировка по нескольким таблицам

Вывести наименование города, код города, а также количество людей и средний возраст людей из таблицы MAN, проживающих в данном городе, объединив таблицы MAN и CITY по колонке CITYCODE.

```
select city.cityname as cityname, count(man.phonenum) as cn, avg(man.yearold) as avgyo
from man inner join city on man.citycode = city.citycode group by cityname
```

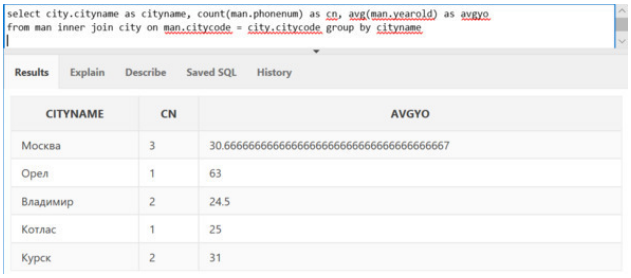


Рисунок 102. Группировка по нескольким таблицам MAN, CITY

Сортировка результатов запросов

В этих сложных запросах можно выполнять сортировку, в том числе и по сгруппированным записям.

Вывести сведения: список марок автомобиля, а также сколько людей приобрели авто указанной марки, объединив таблицы MAN и AUTO по колонке PHONENUM, отсортировать по названию марок авто.

```
select mark, count(man.phonenum) cnt from auto inner join man on man.phonenum =
auto.phonenum group by mark order by mark
```

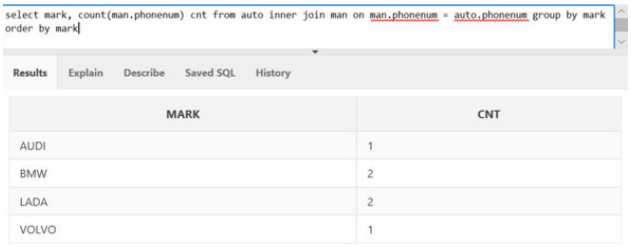


Рисунок 103. Группировка по таблицам AUTO, MAN

Важные замечания

Также в подобных запросах вполне допускается сортировка по агрегатной функции.

Вывести наименование города, код города, а также количество и средний возраст людей из таблицы MAN, проживающих в данном городе. Отсортировать результаты по количеству людей, а также по суммарному возрасту людей.

```
select city.cityname as cityname, count(man.phonenum) as cn, avg(man.yearold) as avgyo
        from man inner join city on man.citycode = city.citycode group by cityname
        order by count(man.phonenum), sum(man.yearold)
```

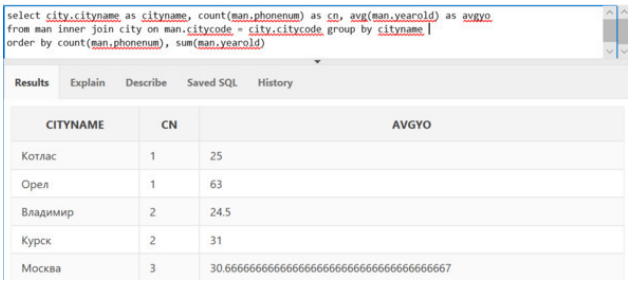


Рисунок 104. Группировка по таблицам MAN, CITY

Вопросы учеников

А можно ли с помощью агрегатной функции COUNT подсчитать количество уникальных элементов?

Да, для этого мы можем использовать уже знакомую нам конструкцию DISTINCT.

Предположим, что нам необходимо вывести список марок автомобиля, а также сколько людей с уникальным именем приобрели авто указанной марки, объединив таблицы MAN и AUTO по колонке PHONENUM.

```
select mark, count(distinct man.firstname) cnt from auto inner join man on man.phonenum =
auto.phonenum group by mark
```

Обратите внимание: мы использовали конструкцию **DISTINCT MAN.FIRSTNAME** в агрегатной функции COUNT.

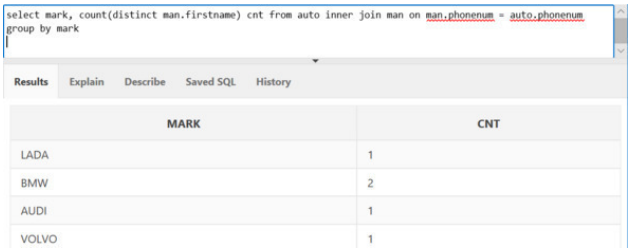


Рисунок 105. Группировка объединенных данных по MARK таблиц AUTO, MAN

Контрольные вопросы и задания для самостоятельного выполнения

1. Выведите марки автомобилей из таблицы AUTO и найдите количество людей, купивших одну и ту же марку авто, не проживающих в одном и том же городе.

2. Выведите названия городов и сумму возрастов, средний возраст жителей этого города из таблицы MAN, отсортируйте данные по количеству жителей из таблицы MAN в каждом из городов.

Шаг 34. HAVING как фильтр для групп и сложные группировки данных. ROWID – уникальный идентификатор строки. Дубликаты строк

Введение

В некоторых задачах необходимо применить фильтр уже для результата группировки записей. Подобный фильтр применяется, например, при ограничении количества выводимых групп.

Теория и практика

Перед нами неизбежно возникает необходимость ограничить результаты вывода групп.

Например, у нас есть группировка автомобилей по марке.

В нашей таблице есть несколько марок машин; теперь попробуйте представить ситуацию, когда в этой таблице были бы машины всех возможных марок.

Для того чтобы установить критерии отбора результатов группировки, в языке SQL есть специальный оператор HAVING, используемый совместно с GROUP BY.

Синтаксис

```
SELECT перечень полей FROM таблица WHERE условия
GROUP BY перечень полей HAVING условие
```

Здесь после HAVING условие – это условие ограничения на вывод групп.

Примеры

Сгруппировать данные в таблице AUTO по цвету (COLOR), вывести только группы красных авто и синих.

```
SELECT color FROM auto GROUP BY color HAVING color in ('СИНИЙ', 'КРАСНЫЙ')
```

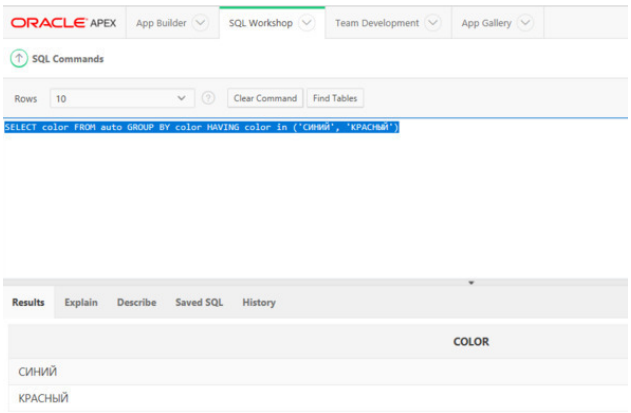


Рисунок 106. Группировка по цвету: таблица AUTO

Сгруппировать данные в таблице AUTO по марке, найти количество в группе, выбрать только группы, где больше одного авто.

```
SELECT mark, COUNT(1) cn FROM auto GROUP BY mark HAVING count(1) > 1
```

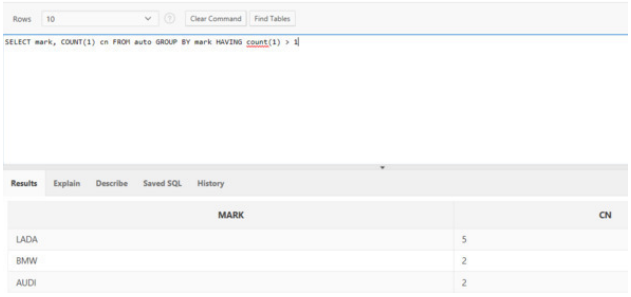


Рисунок 107. Группировка по первой букве

Сгруппировать данные по первой букве имени (FIRSTNAME), найти все группы, где средний возраст (YEAROLD) людей (MAN) больше 29 лет.

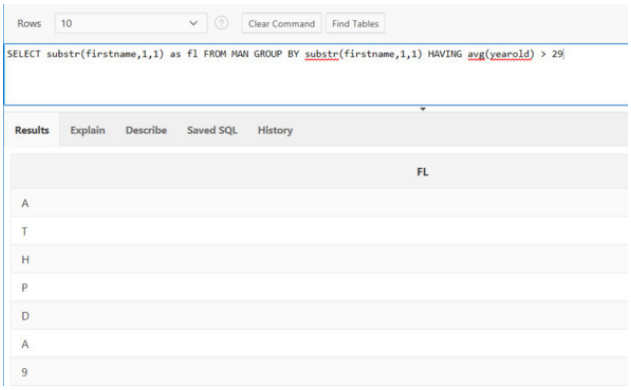


Рисунок 108. Использование HAVING: запрос

```
SELECT substr(firstname,1,1) as fl FROM MAN GROUP BY substr(firstname,1,1) HAVING
avg(yearold) > 29
```

Дубли строк

В ORACLE SQL существует специальный идентификатор для каждой строки таблицы – это колонка ROWID, уникальный идентификатор строки.

Что такое ROWID? Уникальный идентификатор: состоит из номера объекта (32 бита), относительного номера файла (10 бит), номера блока (22 бита) и номера строки (16 бит).

Как использовать ROWID? Если нет другого уникального идентификатора, первичного ключа, то можно опираться на ROWID.

Пример работы с ROWID – находим дубли.

```
select t.rowid, t.nm from t where (t.rowid not in ( select min(t.rowid) from t group by nm ))
```

Пример работы с ROWID – находим количество повторенийю

```
select count(t.rowid)  , nm from t group by nm
```

Пример работы с ROWID – сортировка.

```
select t.rowid , nm from t order by t.rowid
```

Для удаления дубликатов строк в таблице без первичного ключа можно обратиться именно к ROWID-колонке.

Пример удаления дублей из таблицы AUTO

```
delete auto where rowid not in   (select max(rowid) from auto group by regnum, mark, color,
releasedt, phonenum)
```

В этом примере группируем таблицу AUTO по всем колонкам, находим MAX (ROWID) сгруппированных записей и удаляем те записи, ROWID которых нет в сгруппированном наборе данных.

Важные замечания

• Оператор HAVING нельзя использовать в запросах без GROUP BY.
• Оператор HAVING возможно также применять вместе с сортировкой ORDER BY.

Синтаксис

```
SELECT перечень Колонок FROM таблица WHERE условия
GROUP BY перечень Колонок
HAVING условие на отбор записей группировки
OREDER BY Колонки для сортировки группы
```

Примеры

Сгруппировать данные в таблице AUTO по цвету, найти количество авто в каждой группе, отсортировать по цвету.

```
SELECT color FROM auto GROUP BY color HAVING color in( 'КРАСНЫЙ','СИНИЙ') ORDER BY
color
```

Сгруппировать данные в таблице AUTO по цвету, найти количество авто в каждой группе.

Отсортировать количество авто в каждой группе.

```
SELECT color, COUNT(1) ca FROM auto GROUP BY color ORDER BY count(1)
```

Вопросы учеников

Есть ли другие способы удаления дублей из таблиц?

Да, например с использованием EXIST, с использованием аналитических функций. Используем EXIST:

```
delete auto a where not exists (
 select 1   from (select max(rowid) rid from auto group by regnum, mark, color, releasedt,
phonenum) ar where a.rowid = ar.rid);
```

Будет ли работать HAVING в запросах с несколькими таблицами?

Да, ограничений нет.

Пример запроса:

```
SELECT cityname, avg(yearold) FROM CITY c INNER JOIN MAN m ON c.citycode = m.citycode
GROUP BY cityname HAVING avg(yearold)>27
```

Контрольные вопросы и задания для самостоятельного выполнения

1. Напишите запрос, который группирует автомобили по марке, находит количество авто каждой марки и выводит на экран только марки авто BMW, LADA – воспользуйтесь HAVING.
2. Напишите запрос, который бы выводил дублирующиеся строки из таблицы CITY.
3. Напишите запрос, который бы удалял дубликаты строк из таблицы MAN.

Шаг 35. Подзапрос для множеств WHERE IN SELECT

Введение

Ранее мы уже встречались с оператором IN в запросах SQL в инструкции WHERE, в качестве фильтра по перечислению значений.

Сейчас мы познакомимся с еще одним способом использования IN, который достаточно часто применяется на практике.

Теория и практика

Важным свойством использования IN является то, что в качестве источника значений для списка оператора может являться другой подзапрос.

Вспомните синтаксис запроса с инструкцией IN, который мы проходили ранее.

SELECT * или колонки через запятую FROM таблиц WHERE имяколонки IN (перечень значений).

```
SELECT * FROM auto WHERE Color in ('КРАСНЫЙ', 'СИНИЙ', 'ЗЕЛЕНЫЙ')
```

В рассматриваемом нами варианте в этом шаге вместо списка значений мы сможем задать подзапрос, возвращающий необходимые нам значения.

Синтаксис

```
SELECT * или колонки через запятую FROM таблиц WHERE колонки IN (SELECT
колонка_сравнения FROM таблица)
```

Примеры

Вы помните, по каким колонкам связаны таблицы в нашей схеме? Если нет, то посмотрите, пожалуйста, на схему ниже.

AUTO – > MAN = PHONENUM
CITY – > MAN = CITYCODE

Это может понадобиться при рассмотрении примеров.

Выбрать все машины * AUTO, где владельцам (MAN) больше 35 лет (YEAROLD). Использовать IN с подзапросом.

```
SELECT * FROM auto WHERE phonenum IN (SELECT phonenum FROM man WHERE
yearold>35)
```

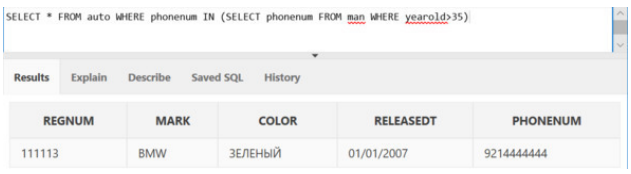


Рисунок 109. Пример запроса с IN SELECT

Выбрать все города (CITY), где есть записи из таблицы MAN. Использовать IN с подзапросом.

```
SELECT * FROM city WHERE citycode IN (SELECT citycode FROM man)
```

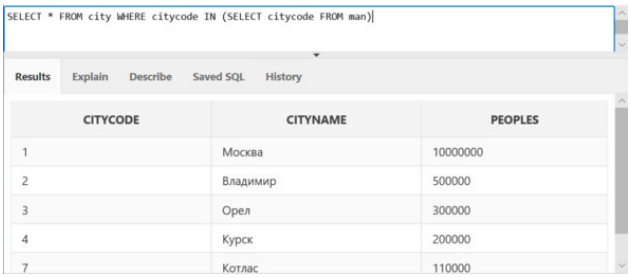


Рисунок 110. Пример запроса к CITY: запрос к MAN

Выбрать все записи из MAN, где люди проживают в городах (CITY) с населением (PEOPLES) больше 700 000 человек. Использовать IN с подзапросом.

```
SELECT * FROM man WHERE citycode IN (SELECT citycode FROM city WHERE peoples >
700000)
```

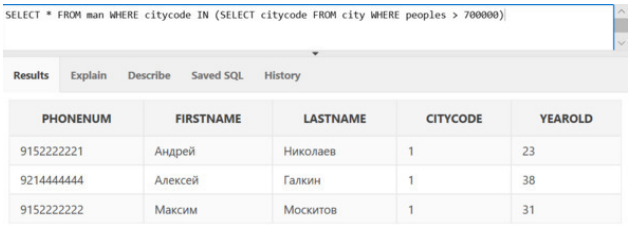


Рисунок 111. Запрос с IN SELECT к таблице MAN

Важные замечания

Тип данных сравниваемого списка SELECT должен совпадать с типом данных колонки сравнения, иначе необходимо прибегнуть к преобразованию типов.

При конструкции IN с подзапросом игнорируются NULL-значения. Для работы с NULL-значениями необходимо использовать функцию преобразования NVL.

Вопросы учеников

Каким образом написать IN по нескольким колонкам, что-то подобное следующему запросу?

```
SELECT * FROM TABLE1 WHERE KEY1,KEY2 IN (SELECT KEY1,KEY2 FROM TABLE2)
```

Нет, такой синтаксис классический SQL не поддерживает, так сделать не получится. Сравнение в IN осуществляется по одной колонке, но всегда можно использовать другие варианты. Например, конструкцию EXISTS.

Контрольные вопросы и задания для самостоятельного выполнения

1. Выбрать все машины (AUTO), где имя владельца MAN (fIStNAME) начинается с буквы A, использовать IN с подзапросом.

2. Выбрать все города (CITY), где есть записи из таблицы MAN и людям больше 35 лет (YEAROLD), использовать IN с подзапросом.

3. Выбрать все машины (AUTO), где возраст владельца больше 37 лет (YEAROLD) и длина имени больше пяти букв (FIRSTNAME), использовать IN с подзапросом.

День седьмой

Шаг 36. Подзапросы EXISTS

Введение

Как правило, запросы SQL с конструкцией EXISTS воспринимаются учениками как наиболее сложный материал.

Теория и практика

Итак, подзапрос с использованием EXISTS является наиболее сложным для понимания типом подзапроса в SQL.

Но мы попробуем разобраться, и здесь очень важно осознать, что подзапрос EXIST является предикатом, возвращает нам либо истину, либо ложь, то есть подзапрос с EXIST – это критерий того, будет на экран выведена данная строка либо нет. Если подзапрос возвращает хоть одну строку, то внешний запрос выводит данные – конкретную связанную строчку, если нет, данные не выводятся.

Также подзапрос EXISTS подразумевает объединение, то есть внутренний подзапрос связывается определенным отношением с внешним запросом.

Рассмотрим синтаксис EXISTS

```
SELECT поля FROM таблица1 т1
WHERE EXISTS (SELECT 1 FROM таблица2 т2 WHERE
t1.key = t2.key
)
```

Здесь (SELECT 1 FROM таблица2 т2 WHERE t1.KEY = t2.KEY) является внутренним подзапросом, который либо возвращает строку, либо нет, отношением, влияющим на поведение внешнего запроса.

Если (SELECT 1 FROM таблица2 т2 WHERE t1.KEY = t2.KEY) возвращает какие-либо данные для внешнего запроса, тогда **SELECT поля FROM таблица1 т1** выводит соответствующую строку на экран.

Впрочем, гораздо проще всего понять работу EXIST на примерах.

Итак, примеры:

Вывести на экран все города * из таблицы CITY, для которых есть соответствующая запись таблицы MAN, использовать EXISTS, связь по полю CITYCODE.

```
SELECT * FROM city c WHERE EXISTS (SELECT 1 FROM man m WHERE c.citycode = m.citycode )
```

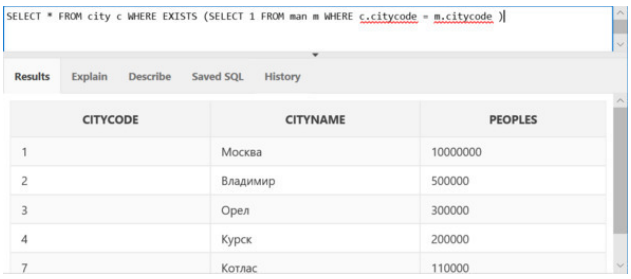


Рисунок 112. Запрос с EXISTS к таблице MAN

Подзапрос EXISTS выбирает данные, связанные с внешним запросом по колонке CITYCODE, то есть для каждой возвращаемой строки внешнего запроса SELECT * FROM CITY проверяется по CITYCODE наличие такой строки во внутреннем запросе SELECT 1 FROM MAN m WHERE c.CITYCODE = m.CITYCODE.

Если подзапрос находит такую строку, то строчка из внешнего запроса выводится на экран.

Вывести имя и фамилию человека (FIRSTNAME, LASTNAME) из таблицы MAN, который проживал бы в городе с населением (PEOPLES) больше 1 миллиона человек. Использовать EXISTS.

```
SELECT firstname, lastname FROM man m WHERE EXISTS (SELECT 1 FROM city c WHERE
c.citycode = m.citycode AND  c.peoples > 1000000)
```

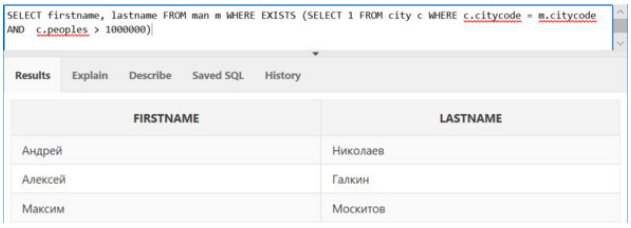


Рисунок 113. Запрос с EXISTS к таблице MAN

Подзапрос EXISTS выбирает данные, связанные с внешним запросом по колонке CITYCODE, то есть для каждой возвращаемой строки внешнего запроса SELECT FIRSTNAME, LASTNAME FROM MAN m проверяется по CITYCODE наличие такой строки во внутреннем запросе SELECT 1 FROM CITY c WHERE c.CITYCODE = m.CITYCODE, в котором есть дополнительное условие на количество населения c.PEOPLES> 1 000 000. Если подзапрос находит такую строку, то строчка из внешнего запроса выводится на экран.

Важные замечания

Одним из важных правил использования конструкции EXISTS является объединение внутреннего подзапроса с внешним запросом по ключевым колонкам. Условия данного объединения надо внимательно писать: если неправильно задать эти параметры, ключевые колонки, то результат запроса будет неверным.

Также эффективно использовать EXIST с логическим операндом NOT, например, используя EXISTS, вывести на экран все города (*) из таблицы CITY, для которых нет соответствующей записи в таблице MAN, связь по полю CITYCODE.

```
SELECT * FROM city c WHERE NOT EXISTS (SELECT 1 FROM man m WHERE c.citycode =
m.citycode )
```

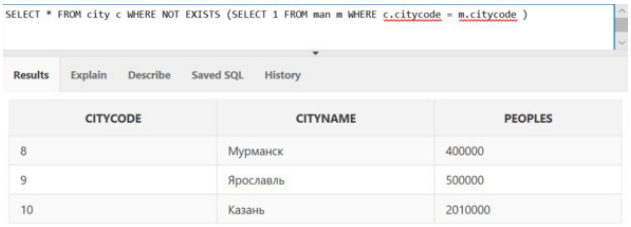


Рисунок 114. Запрос с EXISTS к таблице CITY

Вопросы учеников

В прошлом шаге вы говорили, что можно с помощью EXISTS заменить конструкцию IN. Приведите пример.

Например, у нас есть таблицы TABLE1, TABLE 2, поля связи KEY1, KEY2. Необходимо вывести на экран те данные из таблицы TABLE1, для которых присутствуют соответствующие строки в TABLE2 по KEY1, KEY2.

```
SELECT * FROM table1 WHERE EXISTS (SELECT 1 FROM TABLE2 WHERE table1.key1 =
table2.key1 AND table1.key2 = table2.key2)
```

У вас в примерах есть синтаксис вида SELECT 1 FROM MAN m WHERE c.CITYCODE = m.CITYCODE. Почему именно единица? Поясните, как это работает.

Если в подзапросе есть хоть одна строка, то будет выведена единица, если нет, то подзапрос ничего не вернет, сама цифра 1 здесь не принципиальна.

Контрольные вопросы и задания для самостоятельного выполнения

1. Вывести все автомобили из таблицы AUTO, для которых нет соответствующих записей в таблице MAN. Использовать EXISTS (связь по PHONENUM).

2. Вывести все автомобили из таблицы AUTO, где есть записи в таблице MAN, где возраст людей больше 35 лет, используя EXISTS (связь по PHONENUM).

Шаг 37. Подзапрос как новая колонка запроса

Введение

Помните перечисление колонок таблицы после команды SELECT? В языке SQL есть возможность задать новую колонку как результат выполнения подзапроса. Разберемся, как это сделать.

Теория и практика

Возможности использования подзапросов.

Язык SQL позволяет использовать подзапрос для вывода значений в качестве дополнительных колонок запроса.

Синтаксис

SELECT колонка 1, колонка 2, колонка 3, (SELECT колонка FROM другая таблица WHERE таблица1.колонка связи = другая таблица. колонка связи) псевдоним колонки FROM

Помните, по каким колонкам объединяются таблицы в нашей схеме?
AUTO – > MAN = PHONENUM
CITY – > MAN = CITYCODE

Примеры

Выбрать из CITY наименование города (CITYNAME) и наименование города (CITYNAME), где код города больше кода данного города на 1.

```
SELECT cityname, (select cityname FROM city c1 WHERE c1.citycode = c.citycode+1) ctn1
FROM city c
```

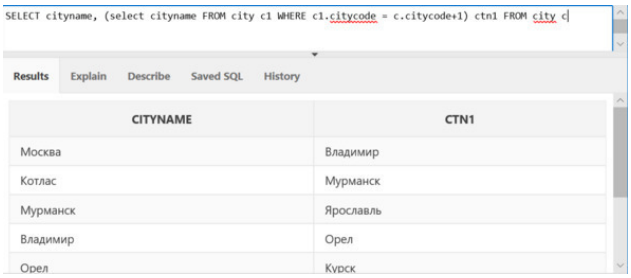


Рисунок 115. Запрос с подзапросом в колонке запроса

Выбрать имя, фамилию и город проживания (FIRSTNAME, LASTNAME, CITYNAME) человека.

```
SELECT  firstname,lastname, (SELECT cityname FROM city WHERE city.citycode =
man.citycode) ctname FROM man
```

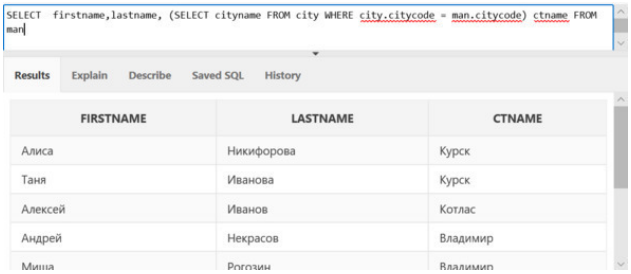


Рисунок 116. Запрос с подзапросом в колонке запроса к таблице MAN

Выбрать марки AUTO (MARK), а также сколько человек обладает такой машиной (MARK) из таблицы MAN.

```
SELECT DISTINCT mark, (SELECT count(1) FROM man WHERE man.phonenum in (SELECT
phonenum FROM auto a1 WHERE a1.mark=a.mark)) as cnt FROM auto a
```

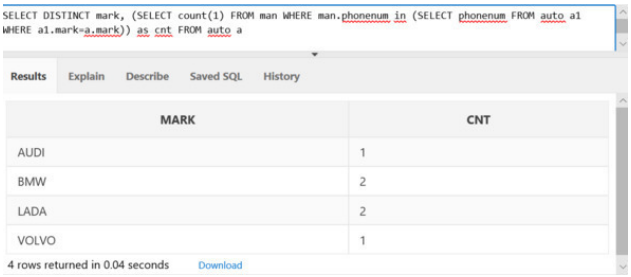


Рисунок 117. Запрос с подзапросом в колонке запроса к таблице MAN, AUTO

Важные замечания

Такая конструкция порой бывает незаменима при практическом применении, однако у применения этого подзапроса также есть ряд ограничений:

• Подзапрос в качестве колонки запроса не должен возвращать более одной строки, в противном случае возникнет ошибка.

• Такой подзапрос может вернуть только значение одной колонки, синтаксис допускает только такой вариант использования; впрочем, всегда можно написать несколько похожих подзапросов.

Вопросы учеников

Есть ли какие-либо ограничения по типам данных при создании подзапроса в качестве колонки запроса?

Скорее всего, нет. Я считаю, что этот синтаксис будет работать даже в случае запросов, которые возвращают CLOB- или BLOB-значения.

Контрольные вопросы и задания для самостоятельного выполнения

1. Выбрать марку авто, цвет, имя человека, который владеет машиной, использовать подзапрос для новой колонки таблицы.

2. Выбрать марку авто, цвет, имя человека, который владеет машиной, и в каком городе живет этот человек, использовать подзапрос для новой колонки таблицы.

3. Выбрать из MAN имя, фамилию и город проживания человека, сколько человек проживает в данном городе. Использовать подзапрос для новой колонки таблицы.

Шаг 38. Подзапрос как источник данных после FROM

Введение

Давайте познакомимся с еще одним замечательным способом использования подзапросов – использованием подзапроса как источника данных после ключевого слова FROM. Данный способ очень часто применяется на практике.

Теория и практика

Язык SQL предоставляет возможность использовать подзапрос вместо таблицы после FROM.

Синтаксис этого магического способа

```
SELECT перечень полей или * FROM (SELECT перечень полей или * FROM таблица2
) псевдоним для внешнего подзапроса WHERE ограничения на псевдоним GROUP BY
ORDER BY
```

Примеры

Выбрать все красные (COLOR) автомобили * AUTO, используя подзапрос как таблицу.

```
SELECT * FROM ( SELECT * FROM auto WHERE color = 'КРАСНЫЙ' ) t1
```

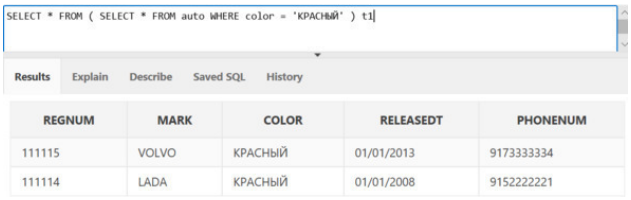


Рисунок 118. Запрос с подзапросом как источник данных: таблица AUTO

Выбрать все автомобили * и владельцев *, используя подзапрос как таблицу.

```
SELECT * FROM     (SELECT * FROM auto  INNER JOIN man ON auto.phonenum =
man.phonenum ) t1
```

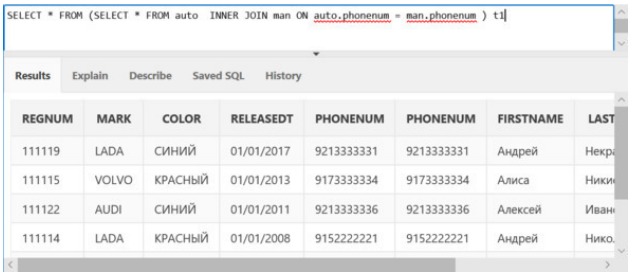


Рисунок 119. Запрос с подзапросом как источник данных: таблица MAN

Выбрать все города (CITYNAME) и население городов (PEOPLES), используя подзапрос как таблицу.

```
SELECT * FROM (SELECT cityname, peoples FROM city) t1
```

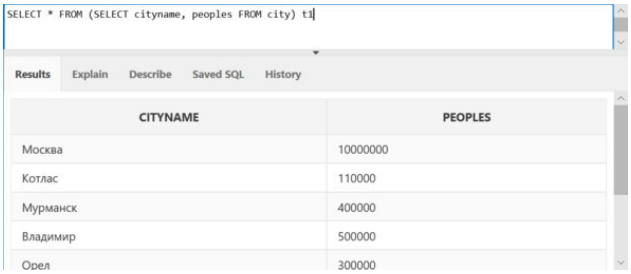


Рисунок 120. Запрос с подзапросом как источник данных: таблица CITY

Важные замечания

Также с помощью такого способа связать можно два подзапроса конструкцией JOIN.

Связать два подзапроса: один – все люди (MAN), которым больше 35 лет (YEAROLD), второй – все автомобили (AUTO), произведенные после 2003 года (RELEASEDT).

```
SELECT * FROM  (SELECT * FROM man WHERE yearold > 35 ) m INNER JOIN (SELECT * FROM
auto WHERE releasedt >= date'2004-01-01') n ON n.phonenum = m.phonenum
```

Обратите внимание, что при использовании подзапроса как источника данных необходимо для каждого такого подзапроса применять уникальные псевдонимы, такие как псевдоним n в нашем примере.

Вопросы учеников

Можно ли в таком виде запросов использовать запросы с таблицей DUAL?

Да, вот пример:

Можно ли в таком виде запросов использовать разные типы объединений?

Да, можно использовать любые типы объединений JOIN.

Контрольные вопросы и задания для самостоятельного выполнения

1. Выбрать все автомобили BMW из AUTO, используя подзапрос как таблицу после FROM.

2. Выбрать все автомобили из AUTO и их владельцев, которым больше 25 лет, используя подзапрос как таблицу.

3. Выбрать все города, в которых больше миллиона жителей, используя подзапрос после FROM.

Шаг 39. Повторение темы подзапросов. Подзапросы в запросах с группировкой данных

Введение

Подзапросы можно сочетать с запросами с группировкой данных. Это интересная возможность, которая позволяет разрешить некоторые проблемы.

Теория и практика

Продемонстрируем, как работают подзапросы в запросах с группировками данных.

Пример 1. Выбрать из таблицы MAN имя и возраст, выбрать из таблицы CITY наименование города подзапросом, сгруппировать данные по названию города, найти сумму возрастов всех жителей каждого города с заданным именем, использовать тип подзапроса «колонка запроса».

```
SELECT firstname, SUM(yearold), cityname FROM (
SELECT firstname as firstname, yearold as yearold, (select cityname from city c where
m.citycode = c.citycode) as cityname  FROM man m ) mv
GROUP BY firstname,  cityname
```

Обратите внимание: в данном примере мы перед тем как сгруппировать данные, используем выборку с подзапросом «колонка таблицы» в качестве внутреннего подзапроса.

FIRSTNAME	SUM(YEAROLD)	CITYNAME
Таня	38	Владимир
915915%	81	Москва
Donald	70	Калемоз
Таня	31	Курск
Таня	74	Орел
CHARLI	25	-

Пример 2. Выбрать только те названия городов из таблицы CITY, где в таблице MAN более одного человека, использовать подзапрос.

```
select * from CITY where (SELECT count(citycode) FROM man WHERE man.citycode =
city.citycode )>1
```

В этом запросе нет группировки, но используется агрегатная функция COUNT.

CITYCODE	CITYNAME	PEOPLES
1	Москва	10000040
2	Владимир	500010
4	Курск	200010
13	Калемоз	113010

Пример 3. Найти количество автомобилей каждой марки, которые купили люди, проживающие в городах с населением более миллиона человек. Используйте подзапрос с IN.

```
SELECT mark, count(regnum) FROM auto
WHERE phonenum in (SELECT phonenum FROM man WHERE citycode in (select citycode
FROM city WHERE peoples > 100000))
GROUP by mark
```

В этом запросе используется два вложенных подзапроса IN, записи по которым фильтруются, после чего выполняется группировка по отфильтрованным значениям.

MARK	COUNT(REGNUM)
BMW	1
LADA	2
BMWE60	1
VOLVO	1

Пример 4. Найти количество автомобилей каждой марки, которые купили люди не старше 35 лет. Используйте подзапрос с EXISTS.

```
SELECT mark, count(regnum) FROM auto WHERE EXISTS (SELECT phonenum FROM man
WHERE man.phonenum = auto.phonenum AND NOT yearold>35) GROUP by mark
```

В данном запросе используется логический оператор NOT, чтобы правильно перевести в SQL условие – **не старше**.

MARK	COUNT(REGNUM)
LADA	1
BMW	1
BMWE60	1
VOLVO	1

Важные замечания

Подзапросы можно сочетать с группировками, но важно понимать, что подзапрос как колонка запроса не может быть частью групповой функции, то есть следующий запрос будет неверен.

Примеры неправильных запросов:

```
SELECT firstname, SUM(yearold), (select cityname from city c where m.citycode = c.citycode)
as cityname FROM man GROUP BY firstname,  cityname
```

Неправильный запрос – другой синтаксис:

```
SELECT firstname, SUM(yearold), (select cityname from city c where m.citycode = c.citycode)
as cityname FROM man GROUP BY firstname,  (select cityname from city c where m.citycode =
c.citycode)
```

Подзапрос как колонка запроса не может вернуть более одной строки.

Если мы используем подзапрос как в примере 3, то такой подзапрос также не может вернуть более одной строки.

Вопросы учеников

Можно переписать первый пример другим способом, без подзапросов?

Да, эта задача имеет несколько решений, предлагаю вам их найти самостоятельно.

В примере 3 мы использовали несколько вложенных подзапросов. Как еще можно решить эту задачу, чтобы не использовать вложенные подзапросы?

Задача решается различными способами, например: запрос как источник данных, соединение таблиц.

Почему ни в одном из практических примеров не было подзапроса как источника данных для запроса?

Подобный пример мы разберем в следующем шаге.

Контрольные вопросы и задания для самостоятельного выполнения

1. Повторите материалы данного шага.
2. Перепишите запрос из примера 3 с использованием подзапроса EXISTS.
3. Выберите только те названия городов из таблицы CITY, где в таблице MAN не более одного человека, используйте EXISTS.
4. Найдите количество автомобилей каждой марки, которые купили люди старше 29 лет. Используйте подзапрос с IN.

Шаг 40. Сочетание разных типов подзапросов

Введение

В прошлых шагах мы с вами рассмотрели возможности работы с подзапросами, а также различные типы подзапросов и их использование в языке SQL.

Перечислим типы подзапросов, с которыми мы познакомились:

– подзапросы как дополнительная колонка в основном запросе;

– подзапросы с оператором IN;

– подзапросы, которые идут после FROM в качестве источника данных, то есть как бы вместо таблицы.

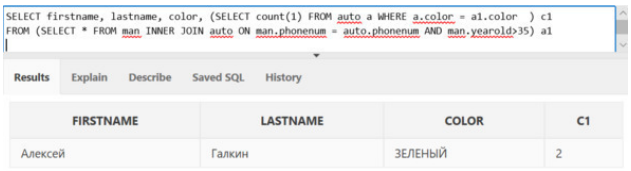
Теория и практика

Попробуем сейчас сочетать эти несколько типов подзапросов в одном главном большом запросе, в одной большой команде SELECT.

Сразу перейдем к практике.

Вывести на экран данные из таблицы MAN и таблицы AUTO: имя, фамилию (FIRSTNAME, LASTNAME) человека, цвет машины (COLOR), марку машины (MARK) для людей, чей возраст (YEAROLD) больше 35 лет, и как дополнительную колонку вывести с помощью подзапроса информацию о том, сколько машин этого же цвета (COLOR) в таблице AUTO.

```
SELECT firstname, lastname, color, (SELECT count(1) FROM auto a WHERE a.color = a1.color  )
c1
        FROM (SELECT * FROM man INNER JOIN auto ON man.phonenum = auto.phonenum
        AND man.yearold>35) a1
```

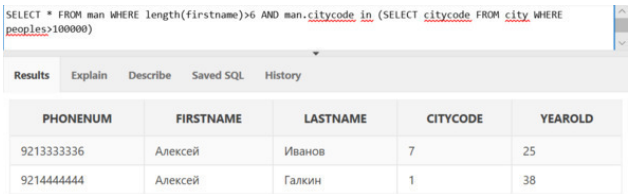


FIRSTNAME	LASTNAME	COLOR	C1
Алексей	Галкин	ЗЕЛЕНЫЙ	2

Рисунок 121. Запрос MAN, AUTO сложный

Вывести на экран данные из таблицы MAN *, где имя человека (FIRSTNAME) больше шести букв и эти люди живут в городе с населением (PEOPLES) больше 1 миллиона человек.

```
SELECT * FROM man WHERE length(firstname)>6 AND man.citycode in (SELECT citycode
FROM city WHERE peoples>1000000)
```

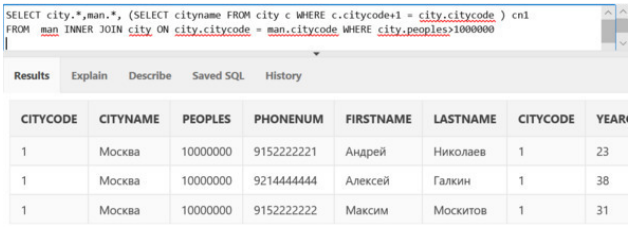


PHONENUM	FIRSTNAME	LASTNAME	CITYCODE	YEAROLD
9213333336	Алексей	Иванов	7	25
9214444444	Алексей	Галкин	1	38

Рисунок 122. Запрос MAN, CITY

Вывести на экран данные из таблицы CITY и таблицы MAN, где население (PEOPLES) больше 1 миллиона человек, вывести как дополнительный столбец город (CITYNAME), у которого код города больше на 1.

```
SELECT city.*,man.*, (SELECT cityname FROM city c WHERE c.citycode+1 = city.citycode ) cn1
FROM  man INNER JOIN city ON city.citycode = man.citycode WHERE city.peoples>1000000
```



```
SELECT city.*,man.*, (SELECT cityname FROM city c WHERE c.citycode+1 = city.citycode ) cn1
FROM  man INNER JOIN city ON city.citycode = man.citycode WHERE city.peoples>1000000
```

Results | Explain | Describe | Saved SQL | History

CITYCODE	CITYNAME	PEOPLES	PHONENUM	FIRSTNAME	LASTNAME	CITYCODE	YEAR
1	Москва	10000000	9152222221	Андрей	Николаев	1	23
1	Москва	10000000	9214444444	Алексей	Галкин	1	38
1	Москва	10000000	9152222222	Максим	Москитов	1	31

Рисунок 123. Запрос MAN, AUTO: сложный запрос

Важные замечания

Подзапрос, который возвращает единственную запись, можно также использовать в конструкции WHERE при проверке на равенство значений.

Приведу пример:

```
SELECT * FROM man WHERE man.citycode = (SELECT citycode FROM city WHERE cityname =
'Москва')
```

Естественно, такой запрос должен возвращать одну-единственную строку, а также выводить в качестве результата только одну колонку.

При таком подходе считается стандартной практикой использование псевдонимов для подзапросов.

Вопросы учеников

Если в одном из подзапросов будут сгруппированные данные с помощью GROUP BY, такой запрос будет работать?

Разумеется, будет, и вот пример такого запроса. Подобные запросы мы разбирали в предыдущем шаге.

```
SELECT * FROM (select color , count(1) as countauto from auto GROUP BY COLOR) cl
```

Контрольные вопросы и задания для самостоятельного выполнения

1. Вывести все автомобили AUTO *, которые приобрели люди, проживающие в городе (CITY) с населением (POPULATION) больше 1 миллиона человек. Вывести как колонку, как дополнительное поле подзапрос «название города» (CITYNAME) рядом с маркой автомобиля.

2. Вывести все автомобили AUTO *, которые приобрели люди, проживающие в городе (CITY) с населением (PEOPLES) больше 1 миллиона человек. Вывести как колонку подзапрос «название города» (CITYNAME) рядом с маркой автомобиля.

День девятый

Шаг 41. Предикаты ANY, SOME и ALL

Введение

Рассмотрим на этом шаге незаслуженно забытые, нечасто используемые на практике конструкции языка SQL – ANY, ALL. Тем не менее данные конструкции обладают исключительными возможностями.

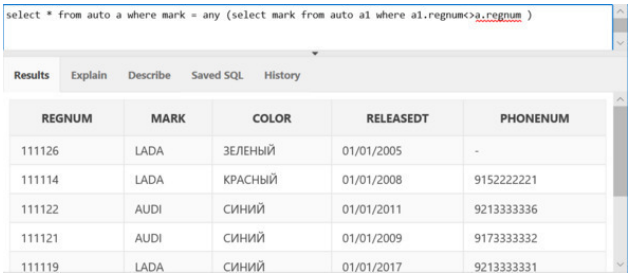
Теория и практика

Синтаксическая конструкция ANY – это предикат, который является верным, если каждое из всех значений, выведенных подзапросом, удовлетворяет условию для текущей строки внешнего запроса.

Давайте рассмотрим на примерах:

Вывести из таблицы AUTO только те машины, для которых в таблице AUTO есть машины такой же марки.

```
select * from auto a where mark = any (select mark from auto a1 where
a1.regnum<>a.regnum )
```



```
select * from auto a where mark = any (select mark from auto a1 where a1.regnum<>a.regnum )
```

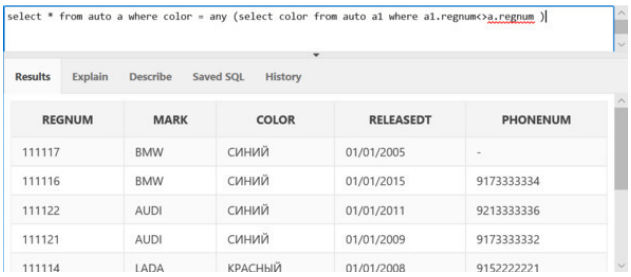
Results | Explain | Describe | Saved SQL | History

REGNUM	MARK	COLOR	RELEASEDT	PHONENUM
111126	LADA	ЗЕЛЕНЫЙ	01/01/2005	-
111114	LADA	КРАСНЫЙ	01/01/2008	9152222221
111122	AUDI	СИНИЙ	01/01/2011	9213333336
111121	AUDI	СИНИЙ	01/01/2009	9173333332
111119	LADA	СИНИЙ	01/01/2017	9213333331

Рисунок 124. Работа предиката ANY

Вывести из таблицы AUTO те машины, для которых в таблице AUTO есть машины такого же цвета.

```
select * from auto a where color = any (select color from auto a1 where
a1.regnum<>a.regnum )
```



```
select * from auto a where color = any (select color from auto a1 where a1.regnum<>a.regnum )
```

Results | Explain | Describe | Saved SQL | History

REGNUM	MARK	COLOR	RELEASEDT	PHONENUM
111117	BMW	СИНИЙ	01/01/2005	-
111116	BMW	СИНИЙ	01/01/2015	9173333334
111122	AUDI	СИНИЙ	01/01/2011	9213333336
111121	AUDI	СИНИЙ	01/01/2009	9173333332
111114	LADA	КРАСНЫЙ	01/01/2008	9152222221

Рисунок 125. Работа предиката ANY: таблица AUTO

Предикат ALL является верным, если каждое значение, выбранное подзапросом, удовлетворяет условию в предикате внешнего запроса.

Ключевое отличие от предиката и ANY в том, что мы можем использовать операции больше и меньше (> или <).

Следующие запросы являются примером данного предиката.

Вывести на экран с помощью SELECT всех людей из таблицы MAN, чей возраст больше возраста каждого человека, который проживает в городе Владимир.

```
SELECT * FROM man WHERE yearold  > ALL(select yearold from man where citycode = (select
citycode from city where cityname = 'Владимир'))
```

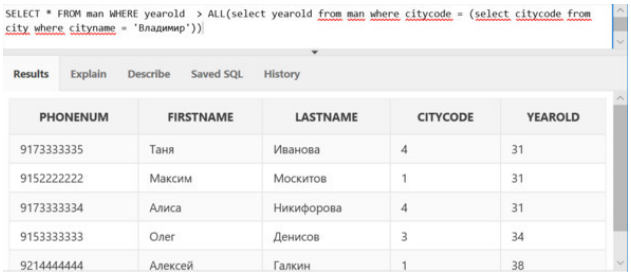


Рисунок 126. Работа предиката ALL: таблицы CITY, MAN

Вывести на экран с помощью SELECT всех людей из таблицы MAN, чей возраст меньше каждого человека, у которого автомобиль BMW.

```
SELECT * FROM man WHERE yearold  < ALL(select yearold from man where phonenum in
(select phonenum from auto where mark = 'BMW'))
```

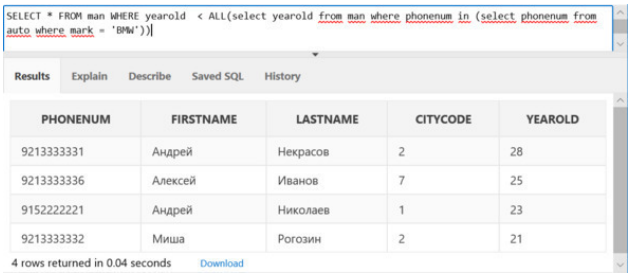


Рисунок 127. Работа предиката ALL: таблицы AUTO, MAN

Важные замечания

Обратите внимание, что конструкции ALL и ANY используются в WHERE как результат сравнения значения ячейки в заданной колонке со списком значений. То есть сначала название поля, а затем уже подзапрос.

Вопросы учеников

Можно ли использовать ALL и ANY со знаком неравенства?

Да, можно, и вот примеры таких запросов.

Вывести на экран с помощью SELECT всех людей из таблицы MAN, чей возраст не равен возрасту каждого человека, у которого автомобиль BMW.

```
SELECT * FROM man WHERE yearold  != ALL(select yearold from man where phonenum in
(select phonenum from auto where mark = 'BMW'))
```

Вывести из таблицы AUTO машины, если в таблице AUTO нет машин такого же цвета.

```
select * from auto a where color != any (select color from auto a1 where
a1.regnum<>a.regnum )
```

Подобным образом можно оформить такой же запрос с конструкцией NOT.

```
SELECT * FROM man WHERE NOT yearold  = ALL(select yearold from man where phonenum in
(select phonenum from auto where mark = 'BMW'))
```

Контрольные вопросы и задания для самостоятельного выполнения

1. Повторить запросы из этого шага.

2. Вывести все города, где количество проживающих людей больше, чем количество людей в городах с CITYCODE 5, 6, 7.

3. Вывести на экран с помощью SELECT всех людей из таблицы MAN, чей возраст меньше каждого человека, у которого автомобиль BMW.

Шаг 42. Преобразование типов данных

Введение

Очень часто требуется при решении различных задач преобразовать данные от одного к другому типу, например преобразовать число к строке.

Теория и практика

В SQL ORACLE диалекта существует несколько функций для преобразования типов данных.

Есть универсальная функция для преобразования различных типов данных к текстовому формату.

Универсальная функция **TO_CHAR**.

Рассмотрим несколько примеров работы функции **TO_CHAR**.

Синтаксис

```
SELECT перечень полей, to_char(поле, 'формат') FROM  имя таблицы WHERE условия отбора строк
```

Преобразование даты в текстовый формат

YYYY – полный формат года;
YY – краткий формат года;
MM – месяц;
DY – день недели;
hh24 – часы, также возможны варианты с AM и PM, подробнее в документации;
mi – минуты;
ss – секунды.

Примеры работы функции TO_CHAR для преобразования дат в строку

Показать текущий день недели.

```
select to_char(sysdate, 'dy') as dn from dual
```

Показать текущую дату как в примере 22-10-2018 23:11:11, преобразовав к строке.

```
select to_char(sysdate, 'dd-mm-yyyy hh24:mi:ss') as dt from dual
```

Показать текущую дату как в примере 22.10.18 23:11:11, преобразовав к строке.

```
select to_char(sysdate, 'dd.mm.yy hh24:mi:ss') as dt from dual
```

Показать текущую дату как в примере 2018.10.22 23:11:11, преобразовав к строке.

```
select to_char(sysdate, 'dd.mm.yy hh24:mi:ss') as dt from dual
```

Преобразование числовых значений в текстовый формат

0 – число с лидирующим 0;
99 – число.

Примеры
Преобразовать в строку 12.033.

```
select to_char(12.033,'00.999') from dual
```

Преобразовать в строку 0.033 как.033.

```
select to_char(0.033,'9.999') from dual
```

Преобразовать в строку 123.0334.

```
select to_char(123.0334,'000.999') from dual
```

Конструкция CAST
Весьма полезной может быть функция преобразования типов – CAST.
Она имеет следующий синтаксис:

```
select cast(поле,тип к которому пребраз) from табл where усл
```

Примеры
Привести число 1000 к типу VARCHAR2 (10).

```
select cast(1000 as varchar2(10)) as cs from dual
```

Привести строку «1000.01» к типу NUMBER.

```
select cast('1000' as number) as cs from dual
```

Важные замечания

Следует понимать, что описанные функции преобразования данных весьма специальные и могут применяться лишь в ORACLE SQL диалекте, а для других систем, таких как MYSQL, MS SQL, PostgreSQL, следует ознакомиться с соответствующим разделом документации по функциям преобразования типов.

При использовании CAST необходимо, чтобы типы данных совпадали – то есть если мы выполняем преобразование к типу NUMBER, то значение должно переводиться в номер, в значении не должно быть лишних символов, не являющихся числами.

Вопросы учеников

Можно ли, наоборот, преобразовать строку в DATE?

Да, и для этого есть функция TO_DATE (строка, формат).
Пример использования TO_DATE:

```
select to_date('13-10-2018 16:59:49', 'dd-mm-yyyy hh24:mi:ss') as dn from dual
```

Можно ли при преобразовании больших чисел в строку показать разделители разрядов числа?

Такая возможность есть, вот пример:

```
select to_char(1234325234234.55,'999G999G999G999G999D00' ) nn from dual
```

– 1 234 325 234 234,55.
Или:

```
select to_char(1234325234234.55,'999G999G999G999G999D00' ) nn from dual
```

– 1 234 325 234 234,55.
Здесь D – разделитель дробного числа, а G – разделитель разрядов.

Контрольные вопросы и задания для самостоятельного выполнения

1. Объяснить отличия функции TO_CHAR от функции CAST.
2. Показать текущий день недели.
3. Показать вчерашнюю дату как в примере 22:10:2018 23-11-11.
4. Показать вчерашнюю дату как в примере 22-10-18 23-11-11.
5. Показать вчерашнюю дату как в примере 2018-10-22 23:11:11.
6. Привести строку «20.01» к типу NUMBER.

Шаг 43. Объединение таблицы с самой же собой

Введение

Пожалуй, самый парадоксальный и необычный способ объединения таблиц JOIN заключается в том, что язык SQL допускает объединение таблицы с самой же собой, в этом случае для каждой таблицы создается свой псевдоним и обращение происходит по уникальному псевдониму.

Теория и практика

Синтаксис

```
SELECT поля или * FROM таблица1 t11 inner left right join таблица1 t12 ON условие соединения таблиц WHERE доп условия
```

Альтернативный синтаксис

```
SELECT поля или * FROM таблица1 t11 , таблица1 t12 WHERE t11 код = t12 код ( условие соединения  таблиц )
```

Здесь таблица t11—t12 – это псевдонимы для одной и той же таблицы – **таблица1**.

Примеры

Выбрать из таблицы CITY названия городов и популяцию, а также названия городов и популяцию, где код города = код текущего города +1.

```
select c1.citycode, c2.citycode, c2.cityname, c2.peoples from city c1 left join city c2 on c1.citycode +1= c2.citycode
```

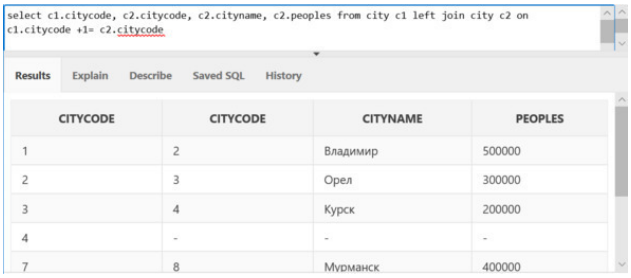
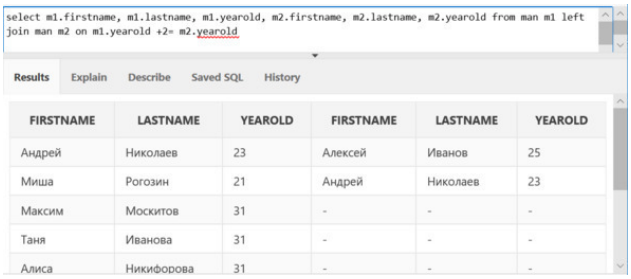


Рисунок 128. Запрос к CITY: таблица с самой же собой

Выбрать из таблицы MAN имя, фамилию человека, возраст, а также имя, возраст и фамилию человека, который старше данного человека на два года.

```
select m1.firstname, m1.lastname, m1.yearold, m2.firstname, m2.lastname, m2.yearold from man m1 left join man m2 on m1.yearold +2= m2.yearold
```



```
select m1.firstname, m1.lastname, m1.yearold, m2.firstname, m2.lastname, m2.yearold from man m1 left
join man m2 on m1.yearold +2= m2.yearold
```

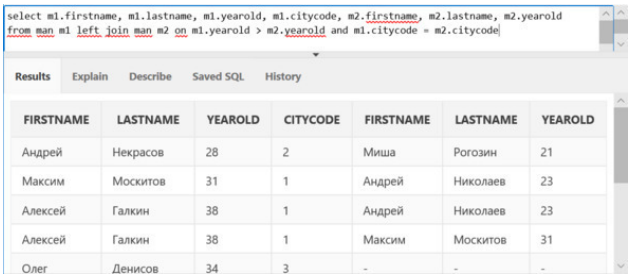
Results | Explain | Describe | Saved SQL | History

FIRSTNAME	LASTNAME	YEAROLD	FIRSTNAME	LASTNAME	YEAROLD
Андрей	Николаев	23	Алексей	Иванов	25
Миша	Рогозин	21	Андрей	Николаев	23
Максим	Москитов	31	-	-	-
Таня	Иванова	31	-	-	-
Алиса	Никифорова	31	-	-	-

Рисунок 129. Запрос к MAN: таблица с самой же собой

Выбрать из таблицы MAN имя, фамилию человека, возраст, а также имя и фамилию, возраст человека, который старше и живет в том же городе.

```
select m1.firstname, m1.lastname, m1.yearold, m1.citycode, m2.firstname, m2.lastname,
m2.yearold from man m1 left join man m2 on m1.yearold > m2.yearold and m1.citycode =
m2.citycode
```



```
select m1.firstname, m1.lastname, m1.yearold, m1.citycode, m2.firstname, m2.lastname, m2.yearold
from man m1 left join man m2 on m1.yearold > m2.yearold and m1.citycode = m2.citycode
```

Results | Explain | Describe | Saved SQL | History

FIRSTNAME	LASTNAME	YEAROLD	CITYCODE	FIRSTNAME	LASTNAME	YEAROLD
Андрей	Некрасов	28	2	Миша	Рогозин	21
Максим	Москитов	31	1	Андрей	Николаев	23
Алексей	Галкин	38	1	Андрей	Николаев	23
Алексей	Галкин	38	1	Максим	Москитов	31
Олег	Денисов	34	3	-	-	-

Рисунок 130. Запрос к MAN: таблица с самой же собой

Важные замечания

Разумеется, при таком виде объединения просто необходимо использование разных псевдонимов для каждого дубликата таблицы, иначе СУБД не сможет корректно выполнить такой запрос.

Вопросы учеников

В каком случае такой вид объединения работать не будет?

Затруднительно представить объединение таблицы с самой собой в конструкциях CROSS JOIN и FULL JOIN.

Контрольные вопросы и задания для самостоятельного выполнения

1. Выбрать из таблицы CITY названия города, а также названия города, где код города = код текущего города +2.

2. Выбрать из таблицы MAN имя, фамилию человека, а также имя и фамилию человека, который старше данного человека на три года.

3. Выбрать из таблицы MAN имя, фамилию человека, а также имя и фамилию человека, который живет в том же городе.

Шаг 44. Операторы для работы с множествами – UNION, UNION ALL

Введение

Операторы языка SQL **UNION** и **UNION ALL** позволяют сделать объединение из нескольких запросов SQL специальным образом. Если при объединении **JOIN** колонки из разных таблиц располагаются горизонтально друг за другом, то данные при объединении **UNION** выводятся последовательно, как один набор данных под другим.

Теория и практика

Итак, **UNION** – специальный оператор языка SQL для работы с множествами.

UNION объединяет наборы данных – строчки из наборов данных, непосредственно одна за другой.

Также часто применяется оператор UNION ALL, который работает так же, как и UNION, но в отличие UNION не выводит записи, если они дублируются, то есть в выводе не будет дублей строк. UNION ALL, напротив, дубли не убирает и выводит дублирующиеся строки на экран.

Примеры

Объединим номера телефонов из таблицы AUTO и таблицы MAN с помощью UNION.

```
select  phonenum from auto union select  phonenum from man
```

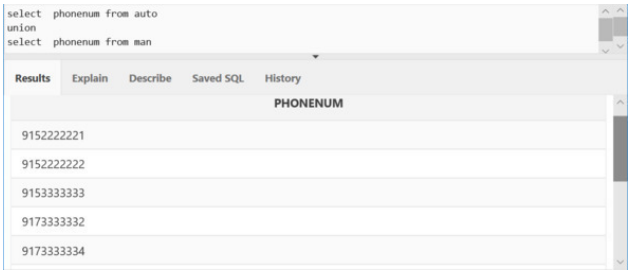


Рисунок 131. Запрос с использованием UNION: таблица MAN

Объединим номера телефонов из таблицы AUTO и таблицы MAN с помощью UNION ALL.

```
select  phonenum from auto union all select  phonenum from man
```

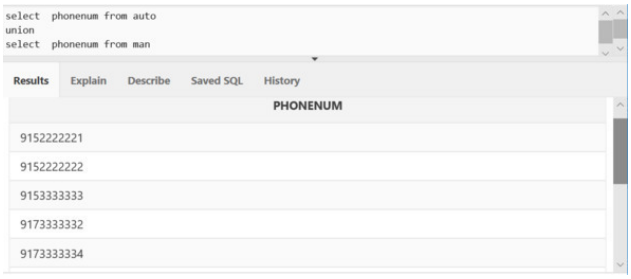


Рисунок 132. Запрос с использованием UNION: таблицы AUTO, MAN

Объединим данные из таблицы AUTO и таблицы AUTO1.

```
select * from auto union all select * from auto1
```

Важные замечания

При использовании операторов UNION есть несколько важных ограничений, а именно: данные в наборах должны быть однотипны, то есть последовательность полей и типы данных в обоих наборах должны быть одинаковы. А также, естественно, количество выводимых колонок должно совпадать в каждом наборе.

Вопросы учеников

Какой оператор отрабатывает быстрее – UNION или UNION ALL?

Несомненно, в больших наборах данных быстрее будет работать UNION ALL, так как UNION убирает дубли и затрачивает на это дополнительные ресурсы системы.

Как сделать сортировку данных в запросе с UNION?

Вот пример, как бы сделал я.
Выбрать строчки из AUTO и AUTO1, отсортировать данные по COLOR.

```
select * from ( select  * from auto union all select  * from auto1) u order by color
```

Контрольные вопросы и задания для самостоятельного выполнения

1. Используя UNION выведите из таблицы AUTO синие автомобили и зеленые автомобили из AUTO1.

2. Используя UNION ALL, выведите из таблицы AUTO все автомобили и зеленые автомобили AUTO.

Шаг 45. Операторы MINUS, INTERSECT

Введение

Если мы знакомимся с операторами для работы с множествами, тогда нельзя не упомянуть об операторах языка SQL – MINUS и INTERSECT.

Теория и практика

MINUS вычитает из первого набора данных второй набор данных, то есть в результате выполнения SQL-запроса с оператором MINUS на экран будут выведены лишь те строки из первого набора, которых нет во втором наборе – в подзапросе, следующем непосредственно после оператора MINUS.

Примеры

Вывести те номера телефонов, которые есть в MAN, но которых нет в таблице AUTO.

```
select  phonenum from man MINUS select  phonenum from auto
```

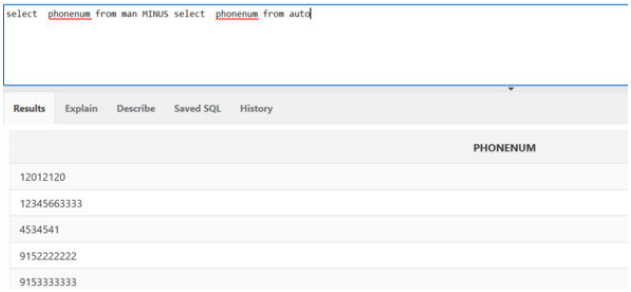


Рисунок 133. Демонстрация работы оператора MINUS

Вывести строки из таблицы AUTO, которых нет в таблице AUTO1.

```
select * from auto MiNUS select * from auto1
```

Оператор INTERSECT выведет только те строки, которые есть и в первом, и во втором наборе данных, то есть пересечение множеств.

Вывести те телефоны, которые есть в таблице MAN и таблице AUTO.

```
select  phonenum from man INTERSECT select  phonenum from auto;
```

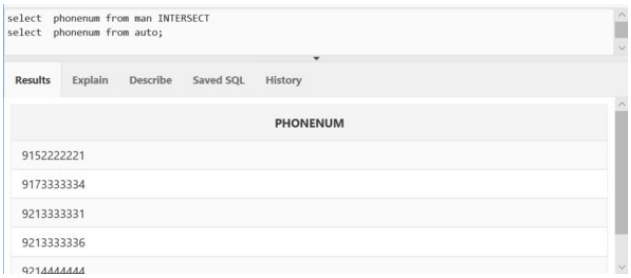


Рисунок 134. Демонстрация работы INTERSECT

Вывести CITYCODE, которые есть в таблицах MAN и CITY.

```
select  citycode from man INTERSECT select  citycode from city
```

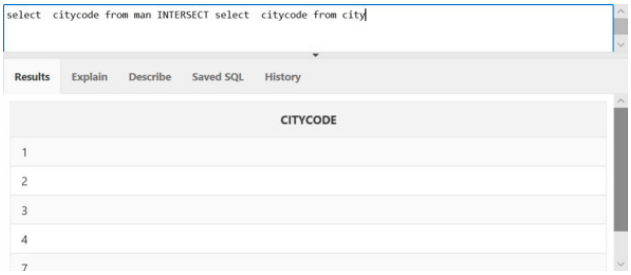


Рисунок 135. Демонстрация работы INTERSECT

Важные замечания

Так же, как и в предыдущем шаге, данные в наборах должны быть однотипны, то есть последовательность полей и типы данных в обоих наборах должны быть одинаковы. А также, естественно, количество выводимых колонок должно совпадать в каждом наборе.

Вопросы учеников

Можно ли заменить оператор MINUS другими операторами языка SQL?

Да, например оператором IN или NOT IN, вот пример.
Вывести те номера телефонов, которые есть в MAN, но которых нет в таблице AUTO.

```
select phonenum from man where man.phonenum not in (select phonenum from auto)
```

Можно ли использовать NOT с операторами MINUS и INTERSECT?

Нет, синтаксис SQL не позволяет этого сделать.

Контрольные вопросы и задания для самостоятельного выполнения

1. Найдите, какие строки есть в AUTO, но нет в AUTO1. Используйте MINUS.
2. Найдите совпадение из AUTO и AUTO1. Используйте INTERSECT.

День десятый

Шаг 46. Повторение материала. Сочетание операторов множеств и предикатов

Введение

Повторим материалы предыдущего занятия. Продемонстрируем, как можно сочетать предикаты ANY, SOME, ALL с подзапросами и операторами MINUS, INTERSECT, UNION.

Теория и практика

Логические операторы можно эффективно сочетать с различными типами подзапросов, а также с MINUS, INTERSECT, UNION. Приведем несколько примеров таких эффективных сочетаний.

Пример 1. Выберите из таблицы AUTO автомобили, которые принадлежат людям, для которых нет соответствия в таблице CITY, используйте MINUS.

```
SELECT * FROM auto WHERE auto.phonenum IN (SELECT phonenum FROM man WHERE
citycode in (SELECT citycode FROM AUTO MINUS SELECT citycode FROM city))
```

Данный запрос не вернул ни одной записи.

Пример 2. Немного переделаем задание.

Выберите из таблицы AUTO автомобили, которые принадлежат людям, для которых есть записи в таблице CITY, где PEOPLES <100 000, используйте MINUS.

```
SELECT * FROM auto WHERE auto.phonenum IN (SELECT phonenum FROM man WHERE
citycode in (SELECT citycode FROM AUTO MINUS SELECT citycode FROM city WHERE peoples
< 1000000 ))
```

Запрос демонстрирует использование MINUS с подзапросом.

REGNUM	MARK	COLOR	RELEASEDT	PHONENUM
111114	LADA	КРАСНЫЙ	01/01/2008	9152222221
61656842	BMWE60	black	01/01/2010	777888

Рисунок 136. Демонстрация работы MINUS с подзапросом

Пример 3. Выберите из таблицы AUTO записи, для которых есть соответствие в таблицах MAN и MAN1.

```
SELECT * FROM auto WHERE auto.phonenum = ANY(SELECT phonenum FROM man UNION
SELECT phonenum FROM man1 )
```

REGNUM	MARK	COLOR	RELEASEDT	PHONENUM
61656842	BMWE60	black	01/01/2010	777888
111114	LADA	КРАСНЫЙ	01/01/2008	9152222221
111116	BMW	СИНИЙ	01/01/2015	9173333334
111115	VOLVO	КРАСНЫЙ	01/01/2013	9173333334
111119	LADA	СИНИЙ	01/01/2017	9213333331

Рисунок 137. Демонстрация работы ANY

Важные замечания

При использовании операторов MINUS, UNION, INTERSECT необходимо следить, чтобы наборы данных совпадали, чтобы наборы колонок и типы данных в этих колонках были идентичны.

Вопросы учеников

Может ли оператор UNION использоваться в запросах-группировках?

Да, вот пример такого запроса:

```
SELECT firstname, count(lastname) FROM (SELECT * FROM man UNION SELECT * FROM
man1 ) m1 GROUP BY firstname
```

Можно ли создать запрос, который бы работал со вложенным UNION?

Пример подобного запроса:

```
SELECT phonenum FROM AUTO UNION (SELECT phonenum FROM man UNION SELECT
phonenum FROM man1 )
```

Чем можно заменить MINUS INTERSECT?

Запросами с использованием подзапросов EXISTS и IN. Попробуйте написать подобные запросы самостоятельно.

Контрольные вопросы и задания для самостоятельного выполнения

1. Повторите материалы данного шага.

2. Объясните использование операторов работы с множествами (UNION, MINUS, INTERSECT) совместно с подзапросами.

3. Напишите запрос, выбирающий записи из MAN, для которых есть записи в таблицах AUTO и AUTO1.

4. Выберите из таблицы AUTO записи, для которых есть соответствие в таблице MAN и которых нет в MAN1.

Шаг 47. Обновление данных и удаление данных с использованием подзапросов

Введение

Сложные подзапросы, которые мы изучали в предыдущих шагах, очень часто используются в командах обновления данных и удаления данных.

Теория и практика

Обновление строк UPDATE

Напишем запрос для обновления данных в таблице MAN, где люди проживают в городах с населением более миллиона жителей.

Задание звучит следующим образом.

Обновите возраст людей в таблице MAN (YEAROLD +1) у тех людей, которые проживают в городах с населением (PEOPLES) более миллиона человек.

```
update man set yearold = yeaold + 1 where citycode in  (SELECT citycode FROM city WHERE
peoples > 100000);
```

Обновите возраст людей в таблице MAN (YEAROLD +1) только у тех людей, для которых людей с таким же возрастом нет в таблице MAN.

```
update man m1 set yearold = yearold+1 where not exists(select 1 from man m2 where
m2.yearold=m1.yearold)
```

Обновите возраст людей, чье имя начинается с буквы «О», на сумму всех возрастов людей из таблицы MAN с именем Алексей.

```
update man set yearold = (select sum(yearold) from man where firstname = 'Алексей') where
firstname like 'O%';
```

Удаление строк DELETE

Подобный синтаксис можно использовать и при удалении данных с помощью команды DELETE.

Например, удалите из таблицы AUTO1 все записи, которых нет в таблице AUTO.

```
delete auto1 where auto1.regnum not in (select regnum from auto1)
```

Важные замечания

Команды UPDATE и DELETE требуют завершения командой COMMIT для фиксации изменений в базе данных.

Следует с особой осторожностью относиться к выполнению данных команд, так как после их выполнения восстановить данные в первоначальном виде может быть довольно сложно.

> *Важный лайфхак: лично я сначала всегда пытаюсь написать запрос SELECT, который вернет мне нужные данные, а уже потом преобразую этот запрос в команду UPDATE или DELETE.*

Вопросы учеников

Понятно, что различные типы подзапросов можно использовать с операторами UPDATE, DELETE, а как воспользоваться конструкциями MINUS и INTERSECT для этих команд?

Да, это возможно. Например, удалите из таблицы AUTO1 все записи, которых нет в таблице AUTO.

```
delete auto1 where auto1.regnum in (select regnum from auto minus select regnum from
auto1)
```

Контрольные вопросы и задания для самостоятельного выполнения

1. Напишите команду для удаления из таблицы MAN всех строк с информацией о людях, проживающих в городах с населением 200 000.

2. Обновите номера автомобилей, добавив к номеру 111 в конце строки, только тех, для которых нет совпадающих телефонов в таблице MAN.

Шаг 48. Нормализация. Проектирование базы данных. Основы

Введение

Нормализация – это правила, позволяющие улучшить, оптимизировать базу данных.

Нормализация ставит задачу обеспечить поддержку целостности данных. Нормализация – процесс, обеспечивающий хранение одного элемента в одном-единственном экземпляре в базе данных, позволяя тем самым избегать дублирования.

База данных считается нормализованной, если ее таблицы представлены как минимум в третьей нормальной форме.

Процесс нормализации данных заключается в устранении избыточности данных в таблицах, в разбиении больших таблиц с множеством колонок на таблицы с меньшим количеством колонок.

То есть нормализация – это процесс приведения базы данных и структуры хранения данных к нормальным формам.

Существует шесть нормальных форм, для создания нормализованной базы данных достаточно знать первые три формы.

Теория и практика

Ненормализованная база данных

У таблиц нет первичных ключей, информация в строках таблиц дублируется, в одной колонке таблицы содержится несколько неатомарных значений.

Пример такой таблицы:

NAME	CARS	CITY
ПЕТРОВ	BMW 12312;ВАЗ 2432	Москва
ПЕТРОВ	ТАВРИЯ 132332	МИНСК;Москва
Иванов 495-233	Scoda ; Mersedes	Москва 10000000
Иванов 495-233	Volvo	Москва
Сидоров 23423-23		Минск
Сидоров	Tesla	NewYork

Рисунок 136. Таблица ненормализованная

У таблицы нет первичных ключей, и в колонках CITY и CARS содержатся неатомарные значения.

Первая нормальная форма

Более организованно выглядит база, когда данные находятся в первой нормальной форме.

Для этого таблицы в базе должны удовлетворять следующим правилам.

Каждый атрибут отношения должен хранить атомарное значение, каждое отношение (строка в таблице) должно содержать одинаковое количество атрибутов (столбцов), это первая нормальная форма.

• Запрещает повторяющиеся колонки (содержащие одинаковую по смыслу информацию);

• запрещает множественные колонки (содержащие значения типа списка и т.п.);

• требует определить первичный ключ для таблицы, то есть ту колонку или комбинацию колонок, которые однозначно определяют каждую строку.

Пример таблицы в первой нормальной форме:

PHONE	CARN	CAR	NAME
123-232-232	3142	BMW	Иванов
123-232-232	3423432	LADA	Иванов
3232-2324-23	235345	УАЗ	Петров

Рисунок 137. Первая нормальная форма

• В таблице нет множественных колонок, в таблице нет колонок, содержащих одинаковую по смыслу информацию;

• комбинация колонок CARN, PHONE – это уникальное сочетание, однозначно определяющее строку.

Вторая нормальная форма

Считается, что база данных находится во второй нормальной форме, если соблюдаются следующие условия:

• таблица уже находится в 1НФ и при этом все неключевые атрибуты зависят только от первичного ключа, то есть

• вторая нормальная форма требует, чтобы неключевые колонки таблиц зависели от первичного ключа в целом, но не от его части;

• если таблица находится в первой нормальной форме и первичный ключ у нее состоит из одного столбца, то она автоматически находится и во второй нормальной форме.

Пример

Покупатели

PHONE	STATUS	NAME
123-232-232	Водит	Иванов
3232-2324-23	Крут вод	Петров

Рисунок 138. Вторая нормальная форма

Машины

CARN
3142

Рисунок 139. Вторая нормальная форма

В данном случае у нас есть таблицы, для каждой из которых соблюдаются правила второй нормальной формы. В таблице 1НФ у нас был составной ключ и зависимость записей от части ключа. В этом примере для машин первичный ключ CARN, для покупателей – PHONE.

Третья нормальная форма

Третья нормальная форма – это когда таблица находится во второй нормальной форме, каждый неключевой атрибут зависит только от первичного ключа и такие атрибуты не зависят друг от друга.

Покупатели

PHONE	NAME
123-232-232	Иванов
3232-2324-23	Петров

Машины

CARN	CAR
3142	BMW
3423432	LADA
235345	УАЗ

Транзакции

TRNO	CARN	PHONE
1	3142	123-232-232
2	3423432	123-232-232
3	235345	

Рисунок 140. Третья нормальная форма

Атрибуты – колонки таблиц не находятся в зависимости друг от друга и зависят только от первичного ключа.

Важные замечания

Для первой нормальной формы также важно соответствие базы данных следующим требованиям:

• строки таблиц не должны зависеть друг от друга, то есть первая запись не должна влиять на вторую и наоборот, вторая на третью и т. д. Порядок размещения записей в таблице не имеет никакого значения;

• порядок размещения колонок таблицы также не должен иметь никакого значения.

Вопросы учеников

Нормализация базы данных до третьей нормальной формы является обязательной всегда?

Нет, на практике очень часто встречается, что приложение не соответствует 3НФ, это делается ради производительности. Например, для большинства DWH-проектов данные ненормализованные.

А какие еще есть нормальные формы?

Всего пять нормальных форм плюс нормальная форма Бойса Кодда, вы можете ознакомиться более подробно с важной темой нормализации базы данных самостоятельно.

Контрольные вопросы и задания для самостоятельного выполнения

1. Повторите материалы данного шага.
2. Запомните определения нормальных форм.
3. Дайте определение первой нормальной формы.
4. Объясните отличия первой нормальной формы от второй нормальной формы.

Шаг 49. Сложные задачи с собеседований в крупные компании с решениями

Введение

Здесь собраны лучшие задачи с собеседований в крупные компании, банки, западные фирмы.

Задачи адаптированны к нашей учебной схеме, с готовыми решениями.

Сморите, запоминайте и попробуйте решить эти задачи сами.

Задачи

1. Найти количество букв «о» в слове «молоко» с помощью запроса.

```
select length('молоко') - length(replace('молоко', 'о')) from dual
```

2. Какой запрос будет работать быстрее:

```
select cityname from city where populations > 100000
union select name from city
```

или же альтернативный вариант?

```
select cityname from city where populations > 100000
 union all select name from city
```

Второй, так как первый убирает дубли и тратит на это ресурсы.

3. Выбрать код, название города и популяцию, код города и популяцию, где код города больше текущего кода города на 2.

```
select * from city c left join city c1 on c.citycode +2= c1.citycode
```

4. Вывести марку авто, а также сколько авто такой марки в таблице, вывести записи, где количество авто такой марки больше 2; записи не должны дублироваться.

```
select mark, count(1) from auto a group by mark having count(1)>1
```

5. Вывести марку и цвет автомобиля, имя и фамилию покупателя для всех покупателей, которые живут в городе с населением больше 1 миллиона человек.

```
select mark, color, firstname, lastname from auto a inner join man m on m.phonenum =
a.phonenum where m.citycode in (select citycode from city c where c.peoples> 1000000 )
```

6. Вывести на экран данные о людях, которые живут в городах с населением больше 1 000 000, если людей с таким же именем нет в таблице MAN.

```
select * from man m where m.citycode in (select citycode from city where peoples > 1000000)
and not exists
(select * from man m1 where m.firstname = m1.firstname  and m.phonenum <>
m1.phonenum )
```

7. Вывести на экран наименование города, если в таблице нет города, который начинается на такую же букву.

```
select * from city c where substr(c.cityname,1,1) not in (select substr(c1.cityname,1,1) from
city c1 where c1.citycode<> c.citycode)
```

8. Вывести на экран, сколько машин каждого цвета для машин марок BMW и LADA.

```
select color, mark, count(1) from auto where mark in ('BMW','LADA') group by color, mark
```

9. Подсчитать количество BMW в AUTO.

```
select count(1) as countbmw from auto where mark = 'BMW'
```

10. Вывести на экран марку авто и количество AUTO не этой марки.

```
select distinct mark, (select count(1) from auto a1 where a1.mark != a.mark) as c from auto a
```

Шаг 50. Сложные задачи и вопросы для самостоятельного выполнения

Введение

На этом шаге вам предстоит познакомиться с задачами для самостоятельного выполнения. Решение этих задач без посторонней помощи и подсказок покажет, насколько хорошо вы усвоили материал предыдущих шагов. Если вы чего-либо не помните, не знаете, просто пролистайте книгу к началу и повторите те шаги, с которыми возникли сложности.

Задачи

• Напишите запрос на обновление MAN, который добавит в конце номера телефона символ «звездочка», если данный человек живет в городе с населением более 2 миллионов человек. Используйте таблицы CITY (PEOPLES), MAN (PHONENUM).

• Выведите на экран дату из таблицы AUTO в формате как в примере 22-12-1999 11:11:11, напишите соответствующий запрос.

• Подсчитайте средний возраст людей из таблицы MAN, чья фамилия начинается с буквы А.

• Выведите первую букву названия города из CITY, а также имена людей, начинающихся с этой буквы, если эти люди проживают в данном городе.

• Выведите название города из CITY, а также количество людей, которые проживают в этом городе, из таблицы MAN, которым больше 28 лет.

• Напишите запрос, который выводит на экран номер и марку автомобиля и номер автомобиля, выпущенного на 2 года позже данной даты.

• Найдите с помощью запроса количество букв «а» в имени каждого человека из MAN, выведите **FIRSTNAME**, **LASTNAME** и результат вычислений.

• Найдите количество автомобилей каждого цвета и в отдельной колонке посчитайте, сколько авто такого цвета AUTO (COLOR) выпущено после 2000 года. Напишите запрос, возвращающий следующую информацию: цвет, количество, количество после 2000 года.

• Выведите все автомобили, которые купили люди, проживающие в городе, где в названии есть буква «и».

• Выведите с помощью запроса, сколько строк останется, если убрать из таблицы людей с одинаковыми именами.

День одиннадцатый

Шаг 51. SQL – расширенные знания. Чем дальше, тем… интереснее

Введение

Вот и пройдена первая часть нашей книги. Изучено множество тем.
Попробуйте задать себе следующие вопросы:

• Какие шаги вызвали у вас наибольшие сложности?
• Какие темы показались вам наиболее простыми в освоении?
• Какие темы все же следовало объяснить как-то по-другому с точки зрения подачи материала?

Если у вас остались вопросы по пройденному материалу, задайте их в группе Facebook, указанной на сайте школы SQLADV.ru.

Предлагаю вам для закрепления знаний ответить на несколько теоретических вопросов по темам, которые мы уже прошли.

Контрольные вопросы для закрепления материала

Вот несколько вопросов для проверки и закрепления ваших знаний:

1. Что такое первичный ключ в таблице?
2. Как работает HAVING в запросах с группировками?
3. Поясните назначение оператора INTERSECT.
4. Поясните назначение функции LENGTH при работе со строками.
5. Поясните назначение функции SYSDATE.
6. Как вывести на экран текущий день недели, используя SYSDATE и TO_CHAR?
7. Как удалить все данные из таблицы?
8. Как подсчитать количество строк в таблице?
9. Для чего используется INNER JOIN?
10. Что такое нормализация? Объясните определение первой, второй и третьей нормальных форм.

Теория и практика

Вторая часть книги представляет собой расширенный углубленный курс языка SQL.

В следующих шагах мы узнаем:

• что такое аналитический SQL, где он применяется, как его использовать, назначение основных аналитических функций;

• многообразие действий сложного оператора MODEL, который позволяет использовать агрегатные функции, сложные группировки, построение математических итераторов;

• конструкцию CONNECT BY для создания итераторов и построения иерархических запросов;

• операторы множественной вставки и множественного обновления;

• распределенные базы данных и конструкцию DATABASE LINK;

• ретроспективные запросы и восстановление данных;

• работу с блокировками и доступ одновременно нескольких сессий к базе данных

• и множество прочих интереснейших материалов, которые можно будет с успехом использовать в дальнейшей работе.

Важные замечания

Для решения практических примеров в некоторых дальнейших главах книги понадобится установка дополнительного программного обеспечения. Требуется установить специальный сервер ORACLEXE.

Также необходимо установить и загрузить специальную утилиту SQL DEVELOPER с официального сайта ORACLE. Данные программы распространяются по свободной лицензии и устанавливаются довольно несложно. Сам процесс установки быстрый и не требует специальных знаний.

На официальном сайте ORACLE.COM необходимо перейти в раздел DOWNLOAD, потом, после регистрации на сайте, скачать необходимое программное обеспечение.

Из множества дистрибутивов следует выбрать SQL DEVELOPER WINdows 64-bit with JDK 8 INcluded и ORACLEXE11 WIN 64 (DATABASE 11g Express Edition).

Установка

Вначале устанавливается ORACLEXE. Загрузите установочный файл, запустите его и следуйте инструкциям установщика.

Когда установщик попросит вас ввести специальный пароль, запомните или запишите пароль, который ввели.

SQL DEVELOPER распаковывается в заданную папку и запускается из этой папки, для запуска программы используется файл plsqldeveloper. exe.

Соединение с базой данных под учетной записью администратора

Запустите программу SQL DEVELOPER.
Выберите новое подключение (в левом верхнем углу – зеленая клавиша плюс «+»).
Заполните следующие поля:

CONNECTION NAME – заполняем SYSCONNECT, хотя можно и другое имя;
USER NAME – заполняем SYS;
PASSWORD – тот пароль, который вы набрали при установке программы.

Отметьте галочку напротив SAVE PASSWORD – CONNECTION TYPE необходимо отметить.
ROLE – необходимо отметить как SYSDBA.

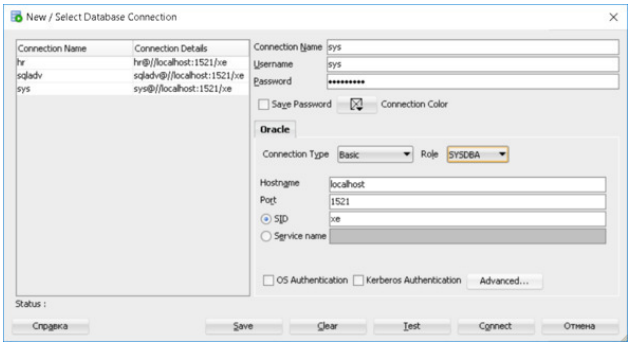


Рисунок 141. Создание подключения

На сайте ORACLE есть подробная инструкция по установке данных программных продуктов. Также множество информации можно найти в интернете на соответствующих тематических ресурсах.

Вопросы учеников

Для изучения скольких шагов потребуется установка дополнительного программного обеспечения?

Примерно семь шагов из книги требуют установки дополнительного ПО.

Что если я не буду проходить эти семь шагов и ограничусь только теорией?

Тогда у вас не получится освоить весь материал из книги, но лучше сделать то, что возможно.

Пройдите сначала те шаги, которые не требуют установки дополнительного ПО, а уже затем шаги, где установка ПО необходима.

Не получается установить эти программы. Что делать?

На официальном сайте ORACLE есть подробная инструкция по их установке. Может быть, вы выбираете неподходящий для вашей системы дистрибутив.

Шаг 52. Вставка данных из запроса

Введение

В SQL есть возможность добавлять данные в таблицу с помощью запроса SELECT. Это особенно удобно, когда нам следует осуществить вставку довольно большого количества данных.

Теория и практика

В такой вставке данных можно задействовать всю мощь оператора выборки SELECT и использовать группировки, агрегатные функции и другие возможности.

Заполнить таблицу данными на основе запроса с помощью команды INSERT.
Синтаксис:

```
INSERT INTO table2
(column_name(s))
SELECT column_name(s) FROM table1,tablen...;
```

где TABLE2 – таблица, куда осуществляется вставка, а SELECT – произвольный запрос.
Примеры

```
iNSERT INTO city1(cityname, citycode, peoples) Select cityname, citycode, peoples+100  from
city WHERE peoples > 1000000
```

В таблицу CITY1 будут добавлены те строки из таблицы CITY, где PEOPLES> 1 000 000, и к полю PEOPLES будет добавлено 100.

```
INSERT NTO CARNAME(NM) SELECT mark||Color FROM auto WHERE mark = 'BMW'
```

В таблицу CARNAME в колонку nm добавляем все марки автомобилей, объединенные с цветом.
Допустим и такой способ записи оператора вставки:

```
iNSERT INTO man1 Select * from MAN
```

То есть в таблицу MAN1 добавляются все записи из таблицы MAN.

Важные замечания

Необходимо, чтобы количество, структура и типы колонок таблицы после оператора INSERT соответствовали структуре и типам колонок в операторе вставки SELECT.

Например, CITY1 имеет такую же структуру, как и таблица CITY.

```
INSERT INTO city(citycpde,cityname) SELECT * FROM city;
```

Это неправильный запрос, так как запрос SELECT возвращает больше колонок, чем перечисленные колонки после команды INSERT (CITYCODE, CITYNAME);

Правильные варианты запросов:

```
INSERT INTO city1(citycpde,cityname) SELECT citycode,cityname FROM city;
INSERT INTO city1 SELECT *FROM city;
```

Следует помнить, что, как и обычная команда вставки INSERT, INSERT INTO SELECT также должна завершаться оператором COMMIT или ROLLBACK для фиксации изменений или отката изменений.

Вопросы учеников

Что делать, если одна из колонок таблицы имеет другой тип данных, отличный от типа колонки в операторе SELECT?

В этом случае необходимо выполнить преобразование типов с помощью функции **CAST, TO_NUMBER, TO_CHAR**.

Я пытаюсь вставить тестовые данные указанным способом, но получаю ошибку, что добавляемые данные имеют большую длину, чем длина колонки. Что порекомендуете в этом случае?

Следует воспользоваться функций SUBSTR для усечения записей из запроса до необходимого размера.

В нашей таблице три колонки, я добавляю данные только в две из них, возможно ли такое?

Да, возможно так сделать, но только если третья колонка не является первичным ключом или не имеет ограничения NOT NULL, в данную колонку будет добавлен NULL.

Контрольные вопросы и задания для самостоятельного выполнения

1. Добавить в таблицу MAN1 записи из MAN, где возраст людей больше 33 лет.

2. Добавить в таблицу MAN1 записи из MAN, где возраст людей больше 50 лет, и к фамилии участника добавить количество лет – пример «Петров54».

3. Добавить в таблицу MAN1 записи из MAN, где возраст людей 19, но к полю «возраст» (AGE) необходимо добавить AGE +1.

Шаг 53. Создание таблиц на основе запроса

Введение

Язык SQL позволяет создавать таблицы на основе запроса. При этом в таком запросе можно использовать группировки, операторы объединения таблиц, математические расчеты.

Теория и практика

Создание таблицы и заполнение таблицы на основе запроса.
Создание таблицы:

```
CREATE TABLE tablename AS
SELECT список полей FROM таблица WHERE GROUP BY
```

Создается таблица по запросу SELECT.

Пример:

```
CREATE TABLE auto1 as SELECT * FROM auto
```

Создание дубликата таблицы AUTO.
Пример:

```
CREATE TABLE aobj1 as SELECT * FROM auto WHERE rownum < 4
```

Создание таблицы AOBJ1 – первые четыре строки из таблицы AUTO.
Пример:

```
CREATE TABLE aobj2 as SELECT color, mark FROM auto WHERE rownum < 3
```

Создание таблицы AOBJ2 – первыми добавлены первые две строки из таблицы AUTO.
Пример:

```
CREATE TABLE aobj3 as SELECT firstname, lastname FROM city WHERE peoples>100000
```

Также мы можем заполнить таблицу данными на основе запроса.

Важные замечания

Для того чтобы скопировать только структуру таблицы, без данных, можно воспользоваться следующим приемом:

CREATE TABLE TAB1 AS SELECT * FROM TAB2 WHERE 1=0

Естественно, такой способ не позволяет создавать ключи и индексы. Их необходимо будет создать отдельным скриптом.

Этот метод работает со всеми типами колонок, в том числе и с колонками BLOB, CLOB.

Вопросы учеников

Нужны ли специальные права для создания таблиц?

Для создания в своей схеме не нужны. Для создания в другой схеме необходима привилегия CREATE ANY TABLE.

Используются соответствующие соединения между базами данных, создание таблицы на основе запроса завершается сообщением об ошибке.

При создании таблицы через соединения DB_LINK подобным способом действительно могут быть определенные сложности. Они связаны с передачей данных в колонках CLOB, BLOB, BFILE. Как решить эту проблему, будет рассказано в одном из следующих шагов.

Контрольные вопросы и задания для самостоятельного выполнения

1. Повторите материалы данного шага.
2. Создайте дубликат таблицы CITY – CITY1.
3. Создайте таблицу на основе запроса к таблице AUTO с полями «цвет автомобиля», «количество автомобилей».
4. Создайте дубликат таблицы AUTO – AUTO1.

Шаг 54. PIVOT – переворачиваем запрос с группировкой

Введение

Иногда необходимо транспонировать таблицу или запрос, то есть чтобы данные, которые были в строчках таблицы, стали бы столбцами. Данные в ячейках таблицы становятся заголовками столбцов.

Теория и практика

Для транспонирования таблицы с группировкой в ORACLE SQL используется специальный оператор PIVOT.

Синтаксис конструкции:

```
SELECT * FROM
(
 SELECT fieldlist
 FROM tables
 WHERE conditions
)
PIVOT
(
 aggregate_function(field)
 FOR field2
 IN ( expr1, expr2, ... expr_n) | subquery
) ORDER BY expression [ ASC | DESC ];
```

Для того чтобы продемонстрировать работу данной функции, создадим специальный пример.

СОЗДАДИМ ДЕМО-ТАБЛИЦУ.

Условимся, что перечень значений свойств был ограничен.

Для примера создадим таблицу со следующими полями:

- идентификатор;
- название;
- цвет.

По условиям задачи у нас ЗАДАННОЕ количество цветов – красный, зеленый, синий.

```
CREATE TABLE t1(tid NUMBER, tname VARCHAR2(20),tprop VARCHAR2(20));
```

Заполним таблицу данными, выделяя каждую команду отдельно.

```
INSERT INTO t1 VALUES(1,'круг','красный');
INSERT INTO t1 VALUES(2,'круг','зеленый');
INSERT INTO t1 VALUES(3,'круг','зеленый');
INSERT INTO t1 VALUES(4,'круг','синий');
INSERT INTO t1 VALUES(5,'квадрат','красный');
INSERT INTO t1 VALUES(6,'квадрат','синий');
INSERT INTO t1 VALUES(7,'квадрат','красный');
INSERT INTO t1 VALUES(8,'ромб','красный');
INSERT INTO t1 VALUES(9,'ромб','синий');
COMMIT;
```

ВЫПОЛНИМ ЗАПРОС, чтобы посмотреть, что у нас получилось.

```
SELECT tname, tprop, COUNT(tprop) count FROM t1 GROUP BY tname, tprop
```

Другой пример – запрос с группировкой значений.

```
SELECT tname, SUM(DECODE(tprop,'синий',1,0)) "синий",
SUM(DECODE(tprop,'красный',1,0)) "красный" , SUM(DECODE(tprop,'зеленый',1,0))
"зеленый"  FROM t1 GROUP BY tname
```

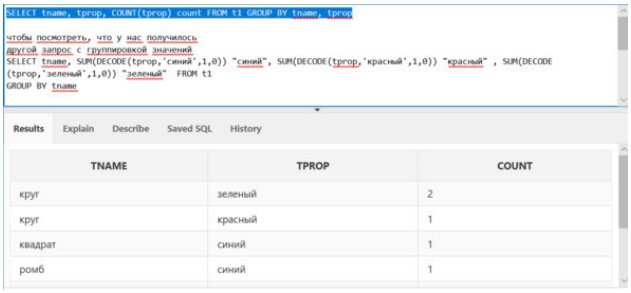


Рисунок 142. Использование PIVOT

И перепишем уже запрос с использованием PIVOT.

```
SELECT * FROM (select tname, tprop from t1)
  PIVOT ( count(tprop) FOR tprop IN  ('красный','синий','зеленый'));
в данном случае count(tprop)  это  aggregate_function(field)
IN  ('красный','синий','зеленый')); -- значения транспонирования
```

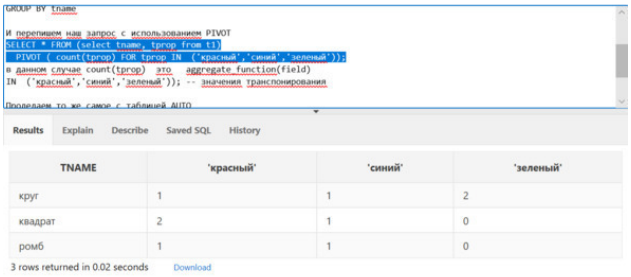


Рисунок 143. Запрос с группировкой: демонстрация PIVOT

Проделаем то же самое с таблицей AUTO.

```
SELECT * FROM (select mark, color from auto)
 PIVOT ( count(color) FOR color IN  ('КРАСНЫЙ','СИНИЙ','ЗЕЛЕНЫЙ'));
```

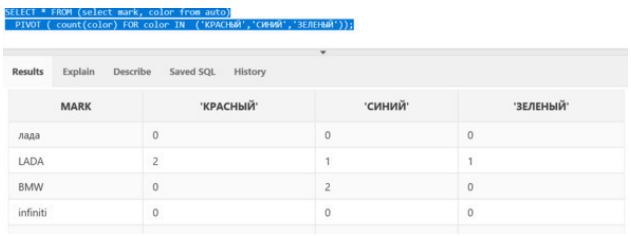


Рисунок 144. Демонстрация использования AUTO

Удалим тестовую таблицу t1.

Важные замечания

Для функции PIVOT существует обратная функция **UNPIVOT** – для преобразования строки запроса в столбец.

Вопросы учеников

Какие агрегатные функции мы можем использовать в конструкции PIVOT?

Такие же, как и в запросах с группировкой GROUP BY. Это могут быть SUM, COUNT, MIN, MAX, AVG.

Как задать свои наименования столбцов?

После оператора FOR можно задавать свои наименования столбцов через инструкцию AS.

Пример:

```
SELECT * FROM (select mark, color from auto)
 PIVOT ( count(color) FOR color IN  ('КРАСНЫЙ' as red,'СИНИЙ' as blue ,'ЗЕЛЕНЫЙ' as
green));
```

Контрольные вопросы и задания для самостоятельного выполнения

1. Сделайте то же самое, только чтобы в первом столбце были цвета, а в названиях полей – названия геометрических фигур, то есть транспонирование по названию фигур геометрии.
2. Повторите с использованием PIVOT запрос с марками авто AUTO (MARK).

Шаг 55. Использование итераторов

Введение

Язык SQL позволяет программировать специальные последовательности целых чисел (итераторы), которые могут использоваться в запросах.

Теория и практика

Создание итераторов реализуется с помощью конструкции CONNECT BY.
Синтаксис

```
SELECT LEVEL AS lv FROM dual CONNECT BY LEVEL<N
```

В переменную LEVEL будут выведены последовательно значения от 1 до N.
Простой пример: выведем в запросе все числа от 1 до 100.

```
SELECT LEVEL AS lv FROM dual CONNECT BY LEVEL<101
```

То есть специальное ключевое слово LEVEL выполняет роль последовательности от 0 до ограничения LEVEL <101.

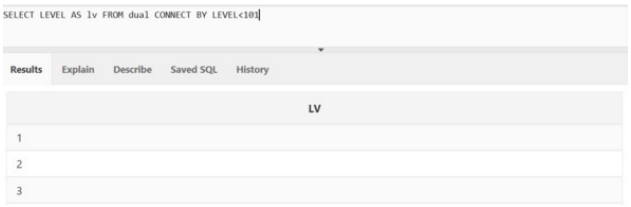


Рисунок 145. Демонстрация работы итератора

Пример итератора Connect By для работы с датами на ORACLE SQL.
Вывести календарь на следующие 30 дней.

```
SELECT sysdate+LEVEL AS dtl, TO_CHAR(sysdate+LEVEL, 'dy') AS dn FROM dual CONNECT BY
level<30
```

Здесь level применяется одновременно с функцией SYSDATE.

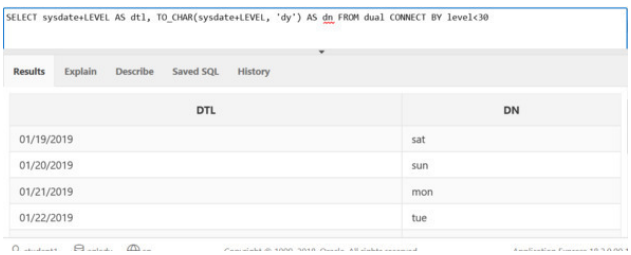


Рисунок 146. Использование итераторов

Вывести календарь на следующие 30 дней, без субботы и воскресенья.

```
SELECT * FROM (  SELECT sysdate+level AS lv, TO_CHAR(sysdate+LEVEL, 'dy') AS ddy FROM
dual CONNECT BY level<30 ) cl WHERE cl.ddy NOT IN ('сб','вс');
```

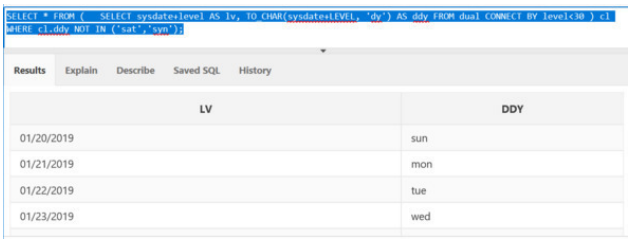


Рисунок 147. Использование итераторов

Вывести числа от 1 до 100, кроме чисел, которые делятся на 3 нацело.

```
SELECT * FROM (
SELECT level lv FROM dual CONNECT BY LEVEL<101) WHERE MOD(lv,3)<>0
```

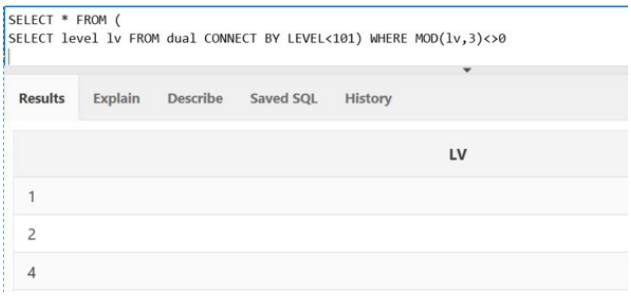


Рисунок 148. Использование итератора MOD

Важные замечания

Данную конструкцию итератора CONNECT BY можно использовать в сочетании с другими запросами SQL, например:

```
SELECT * FROM city WHERE citycode in (SELECT level FROM DUAL CONNECT BY Level<11)
```

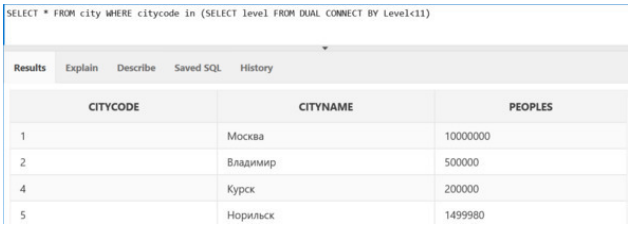


Рисунок 149. Использование итератора

Иногда это может быть удобно и востребовано на практике.

Вопросы учеников

Я пишу следующий запрос: SELECT level FROM DUAL CONNECT BY (Level <11 and level> 5), но результат неправильный, выводится только одна строка, почему так происходит?

Подобный синтаксис, действительно, хоть и не вызывает ошибку, но приводит к некорректным результатам.

Попробуйте сделать следующим образом:

```
SELECT level+5  FROM DUAL CONNECT BY (Level<11 - 5)
```

Чем отличается Sequence от итераторов в запросах с CONNECT BY?

Это совершенно разные сущности, хотя результат их работы в чем-то схож. SEQUENCE представляет собой реальный физический объект в базе данных, обращение к которому позволяет вывести на экран заданную последовательность значений, запрос CONNECT BY LEVEL – это просто способ сделать почти то же самое в запросе, без создания SEQUENCE.

Контрольные вопросы и задания для самостоятельного выполнения

1. Вывести с помощью запроса значение sin числа от 0 до 1 с шагом 0,1.

2. Вывести с помощью запроса число и день недели (TO_CHAR (DATE,«DY»)) всех дней за 2008 год.

3. Вывести с помощью запроса все числа от 500 до 1000, кроме тех чисел, которые делятся нацело на 10.

День двенадцатый

Шаг 56. Иерархические запросы CONNECT BY

Введение

Достаточно часто необходимо представить данные в иерархическом виде. Например, есть иерархическая система подчинения подразделений на предприятии, либо состав изделий, когда одно изделие состоит из других, а те, в свою очередь, также состоят из более мелких деталей.

Для этого используется древовидная иерархическая система, такая, где есть идентификатор и есть идентификатор родителя (родительского узла) ID, PARENTED. Для работы с подобной иерархической структурой также используется конструкция SQL – CONNECT BY.

Теория и практика

Оператор CONNECT BY используется для запросов, где требуется работа со сложной древовидной иерархией.

Например, уровни подчинения сотрудников компании или сложная структура отделов на предприятии.

```
select level... start with нач условие обхода дерева
 connect by PRIOR nocycle recurse-condition ORDER SIBLINGS By с учетом уровня
```

где LEVEL – специальная переменная, означающая уровень иерархии;
START WITH – ключевой оператор древовидной структуры: с какого элемента необходимо начинать обход;
INITIAL-CONDITION – условное выражение для обозначения начала обхода.
Пример: создадим и заполним таблицу.

```
CREATE TABLE parentchild(id NUMBER PRIMARY KEY, name VARCHAR2(50), idparent
NUMBER);
```

Заполним таблицу, выполняя последовательно каждую команду.

```
INSERT INTO parentchild VALUES (1,'Олег', null) ;
INSERT INTO parentchild VALUES (2,'Витя', 1) ;
INSERT INTO parentchild VALUES (3,'Саша', 2) ;
INSERT INTO parentchild VALUES (4,'Таня', 3) ;
INSERT INTO parentchild VALUES (5,'Женя', 3) ;
INSERT INTO parentchild VALUES (6,'Костя',4) ;
INSERT INTO parentchild VALUES (7,'Миша', 2) ;
INSERT INTO parentchild VALUES (8,'Леша', 5) ;
INSERT INTO parentchild VALUES (9,'Макс', 5) ;
INSERT INTO parentchild VALUES (10,'Вася', 5) ;
COMMIT;
```

Здесь ID – идентификатор сотрудника, NAME – наименование сотрудника, IDPARENT – идентификатор руководителя данного сотрудника.

Пример позволяет показать иерархию.

```
SELECT LEVEL, lpad('_', 4*LEVEL,'_')||name  as nm
FROM parentchild
START WITH idparent is null
CONNECT BY  PRIOR id=idparent ORDER SIBLINGS By name
```

LEVEL покажет уровень иерархии.

В данном примере генерируется количество символов подчеркивания, соответствующее уровню иерархии LEVEL.

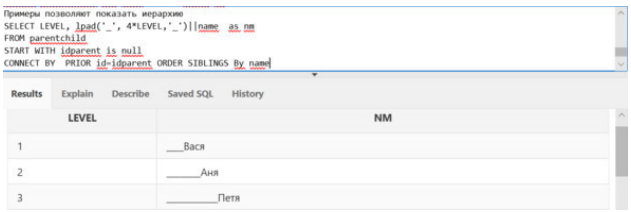


Рисунок 150. Использование CONNECT BY START WITH PRIOR

Важные замечания

В данном способе воспроизведения иерархии с использованием запросов SQL CONNECT BY не работает классическая сортировка ORDER BY, попытка сортировки запроса с помощью ORDER BY ломает иерархию, поэтому для сортировки подобных запросов используется специальная директива ORDER SIBLINGS BY.

Есть возможность представить иерархию в виде пути от каталога к каталогу. Для этого необходимо воспользоваться функцией SYS_CONNECT_BY_PATH (NAME, '/»).

```
SELECT LEVEL, lpad('_', 4*LEVEL,'_')||name  as nm, SYS_CONNECT_BY_PATH(name, '/')FROM
parentchild START WITH idparent is null CONNECT BY  PRIOR id=idparent ORDER SIBLINGS By
name
```

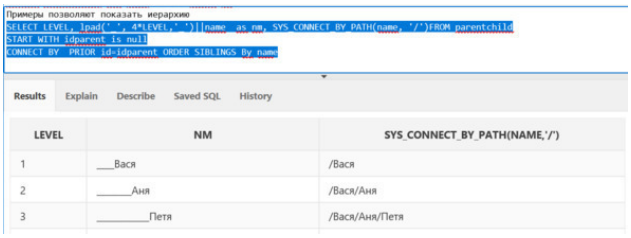


Рисунок 151. CONNECT BY: пример

Иногда при таком способе организации данных возникают так называемые циклические ссылки, это когда родительская запись ссылается на подчиненную, а подчиненная, в свою очередь, на родительскую. Обращение в запросе к таким данным приводит к бесконечному выполнению цикла и ошибке.

Избежать подобной проблемы позволяет использование специальной директивы NOCYCLE.

Переделаем запрос из примера с использованием данной директивы.

```
SELECT LEVEL, lpad('_', 4*LEVEL,'_')||name  as nm, SYS_CONNECT_BY_PATH(name, '/')FROM
parentchild START WITH idparent is null CONNECT BY NOCYCLE PRIOR id=idparent ORDER
SIBLINGS By name
```

Вопросы учеников

Такая конструкция для вывода иерархических данных существует только в диалекте ORACLE SQL?

Нет, в MS SQL и PostgreSQL также существуют подобные операторы. Это оператор WITH RECURSIVE.

```
WITH RECURSIVE
 Rec (id, pid, name)
 AS (
  SELECT id, pid, title FROM parentchild   UNION ALL
  SELECT Rec.id, Rec.pid, Rec.title
   FROM Rec, parentchild
   WHERE Rec.id = parentchild.pid
  )
 SELECT * FROM Rec
 WHERE pid is null;
```

Для MySQL такой конструкции не существует.

Как начать обход дерева не с первого, а со второго, третьего элемента?

Для этого в директиве START WIDTH следует указать идентификатор того элемента, с которого начинается обход древовидной структуры.

Например,

```
SELECT LEVEL, lpad('_', 4*LEVEL,'_')||name  as nm, SYS_CONNECT_BY_PATH(name, '/')FROM
parentchild START WITH idparent =1 CONNECT BY  PRIOR id=idparent
```

Здесь START WITH IDPARENT =1, обход начинается с первого элемента.

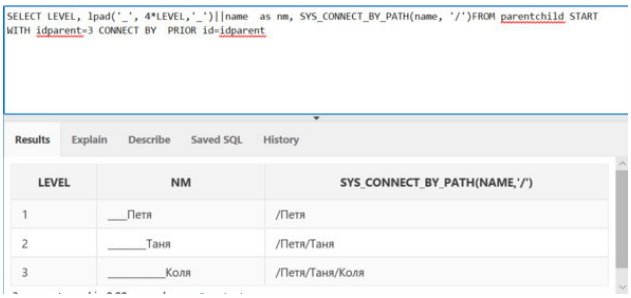


Рисунок 152. CONNECT BY: пример SYS_CONNECT_BY_PATH

Контрольные вопросы и задания для самостоятельного выполнения

1. Повторите материалы данного шага.

2. Создайте таблицу «Детали», где одна деталь может быть частью другой, выведите на экран иерархию деталей.

3. Создайте циклическую иерархию, запустите запрос с использованием NOCYRCLE и без использования NOCYRCLE, проанализируйте результат.

Шаг 57. Условные выражения в SQL-запросе. DECODE/CASE

Введение

В языке SQL есть специальные команды для условных выражений в запросах, то есть выражений, на критерии истинности которых осуществляется ветвление запроса. Аналогичные конструкции есть в алгоритмических языках, таких, например, как JAVA, PASCAL или C ++.

В алгоритмических языках это операторы IF и CASE.

В языке SQL похожая функциональность реализуется с помощью операторов DECODE, CASE.

Теория и практика

Условные выражения в SQL ORACLE диалекта реализуются с помощью операторов CASE и DECODE.

Разберем каждый из условных операторов отдельно с примерами.

DeCODE выводит значение в зависимости от логического выражения.

SELECT DeCODE (выражение, значение, значение если выражение = значение, значение если выражение!= значение) FROM TABLE;

Пример 1

```
select decode(sqrt(4),2,'равно 2','не равно 2') AS EXP from dual
```

равно 2.

Пример 2

```
SELECT cityname, decode(citycode,1,'city1','не city1') as exp FROM City WHERE ROWNUM< 20
```

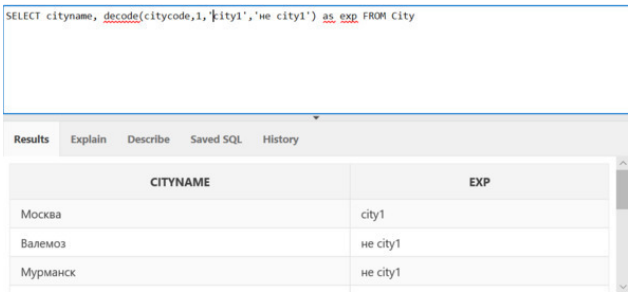


Рисунок 153. Запрос: использование DECODE

Более сложный вариант оператора DECODE:

```
select decode(sqrt(4),1,'равно 2',    'не равно 2') AS EXP from dual
    --равно 2
```

DeCODE (выражение, значение1, значение если выражение= значение1,
– Значение2, значение если выражение= значение2,
– Значениеn, значение если выражение= значениеn,
– Значение если выражение! = значение) FROM TABLE;

Пример 1

```
select decode(sqrt(16),1,'равно 1',2,'равно 2',4,'равно 4', 'не равно 1,2,4') AS EXP from dual
--равно 4
```

Пример 2

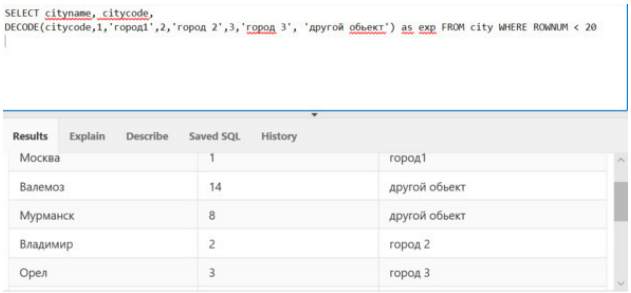


Рисунок 154. Запрос: использование оператора DECODE

```
case
это еще один оператор для условных выражений
CASE [ выражение ]
  WHEN значение если выражение= значение1 THEN result_1
  WHEN значение если выражение= значение2 THEN result_2
  ...
  WHEN значение если выражение= значениеn THEN result_n
  ELSE result (если не равно ни одному из Значений)
END
```

Пример 1

```
select case sqrt(16)
      when 1 then 'равно 1'
      when 2 then 'равно 2'
      when 4 then 'равно 4'
      else 'не равно 1,2,4'
      end as cse from dual;
-- равно 4
```

Пример 2

```
SELECT cityname, citycode,
   case citycode
   WHEN 1 THEN 'город 1'
   WHEN 2 THEN 'город 2'
   WHEN 3 THEN 'город 3'    ELSE 'другой город' end as cs FROM city
```

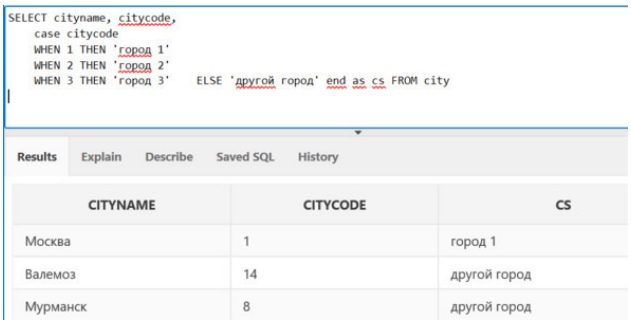


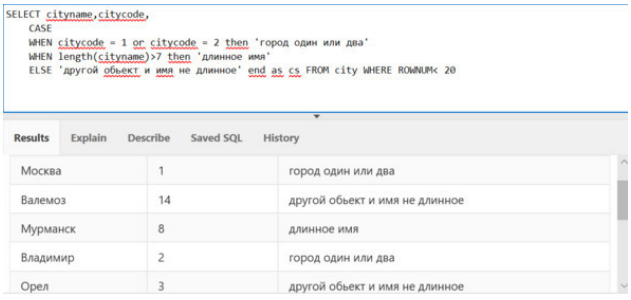
Рисунок 155. Запрос: использование оператора CASE

Еще один вариант CASE:

```
CASE
  WHEN выражение1 THEN result_1 – выражение 1 истинно
  WHEN выражение2 THEN result_2 —выражение 2 истинно
  ...
  ELSE result (если любое выражение не истинно)
END
```

Пример

```
SELECT cityname,citycode,
   CASE    WHEN citycode = 1 or citycode = 2 then 'город один или два'
   WHEN length(cityname)>7 then 'длинное имя'
   ELSE 'другой обьект и имя не длинное' end as cs FROM city WHERE ROWNUM< 20
```



Важные замечания

Если в операторе CASE нет предложения ELSE, тогда оператор CASE вернет NULL.

```
select case sqrt(144)
    when 1 then 'равно 1'
    when 2 then 'равно 2'
    when 4 then 'равно 4'
    else 'не равно 1,2,4'
    end as cse from dual;
```

– результат NULL.

Если в операторе CASE несколько одинаковых сравнений, то правильным будет результат последнего из них.

Вопросы учеников

Существуют ли подобные операторы в других базах данных, например в MS SQL или PostgreSQL?

Это также оператор CASE с аналогичным синтаксисом и назначением.

Как сравнивать интервалы с помощью DECODE? Например, 1—10 – ответ 1, 20—30 – ответ 2, 30—40 – ответ 3.

Вот пример такого сравнения, здесь речь идет об интервале каталогов:

```
SELECT cat_id,
DECODE(TRUNC ((cat_id - 1) / 10), 0, 'catalog 1',
                        1, ' catalog 2',
                        2, ' catalog 3',
                        'unknown') result
FROM catalog;
```

Контрольные вопросы и задания для самостоятельного выполнения

1. Повторите материалы данного шага.

2. Из таблицы AUTO выберите сведения о цвете автомобиля, марке автомобиля, если номер автомобиля начинается с цифры 1 – выведите в отдельной колонке «регион 1», с цифры 2 – «регион 2», с цифры 3 – «регион 3». Используйте DECODE.

3. Сделайте предыдущее задание с использованием CASE.

Шаг 58. Временные таблицы. Когда лучше применять

Введение

Временные таблицы в SQL выполняют разные функции, например разумно применять временные таблицы, когда необходимо сохранить предварительный результат некоторого сложного расчета, подвести предварительные итоги по списку операций или подготовить данные для отчета перед отправкой на печать.

Теория и практика

Временные таблицы существуют в ORACLE начиная с версии 8.
Они предназначаются для хранения данных на протяжении сеанса или транзакции.
Временная таблица поддерживает использование индексов и ограничений.

Временные таблицы (GLOBAL TEMPORARY), в отличие от таблиц регулярных, целесообразно использовать в тех случаях, когда сохраняемые данные часто изменяются и непостоянны.

Данные во временной таблице будут видны только в той сессии или транзакции, которая поместила эти данные в таблицу.

После создания временной таблицы ее описание сохраняется в словаре данных ORACLE.

Место в сегменте под данные выделяется динамически, в момент выполнения первой команды DML (SELECT, INSERT, UPDATE) для этой временной таблицы.

Временная таблица описывается таким же образом, что и обычная регулярная таблица, и все данные такой таблицы подразделяются:

- на данные, используемые только в данной сессии,
- или же данные, используемые только в данной транзакции.

Специфику поведения данных относительно сессии определяют ключевые слова ON COMMIT DELETE и ON COMMIT PRESERVE в команде CREATE TABLE.

- ON COMMIT DELETE ROWS используется во временных таблицах, данные которой существуют в пределах одной транзакции;
- ON COMMIT PRESERVE ROWS используется во временных таблицах, данные которой существуют в пределах одной сессии.

СУБД ORACLE удаляет все строки из временной таблицы – очищает таблицу после завершения сессии или транзакции.

```
CREATE GLOBAL TEMPORARY TABLE TEMP_AUTO
(
NO NUMBER(2,0),
MARK VARCHAR2(14)
)    ON COMMIT PRESERVE ROWS;
```

Данный пример демонстрирует создание временной таблицы, данные которой относятся к сессии.

Пример создания временной таблицы с использованием запроса:

```
CREATE GLOBAL TEMPORARY
TABLE tmpauto
ON COMMIT PRESERVE ROWS
AS SELECT * FROM auto WHERE rownum < 2000
```

Демонстрация отличия временной таблицы от таблицы регулярной.

Создаем две таблицы

- временную

```
CREATE GLOBAL TEMPORARY TABLE TMP_T
(
ID NUMBER(32)
)    ON COMMIT DELETE ROWS;
```

- регулярную

```
CREATE TABLE REG_T
(
ID NUMBER(32)
   ) ;
```

Не будем нагружать скрипты излишним синтаксисом.

Добавляем данные:

```
INSERT INTO TMP_T(ID) VALUES (1);INSERT INTO TMP_T(ID) VALUES (2);
INSERT INTO TMP_T(ID) VALUES (3);INSERT INTO REG_T(ID) VALUES (1);
INSERT INTO REG_T(ID) VALUES (2);INSERT INTO REG_T(ID) VALUES (3);
```

Выполняем запрос к временной таблице:

```
SELECT * FROM TMP_T;
```

Результат
Нет данных

К регулярной таблице:

```
SELECT * FROM REG_T;
```

Результат:
– 1
– 2
– 3

Закрываем сессию, соединяемся с базой снова. Выполняем запрос к регулярной таблице.

```
SELECT * FROM TMP_T;
```

Результат:
1
2
3

К временной таблице:

```
SELECT * FROM REG_T;
```

Результат:
No DATA.

Итак, данные во временной таблице сохраняются только в текущей сессии или транзакции в зависимости от выражения ON COMMIT в скрипте создания таблицы.

Отличие ON COMMIT PRESERVE ROWS от ON COMMIT DELETE ROWS

Создаем две таблицы ON COMMIT PRESERVE ROWS.

```
CREATE GLOBAL TEMPORARY TABLE SCOTT.TMP_PRE_ROWS
(    ID NUMBER(32)     )    ON COMMIT PRESERVE ROWS;
ON COMMIT DELETE ROWS
CREATE GLOBAL TEMPORARY TABLE SCOTT.TMP_DEL_ROWS
(    ID NUMBER(32)     )    ON COMMIT DELETE ROWS;
```

Добавляем данные в созданные таблицы.

```
INSERT INTO TMP_DEL_ROWS(ID) VALUES (1);
INSERT INTO TMP_DEL_ROWS(ID) VALUES (2);
INSERT INTO TMP_DEL_ROWS(ID) VALUES (3);
INSERT INTO TMP_PRE_ROWS(ID) VALUES (1);
INSERT INTO TMP_PRE_ROWS(ID) VALUES (2);
INSERT INTO TMP_PRE_ROWS(ID) VALUES (3);
```

Смотрим:

```
SELECT * FROM TMP_PRE_ROWS;
```

Результат:
1
2
3

```
SELECT * FROM TMP_DEL_ROWS;
```

Результат:
1
2
3
Выполняем команду COMMIT, после чего вновь выполняем запросы.

```
SELECT * FROM SCOTT.TMP_PRE_ROWS;
```

Результат:
1
2
3

```
SELECT * FROM SCOTT.TMP_DEL_ROWS;
```

Результат:
No DATA.

Важные замечания

Для создания временных таблиц, работы с временными таблицами необходимы соответствующие привилегии.

После рестарта экземпляра базы данных данные во временных таблицах не сохраняются.

Если вы используете индексы во временных таблицах, то следует запомнить, что эти индексы нельзя пересоздавать после вставки данных.

Вопросы учеников

Вы рассказали про временные таблицы в ORACLE SQL. Есть ли временные таблицы в MS SQL БД и в postgreSQL?

Временные таблицы в ORACLE существенным образом отличаются от тех же временных таблиц в MS SQL Server.

Есть ряд ограничений, не позволяющих использовать их так, как вы бы сделали, например, в MS SQL Server. Главное отличие в том, что в Oracle временная таблица создается как часть словаря данных и является по сути статической.

Контрольные вопросы и задания для самостоятельного выполнения

1. Повторите материалы данного шага.
2. Создайте две временные таблицы tmdr и tmpr для хранения данных из таблицы CITY ON COMMIT DELETE ROWS и ON COMMIT PRESERVE ROWS.
3. Разберитесь в специфике работы с этими таблицами.

Шаг 59. Регулярные выражения в SQL

Введение

Регулярные выражения (англ. regular expressions, сокр. RegExp, RegEx, жарг. регэкспы или регексы) – это формальный язык поиска и осуществления манипуляций с подстроками в тексте, основанный на использовании метасимволов (символов-джокеров, англ. wildcard characters).

По сути это строка-образец (англ. pattern, по-русски ее часто называют шаблоном, маской), состоящая из символов и метасимволов и задающая правило поиска.

Регулярные выражения сделали революционные преобразования в использовании при поиске данных, при поиске соответствий.

Современные языки программирования, такие как Java, Go, Java Script, имеют специальные средства в своем арсенале для работы с регулярными выражениями.

В SQL диалекта ORACLE есть также множество операторов для работы с регулярными выражениями.

Теория и практика

Для работы с регулярными выражениями в ORACLE SQL используются следующие операторы: REGEXP_LIKE, REGEXP_REPLACE, REGEXP_SUBSTR, REGEXP_COUNT, REGEXP_INSTR.

Примеры

Рассмотрим работу каждой из этих команд, для наглядности создадим таблицу и заполним ее следующими данными:

• фамилия, дата рождения, город проживания – данные представлены сплошным символьным буфером, с разделителем»,».

```
CREATE TABLE regtest
AS SELECT 'Зайцев,01111998,Киев' dt FROM dual UNION
SELECT 'Иванов,01011982,Воронеж' dt FROM dual UNION
SELECT 'Петров,01011988,Москва' dt FROM dual;
```

REGEXP_LIKE

REGEXP_LIKE выбирает из таблицы все строки, соответствующие заданному шаблону регулярного выражения REGEXP.

Пример использования REGEXP_LIKE (выражение; «REGEXP шаблон»): выберем из таблицы все строчки, которые содержат дату рождения в заданном формате REGEXP [0—9] {8}.

```
SELECT * FROM regtest WHERE REGEXP_LIKE(dt,'[0-9]{8}')
```

Рисунок 156. Пример использования REGEXP: запрос к REGTEST

Добавим строку с нестандартным форматом даты.

```
INSERT INTO regtest VALUES ('Волков,010A1988,Дмитров');
select * from regtest where regexp_like(dt,'[0-9]{8}')
```

REGEXP_REPLACE

REGEXP_REPLACE заменяет шаблон регулярного выражения REGEXP в строке на заданную строку.

Синтаксис:

REGEXP_REPLACE (выражение; «REGEXP шаблон», «REGEXP шаблон» или константа) :

Заменим дату рождения в заданном формате REGEXP [0—9] {8} на выражение MYDATE.

```
SELECT REGEXP_REPLACE(t.dt,'[0-9]{8}','mydate') rr FROM regtest t
```

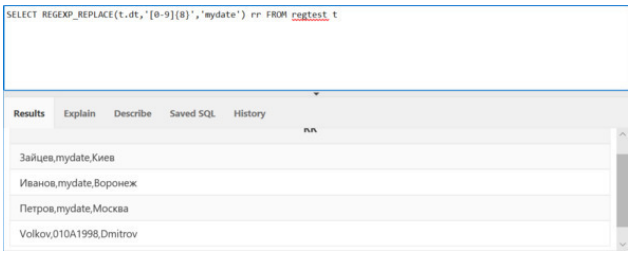


Рисунок 157. работа с REGEXP: запрос к таблице REGTEST

Результат:
Зайцев, MYDATE, Киев
Иванов, MYDATE, Воронеж
Петров, MYDATE, Москва
Волков, 010A1988, Дмитров

REGEXP_SUBSTR

REGEXP_SUBSTR выделяет из строки заданный REGEXP шаблон.

Пример использования REGEXP_SUBSTR (выражение; «REGEXP шаблон»; вхождение; параметр сопоставления) :

Выделим дату рождения в заданном формате REGEXP [0—9] {8} из общей строки.

```
SELECT REGEXP_SUBSTR(t.dt,'[0-9]{8}') rr,t.dt rr FROM regtest t
```

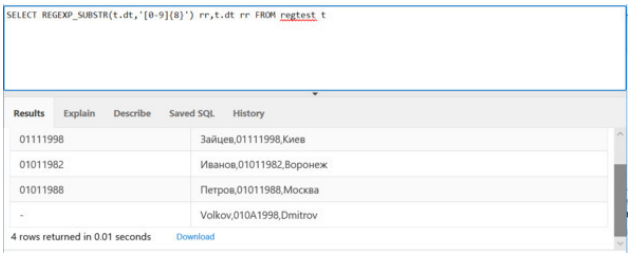


Рисунок 158. Запрос к таблице REGTEST: демонстрация REGEXP_SUBSTR

REGEXP_INSTR

REGEXP_INSTR определяет номер первого символа вхождения REGEXP шаблона в строку.

```
SELECT REGEXP_INSTR(t.dt,'[0-9]{8}') rr,t.dt rr FROM regtest t
```

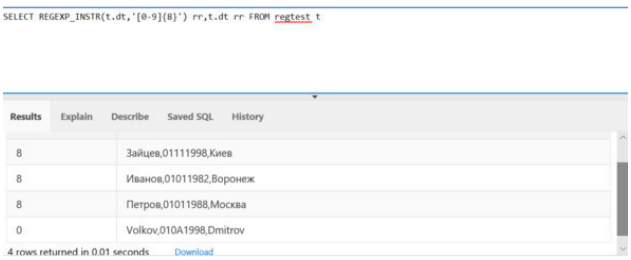


Рисунок 159. Демонстрация REGEXP_INSTR

REGEXP_COUNT

REGEXP_COUNT определяет количество вхождений REGEXP шаблона в строку.

Пример использования REGEXP_COUNT (выражение; «REGEXP шаблон») :

Определим количество цифр REGEXP [0—9] в строках таблицы.

```
SELECT REGEXP_COUNT(t.dt,'[0-9]{1}') rr,t.dt rr FROM regtest t
```

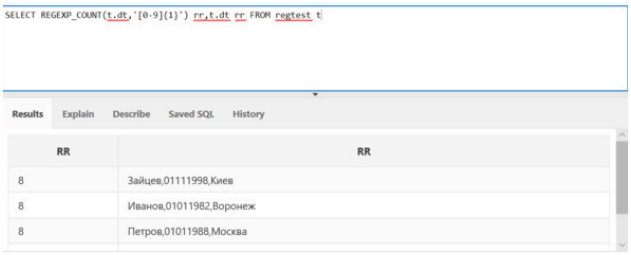


Рисунок 160. Демонстрация REGEXP_COUNT: запрос к таблице REGTEST

Важные замечания

На самом деле в REGEXP_LIKE, REGEXP_SUBSTR имеется ряд дополнительных параметров, которые влияют на результат работы данных функций.

Некоторые дополнительные интересные примеры использования REGEXP.

Выбираем подстроку с разделителями, есть строка Зайцев, 01111998, Киев = Зайцев 01111998 Киев

Для примера выберем только фамилию и город.

```
SELECT REGEXP_SUBSTR(t.dt,'[^,]+',1,1) f, regexp_substr(t.dt,'[^,]+',1,3) city FROM regtest t
```

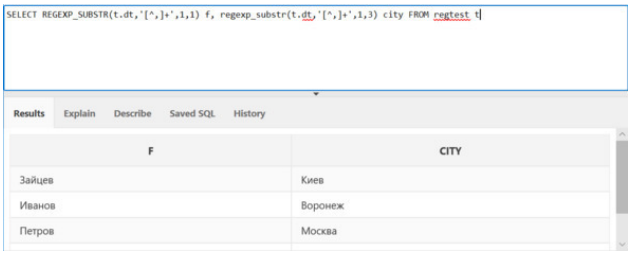


Рисунок 161. Запрос к таблице REGTEST: демонстрация REGEXP_SUBSTR

Воспользовавшись данным методом в сочетании с командой CONNECT BY, можно преобразовать каждую подстроку с разделителями в строку таблицы.

```
SELECT REGEXP_SUBSTR('Петров,01011988,Москва','[^,]+',1,level) ll
    FROM dual
  CONNECT BY level <=  REGEXP_COUNT('Петров,01011988,Москва',',')+1
```

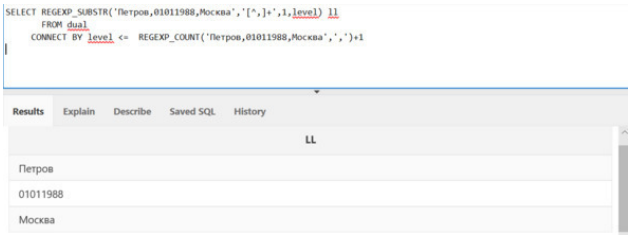


Рисунок 162. Демонстрация REGEXP_SUBSTR и LEVEL

Или более простой пример: найдем количество букв «о» в слове «молоко».

```
SELECT REGEXP_COUNT('Молоко','o') FROM Dual
```

Вопросы учеников

Приведите еще несколько примеров использования регулярных выражений в SQL.

Наиболее часто регулярные выражения используются при проверке номеров телефонов на соответствие заданной маске, веб-страниц, HTML-, XML-документов.

Проверка телефона на соответствие заданной маске:

```
SELECT phonenum FROM phones WHERE NOT REGEXP_LIKE(phones, '\d{3}\.\d{3}\.\d{4}')
```

Парсинг и замена XML-выражения:

```
select regexp_replace(regexp_substr(col,'<old_template_code>.*</old_template_code>'),
                      '(<old_template_code>)(.*)(</old_template_code>)',
                       '\2')from table;
```

Контрольные вопросы и задания для самостоятельного выполнения

1. Повторить запросы ЭТОГО шага.
2. Подсчитать количество букв «т» в слове «трактор», используя регулярные выражения.
3. Используя регулярные выражения, выбрать все телефоны из таблицы AUTO, начинающиеся с 911, 915 (REGEXP_LIKE, REGEXP_SUBSTR).

Шаг 60. Аналитический SQL. Запросы рейтингов. Накопительный итог

Введение

В ORACLE SQL существует специальный тип запросов – аналитические запросы, или запросы с аналитическими функциями. Данные функции используют в качестве одного из аргументов набор данных, который является предварительным результатом обработки основного запроса.

Данные функции используются довольно часто и применяются при обработке сложной финансовой, статистической информации.

Теория и практика

Аналитические запросы в практическом плане чаще применяются для решения следующих специализированных задач:

• запросы с рейтингами;

• запросы с накопительным итогом;

• запросы для вычисления результата в рамках временного окна;

• запросы для поиска следующих и предыдущих значений после текущей записи в источнике данных по некоторому признаку.

На данном шаге разберем запросы с рейтингами и запросы с накопительным итогом. Следующий шаг будет посвящен более сложной теме – вычислению результата в рамках временного окна и поиску следующих и предыдущих значений после текущей записи в источнике данных по некоторому признаку.

Рассмотрим общую структуру аналитических запросов:

```
SELECT аналитическая функция OVER([PARTITION партицирование...]
ORDER BY (упорядочивание выражение 2 [,...] [{ASC/DESC}] [{NULLS FIRST/NULLS LAST}]) a
from t
```

Аналитическая функция – одна из аналитических функций, ниже по тексту есть описание данных функций.

• OVER – специальная конструкция, показывающая базе данных, что в запросе используется аналитический SQL;

• PARTITION – партицирование – это логическая модель разделения данных в запросе, некий сегмент данных, объединенных по общему признаку (обычно это одна или несколько колонок запроса);

• ORDER BY – упорядочивание – показывает, как будут отсортированы данные в рамках заданного сегмента (ASC/DESC): порядок, по возрастанию или по убыванию.

В аналитическом SQL наиболее важная составляющая запроса – аналитические функции.

Разберем основные аналитические функции. Всего аналитических функций более 30, но наиболее часто употребляются именно следующие:

• ROW_NUMBER () – номер строки в группе;

• LAG (f, n,m): f – имя колонки, n – предыдущее значение в группе, m – значение по умолчанию;

• LEAD (f, n,m): f – имя колонки, n – последующее значение в группе, m – значение по умолчанию;

• FIRST_VALUE (f): f – имя колонки, первое значение в группе;

• LAST_VALUE (f): f – имя колонки, последнее значение в группе;

• STD_DEV (f): f – имя колонки, значение стандартного распределения в группе;

• SUM (f): f – имя колонки, накопительная сумма по группе;

• AVG (f): f – имя колонки, среднее по группе заданных групп;
•RANK (f): f – имя колонки, относительный ранг записи в группе;
•DENSe_RANK (f): f – имя поля, абсолютный ранг записи в группе.

Проще всего разобраться с аналитическими функциями и аналитическим SQL на примерах.

Подготовим необходимые данные для демонстрации:

```
CREATE TABLE PersonA(Tbn NUMBER PRIMARY KEY, NAME VARCHAR2(20), otd NUMBER, sal
NUMBER);
```

– табельный номер, имя, отдел, зарплата.

```
INSERT INTO PersonA(Tbn,name,otd,sal) VALUES(1, 'Аня',10,9000);
INSERT INTO PersonA(Tbn,name,otd,sal) VALUES(2, 'Саша',10,5500);
INSERT INTO PersonA(Tbn,name,otd,sal) VALUES(3, 'Таня',10,7000);
INSERT INTO PersonA(Tbn,name,otd,sal) VALUES(4, 'Ваня',20,2300);
INSERT INTO PersonA(Tbn,name,otd,sal) VALUES(5, 'Олег',20,4300);
INSERT INTO PersonA(Tbn,name,otd,sal) VALUES(6, 'Коля',20,3900);
INSERT INTO PersonA(Tbn,name,otd,sal) VALUES(7, 'Таня',30,7000);
INSERT INTO PersonA(Tbn,name,otd,sal) VALUES(8, 'Макс',30,9000);
INSERT INTO PersonA(Tbn,name,otd,sal) VALUES(9, 'Таня',30,8500);
INSERT INTO PersonA(Tbn,name,otd,sal) VALUES(10,'Макс',30,9900);
INSERT INTO PersonA(Tbn,name,otd,sal) VALUES(11,'Олег',30,9900);
INSERT INTO PersonA(Tbn,name,otd,sal) VALUES(12,'Макс',30,9900);
INSERT INTO PersonA(Tbn,name,otd,sal) VALUES(13,'Макс',30,9900);
INSERT INTO PersonA(Tbn,name,otd,sal) VALUES(14,'Макс',30,9900);
INSERT INTO PersonA(Tbn,name,otd,sal) VALUES(15,'Макс',30,9900);
INSERT INTO PersonA(Tbn,name,otd,sal) VALUES(16,'Макс',30,7000);
INSERT INTO PersonA(Tbn,name,otd,sal) VALUES(17,'Таня',30,3500);
```

У нас есть таблица PERSONA, в которой есть колонки «табельный номер сотрудника», «имя сотрудника», «номер отдела, в котором работает сотрудник» и «оклад сотрудника в долларах».

Запросы списка лидеров

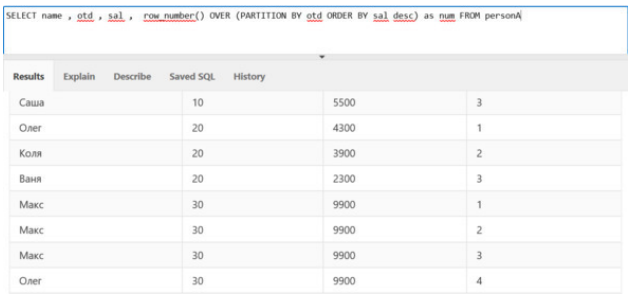
Первые по заплате в рамках отдела, первые по продажам, люди самой высокой зарплатой, рейтинги сотрудников по показателям – все это запросы списка лидеров.

Первые три сотрудника с самой высокой зарплатой по отделам (сегмент по отделу).

Данный запрос делит сотрудников на отделы по номеру отдела PARTITION BY OTD, сортирует сотрудников по заплате и с помощью внешнего запроса выводит на экран первых трех сотрудников с самой высокой зарплатой.

Выполним сначала следующий запрос:

```
SELECT name , otd , sal ,  row_number() OVER (PARTITION BY otd ORDER BY sal desc) as num
FROM personA
```



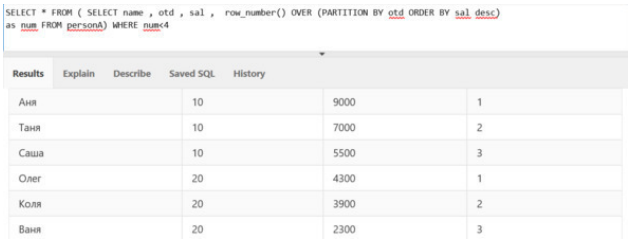
Results	Explain	Describe	Saved SQL	History
Саша	10	5500	3	
Олег	20	4300	1	
Коля	20	3900	2	
Ваня	20	2300	3	
Макс	30	9900	1	
Макс	30	9900	2	
Макс	30	9900	3	
Олег	30	9900	4	

Рисунок 163. Демонстрация работы аналитических функций

Колонка NUM, полученная с помощью аналитической функции ROW_NUMBER () и аналитического подзапроса, возвращает номер строки в наборе данных, разбитом по колонке OTD, отсортированном по SAL.

Преобразуем запрос, чтобы вывести только первых трех сотрудников.

```
SELECT * FROM ( SELECT name , otd , sal ,  row_number() OVER (PARTITION BY otd ORDER BY
sal desc)
as num FROM personA) WHERE num<4
```



Results	Explain	Describe	Saved SQL	History
Аня	10	9000	1	
Таня	10	7000	2	
Саша	10	5500	3	
Олег	20	4300	1	
Коля	20	3900	2	
Ваня	20	2300	3	

Рисунок 164. Демонстрация работы аналитических функций

Еще один пример: данные делятся по наименованию отдела OTD, а сортировка данных в рамках сегмента по имени – NAME.

По наименованию (PARTITION по отделу), сортировка по NAME.

```
SELECT * FROM ( SELECT name , otd , sal ,  row_number() OVER (PARTITION BY otd ORDER BY
name) as num
from personA) WHERE num<4
```

```
SELECT * FROM test_a T WHERE  T.ID IN (select ID FROM TEST_A MINUS select ID FROM
TEST_B)
еще решения
select * from test_A LEFT JOIN test_b ON test_A.ID = test_B.ID WHERE  test_B.ID IS NULL
select * from test_A where test_A.ID <> all(SELECT test_B.ID From test_b)
select * from test_A minus select * from test_A where test_A.ID IN (SELECT (ID) from TEST_B)
```

Вывести рабочие дни в следующем месяце, выходные не учитываются.

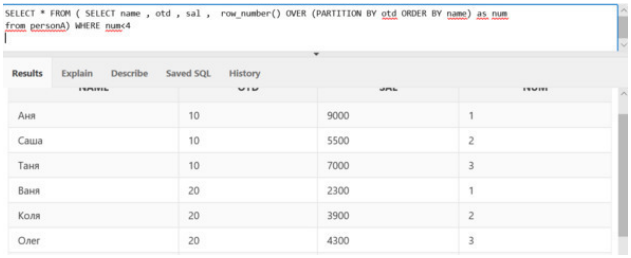


Рисунок 165. Демонстрация работы аналитических функций: запрос к PERSONA

Запросы с накопительным итогом

Накопление суммы в рамках отдела, накопление прибыли в рамках заданного месяца, накопление бонусов в рамках карты с сортировкой по дате – все это запросы с накопительным итогом.

Запросы с накопительным итогом пишутся аналогично запросам, которые мы только что проходили, различаются только функции. В данном случае рассмотрим аналитическую функцию **SUM**.

Накопительный итог по зарплате.

В данном запросе в колонке Num формируется накопительный итог по зарплате – аналитическая функция **SUM (SAL)**, в рамках отдела с сортировкой по зарплате **OVER (PARTITION BY OTD ORDER BY SAL)**.

```
SELECT name , otd , sal ,  sum(sal) OVER (PARTITION BY otd ORDER BY sal) as num FROM
personA
```

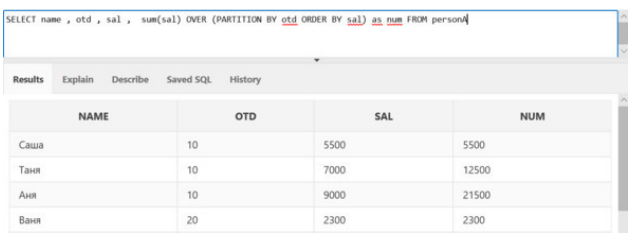


Рисунок 166. Демонстрация работы аналитических функций: запрос к PERSONA

Накопительный итог по зарплате, но с сортировкой по NAME.

В данном запросе в колонке Num формируется накопительный итог по зарплате – аналитическая функция **sum (sal)**, в рамках отдела с сортировкой по колонке name – OVER **(PARTITION BY otd ORDER BY name)**.

```
SELECT name , otd , sal ,  sum(sal) OVER (PARTITION BY otd ORDER BY name) as num FROM personA
```

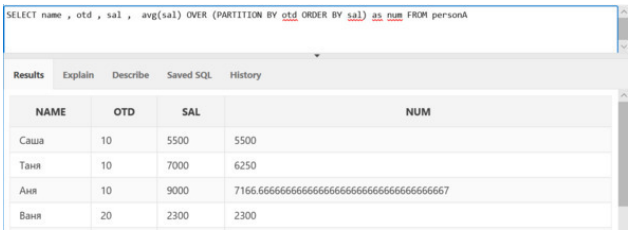


Рисунок 167. Демонстрация работы аналитических функций: запрос к PERSONA

Попробуем найти среднее значение по зарплате с накоплением в рамках отдела, используем avg (sal).

```
SELECT name , otd , sal ,  avg(sal) OVER (PARTITION BY otd ORDER BY sal) as num FROM personA
```

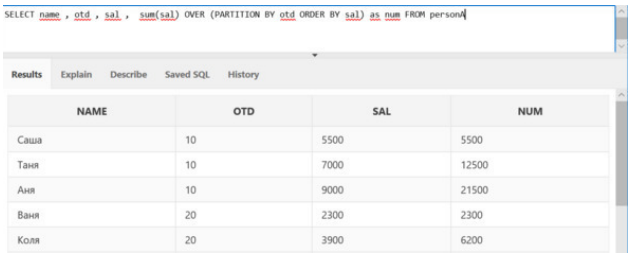


Рисунок 168. Демонстрация работы аналитических функций: запрос к PERSONA

Важные замечания

В первом примере мы использовали аналитическую функцию ROW_NUMBER (), но предположим, что у нескольких людей в рамках одного отдела одинаковая зарплата, а значит в рейтинге эти люди должны быть на одной позиции. Получается, что запрос из первого примера не совсем корректный.

Для правильной работы запроса из первого примера лучше использовать специальную функцию, которая называется RANK ().

Более корректно: используем rank ().

```
SELECT * FROM ( SELECT name , otd , sal
, rank() OVER (PARTITION BY otd ORDER BY sal desc) as num FROM personA) WHERE num<4
```

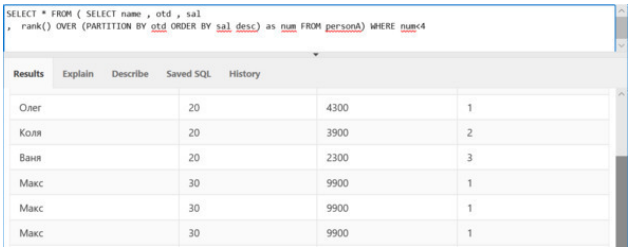


Рисунок 169. Демонстрация работы аналитических функций: запрос к PERSONA

Посмотрите, как изменились результаты: теперь, когда у людей в рамках отдела одинаковая зарплата, им присваивается одинаковый рейтинг – Макс, Макс и Макс из отдела 30.

Пример демонстрирует отличие rank () от row_number ().

Вопросы учеников

Чем отличаются аналитические функции AVG и SUM от соответствующих агрегатных функций, используемых при простой группировке?

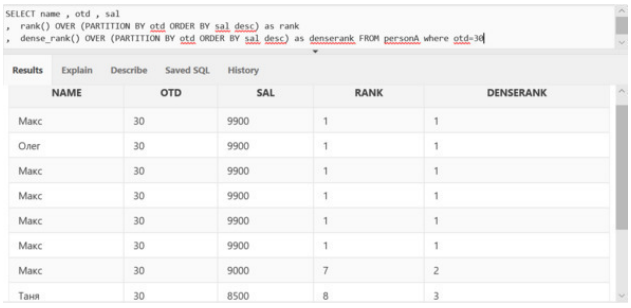
Они имеют совершенно разное назначение: благодаря конструкции OVER в запросе СУБД понимает, что в запросе используется аналитический SQL, тогда будет возвращен накопительный итог, иначе функция будет работать как обычная агрегатная функция.

В чем отличие аналитической функции RANK (f) от функции DENSe_RANK (f)?

RANK (f) считает абсолютный ранг записи в рамках сегмента – отдела в нашем примере, DENSe_RANK вычисляет относительный ранг, с учетом изменения результата.

Посмотрите следующий пример, чтобы лучше понять, как работает RANK ().

```
SELECT name , otd , sal
, rank() OVER (PARTITION BY otd ORDER BY sal desc) as rank
, dense_rank() OVER (PARTITION BY otd ORDER BY sal desc) as denserank FROM personA
```



Results | Explain | Describe | Saved SQL | History

NAME	OTD	SAL	RANK	DENSERANK
Макс	30	9900	1	1
Олег	30	9900	1	1
Макс	30	9900	1	1
Макс	30	9900	1	1
Макс	30	9900	1	1
Макс	30	9900	1	1
Макс	30	9000	7	2
Таня	30	8500	8	3

Рисунок 170. Демонстрация работы аналитических функций: запрос к PERSONA

В данном случае RANK Тани = 3, а dense_RANK Тани = 8.

Можно ли в примере в запросах с рейтингом не использовать сортировку ORDER BY?

Нет, в таком случае данный запрос будет бессмысленным, поскольку параметр, по какому признаку сортировать данные в рейтинге, есть основа самого рейтинга.

Что если в запросе с накопительным итогом не использовать ORDER BY?

Тогда функция вернет окончательный итог без учета накоплений.

Пример запроса:

```
SELECT name , otd , sal , sum(sal) OVER (PARTITION BY otd) as num FROM personA
```

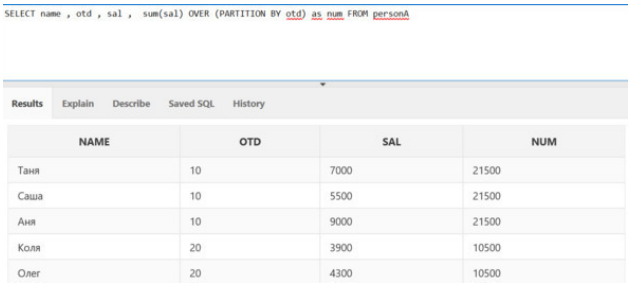


Рисунок 171. Демонстрация работы аналитических функций: запрос к PERSONA, первые N

Что означает в синтаксисе аналитических запросов NULLS FIRST/ NULLS LAST?

Это параметры значат, что если аналитическая функция встречает в запросе пустое значение, то помещает их в конец списка NULLS LAST или в начало списка NULLS FIRST.

Контрольные вопросы и задания для самостоятельного выполнения

1. Разберите запросы из тестовых примеров.
2. Выведите на экран 5 сотрудников с самой высокой зарплатой personA.
3. Выведите на экран 5 сотрудников с самой высокой зарплатой, выведите зарплату, которая меньше, чем у данного в рейтинге personA.
4. Повторите запрос с накопительным итогом зарплаты по отделам с сортировкой по номеру отдела, поясните результат.
5. Поясните отличие аналитической функции ROW_NUM от аналитической функции ORDER BY.

День тринадцатый

Шаг 61. Аналитический SQL. Конструкции окна. Первая и последняя строки

Введение

В запросах с использованием аналитического SQL есть возможность работать с окном данных и использовать записи из набора данных, которые являются следующими либо предыдущими по отношению к текущей записи.

Это сложная тема, поэтому я выделил ее в отдельный шаг.

Теория и практика

Разберемся, что такое:
• запросы для вычисления результата в рамках временного окна;
• запрос для поиска следующих и предыдущих значений после текущей записи в источнике данных по некоторому признаку.
Рассмотрим вновь синтаксис аналитического запроса:

```
SELECT аналитическая функция OVER([PARTITION партицирование...]
ORDER BY (упорядочивание выражение 2 [,...] [{ASC/DESC}] [{NULLS FIRST/NULLS LAST}]) a
from t
ROWS | RANGE} {{UNBOUNDED | выражение} PRECEDING | CURRENT ROW
ROWS | RANGE}
BETWEEN
UNBOUNDED PRECEDING | CURRENT ROW |
UNBOUNDED | выражение 1}{PRECEDING | FOLLOWING
AND
UNBOUNDED FOLLOWING | CURRENT ROW |
UNBOUNDED | выражение 2}{PRECEDING | FOLLOWING
```

Здесь PRECEDING и FOLLOWING задают верхнюю и нижнюю границы окна агрегирования (то есть определяют строки интервала для агрегирования).

```
SELECT name , otd , sal
, avg(sal) over (PARTITION BY otd order by sal ROWS BETWEEN 3 PRECEDING AND CURRENT
ROW) as num FROM personA
```

В запросе используется конструкция окна ROWS BETWEEN N PRECEDING AND CURRENT ROW для вычисления среднего avg (sal), считаются 3 предыдущие строки перед текущей строкой.

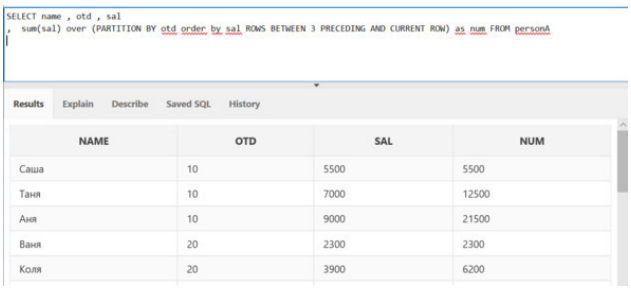
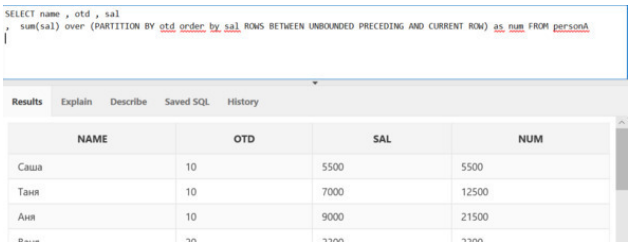


Рисунок 172. Демонстрация работы аналитических функций: запрос к PERSONA, расчет среднего

```
SELECT name , otd , sal
, sum(sal) over (PARTITION BY otd order by sal ROWS BETWEEN UNBOUNDED PRECEDING
AND CURRENT ROW) as num FROM personA
```



NAME	OTD	SAL	NUM
Саша	10	5500	5500
Таня	10	7000	12500
Аня	10	9000	21500

Рисунок 173. Демонстрация работы аналитических функций: запрос к PERSONA, расчет среднего

В запросе используется конструкция окна ROWS BETWEEN N PRECEDING AND CURRENT ROW для вычисления среднего avg (sal), считаются все (UNBOUNDED) предыдущие строки перед текущей строкой.

В аналитическом SQL существуют аналитические функции, которые определяют значение n предыдущих или последующих строк по отношению к текущей строке в группе.

Разберем данные аналитические функции:

• LAG (f, n, m): f – имя поля, n – предыдущее значение в группе, m – значение по умолчанию;

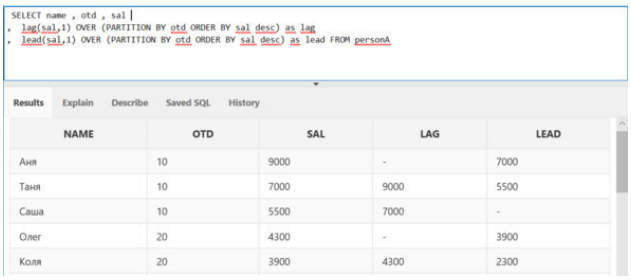
• LEAD (f, n, m): f – имя поля, n – последующее значение в группе, m – значение по умолчанию;

•FIRST_VALUE (f): f – имя поля, первое значение в группе;

• LAST_VALUE (f): f – имя поля, последнее значение в группе.

Несколько примеров использования данных аналитических функций.

```
SELECT name , otd , sal
, lag(sal,1) OVER (PARTITION BY otd ORDER BY sal desc) as lag
, lead(sal,1) OVER (PARTITION BY otd ORDER BY sal desc) as lead FROM personA
```



NAME	OTD	SAL	LAG	LEAD
Аня	10	9000	-	7000
Таня	10	7000	9000	5500
Саша	10	5500	7000	-
Олег	20	4300	-	3900
Коля	20	3900	4300	2300

Рисунок 174. Демонстрация работы аналитических функций: запрос к PERSONA, LAG, LEAD

Выводится зарплата сотрудника, предыдущая зарплата по рейтингу **LAG (SAL,1**), следующая зарплата по рейтингу **LEAD (SAL,1)**.

```
SELECT name , otd , sal
, FIRST_VALUE(sal) OVER (PARTITION BY otd ) as first , LAST_VALUE(sal) OVER (PARTITION BY
otd ) as last FROM personA
```

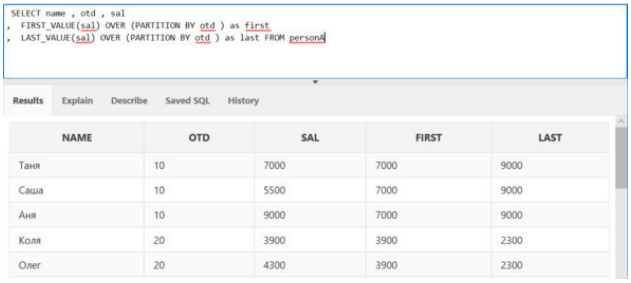


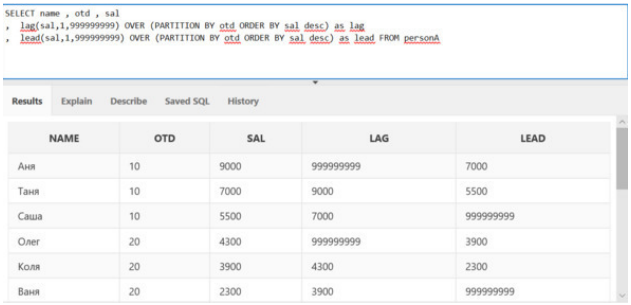
Рисунок 175. Демонстрация работы аналитических функций: запрос к PERSONA, FIRST_VALUE, LAST_VALUE

Важные замечания

В функциях LAG и LEAD есть еще один важный параметр – это значение, которое будет возвращать функция, если результат будет равен NULL.

Пример такого запроса:

```
SELECT name , otd , sal
, lag(sal,1,999999999) OVER (PARTITION BY otd ORDER BY sal desc) as lag
, lead(sal,1,999999999) OVER (PARTITION BY otd ORDER BY sal desc) as lead FROM personA
```



Results | Explain | Describe | Saved SQL | History

NAME	OTD	SAL	LAG	LEAD
Аня	10	9000	999999999	7000
Таня	10	7000	9000	5500
Саша	10	5500	7000	999999999
Олег	20	4300	999999999	3900
Коля	20	3900	4300	2300
Ваня	20	2300	3900	999999999

Рисунок 176. Демонстрация работы аналитических функций: запрос к PERSONA, функции LAG, LEAD

В данном случае пустые значения функций LAG, LEAD заменятся на 999999999.

Обратите внимание, что при использовании FIRST_VALUE, LAST_VALUE указывать сортировку не обязательно. Тогда данные функции будут работать как агрегатные функции MIN и MAX в рамках сегмента.

При указании сортировки LAST_VALUE будет возвращать последнее значение в рамках накопительного итога.

Вопросы учеников

Можно ли использовать LAG, LEAD вне рамок сегмента?

Да, и эта возможность довольно часто используется на практике.

Пример:

```
SELECT name , otd , sal
, lag(sal,1) OVER (ORDER BY sal desc) as lag
, lead(sal,1) OVER (ORDER BY sal desc) as lead FROM personA
```

Как найти в рамках нашего примера зарплату 2 за 3 строчки перед текущей в рамках рейтинга по величине зарплаты?

Второй параметр в *LAG, LEAD* обозначает количество строк перед или после текущей строки, за которой возвращается значение.

```
SELECT name , otd , sal
, lag(sal,3) OVER (ORDER BY sal desc) as lag
, lead(sal,3) OVER (ORDER BY sal desc) as lead FROM personA
```

В конструкции окна аналитической функции есть возможность оперировать интервалами времени. Приведите пример.

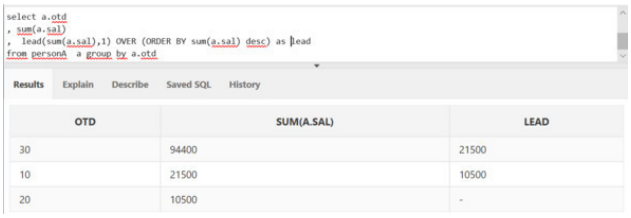
Это возможно с данными типа DATE или TIMESTAMP. Пример интервала за 20 дней назад от даты, соответствующей текущей строке запроса.

```
SELECT name, datem, sal,
AVG(sal)
OVER (ORDER BY datem
RANGE BETWEEN NUMTOYMINTERVAL(20, 'DAY') PRECEDING
AND CURRENT ROW) avg_sal
FROM person1date;
```

Можно ли использовать аналитический SQL, аналитические функции вместе с операциями группировки в запросе?

Да, иногда требуется совместное использование аналитического SQL и групповых операций, приведу пример подобного запроса.

```
select a.otd, sum(a.sal)
, lead(sum(a.sal),1) OVER (ORDER BY sum(a.sal) desc) as lead
from personA  a group by a.otd
```



```
select a.otd
, sum(a.sal)
, lead(sum(a.sal),1) OVER (ORDER BY sum(a.sal) desc) as lead
from personA  a group by a.otd
```

Results | Explain | Describe | Saved SQL | History

OTD	SUM(A.SAL)	LEAD
30	94400	21500
10	21500	10500
20	10500	-

Рисунок 176. Демонстрация работы аналитических функций: запрос к PERSONA, функции LAG, LEAD

Контрольные вопросы и задания для самостоятельного выполнения

1. Выведите значение зарплаты сотрудника в отделе, следующую зарплату по рейтингу.
2. Выведите значение зарплаты сотрудника в отделе, предыдущую зарплату по рейтингу.
3. Выведите значение зарплаты сотрудника в отделе, среднюю зарплату по рейтингу за предыдущие две строки.
4. Выведите значение зарплаты сотрудника в отделах, максимальную и минимальную зарплату в отделах.

Шаг 62. Конструкция KEEP FIRST/LAST

Введение

Конструкция KEEP FIRST/LAST используется в SQL ORACLE для вычисления значения первой или последней записи в заданной подгруппе, отсортированной по некоторому признаку.

Также позволяет найти результат агрегатной функции по сгруппированным данным, если таких значений несколько.

Теория и практика

С помощью KEEP FIRST/LAST реализована возможность выбрать первое (последнее) значение в отсортированном наборе внутри группы.

Проще понять это на примере.

Создадим таблицу курсов валют.

```
CREATE TABLE prices
(
 ticker VARCHAR2(3), -- валюта сокращ
 pdate DATE, -- дата
 price FLOAT -- курс валюты
);
```

Заполним таблицу тестовыми данными.

За каждый день в таблице может быть несколько курсов заданной валюты.

```
INSERT INTO prices( ticker, pdate, price) VALUES ('usd', DATE'2016-10-22', 55.11);
INSERT INTO prices( ticker, pdate, price) VALUES ('usd', DATE'2016-10-22', 56.11);
INSERT INTO prices( ticker, pdate, price) VALUES ('usd', DATE'2016-10-25', 57.11);
INSERT INTO prices( ticker, pdate, price) VALUES ('usd', DATE'2016-10-25', 57.22);
INSERT INTO prices( ticker, pdate, price) VALUES ('usd', DATE'2016-10-27', 58.11);
INSERT INTO prices( ticker, pdate, price) VALUES ('usd', DATE'2016-11-29', 57.11);

INSERT INTO prices( ticker, pdate, price) VALUES ('eur', DATE'2016-11-22', 65.11);
INSERT INTO prices( ticker, pdate, price) VALUES ('eur', DATE'2016-11-22', 65.12);
INSERT INTO prices( ticker, pdate, price) VALUES ('eur', DATE'2016-11-25', 67.11);
INSERT INTO prices( ticker, pdate, price) VALUES ('eur', DATE'2016-11-27', 68.11);

INSERT INTO prices( ticker, pdate, price) VALUES ('chf', DATE'2016-11-22', 88.11);
INSERT INTO prices( ticker, pdate, price) VALUES ('chf', DATE'2016-11-25', 88.33);
INSERT INTO prices( ticker, pdate, price) VALUES ('chf', DATE'2016-11-25', 89.33);
```

Обратите внимание: за каждую дату может быть несколько разных курсов одной и той же валюты. Напишем запрос, который бы выбирал минимальное и максимальное значения курса каждой валюты за наибольшую дату и за наименьшую, наиболее отдаленную во времени, дату.

```
SELECT DISTINCT ticker
, (SELECT MIN(price) FROM prices p1 WHERE p1.pdate  =
   (SELECT MIN(p3.pdate) FROM prices p3 WHERE p3.ticker = P.ticker) AND p1.ticker =
P.ticker) minfirstprice
, (SELECT MAX(price) FROM prices p2 WHERE p2.pdate =
   (SELECT MAX(p3.pdate) FROM prices p3 WHERE p3.ticker = P.ticker) AND p2.ticker =
P.ticker) maxlastprice
FROM prices P ;
```

Минимальное и максимальное значения курса за наибольшую дату.
Получилось довольно сложно и громоздко, но результат правильный.

```
chf     88,11   88,11   88,33   89,33   88,11   177,66
eur     65,11   65,12   68,11   68,11   130,23  68,11
usd     55,11   56,11   57,11   57,11   111,22  57,11
```

А теперь используем KEEP FIRST, KEEP LAST. Посмотрите, насколько улучшился наш запрос.

```
SELECT
 ticker,
 min(price) KEEP (DENSE_RANK FIRST ORDER BY pdate) as minfirstprice,
 max(price) KEEP (DENSE_RANK FIRST ORDER BY pdate) as maxfirstprice,
 min(price) KEEP (DENSE_RANK LAST ORDER BY pdate) as minlastprice,
 max(price) KEEP (DENSE_RANK LAST ORDER BY pdate) as maxlastprice
FROM prices GROUP BY ticker ORDER BY ticker;
```

Немного подробнее о том, что мы здесь написали.

Функция SUM, MAX, MIN находит сумму, максимальное и минимальное значения из курсов валют за последнюю и первую дату в группе ticker.

В конструкции KEEP FIRST, KEEP LAST мы можем использовать следующие агрегатные функции: **MIN, MAX, SUM, AVG, COUNT, VARIANCE, STDDEV**.

Конструкция KEEP (DENSE_RANK LAST ORDER BY PDATE) означает, что мы осуществляем работу с последними LAST значениями, с сортировкой по полю PDATE; в свою очередь, DENSE_RANK FIRST ORDER BY PDATE вернет агрегатной функции MAX (PRICE) первые значения PRICE, отсортированные по PDATE.

Важные замечания

Конструкцию можно использовать совместно с аналитическими функциями.

Использование KEEP FIRST LAST вместе с аналитической функцией.

Аналитические функции, которые могут применяться: **MIN, MAX, SUM, AVG, COUNT, VARIANCE, STDDEV**.

```
SELECT  ticker,  price,
 min(price) KEEP (DENSE_RANK FIRST ORDER BY pdate) OVER (PARTITION BY ticker)
minfirstprice,
 mAX(price) KEEP (DENSE_RANK FIRST ORDER BY pdate) OVER (PARTITION BY ticker)
mAXfirstprice,
 max(price) KEEP (DENSE_RANK LAST ORDER BY pdate) OVER (PARTITION BY ticker)
minlasttprice,
 mIN(price) KEEP (DENSE_RANK LAST ORDER BY pdate) OVER (PARTITION BY ticker)
maxlastprice
FROM prices
ORDER BY PRICE;..
```

Более подробно про использование аналитических функций можно посмотреть на предыдущих двух шагах.

Вопросы учеников

Можно ли конструкцию KEEP заменить простыми запросами?

Да, можно построить запрос с использованием DECODE, CASE.

Можно ли применить конструкцию KEEP совместно с аналитическими функциями?

Да, с аналитическими функциями можно использовать конструкцию FIRST LAST.

Контрольные вопросы и задания для самостоятельного выполнения

1. Для автомобилей из таблицы AUTO из нашей учебной схемы найдите цвет авто с максимальной или минимальной датой выпуска, используйте KEEP FIRST LAST.

2. Для городов CITY из нашей учебной схемы найдите название города с максимальным количеством населения, используйте KEEP FIRST LAST.

Шаг 63. Конструкция WITH

Введение

Для повышения наглядности SQL и читаемости запросов SELECT, для удобства разработки начиная с версии 9 в SQL диалекта ORACLE добавлен специальный оператор WITH.

Теория и практика

Оператор WITH позволяет заранее формировать внутренний подзапрос и далее обращаться к данному подзапросу по синониму в основном запросе.

Очень часто используется совместно с XML Type и XML SEQ, а также в сложных запросах с множеством реляционных отношений.

Синтаксис
Простой WITH

```
With t1 псевдоним подзапроса
as ( select ... -- внутренни подзапрос
) select перечень полей или * from t1 -- обращение в внтреннему подзапросу
```

Здесь t1 – псевдоним запроса, к которому идет обращение в основной команде SELECT.

Сложный WITH

```
WITH
T1 as (SELECT field_list FROM T list join WHERE cond group by ...) ,
T2 as (SELECT field_list FROM T list join WHERE cond2 group by ..) , tn as ....
SELECT * FROM T1, T2 where t1.cond= t2.cond
```

Примеры
Простой оператор WITH:

```
WITH T AS (SELECT CITYCODE FROM CITY) , T2 AS
(SELECT CITYNAME FROM CITY A , T WHERE T.CITYCODE = A.CITYCODE)
SELECT CITYNAME FROM T2
```

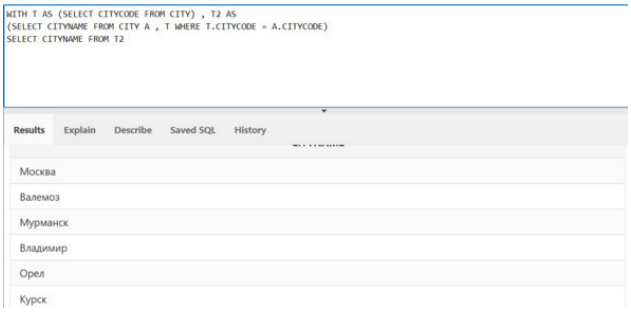
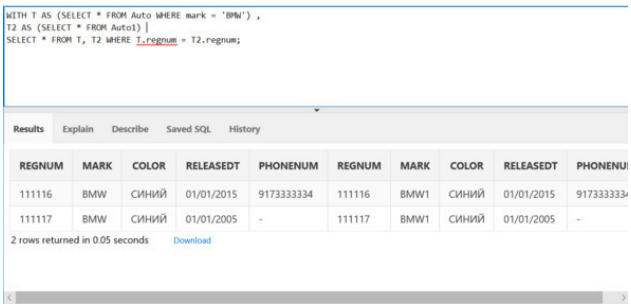


Рисунок 178. Запрос к CITY: использование WITH

Сложное обращение

```
WITH T AS (SELECT * FROM Auto WHERE mark = 'BMW') ,
T2 AS (SELECT * FROM Auto1)
SELECT * FROM T, T2 WHERE T.regnum = T2.regnum;
```



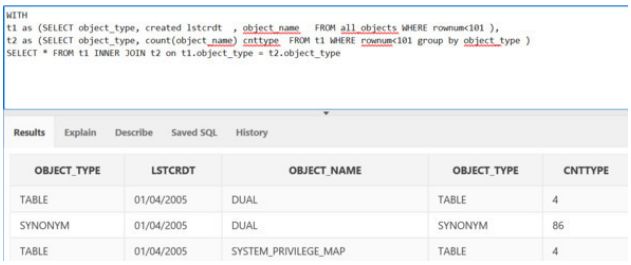
REGNUM	MARK	COLOR	RELEASEDT	PHONENUM	REGNUM	MARK	COLOR	RELEASEDT	PHONENU
111116	BMW	СИНИЙ	01/01/2015	9173333334	111116	BMW1	СИНИЙ	01/01/2015	917333333
111117	BMW	СИНИЙ	01/01/2005	-	111117	BMW1	СИНИЙ	01/01/2005	-

2 rows returned in 0.05 seconds Download

Рисунок 179. Запрос к AUTO1: использование WITH

Два запроса из WITH

```
WITH
t1 as (SELECT object_type, created lstcrdt  , object_name FROM all_objects),
t2 as (SELECT object_type, count(object_name) cnttype FROM t1 group by object_type)
SELECT * FROM t1 INNER JOIN t2 on t1.object_type = t2.object_type
```



OBJECT_TYPE	LSTCRDT	OBJECT_NAME	OBJECT_TYPE	CNTTYPE
TABLE	01/04/2005	DUAL	TABLE	4
SYNONYM	01/04/2005	DUAL	SYNONYM	86
TABLE	01/04/2005	SYSTEM_PRIVILEGE_MAP	TABLE	4

Рисунок 180. Запрос к ALL_OBJECTS: использование WITH

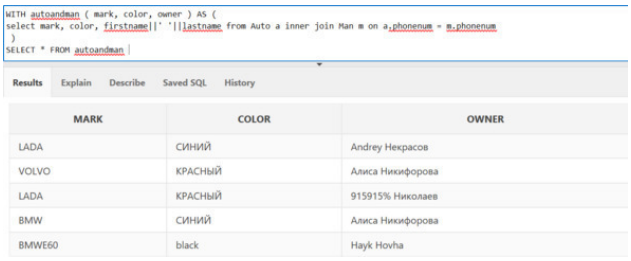
В этом запросе используется системное представление ALL_OBJECTS, с которым мы познакомимся в следующих шагах.

Важные замечания

Конструкция WITH может также быть записана следующим образом:

```
WITH autoandman ( mark, color, owner ) AS (
select mark, color, firstname||' '||lastname from Auto a inner join Man m on a.phonenum =
m.phonenum
 )
SELECT * FROM autoandman
```

Обратите внимание, что колонка owner является составной.



Results | Explain | Describe | Saved SQL | History

MARK	COLOR	OWNER
LADA	СИНИЙ	Andrey Некрасов
VOLVO	КРАСНЫЙ	Алиса Никифорова
LADA	КРАСНЫЙ	915915% Николаев
BMW	СИНИЙ	Алиса Никифорова
BMWE60	black	Hayk Hovha

Конструкция WITH может быть использована для построения рекурсивных запросов, это важное свойство WITH, которое достаточно часто используется на практике.

```
WITH
    n1(n) AS (
      SELECT 1 as n FROM dual UNION ALL
      SELECT 2 as n FROM dual UNION ALL
      SELECT 3 as n FROM dual
    )
   , n2(n) AS (
      SELECT n FROM n1 UNION ALL
      SELECT n + 1 AS n
      FROM   n2
     WHERE n < 5
    )  SELECT* FROM n2
```

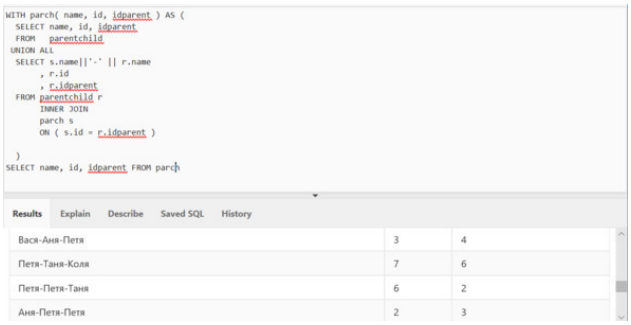
Results | Explain | Describe | Saved SQL | History

1
2
3
2
3
4

Второй пример:

```
WITH parch( name, id, idparent ) AS (
 SELECT name, id, idparent
 FROM   parentchild
UNION ALL
 SELECT s.name||'-' || r.name
     , r.id
     , r.idparent
 FROM parentchild r
     INNER JOIN
     parch s
     ON ( s.id = r.idparent )

 )
SELECT name, id, idparent FROM parch
```



Вопросы учеников

Можно ли использовать конструкцию WITH с XML?

Да, конструкция WITH с XML работает с разбором XML-документов. Пример подобного запроса мы разберем далее, когда будем изучать работу с XML.

Можно ли использовать конструкцию WITH с CONNECT BY в иерархических запросах?

Вот пример подобного запроса:

```
With t1(lev,name,path) as (
SELECT LEVEL, lpad('_', 4*LEVEL,'_')||name  as nm, SYS_CONNECT_BY_PATH(name, '/')FROM
parentchild START WITH idparent is null CONNECT BY  PRIOR id=idparent )
SELECT * FROM t1
```

Может ли быть использован WITH совместно с UNION?

Да, например таким образом:

```
WITH
 n1 as (
SELECT * FROM auto Union SELECT * FROM auto1
)
SELECT * FROM auto1
```

Контрольные вопросы и задания для самостоятельного выполнения

1. Повторите материалы данного шага.
2. Поясните назначение оператора WITH.
3. Поясните, как используется WITH в иерархических запросах.

Шаг 64. Конструкция With и функции

Введение

В SQL диалекте ORACLE 12C есть возможность определить функцию или процедуру на языке PL/SQL и использовать с помощью оператора WITH, как обычный SQL.

Теория и практика

Специальная инструкция WITH с FUNCTION используется, когда необходимо вернуть данные, преобразованные с помощью сложного нелинейного алгоритма.

Синтаксис

```
WITH
 PROCEDURE <NAME_PROCEDURE>
 BEGIN
 ...
 END;

 FUNCTION <NAME_FUNCTION>
 BEGIN
  ...
 END;
SELECT <NAME_FUNCTION>
FROM <TABLE>;
```

Примеры

Вывести на экран марки автомобиля из таблицы AUTO.

```
WITH
 FUNCTION reversive_fnc(p_name VARCHAR2) RETURN VARCHAR2
 is i NUMBER; v VARCHAR2(50);
 begin
  FOR i IN 1..LENGTH(p_name) LOOP
   v := v || SUBSTR(p_name, LENGTH(p_name)-i+1, 1);
  END LOOP;
  return v;
 end;
SELECT DISTINCT reversive_fnc(mark) as rname, mark FROM auto
```

Используемая функция REVERSIVE_FNC выводит слово справа налево по буквам.

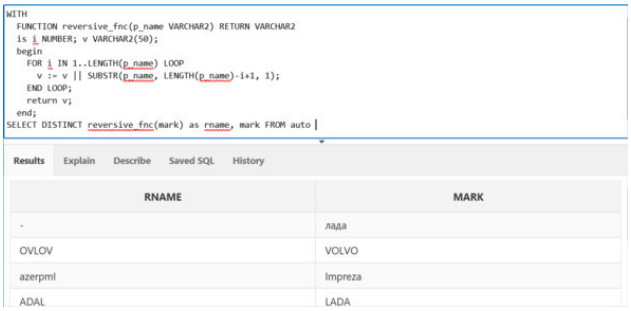


Рисунок 181. Использование WITH и FUNCTION

Еще пример. Добавить к идентификатору города заданное количество нулей, преобразовать к числу.

```
WITH
 FUNCTION incid_fnc(p_id NUMBER, p_count NUMBER) RETURN NUMBER
 is
 begin
  return TO_NUMBER(rpad(p_id , p_count, '0'));
 end;
SELECT citycode, incid_fnc(peoples, 10) fn FROM city
```

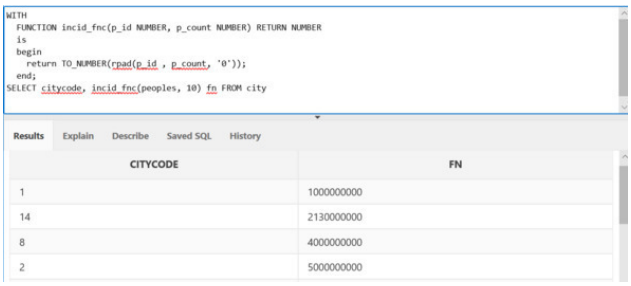


Рисунок 182. Использование WITH и FUNCTION

Важные замечания

Подобное использование WITH, подобный механизм используется в СУБД ORACLE начиная с версии 12C.

В функциях WITH можно использовать динамический SQL и курсоры, однако нельзя использовать команды модификации данных.

Вопросы учеников

Можно ли использовать в WITH не только функции, но и процедуры?

Такой способ есть, но напрямую использовать процедуры все же нельзя.

Пример, как это сделать:

```
WITH
 procedure prcplus(val number, out_val out number)
 is
 begin
  out_val := val + 1;
 end;

 function mult(val number)
 return number
 is
  retval number;
 begin
  prcplus(val, retval);
  return retval * 2;
 end;
select mult(citycode), citycode from city;
```

MULT(CITYCODE)	CITYCODE
4	1
6	2
8	3
10	4

В этом примере из функции MULT вызывается процедура prcplus, а сама функция MULT используется в запросе.

Можно ли использовать WITH совместно с командами обновления данных UPDATE?

```
update /*+ WITH_PLSQL */ man1
set yearold= (with
        function plus(val number) return number
        is
        begin
          return val + 1;
        end;
       select plus(yearold) from dual);
```

В данном запросе обновляется возраст людей из MAN1 (+1) с использованием функции PLUS ().

Контрольные вопросы и задания для самостоятельного выполнения

1. Повторить запросы из данного шага.
2. Написать WITH запрос-функцию, вычисляющую количество авто каждого цвета из AUTO, вывести результат: марку машины, количество машин данной марки.
3. Написать WITH запрос-функцию, добавляющую к цвету авто из таблицы AUTO слово «цвет». Результат: BMW цвет белый.
4. Написать WITH запрос-функцию, умножающую PEOPLES на 1000 из таблицы CITY. Результат: Москва 100 000.
5. Написать запрос-функцию из таблицы MAN, объединяющую имя и фамилию.

Шаг 65. Группировки с DECODE и CASE

Введение

Существует интересный способ сгруппировать данные, используя операторы DECODE или CASE. Такой способ применим, когда в одной таблице множество разнообразных данных и нам необходимо собрать статистическую информацию по каждому из значений.

Теория и практика

Простой способ сгруппировать данные в одном запросе.
В ORACLE существует системное представление ALL_OBJECTS.

Необходимо посчитать количество таблиц, индексов, представлений для каждого владельца и общее количество объектов каждого владельца из представления.

Напишем запрос:

```
SELECT owner,
               count(object_name) object_name,
      count(decode(object_type, 'TABLE', 1, null)) tablecount,
      count(decode(object_type, 'INDEX', 1, null)) idxcount,
      count(decode(object_type, 'VIEW', 1, null)) viewcount
  FROM all_objects WHERE object_type in ('TABLE','INDEX','VIEW') AND rownum<100
```

GROUP BY OWNER:

```
SELECT owner,
       count(object_name) object_name,
       count(decode(object_type, 'TABLE', 1, null)) tablecount,
       count(decode(object_type, 'INDEX', 1, null)) idxcount,
       count(decode(object_type, 'VIEW', 1, null)) viewcount
  FROM all_objects WHERE object_type in ('TABLE','INDEX','VIEW') AND rownum<100
 GROUP BY owner
```

Results | Explain | Describe | Saved SQL | History

ICCMS	14	7	7	0
CIS411W_SQL	2	1	1	0
ATDSCHEMA	2	1	1	0
SICOM	2	1	1	0
LTCAMSTELLAND	1	0	1	0
SYS	74	1	0	73

Рисунок 183. Запрос к ALL_OBJECTS: использование для группировок

В одном запросе мы подсчитали количество таблиц, индексов и количество представлений, а также количество других объектов.

Обратимся к нашей схеме.
Посчитаем количество авто BMW, LADA и прочих одним запросом.

```
SELECT  count(1) as all_auto,
      count(decode(mark, 'BMW', 1, null)) as bmwcount,
      count(decode(mark, 'LADA', 1, null)) as ladacount,
      count(decode(mark, 'BMW', null, 'LADA', null , 1)) as othercount
  FROM AUTO
```

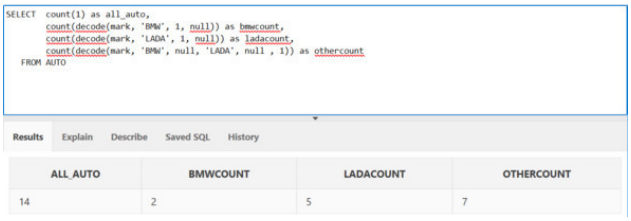


Рисунок 184. Запрос к AUTO: использование для группировок

Важные замечания

В подобных запросах следует учитывать пустые значения ячеек таблицы.

Вопросы учеников

Можно ли использовать другие агрегатные функции, например SUM?

Вот пример: переделаем запрос с использованием агрегатной функции SUM.

```
SELECT
    sum(decode(mark, 'BMW', 1, 0)) as bmwcount,
    sum(decode(mark, 'LADA', 1, 0)) as ladacount,
    sum(decode(mark, 'BMW', 0, 'LADA', 0 , 1)) as othercount
 FROM AUTO
```

Контрольные вопросы и задания для самостоятельного выполнения

1. Повторите материалы данного шага.

2. Напишите запрос, который бы подсчитал количество авто синего, красного и остальных цветов из таблицы AUTO. Используйте DECODE.

3. Напишите запрос, который бы подсчитал людей, которые купили авто, которые не купили авто – в другой колонке, из таблицы MAN. Используйте DECODE.

День четырнадцатый

Шаг 66. Преобразуем запрос в строчку LISTAGG

Введение

В ORACLE SQL есть возможность преобразовать результат запроса в CSV-строку или в любую другую заданную строку с разделителем.

Теория и практика

Функция LISTAGG объединяет значения заданного поля таблицы для каждой группы в строку, через указанный разделитель.

Синтаксис

```
SELECT LISTAGG (поле [, 'разделитель'])
 WITHIN GROUP (order_by_clause сортировка) [OVER partition]
```

Здесь ORDER_BY_CLAUSE – сортировка, колонка, по которой будет отсортирован результат;

OVER PART – сегмент разделения данных.

Разберем на примере – создадим таблицу и заполним ее данными.

```
CREATE TABLE cl_phones(tname VARCHAR2(100), phone VARCHAR2(30));
insert into cl_phones(tname,phone) values ('Андрей','867-843-25');
insert into cl_phones(tname,phone) values ('Андрей','830-044-35');
insert into cl_phones(tname,phone) values ('Андрей','530-055-35');
insert into cl_phones(tname,phone) values ('Vasya','110-044-35');
insert into cl_phones(tname,phone) values ('Vasya','220-055-35');
insert into cl_phones(tname,phone) values ('Mark','930-044-35');
insert into cl_phones(tname,phone) values ('Mark','930-055-35');
```

Запрос для примера:

```
 SELECT tname, LISTAGG(phone, ' ; ') WITHIN GROUP (ORDER BY phone) phonestr FROM
cl_phones GROUP BY tname;
```

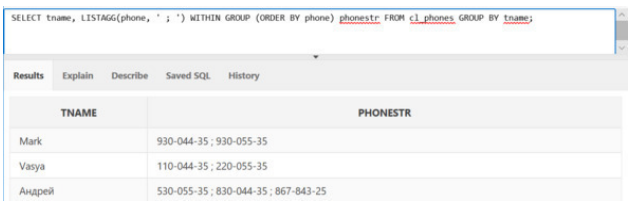


Рисунок 185. Запрос к CL_PHONES: использование LISTAGG

```
DROP TABLE cl_phones;
```

Второй пример.
Создадим таблицу по следующему запросу:

```
CREATE TABLE demo1 AS
SELECT object_name , object_type  FROM all_objects where  object_type = 'TABLE' and
rownum < 15
UNION
SELECT object_name , object_type  FROM all_objects where  object_type = 'INDEX' and
rownum < 15
UNION
SELECT object_name , object_type  FROM all_objects where  object_type = 'VIEW' and
rownum < 15;
```

Запрос

```
SELECT object_type, LISTAGG(object_name, ' ; ') WITHIN GROUP (ORDER BY object_name)
objstr FROM demo1 GROUP BY object_type ;
```

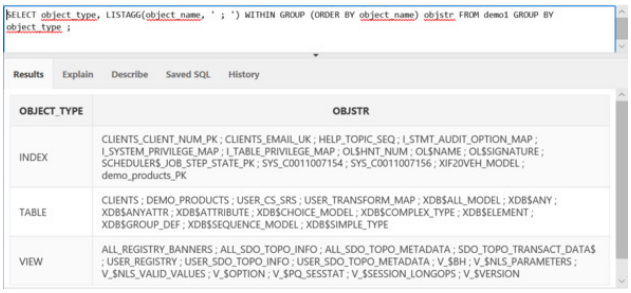
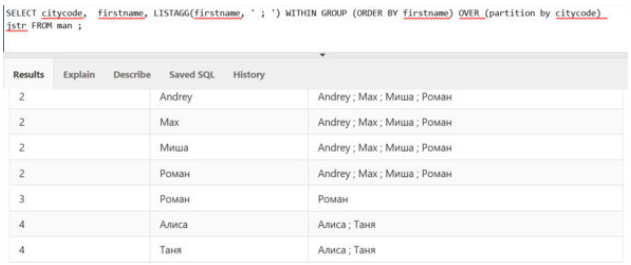


Рисунок 186. Запрос к DEMO1: использование LISTAGG

Важные замечания

Также LISTAGG может работать как аналитическая функция, то есть объединять данные в строку по заданному сегменту (PARTITION).

```
SELECT citycode,  firstname, LISTAGG(firstname, ' ; ') WITHIN GROUP (ORDER BY firstname)
OVER (partition by citycode) jstr FROM man ;
```



```
SELECT citycode,  firstname, LISTAGG(firstname, ' ; ') WITHIN GROUP (ORDER BY firstname) OVER (partition by citycode)
jstr FROM man ;
```

Results | Explain | Describe | Saved SQL | History

2	Andrey	Andrey ; Max ; Миша ; Роман
2	Max	Andrey ; Max ; Миша ; Роман
2	Миша	Andrey ; Max ; Миша ; Роман
2	Роман	Andrey ; Max ; Миша ; Роман
3	Роман	Роман
4	Алиса	Алиса ; Таня
4	Таня	Алиса ; Таня

Рисунок 187. Запрос к MAN: использование LISTAGG с PARTITION

Вопросы учеников

Как использовать функцию LISTAGG совместно с UNION?

Да, такое возможно. Вот пример:

```
SELECT firstname, LISTAGG(lastname, ' ; ')
WITHIN GROUP (ORDER BY lastname) objstr FROM (select * from man union select * from
man1 ) GROUP BY firstname;
```

FIRSTNAME	OBJSTR
915915%	Москитов ; Москитов ; Николаев ; Николаев
Andrey	Maksimov ; Maksimov ; Некрасов ; Некрасов

Контрольные вопросы и задания для самостоятельного выполнения

1. Повторите материалы данного шага.
2. Выведите из таблицы MAN первую букву имени человека (FIRSTNAME) и через точку с запятой все имена (FIRSTNAME), начинающиеся с этой буквы.
3. Выведите из таблицы CITY первую букву города (CITYNAME) и через точку с запятой все города (CITYNAME), начинающиеся с этой буквы.

Шаг 67. Работаем с JSON

Введение

В ORACLE существует возможность работать с JSON-форматом. JSON – специальный формат обмена данными, основанный на текстовом формате. Как и многие другие текстовые форматы, JSON используется при обмене данными между веб-сервисами.

Теория и практика

Проще всего пояснить синтаксис работы JSON в ORACLE на примерах.
Создадим таблицу INFO_USER_V:

```
CREATE TABLE info_user_v (
          id NUMBER primary key -- идентификатор уникальный
        , name VARCHAR2(50) -- наименование
        , json_data VARCHAR2(4000) -- данные JSON
        );
```

В ORACLE 12 существует специальный тип ограничений, чтобы добавлять в колонку только данные JSON-формата и исключить ошибки, связанные с нарушением структуры JSON.
Добавим такую колонку к нашей таблице:

```
ALTER TABLE info_user_v ADD CONSTRAINT c_1_json_data CHECK(json_data is json);
```

Добавляем записи:

```
INSERT INTO info_user_v(id, name, json_data)
VALUES
(1, 'item1', '{
"title":"book",
"name":"Island",
"autor":{
 "firstname":"Robert",
 "lastname":"Stivenson"
 }
}');

INSERT INTO info_user_v (id, name, json_data)
VALUES
(2, 'item2', '{
"title":"new book",
"name":"Game of trones",
"autor":{
 "firstname":"Robert",
 "lastname":"BArateon"
 }
}');
```

Попробуем извлечь данные из таблиц, которые мы создали.

```
SELECT id, json_value(json_data,'$.title') FROM info_user_v
```

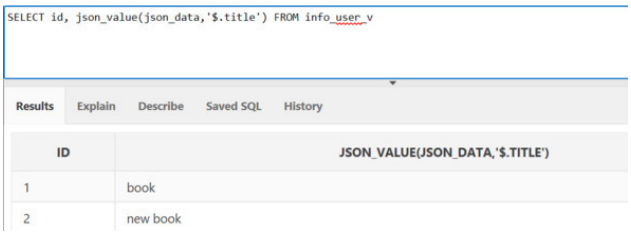


Рисунок 188. Запрос с использованием JSON_VALUE, JSON_DATA

```
SELECT id, json_value(json_data,'$.autor.firstname')
, json_value(json_data,'$.autor.lastname') FROM info_user_v WHERE
json_value(json_data,'$.title') like 'bo%'
```

Рисунок 189. Запрос с использованием JSON_VALUE, JSON_DATA

Еще один способ:

```
SELECT id,tt.*  FROM info_user_v
, JSON_TABLE(json_data,'$.autor'  COLUMNS(firstname varchar2(50 char) path '$.firstname'))
tt;
```

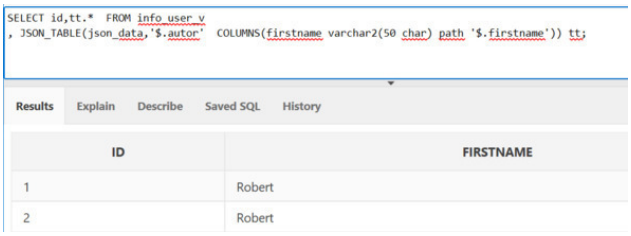


Рисунок 190. Запрос с использованием JSON_TABLE

```
SELECT id,tt.*  FROM info_user_v
, JSON_TABLE(json_data,'$'  COLUMNS(title varchar2(50 char) path '$.title')) tt where tt.title =
'book'
```

Рисунок 191. Запрос с использованием JSON_TABLE, JSON_DATA

Важные замечания

Работа с JSON возможна только в версии ORACLE начиная с 12c.

Вопросы учеников

Можно ли создать представление на основание таблицы json?

Пример создания представления:

```
CREATE OR REPLACE VIEW json_view AS
SELECT ov.a, ov.b, ov.c, data.* FROM old_view ov,
   JSON_TABLE
   (
    ov.my_column format json,
    '$'
    columns
    something VARCHAR2(50 CHAR) format json path '$.x.y.z'
   )
as jsondata;
```

Контрольные вопросы и задания для самостоятельного выполнения

1. Повторите запросы из данного шага.
2. Разберитесь, как работает JSON в ORACLE 12C.
3. Создайте таблицу, где в одной из колонок будут храниться данные формата JSON, вторая колонка – идентификатор.
4. Создайте для данного поля ограничение на корректность формата JSON.
5. Добавьте в колонку JSON данные JSON,»
{
«num»: «123324324»,
«NAME»: «Max»,
«property»: {«COLOR»: «red», «MARK»: «bmw»
– }
6. Извлеките цвет и марку из колонки JSON с помощью SQL-запроса.

Шаг 68. Высший пилотаж SQL. MODEL

Введение

Оператор MODEL в ORACLE SQL позволяет определенным образом эмулировать работу с электронными таблицами. Оператор MODEL имеет сложный синтаксис, но вместе с тем предоставляет более гибкие способы работы с данными.

Теория и практика

Оператор SQL MODEL позволяет рассматривать результат запроса как многомерный массив.

При этом в SQL задаем оси измерения этого массива (идентифицируем данные по осям).

Использование MODEL также позволит нам подводить промежуточные и общие итоги с применением агрегатных функций.

Синтаксис

```
SELECT *FROM table1 -- таблица или запрос
MODEL DIMENSION BY (field1 , field2, ..)--оси, определение осей измерений по которым
мы строим массив (поля для поиска уникальной ячейки)
   MEASURES (field3) -- определяющее поле
   RULES (    cnt['res1', 'res2'] = res3 -- результат который вносится массив
        )  ORDER BY field1; -- сортировка по полю
```

Примеры

Подготовим данные.

Таблица (ручки), где PRT – партия поставки, CNT – количество в данной поставке, COLOR – цвет ручек в поставке (red, green, black).

```
create table pen(prt number, cnt number, color varchar2(15));
```

Заполним таблицу данными (партии 1, 2, 3, 4, 7 с различным количеством ручек, разных цветов).

```
insert into pen(prt,cnt,color) values(1,5,'red');
insert into pen(prt,cnt,color) values(1,5,'black');
insert into pen(prt,cnt,color) values(2,3,'green');
insert into pen(prt,cnt,color) values(2,1,'red');
insert into pen(prt,cnt,color) values(3,1,'red');
insert into pen(prt,cnt,color) values(4,4,'black');
insert into pen(prt,cnt,color) values(7,3,'red');
```

Напишем запрос, демонстрирующий работу MODEL.

```
SELECT * FROM pen
MODEL DIMENSION BY (prt, color) -- измерения оси строим по полям prt, color
   MEASURES (cnt) -- работаем с cnt
   RULES (
      cnt[any, 'red'] = cnt[cv(prt), 'red'] * 10  -- для каждого prt и color = red – cnt в итоговом
запросе умножаем на 10
         )
ORDER BY prt;
```

Results | Explain | Describe | Saved SQL | History

PRT	COLOR	CNT
1	red	50
1	black	5
2	red	10
2	green	3
3	red	10
4	black	4

Рисунок 192. Результаты запроса: использование MODEL

Немного изменим запрос, добавим сумму.

```
SELECT * FROM pen
MODEL DIMENSION BY (prt, color) -- измерения оси строим по полям prt, color
   MEASURES (cnt) -- работаем с cnt
   RULES (
      cnt[NULL, 'SUMM RED'] = sum(cnt)[ANY, 'red'], --  итог, только по red
      cnt[NULL, 'SUMM'] = sum(cnt)[ANY, ANY] -- общий итог
         )
ORDER BY prt;
```

Results | Explain | Describe | Saved SQL | History

PRT	COLOR	CNT
1	black	5
1	red	5
2	red	1
2	green	3
3	red	1

Рисунок 193. Результаты запроса: использование MODEL

Более интересный пример: подводятся итоги по трем партиям ручек 1…3.

```
SELECT * FROM pen
MODEL DIMENSION BY (prt, color) -- измерения оси строим по полям prt, color
   MEASURES (cnt) -- работаем с cnt
   RULES (
      --cnt[any, 'red'] = cnt[cv(prt), 'red'] * 10, -- для каждого prt и color = red - cnt в итговом
запросе умножаем на 10
      cnt[FOR prt FROM 1 TO 3 INCREMENT 1,  'SUMMPART'] = sum(cnt)[cv(prt), ANY], -- итог
по трем партиям
      cnt[NULL, 'SUMM RED'] = sum(cnt)[ANY, 'red'], --  итог, только по red
      cnt[NULL, 'SUMM'] = sum(cnt)[ANY, ANY] -- общий итог
         )
ORDER BY prt;
```

Еще более интересно: дополнительные итоги по цвету ручек!

```
SELECT * FROM pen
MODEL DIMENSION BY (prt, color) -- измерения оси строим по полям prt, color
   MEASURES (cnt) -- работаем с cnt
   RULES (
      --cnt[any, 'red'] = cnt[cv(prt), 'red'] * 10, -- для каждого prt и color = red - cnt в итоговом
запросе умножаем на 10
     -- cnt[FOR prt FROM 1 TO 3 INCREMENT 1,  'SUMMPART'] = sum(cnt)[cv(prt), ANY], -- итог
по трем партиям
      cnt[NULL, FOR color IN (SELECT color FROM pen group by color)] = sum(cnt)[ANY,
cv(color)],  -- ИТОГ ПО ЦВЕТУ РУЧЕК!!!
      cnt[NULL, 'SUMM RED'] = sum(cnt)[ANY, 'red'], --  итог, только по red
      cnt[NULL, 'SUMM'] = sum(cnt)[ANY, ANY] -- общий итог
         )
ORDER BY prt;
```

PRT	COLOR	CNT
1	red	5
1	black	5
2	green	3
2	red	1

Рисунок 194. Результаты запроса: использование MODEL и SUM

Кроме суммы, можно использовать и другие агрегатные функции, например AVG.

```
SELECT * FROM pen
MODEL DIMENSION BY (prt, color) -- измерения оси строим по полям prt, color
   MEASURES (cnt) -- работаем с cnt
   RULES (
      --cnt[any, 'red'] = cnt[cv(prt), 'red'] * 10, -- для каждого prt и color = red - cnt в итговом
запросе умножаем на 10
      -- cnt[FOR prt FROM 1 TO 3 INCREMENT 1,  'SUMMPART'] = sum(cnt)[cv(prt), ANY], -- итог
по трем партиям
      cnt[NULL, FOR color IN (SELECT color FROM pen group by color)] = sum(cnt)[ANY,
cv(color)],  -- ИТОГ ПО ЦВЕТУ РУЧЕК!!!
      cnt[NULL, 'AVG RED'] = avg(cnt)[ANY, 'red']--, --  среднее партия по цвету red
      --cnt[NULL, 'SUMM'] = sum(cnt)[ANY, ANY] -- общий итог
        )
ORDER BY prt;
```

PRT	COLOR	CNT
1	red	5
1	black	5
2	green	3
2	red	1

Рисунок 195. Результаты запроса: использование MODEL

Важные замечания

• Совместно с командой MODEL нельзя использовать предложение PARTITION BY.
• В MODEL подзапрос не может быть соотнесен с внешним запросом.
• При использовании MODEL имена колонок должны быть уникальными.
• Все ссылки на ячейки в MODEL должны быть ссылками на отдельные ячейки.

Вопросы учеников

Можно ли создать представление на основе запроса с MODEL?

Да, в представлениях может использован запрос с MODEL.

Контрольные вопросы и задания для самостоятельного выполнения

1. Повторите материалы данного шага.
2. На основе таблицы AUTO постройте запрос MODEL, взяв за основу запрос из последнего примера.

Шаг 69. MODEL-аналитика, сложные последовательности и массивы

Введение

MODEL также можно использовать для создания двумерных массивов, а также сложных итераторов, сложных последовательностей.

Теория и практика

Мы можем также использовать MODEL для создания массивов.
Создание одномерного массива:

```
SELECT *
FROM dual
  MODEL DIMENSION BY (0 as т1)
  MEASURES (cast(dummy as varchar2(20)) as ct)
  RULES (
    ct[0] = '1',
    ct[1] = '2',
    ct[1] = '3',
    ct[2] = '4',
    ct[3] = '5'
  ) order by 1;
--T1    CT
--0     1
--1     3
--2     4
--3     5
```

Создание двумерного массива:

```
SELECT *
FROM dual
  MODEL DIMENSION BY (0 as nx, 1 as ny)
  MEASURES (cast(dummy as varchar2(20)) as ct)
  RULES (
    ct[0,0] = '111',
    ct[0,1] = '111',
    ct[1,1] = '112',
    ct[2,2] = '211',
    ct[3,3] = '331'
  ) order by 1;
-- NX   NY     CT
--      0      1      111
--      0      0      111
--      1      1      112
--      2      2      211
--      3      3      331
```

Также функция SQL MODEL может использоваться и для создания итераторов подобно CONNECT BY:

- использование итератора;
- простейший пример.

```
select  m1 ,  AA
from
 dual
model dimension by ( 0 as m1 )
 measures    ( cast( 'A ' as varchar2(100) ) as AA )
 rules
  ITERATE (35)
  ( aa[0] = aa[0] || 'A' )
```

– M1 AA
– 0 A AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

Еще пример: итератор в качестве счетчика, но это более сложный пример. Теперь с конструкцией UNTIL наращиваем итерацию, пока aa [0] не будет равно «AAAAAAAA».

```
select
 m1 ,
 AA
from
 dual
model
 dimension by ( 0 as m1 )
 measures    ( cast( 'A' as varchar2(100) ) as AA )
 rules
  ITERATE (35) UNTIL aa[0] = 'AAAAAAAA'
  ( aa[0] = aa[0] || 'A' );
```

Аналогичный пример, но используем команды PREVIOUS () – перед значением и ITERATION_NUMBER – номер итерации.

```
select
 m1 , AA
from  dual model
 dimension by ( 0 as m1 )
 measures    ( cast( 'A' as varchar2(100) ) as AA )
 rules
  ITERATE (15) UNTIL PREVIOUS(AA[0]) = 'AA0A1A2A3' -- пока предыдущее значение AA[0]
  ( aa[0] =aa[0]|| 'A'||ITERATION_NUMBER );
```

Пример простой итерации от 0 до 5:

```
select
 m1 , AA
from  dual model
 dimension by ( 0 as m1 )
 measures     ( cast( 'A' as varchar2(100) ) as AA )
 rules upsert
  ITERATE (5) -- пока предыдущее значение AA[0]
  ( aa[ITERATION_NUMBER] = 'A'||ITERATION_NUMBER *10);
```

Сложная последовательность, в которой следующий номер – произведение двух предыдущих;

• с ограничением результата более 8000.

```
SELECT   m1 , AA
from  dual model
MODEL DIMENSION BY (0 as m1)
   MEASURES (0 as aa)
   RULES
   ITERATE (12) UNTIL (aa[iteration_number] > 8000) (
      aa[iteration_number] =
         CASE iteration_number
            WHEN 0 THEN 1
            WHEN 1 THEN 2
            ELSE aa[iteration_number - 1] * aa[iteration_number-2]
         END );
```

Интереснейший пример с партицированием данных.

Пример с партицированием:

```
SELECT color, iter
FROM (select 'color'||level color from dual connect by level < 4) tab
MODEL
 PARTITION BY ( color )
 DIMENSION by ( 0 as key )
 MEASURES ( cast( null as varchar2(50) ) as iter )
RULES iterate( 10 )
  ( iter[0] = iter[0] || to_char(iteration_number) || ' ' );
```

Важные замечания

• Запрос с MODEL не может быть использован внешним запросом. Однако это может быть запрос с подзапросами, представлениями и т. д.
• Запрос с MODEL не может иметь предложение PARTITION BY.
•Запрос с MODEL не может быть обновлен.

Вопросы учеников

Можно ли использовать запрос MODEL для формирования первичного ключа?

Для формирования первичного ключа необходимо и правильно использовать последовательность SEQUENCE.

Контрольные вопросы и задания для самостоятельного выполнения

1. Внимательно изучите каждый из примеров. Помните, что применение MODEL весьма распространено, очень часто используется при построении кубов в DWH-проектах.
2. Создайте простой итератор с помощью MODEL, от 1 до 100, с шагом 2.

Шаг 70. TIMESTAMP и DATE

Введение

TIMESTAMP – специальный тип данных в СУБД ORACLE, расширяющий возможности типа данных DATE.

TIMESTAMP используется на практике достаточно часто, поэтому данному типу данных посвящен отдельный шаг.

Теория и практика

Тип **TIMESTAMP** представлен в трех вариантах:

• **TIMESTAMP** – год, месяц, часы, минуты, секунды, миллисекунды, сотые доли секунд;

• **TIMESTAMP WITH LOCAL TIME ZONE** – то же, что и предыдущий тип данных, но дополнительно включает еще и следующие значения временных зон: TIME_ZONE_HOUR и TIME_ZONE_MINUTE, или же TIME_ZONE_REGION;

• **TIMESTAMP WITH LOCAL TIME ZONE** – тип данных со значением часового пояса.

Данные типа **TIMESTAMP**.

Для работы с типом TIMESTAMP существует специальная функция **CURRENT_TIMESTAMP**, которая возвращает текущее значение время-дата с данными часового пояса, заданного в параметрах сессии.

У данной функции нет параметров.

Пример использования:

```
ALTER SESSION SET TIME_ZONE = '-3:0';
select CURRENT_TIMESTAMP from dual;
```

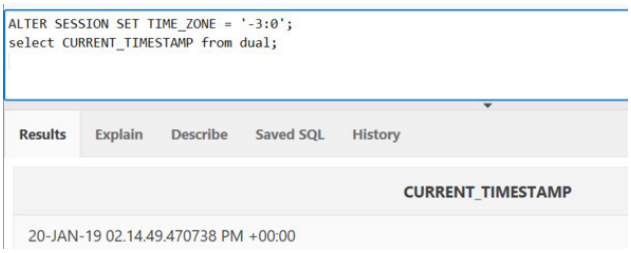


Рисунок 196. Использование TIMESTAMP

```
ALTER SESSION SET TIME_ZONE = '-4:0';
select CURRENT_TIMESTAMP from dual;
```

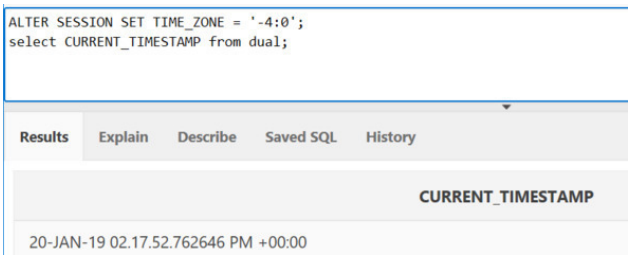


Рисунок 197. Использование TIMESTAMP WITH TIMEZONE

Смещение часового пояса сессии изменилось с -5:0 в -1:0, в результате чего функция CURRENT_TIMESTAMP вернет в качестве значения текущей даты и времени на 4 часа в прошлое.

Рассмотрим примеры преобразования **TIMESTAMP** в DATE.

```
SELECT systimestamp, cast(systimestamp as date) FROM dual
```

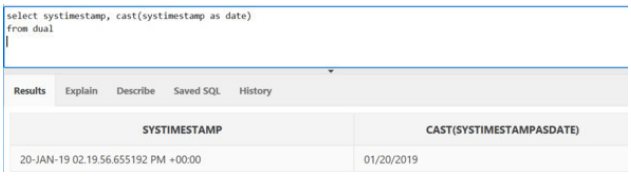


Рисунок 198. Использование TIMESTAMP и DATE

Обратное преобразование из DATE в TIMESTAMP:

```
SELECT CAST(sysdate AS TIMESTAMP) tsdate FROM dual;
```

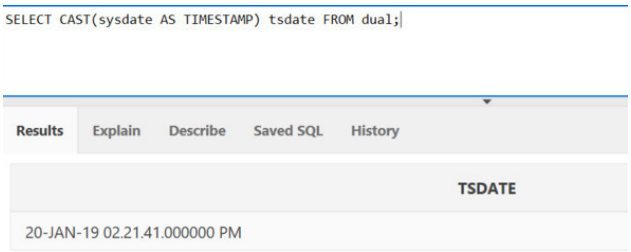


Рисунок 198. Использование TIMESTAMP и DATE

Более сложные преобразования типа **TIMESTAMP**.
Примеры более сложных преобразований с типом **TIMESTAMP**.
TO_TIMESTAMP – преобразование строки к типу **TIMESTAMP**.

```
SELECT TO_TIMESTAMP('2011-05-13 07:15:31.1234', 'YYYY-MM-DD HH24:MI:SS.FF')  FROM
dual;
```

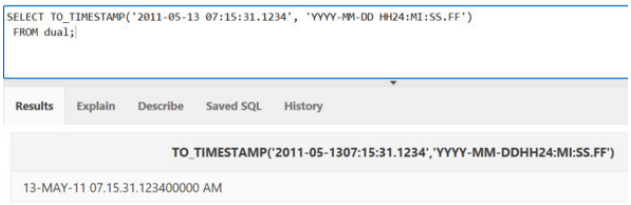


Рисунок 198. Использование TIMESTAMP и DATE

Преобразование временной зоны **NEW_TIME**:

```
SELECT TO_CHAR(NEW_TIME(TO_DATE('25-12-2016 19:45',
   'DD-MM-YYYY HH24:MI'), 'PST', 'EST'), 'DD-MM-YYYY HH24:MI')  FROM dual;
```

Рисунок 200. Использование NEW_TIME

Здесь первый параметр – TIMESTAMP, второй – значение временной зоны, из которой преобразуем, третий – значение временной зоны, в которую преобразуем.

См. таблицу:

AST – ATLANTIC STANDARD TIME
ADT – ATLANTIC DAYLIGHT SAVING TIME
BST – BERING STANDARD TIME
BDT – BERING DAYLIGHT SAVING TIME
CST – CENTRAL STANDARD TIME
CDT – CENTRAL DAYLIGHT SAVING TIME
EST – EASTERN STANDARD TIME
EDT – EASTERN DAYLIGHT SAVING TIME
GMT – GREENWICH MEAN TIME (DATE LINE!)
HST – ALASKA-HAWAII STANDARD TIME
HDT – ALASKA-HAWAII DAYLIGHT SAVING TIME
MST – MOUNTAIN STANDARD TIME
MDT – MOUNTAIN DAYLIGHT SAVING TIME
NST – NEWFOUNDLAND STANDARD TIME
PST – PACIFIC STANDARD TIME
PDT – PACIFIC DAYLIGHT SAVING TIME
YST – YUKON STANDARD TIME
YDT – YUKON DAYLIGHT SAVING TIM

Функция EXTRACT позволяет извлечь следующие значения из **TIMESTAMP**:

MONTH – месяц;
YEAR – год;
MINUTE – минута;
SECOND – секунда;
TIMEZONE_HOUR – час часовой зоны;
TIMEZONE_MINUTE – минута часовой зоны;
TIMEZONE_REGION – регион.

Пример извлечения TIMEZONE_HOUR:

```
SELECT
   EXTRACT(TIMEZONE_HOUR FROM TO_TIMESTAMP_TZ(
    '01-11-2005 19:15:26 -7:15', 'DD-MM-YYYY HH24:MI:SS TZH:TZM'))
    AS TZH   FROM dual;
```

Рисунок 201. Использование TO_CHAR и NEW_TIME

```
SELECT
   EXTRACT(hour FROM TO_TIMESTAMP_TZ(
    '01-01-2005 19:15:26 -7:15', 'DD-MM-YYYY HH24:MI:SS TZH:TZM'))
    AS hour1   FROM dual;
```

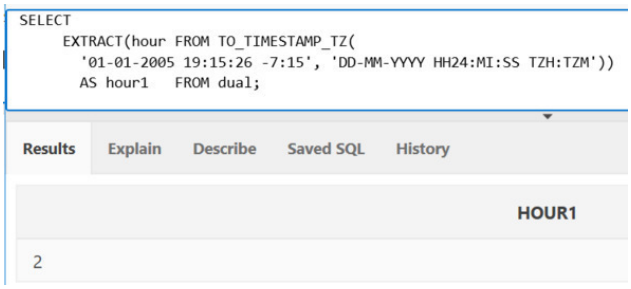


Рисунок 202. Использование EXTRACT с TIMESTAMP

Важные замечания

При преобразовании из текстового значения в **TIMESTAMP** необходимо правильно задавать формат преобразования.

Функция EXTRACT может быть применена не только к колонкам типа данных TIMESTAMP, но и к обычному типу DATE.

Вопросы учеников

Как прибавить или отнять заданный интервал времени из **TIMESTAMP**?

Для этого можно воспользоваться специальной директивой INTERVAL. Как она работает, лучше разобрать на примерах.

```
SELECT TO_TIMESTAMP('2011-05-13 07:15:31.1234', 'YYYY-MM-DD HH24:MI:SS.FF')  + interval
'24' HOUR  FROM dual;
```

Прибавляем интервал 24 часа к заданной дате-времени.

```
SELECT TO_TIMESTAMP('2011-05-13 07:15:31.1234', 'YYYY-MM-DD HH24:MI:SS.FF')  + interval
'24' HOUR FROM dual;
```

Прибавляем интервал 54 минуты к заданной дате-времени.

```
SELECT TO_TIMESTAMP('2011-05-13 07:15:31.1234', 'YYYY-MM-DD HH24:MI:SS.FF')  + interval
'54' minute as nw FROM dual;
```

Прибавляем интервал 3 дня к заданной дате-времени.

```
SELECT TO_TIMESTAMP('2011-05-13 07:15:31.1234', 'YYYY-MM-DD HH24:MI:SS.FF')  + interval
'3' day as nw FROM dual;
```

Прибавляем 30 секунд к заданной дате-времени.

```
SELECT TO_TIMESTAMP('2011-05-13 07:15:31.1234', 'YYYY-MM-DD HH24:MI:SS.FF') + interval
'30' second as nw FROM dual;
```

Прибавляем 3 месяца к заданной дате-времени.

```
SELECT TO_TIMESTAMP('2011-05-13 07:15:31.1234', 'YYYY-MM-DD HH24:MI:SS.FF') + interval
'3' month as nw FROM dual;
```

Необходимо ли всегда использовать тип TIMESTAMP или же можно использовать тип DATE в некоторых задачах?

Тип TIMESTAMP предоставляет значительно больше возможностей, чем DATE, но если вам достаточно DATE для решения ваших задач, то используйте DATE.

Контрольные вопросы и задания для самостоятельного выполнения

1. Объясните принципиальные отличия между типами данных TImeStamp и DATE.
2. Создайте таблицу из двух полей: первое поле – идентификатор, второе – TIMESTAMP (ID, TMS).
3. Добавьте в таблицу запись с идентификатором 1 и текущей датой-временем.
4. С помощью запроса извлеките из таблицы идентификатор «дата-время», выберите час, минуту, секунду из текущего значения «дата-время».

День пятнадцатый

Шаг 71. Фрагментация таблиц, секционирование

Введение

ORACLE поддерживает секционирование таблиц. Секционирование позволяет разделить большую таблицу на более маленькие части по определенному логическому принципу.

Примечание.
Для данного шага необходима установка дополнительного программного обеспечения.

Теория и практика

Фрагментация (секционирование) – это разделение таблицы или индекса на несколько логически связанных частей, фрагментов, секций с неким общим признаком.

Допустим, у нас есть таблица начислений, мы разбиваем эту таблицу на множество секций, например по начислениям за каждый месяц.

Кому и зачем это нужно? Вопрос с секционированием таблиц тесно связан с другим важным вопросом – вопросом масштабируемости проекта.

С помощью фрагментации появляется возможность управления фрагментами (секциями) в больших таблицах, то есть часть ненужных нам данных в текущий момент можно перенести на сторонний носитель.

Оставить для работы, для оптимального доступа к данным только лишь необходимые нам в сейчас секции таблицы.

Также очень часто возникает необходимость быстрого построения индекса по заданному фрагменту, а не по всей таблице.

Для решения данных задач в ORACLE используется фрагментация.

Итак, для демонстрации перечисленных возможностей фрагментации подготовим небольшой тестовый пример.

Для выполнения данного примера нам потребуется войти в систему под пользователем (SYS) с правами администратора.

Создадим три независимых табличных пространства, они нам понадобятся для демонстрационных примеров.

```
CREATE TABLESPACE TBLSP1 DATAFILE 'D:\BASE\TBLSP1.DAT'
SIZE 10M REUSE AUTOEXTEND ON NEXT 2M MAXSIZE 20M;
CREATE TABLESPACE TBLSP2 DATAFILE 'D:\BASE\TBLSP2.DAT'
 SIZE 10M REUSE AUTOEXTEND ON NEXT 2M MAXSIZE 20M;
CREATE TABLESPACE TBLSP3 DATAFILE 'D:\BASE\TBLSP3.DAT'
 SIZE 10M REUSE AUTOEXTEND ON NEXT 2M MAXSIZE 20M;
```

Фрагментация таблиц

В ORACLE используется три типа фрагментации (партицирования) для таблиц.

Фрагментация по диапазону значений

Данные, относящиеся к таблицам, где значения в заданных колонках относятся к некоторому диапазону, распределяются по соответствующим фрагментам (секциям) таблицы.

Например, все проводки 2001—2002 годов помещаются в первую секцию, за 2002—2003 годы во вторую и т. д.

Фрагментация по списку значений

Фрагмент (секция) определяется по элементу списка. Такой способ фрагментации идеально подходит, когда в заданной колонке используется ограниченное число значений.

Фрагментация с использованием хэш-функции

Фагментация по данным заданных столбцов таблицы. ORACLE вычисляет значение специальной хэш-функции, на основании которого определяет, в какой именно фрагмент таблицы поместить заданную запись.

Совмещенный тип фрагментации

Тип фрагментации, совмещающий в себе фрагментацию с использованием хэш-функции и фрагментацию по диапазону значений.

Синтаксис

Для создания и фрагментации таблиц используется дополнительная синтаксическая конструкция в команде CREATE TABLE – PATITION BY.

Обычный синтаксис для создания ферментированной таблицы выглядит следующим образом:

```
CREATE TABLE Имя таблицы
(столбец1 тип, столбец2 тип, столбец n)
PARTITION BY HASH(имя столбца по которому строится хэшфункция)
```

Описание фрагментов:

(partition имя фрагмента1 (партиции) название табличного пространства,

partition имя фрагмента2 (партиции) название табличного пространства)

или

PARTITION BY RANGE (имя столбца для проверки на соответствие диапазону фрагмента)

(PARTITION имя фрагмента1 VALUES LESS THAN

(выражение1)) имя табличного пространства,

(PARTITION имя фрагмента1 VALUES LESS THAN

(выражение2)) имя табличного пространства

Выгажение 1, Выражение 2 задают диапазон для данных для определения фрагмента таблицы (партиции), к которому эти данные относятся.

Специфика использования оператора SELECT для выбора данных из фрагментированных таблиц

С помощью оператора SELECT есть возможность как выбирать все данные из фрагментированной таблицы, так и использовать SELECT для выбора данных из заданного фрагмента таблицы.

```
select * from таблица partition(фрагмент);
```

Пример:

```
select * from pro_hash partition(pt_3);
```

Данный запрос выведет данные из фрагмента таблицы pt_3.
Фрагментация по диапазону значений.
Создадим таблицу проводок с фрагментацией по диапазону значений.

```
CREATE TABLE pro_range
( summ int,
 docdate date,
 docnum number
)
PARTITION BY RANGE(docdate)
(partition pt_1 values less than (to_date('01.02.2014','DD.MM.YYYY')) tablespace TBLSP1,
 partition pt_2 values less than (to_date('01.03.2014','DD.MM.YYYY')) tablespace TBLSP2,
 partition pt_3 values less than (to_date('01.04.2014','DD.MM.YYYY')) tablespace TBLSP3,
 partition p_othermax values less than (maxvalue) tablespace TBLSP3
);
```

Заполним таблицу проводок значениями.

```
insert into pro_range(summ,docdate,docnum)
select trunc(dbms_random.value * 10000) summ,
    to_date('01012014', 'DDMMYYYY') + trunc(dbms_random.value * 99) + 1 dtdoc,
    trunc(dbms_random.value * 50000) numdoc
 from dual
connect by level <= 1100
```

Выберем данные из таблицы.

```
select * from pro_range
```

SUMM DOCDATE DOCNUM

6226 11.01.2014 19170
3561 10.01.2014 41482
7106 09.01.2014 5604
5177 02.01.2014 1917
7090 02.01.2014 26399
692 07.01.2014 39100
8246 23.01.2014 38502
3364 30.01.2014 2813
7014 22.01.2014 41248
2331 11.01.2014 33239

```
select * from pro_range partition(pt_1) where rownum<3;
```

SUMM DOCDATE DOCNUM

6226 11.01.2014 19170
3561 10.01.2014 41482
7106 09.01.2014 5604

```
select * from pro_range partition(pt_2) where rownum<3;
```

SUMM DOCDATE DOCNUM

7131 13.02.2014 15345
2765 26.02.2014 47587
6594 16.02.2014 24707

```
select * from pro_range partition(pt_3) where rownum<3;
```

4832 30.03.2014 10615
4300 06.03.2014 20531
5897 07.03.2014 47711

```
select * from pro_range partition(p_othermax) where rownum<3;
```

SUMM DOCDATE DOCNUM

521 07.04.2014 35258
7416 05.04.2014 32045
1589 07.04.2014 46956

Следующий пример демонстрирует разбиение на фрагменты таблицы в зависимости от года, к которому принадлежит проводка.

```
CREATE TABLE pro_range_year
( summ int,
 docdate date,
 docnum number
)
PARTITION BY RANGE(docdate)
(partition pt_1 values less than ('01-01-2012') tablespace TBLSP1,
 partition pt_2 values less than ('01-01-2013') tablespace TBLSP2,
 partition pt_3 values less than ('01-01-2014') tablespace TBLSP3,
 partition p_othermax values less than (maxvalue) tablespace TBLSP3
);
```

Следующий пример иллюстрирует использование фрагментации таблицы в зависимости от числового значения.

```
CREATE TABLE pro_range_num
( summ int,
 docdate date,
 docnum number
)
PARTITION BY RANGE(docnum)
(partition pt_1 values less than (10) tablespace TBLSP1,
 partition pt_2 values less than (30) tablespace TBLSP2,
 partition pt_3 values less than (40) tablespace TBLSP3,
 partition p_othermax values less than (maxvalue) tablespace TBLSP3
);
```

Фрагментация с использованием списка значений.

Создание таблицы:

```
CREATE TABLE pro_list
( summ int,
 docdate date,
 doctype varchar2(2))
PARTITION BY LIST(doctype)
(
 partition pt_1 values ('PP','PO' ) tablespace TBLSP1,
 partition pt_2 values ('RR') tablespace TBLSP2,
 partition pt_3 values ('RO','SO') tablespace TBLSP3
);
```

Заполняем таблицу:

```
insert into pro_list(summ,docdate,doctype)
select trunc(dbms_random.value * 10000) summ,
    to_date('01012011', 'DDMMYYYY') + trunc(dbms_random.value * 799) + 1 dtdoc,
    decode(trunc(dbms_random.value * 5)+1,1,'PP',2,'PO',3,'RR',4,'RO',5,'SO' ) doctype
 from dual
connect by level <= 1100
```

Проверяем:

```
select * from pro_list
```

SUMM DOCDATE DOCTYPE
1 9445 26.01.2013 PP
2 6120 21.01.2012 PP
3 8374 18.02.2013 PP
4 533 24.04.2011 PP
5 77 27.02.2013 PO
6 5046 12.01.2011 PO

```
select * from pro_list partition(pt_1) where rownum<4;
```

SUMM DOCDATE DOCTYPE
1 9445 26.01.2013 PP
2 6120 21.01.2012 PP
3 8374 18.02.2013 PP

```
select * from pro_list partition(pt_2) where rownum<4;
```

SUMM DOCDATE DOCTYPE
1 3046 21.12.2011 RR
2 2093 07.09.2012 RR
3 9233 14.08.2012 RR

```
select * from pro_list partition(pt_3) where rownum<4;
```

SUMM DOCDATE DOCTYPE

1 7980 11.12.2012 RO
2 9338 01.03.2012 SO
3 9784 22.11.2012 RO

Фрагментация с использованием хэш-функции

Создадим таблицу проводок с фрагментацией по хэш-функции.

```
CREATE TABLE pro_hash
( summ int,
 docdate date,
 docnum number
)
PARTITION BY HASH(docnum)
(partition pt_l tablespace TBLSP1,
 partition pt_2 tablespace TBLSP2,
 partition pt_3 tablespace TBLSP3
);
```

Заполним ее данными:

```
insert into pro_hash(summ,docdate,docnum)
select trunc(dbms_random.value * 10000) summ,
to_date('01012014', 'DDMMYYYY') + trunc(dbms_random.value * 61) + 1   dtdoc,
     trunc(dbms_random.value * 50000) numdoc
 from dual
connect by level <= 1100;
```

Выберем данные из таблицы:

```
select * from pro_hash
```

А также выполним оператор SELECT для каждого из фрагментов (секций) таблицы:

```
select * from pro_hash partition(pt_1);
select * from pro_hash partition(pt_2);
select * from pro_hash partition(pt_3);
```

Смешанный тип фрагментации

Смешанный тип фрагментации предусматривает как фрагментацию по диапазону значений, так и дополнительную фрагментацию по хэш-функции или фрагментацию по списку значений.

```
CREATE TABLE pro_range_hash
( summ int,
 docdate date,
 docnum number
)
PARTITION BY RANGE(docdate)
SUBPARTITION BY HASH(docnum)
SUBPARTITION TEMPLATE(
SUBPARTITION spt_1 TABLESPACE TBLSP1,
SUBPARTITION spt_2 TABLESPACE TBLSP2,
SUBPARTITION spt_3 TABLESPACE TBLSP3
)
(partition pt_1 values less than (to_date('01.02.2014','DD.MM.YYYY')) tablespace TBLSP1,
 partition pt_2 values less than (to_date('01.03.2014','DD.MM.YYYY')) tablespace TBLSP2,
 partition pt_3 values less than (to_date('01.04.2014','DD.MM.YYYY')) tablespace TBLSP3,
 partition p_othermax values less than (maxvalue) tablespace TBLSP3
);
```

Заполним таблицу PRO_RANGE_HASH:

```
insert into pro_range_hash(summ,docdate,docnum)
select trunc(dbms_random.value * 10000) summ,
    to_date('01012011', 'DDMMYYYY') + trunc(dbms_random.value * 799) + 1 dtdoc,
    trunc(dbms_random.value * 50000) numdoc
 from dual
connect by level <= 1100
```

Управление данными во фрагментах таблицы

Попробуем внести данные в таблицу, которые не подходят по условиям ни в один из фрагментов (секций).

```
insert into pro_list(summ,docdate,doctype) values (10,sysdate,'GT');
```

Сервер выведет ошибку Ora-144000. Вставленный ключ секции не соответствует ни одной секции.

Изменим данные таким образом, чтобы изменилась принадлежность записей к фрагменту.

```
update pro_list set pro_list.doctype = 'SO' where pro_list.doctype = 'RR'
```

Получим ошибку обновление ключа секции, что приведет к ее изменению…

Как же сделать возможным перенос строк?
Для этого необходимо включить для выбранной таблицы опцию ROW MOVEMENT.

Выполним скрипт:

```
alter table pro_list enable row movement

выполним наш update повторно

update pro_list set pro_list.doctype = 'SO' where pro_list.doctype = 'RR'
```

На этот раз ошибок не было. Обновление строк в таблице прошло успешно.
А собственно, зачем это все было необходимо? Как это применить?

У вас есть три табличных пространства, нам в данный момент нужна только оперативная информация по документам типа RR из таблицы PRO_LIST в табличном пространстве номер TBLSP2.
Отключим остальные табличные пространства и перенесем их на архивный диск.

```
alter tablespace TBLSP1 offline;
alter tablespace TBLSP3 offline;
```

Выполним запрос по таблице PRO_LIST:

```
Select * from pro_list
```

Получим системную ошибку о том, что сегмент не может быть прочитан.
Скорректируем запрос, так как нам нужны только документы RR.

```
select * from pro_list where pro_list.doctype = 'RR'
```

Запрос выполнен без ошибок.

SUMM DOCDATE DOCTYPE
3046 21.12.2011 RR
2093 07.09.2012 RR
9233 14.08.2012 RR
5474 09.03.2013 RR

Таким образом, применяя фрагментацию в больших таблицах с несколькими миллионами записей, всегда есть возможность освободить часть дискового пространства и перенести неиспользуемые данные на архивный носитель.

Контрольные вопросы и задания для самостоятельного выполнения

1. Повторите материалы данного шага.

2. Создайте секционированную таблицу, основанную на таблице AUTO, где данные бы распределялись по секциям времени, по годам.

Шаг 72. Работаем с XML в SQL

Введение

XML – это специальный расширяемый язык разметки. XML нашел широкое применение в сервисах обмена данными, в интернете, в RSS-сообщениях. Для работы с XML в ORACLE используется специальный тип данных XMLType, при помощи которого работа с XML-выражениями в ORACLE SQL становится максимально удобной и комфортной.

Для работы с XMLType в ORACLE SQL присутствует ряд дополнительных команд, применяемых для обработки XML-документа. Это команды EXTRACT, EXTRACTVALUE, XMLSEQUENCE.

Разберем работу с этими командами подробнее.

Теория и практика

XMLTYPE – специальный тип данных, который позволяет программисту работать с XML-документом, не прибегая к XPath и парсингу строк.

Рассмотрим приемы работы с XMLTYPE.

Создадим таблицу из двух колонок.

Первая колонка – идентификатор, вторая колонка специального типа XMLType.

```
CREATE TABLE messages
  (id        NUMBER PRIMARY KEY
 , xmltest XMLTYPE);
```

Добавим данные, идентификатор XML-документа, и непосредственно сам XML-документ.

```
INSERT INTO messages VALUES
   (100
  , XMLTYPE('
   <note nid="501">
    <to>Tove</to>
    <from>Jani</from>
    <heading>Напоминание</heading>
    <body>В воскресенье!</body>
  </note> '));
INSERT INTO messages VALUES
   (101
  , XMLTYPE('
   <note nid="505">
    <to>Tove</to>
    <from>Jani</from>
    <heading>Напоминание1</heading>
    <body>Привет </body>
  </note> '));
```

Для доступа к XML-атрибутам используются две специальные функции VALUE и EXTRACTVALUE, продемонстрируем их работу на примере.

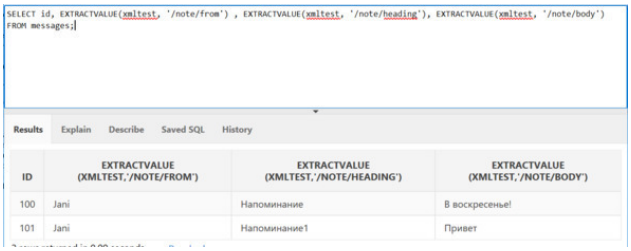
Извлекаем атрибут TO из колонки XMLTEST нашего примера:

```
SELECT id, EXTRACTVALUE(xmltest, '/note/to') FROM messages;
```

ID	EXTRACTVALUE(XMLTEST,'/NOTE/TO')
100	Tove
101	Tove

Извлекаем остальные атрибуты из нашей таблицы из колонки XMLTEST:

```
SELECT id, EXTRACTVALUE(xmltest, '/note/from') , EXTRACTVALUE(xmltest, '/note/heading'),
EXTRACTVALUE(xmltest, '/note/body')
FROM messages;
```

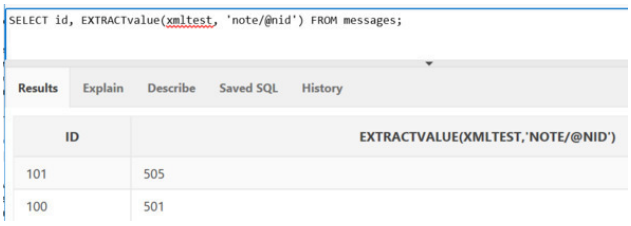


Results | Explain | Describe | Saved SQL | History

ID	EXTRACTVALUE (XMLTEST,'/NOTE/FROM')	EXTRACTVALUE (XMLTEST,'/NOTE/HEADING')	EXTRACTVALUE (XMLTEST,'/NOTE/BODY')
100	Jani	Напоминание	В воскресенье!
101	Jani	Напоминание1	Привет

Следующий пример демонстрирует получение внутреннего атрибута IDn, для этого в EXTRACTVALUE используется специальная запись @nID.

SELECT ID, EXTRACTVALUE (XMLtest, «NOTe/@nID») FROM messages;



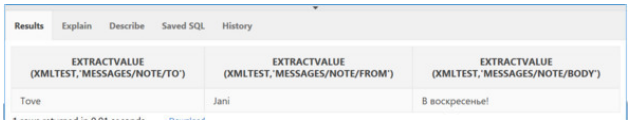
Results | Explain | Describe | Saved SQL | History

ID	EXTRACTVALUE(XMLTEST,'NOTE/@NID')
101	505
100	501

Преобразование строки XML-документа в XMLType.

Пример преобразования типов, также в данном запросе для наглядности используем оператор **WITH**, изученный нами ранее.

```
with demo1 as (select xmltype('<?xml version="1.0" encoding="UTF-8"?>
<messages>
  <note id="501">
    <to>Tove</to>
    <from>Jani</from>
    <heading>Напоминание</heading>
    <body>В воскресенье!</body>
  </note>
</messages>
') xmltest from dual
  )
  select extractvalue(xmltest,'messages/note/to')
       ,extractvalue(xmltest,'messages/note/from')
       ,extractvalue(xmltest,'messages/note/body')
 from demo1;
```



В этом примере, используя **WITH**, мы создали таблицу с одной строкой из XML-документа, воспользовавшись преобразованием XMLType.

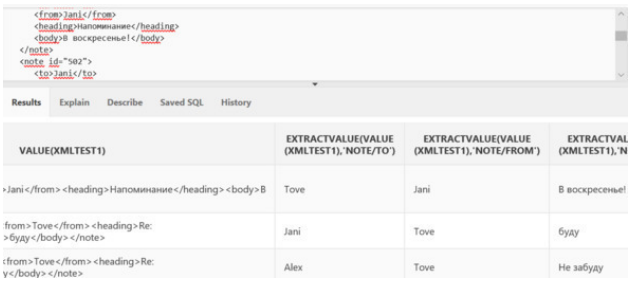
А если нам необходимо преобразовать XML-документ в многострочную таблицу, то следует использовать конструкцию XMLSEQUENCE.

Преобразование XMLType в таблицу

XMLSEQUENCE возвращает XMLSEQUENCETYPE, отображает XML.

Следующий пример демонстрирует, как работать с XMLSEQUENCE для получения данных из XML в табличном виде.

```
with demo1 as (select xmltype('<?xml version="1.0" encoding="UTF-8"?>
<messages>
  <note id="501">
    <to>Tove</to>
    <from>Jani</from>
    <heading>Напоминание</heading>
    <body>В воскресенье!</body>
  </note>
  <note id="502">
    <to>Jani</to>
    <from>Tove</from>
    <heading>Re: Напоминание</heading>
    <body>буду</body>
  </note>
 <note id="503">
    <to>Alex</to>
    <from>Tove</from>
    <heading>Re: Hey</heading>
    <body>Не забуду</body>
  </note>
</messages>') xmltest from dual
  )
  select value(xmltest1),
        extractvalue(value(xmltest1),'note/to')
       ,extractvalue(value(xmltest1),'note/from')
       ,extractvalue(value(xmltest1),'note/body')
 from demo1 s
 , table(XMLSequence(s.xmltest.extract('messages/note'))) xmltest1;
```



В этом примере с помощью инструкции TABLE (XMLSEQUENCE (s. XMLTEST. extract (’messages/NOTe’))) мы создали таблицу, состоящую из XML-документа между тегами NOTE.

Пример

(<NOTe ID=«503»>
<to> Alex </to>
<FROM> Tove </FROM>
<headINg> Re: Hey </headINg>
<body> Не забуду </body>
</NOTe>
)

Из которой извлекли соответствующие значения.

Важные замечания

XMLTYPE появился в ORACLE начиная с версии 9.0, на более ранних версиях ORACLE приведенные в этом шаге команды работать не будут.

Вопросы учеников

Кроме XMLType, какие существуют способы разбора в ORACLE SQL?

Можно использовать XPATH для разбора XML, но это менее удобный способ.

Контрольные вопросы и задания для самостоятельного выполнения

1. Создайте таблицу из двух колонок (ID NUMBER, XMLT XMLtype).
2. Внесите в таблицу следующий XML и ID = 1:

```
<?XML version=«1.0» encodINg=«UTF-8»? >
<BookCatalogue>
<Book>
<Title> YogASana Vijnana: the Science of Yoga </Title>
<Author> Dhirendra Brahmachari </Author>
<DATE> 1966 </DATE>
<ISBN> 81-40-34319-4 </ISBN>
<PublISher> Dhirendra Yoga Publications </PublISher>
<Cost currency=«INR»> 11.50 </Cost>
</Book>
<Book>
<Title> The First AND LASt Freedom </Title>
<Author> J. KrIShnamurti </Author>
<DATE> 1954 </DATE>
<ISBN> 0-06-064831-7 </ISBN>
<PublISher> Harper &amp;amp; Row </PublISher>
<Cost currency=«USD»> 2.95 </Cost>
</Book>
</BookCatalogue>
```

3. Выберите из XML-документа следующую информацию: TITLE, AUTHOR, ISBN в табличном виде.
4. Выберите из данной таблицы CURRENCY при покупке книги.

Шаг 73. Сложные группировки SET GROUP CUBE

Введение

В ORACLE существуют конструкции для подведения подытогов и итогов групповых операций.

Это операции **CUBE**, **ROLLUP**.

Теория и практика

Группировки ROOLUP, CUBE ROLLUP и CUBE – это специальные групповые операции ORACLE.

Используются вместе с группировкой GROUP BY. Применяются начиная с версии ORACLE 8i.

Синтаксис:

```
SELECT выражение1, выражение2, выражение3
 FROM Man GROUP BY ROLLUP(выражение2)
```

ROLLUP – выражение, по которому подводятся итоги.

GROUPING – условное выражение, которое показывает, является ли строчка строчкой подведения итогов.

Очень удобно использовать ROLLUP CUBE при подведении итогов.

```
SELECT substr(firstname,1,2) as Caps, count(firstname) countr FROM Man GROUP BY
substr(firstname,1,2)
```

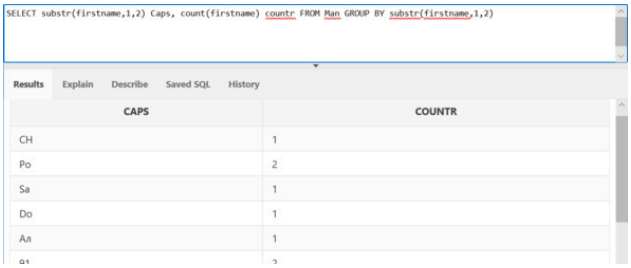


Рисунок 203. Пример использования группировок

Примеры

Попробуем вычислить, сколько имен начинаются с заданной буквы в таблице MAN.

```
SELECT substr(firstname,1,1) Caps, count(firstname) countr FROM Man GROUP BY
substr(firstname,1,1)
```

Итоги нашей выборки подведем с помощью ROLLUP.

```
SELECT substr(firstname,1,1) Caps, count(firstname) countr FROM Man GROUP BY
ROLLUP(substr(firstname,1,1))
```

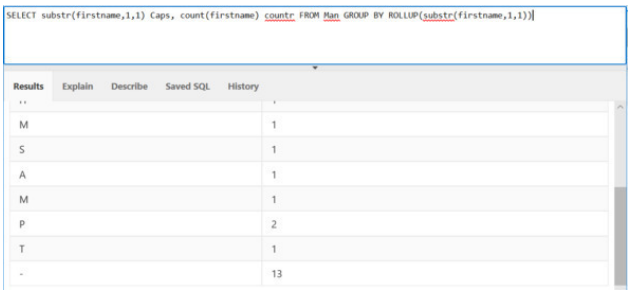


Рисунок 204. Пример использования ROLLUP

Второй пример:

```
SELECT substr(firstname,1,1) Caps, substr(firstname,1,2) Caps2, count(firstname) tr,
grouping(substr(firstname,1,1)) gr1
FROM Man GROUP BY ROLLUP(substr(firstname,1,1),substr(firstname,1,2))
```

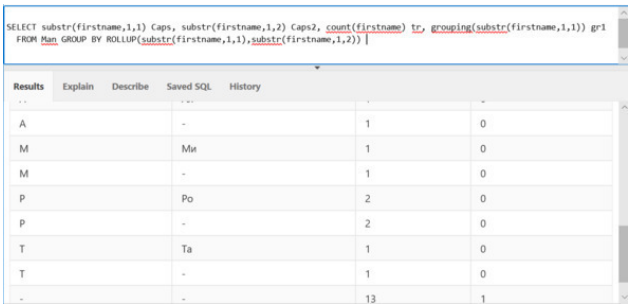


Рисунок 205. Пример использования ROLLUP

Итак, функция ROLLUP подводит итоги по некоторой группе, некоторым сгруппированным значениям. В первом примере это первая буква имени, во втором примере – первые две буквы имени; GROUPING показывает, в какой именно строке подводился итог.

Группировка CUBE также используется для подведения итогов, очень похожа на ROLLUP, но подводит общий итог в конце выборки.

Пример

```
SELECT substr(firstname,1,1) Caps, substr(firstname,1,2) Caps2, count(firstname) tr
 FROM MAN group by cube(substr(firstname,1,1),substr(firstname,1,2))
```

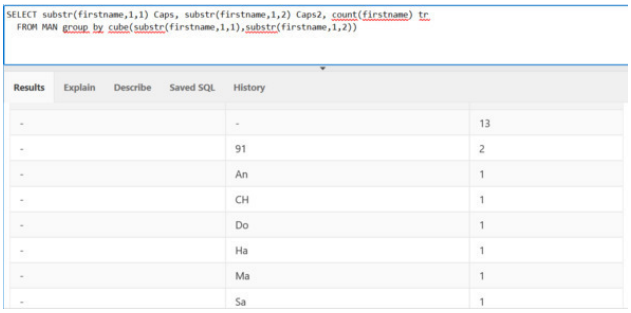


Рисунок 206. Пример использования CUBE

Важные замечания

ROLLUP работает следующим образом:

• сначала вычисляются стандартные агрегированные значения в предложении GROUP BY;

• затем создаются промежуточные итоги высокого уровня для столбцов группировки, которые представляют собой столбцы «колонка2» и «колонка 1», справа налево;

• наконец, создается общий итог.

Вопросы учеников

Есть еще дополнительная функция GROUPING SET, поясните ее работу.

Предложение GROUPING SETS – это расширение предложения GROUP BY, которое можно использовать, чтобы задать одновременно сразу несколько группировок данных.

Пример работы:

```
SELECT mark, color, count(1)
FROM auto1 GROUP BY GROUPING SETS (mark, color, (mark, color));
```

MARK	COLOR	COUNT(1)
LADA	-	1
LADA	RED	1
Impreza	Blue	1
BMWE60	black	1
infiniti	white	1
LADA	КРАСНЫЙ	1
-	-	1
-	RED	1

Без данной функции для получения результата, аналогичного приведенной инструкции SELECT, потребовалось бы несколько запросов, объединенных UNION ALL. Использование нескольких запросов неэффективно, так как требует нескольких обращений для одних и тех же данных.

Контрольные вопросы и задания для самостоятельного выполнения

1. Используя ROLLUP, посчитайте, сколько разных марок автомобилей разного цвета. Подведите итоги с помощью ROLLUP.

2. Объясните различия между ROllUP и CUBE.

3. С помощью CUBE подсчитайте количество людей одного возраста (MAN), подведите итог: всего людей 18, 19, 20 лет.

Шаг 74. Представления

Введение

Ранее мы уже разбирали конструкцию WITH, когда можно предварительно составить запрос и обратиться через псевдоним к этому запросу.

Подобный функционал обеспечивают представления VIEW.

Теория и практика

Представления являются физическим объектом в базе данных, который создается на основе запроса.

Имя представления не может начинаться с цифр и должно быть уникально в рамках схемы, а также не должно совпадать с именем любой из таблиц или зарезервированным словом СУБД.

Обычное представление VIEW физически не содержит данных, запрос из представления выполняется приблизительно, так же как обычный запрос.

Основное назначение представлений в ORACLE – это распределение прав и удобство использования представлений.

Синтаксис

```
CREATE or REPLACE VIEW viewname AS  SELECT * или колонки  FROM таблицы WHERE
условия GROUP BY группировка если надо ORADER BY сортировка
```

где VIEWNAME – название представления, AS – запрос-обращение к представлению.

```
SELECT * FROM viewname WHERE...
```

То есть в качестве источника данных мы используем ранее созданное представление.

Примеры

Представление на основе таблицы AUTO, где только синие авто.

```
CREATE OR REPLACE VIEW v_auto
AS SELECT a.color, a.mark as mark FROM auto a where color = 'СИНИЙ';
```

Выбор данных из созданного представления v_AUTO.

```
SELECT * FROM v_auto;
```

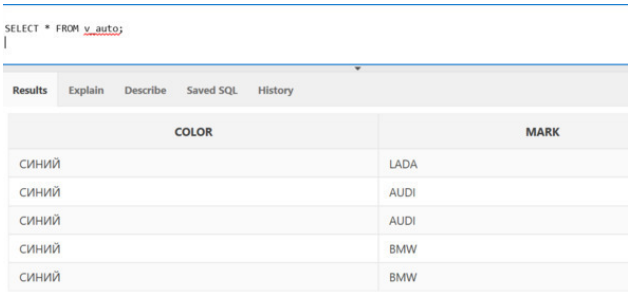


Рисунок 207. Обращение к v_AUTO

Представление на основе таблицы MAN и таблицы AUTO.
Обратиться к данному представлению в запросе, выбрать автомобили BMW.

```
CREATE OR REPLACE VIEW v_man_auto
AS SELECT m.firstname as irstname, m.lastname as lastname, a.mark as mark FROM man m
INNER JOIN auto a ON m.phonenum = a.phonenum;
```

Выбор данных из созданного представления v_MAN_AUTO.

```
SELECT * FROM v_man_auto WHERE mark='BMW';
```

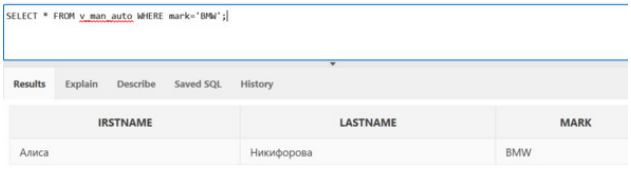


Рисунок 208. Запрос к v_MAN_AUTO

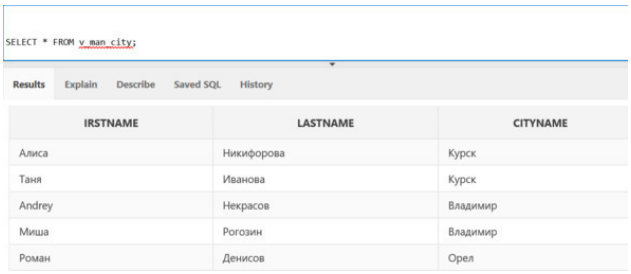
Представление на основе таблицы MAN и таблицы CITY.

Обратиться к данному представлению в запросе:

```
CREATE OR REPLACE VIEW v_man_city
AS SELECT m.firstname as irstname, m.lastname as lastname, c.cityname as cityname FROM
man m INNER JOIN city c ON m.citycode = c.citycode;
```

Выбор данных из представления v_MAN_CITY.

```
SELECT * FROM v_man_city;
```



SELECT * FROM v_man_city;

Results | Explain | Describe | Saved SQL | History

IRSTNAME	LASTNAME	CITYNAME
Алиса	Никифорова	Курск
Таня	Иванова	Курск
Andrey	Некрасов	Владимир
Миша	Рогозин	Владимир
Роман	Денисов	Орел

Рисунок 209. Запрос к v_MAN_CITY

Важные замечания

• На чтение данных из представлений нужны права, которые может выдать администратор системы.

• На создание представлений также нужны права, которые может выдать администратор системы.

• Представления не сохраняют результата запроса, а только текст запроса, каждый раз при обращении к заданному представлению-запросу выполняются вновь.

• Для создания отчетов лучше использовать материализованные преставления или временные таблицы.

Вопросы учеников

Можно ли создать представление на основе запроса другого представления?

Да, никаких ограничений нет.

Можно ли использовать в представлении объекты последовательностей SEQUENCE?

Нет, это ограничение, использовать последовательности в представлениях нельзя.

Можно ли использовать в представлениях одинаковые названия колонок?

Нет, использовать одинаковые названия колонок недопустимо.

Контрольные вопросы и задания для самостоятельного выполнения

1. Расскажите, что такое представление в рамках базы данных.
2. Чем представление отличается от обычного запроса?
3. В чем преимущество использования представлений в базе данных?
4. Создайте представление на основе таблицы CITY.
5. Создайте представление на основе таблиц CITY, MAN с данными из этих двух таблиц (запрос с объединением двух таблиц, связь по колонке CITYCODE).

Шаг 75. Синонимы

Введение

Синонимы (synonyms) – специальные псевдонимы объектов базы данных, применяются для удобства доступа к объектам других схем базы данных, могут использоваться для распределения прав и безопасности доступа к данным. Синонимы могут приватными (private) или общедоступными (public). Общедоступные синонимы видны всем пользователям базы данных, приватные синонимы принадлежат только заданной схеме.

Синонимы создаются для следующих объектов базы данных ORACLE: таблицы, представления, материализованные представления пакетов и процедур.

Теория и практика

Для вышеперечисленных объектов в базе ORACLE можно создать синоним.
В каком случае это целесообразно?
• Во-первых, при обращении к объекту из разных схем для более простой формы записи.
• Для разграничения доступа, безопасности объекта.
• Во-вторых, если объект называется достаточно сложно и для того чтобы упростить обращение к этому объекту.
• Когда обращение к объекту происходит через DB LINK.
Для создания публичного синонима необходима роль CREATE PUBLIC SYNONYM;

```
CREATE PUBLIC SYNONYM синоним   FOR имя_объекта;
```

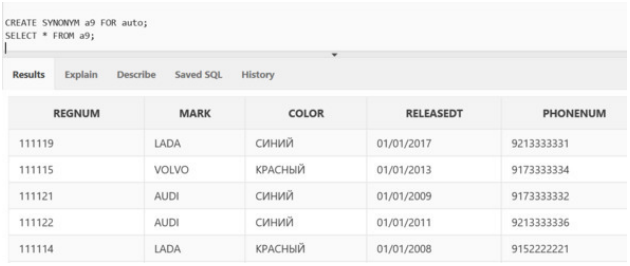
Для создания публичного синонима используется директива PUBLIC перед словом SYNONYM.

Примеры

```
CREATE SYNONYM a9 FOR auto;
```

Обращение к данным через синоним a9:

```
SELECT * FROM a9;
```



```
CREATE SYNONYM a9 FOR auto;
SELECT * FROM a9;
```

Results | Explain | Describe | Saved SQL | History

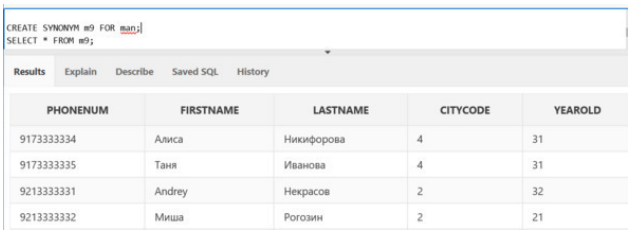
REGNUM	MARK	COLOR	RELEASEDT	PHONENUM
111119	LADA	СИНИЙ	01/01/2017	9213333331
111115	VOLVO	КРАСНЫЙ	01/01/2013	9173333334
111121	AUDI	СИНИЙ	01/01/2009	9173333332
111122	AUDI	СИНИЙ	01/01/2011	9213333336
111114	LADA	КРАСНЫЙ	01/01/2008	9152222221

Рисунок 210. Запрос по синониму a9

Выбираем данные из таблицы с помощью синонима a9.

```
CREATE SYNONYM m9 FOR man;
SELECT * FROM m9;
```

Выбираем данные из таблицы с помощью синонима m9.



```
CREATE SYNONYM m9 FOR man;
SELECT * FROM m9;
```

Results | Explain | Describe | Saved SQL | History

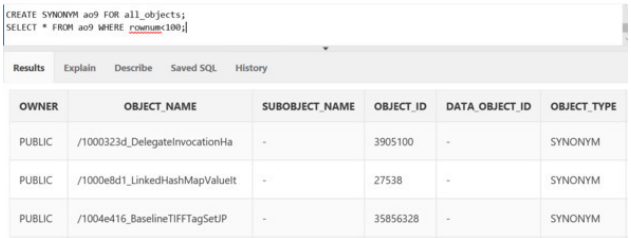
PHONENUM	FIRSTNAME	LASTNAME	CITYCODE	YEAROLD
9173333334	Алиса	Никифорова	4	31
9173333335	Таня	Иванова	4	31
9213333331	Andrey	Некрасов	2	32
9213333332	Миша	Рогозин	2	21

Рисунок 211. Запрос по синониму m9

```
CREATE SYNONYM ao9 FOR all_objects;
```

Создание синонима на основе системного представления ALL_OBJECTS;
Выбираем данные из таблицы с помощью синонима a9.

```
SELECT * FROM ao9 WHERE rownum<100;
```



```
CREATE SYNONYM ao9 FOR all_objects;
SELECT * FROM ao9 WHERE rownum<100;
```

Results | Explain | Describe | Saved SQL | History

OWNER	OBJECT_NAME	SUBOBJECT_NAME	OBJECT_ID	DATA_OBJECT_ID	OBJECT_TYPE
PUBLIC	/1000323d_DelegateInvocationHa	-	3905100	-	SYNONYM
PUBLIC	/1000e8d1_LinkedHashMapValueIt	-	27538	-	SYNONYM
PUBLIC	/1004e416_BaselineTIFFTagSetJP	-	35856328	-	SYNONYM

Рисунок 212. Запрос по синониму a9

Важные замечания

Для просмотра информации обо всех синонимах в базе данных можно использовать следующий запрос:

```
SELECT TABLE_NAME, SYNONYM_NAME
FROM dba_synonyms
WHERE OWNER = 'SQLADV';
```

Для выполнения данного запроса требуются соответствующие права.
Для удаления синонима используется следующая команда:

```
DROP SYNONYM synonymname;
```

Вопросы учеников

Я пытаюсь создать синоним для функции на языке PL SQL, которая находится в пакете, получаю ошибку. В чем причина?

Это ограничение для использования синонимов – объект для синонима не может содержаться в пакете.

Каковы ограничения для имени синонима?

Помимо того, что имя синонима не должно начинаться с цифр или совпадать с зарезервированным словом SQL, также имя синонима должно быть уникальным для всех объектов, которые принадлежат этой же схеме.

Контрольные вопросы и задания для самостоятельного выполнения

1. Можно ли создать синоним для представления?
2. В чем отличие синонимов и представлений?
3. Создайте публичный синоним C1 для таблицы CITY.
4. Создайте приватный синоним C2 для таблицы CITY.
5. Напишите запрос с использованием данных синонимов.
6. Можно ли создать синоним для другого синонима?
7. Поясните разницу между публичным и приватным синонимами.

День шестнадцатый

Шаг 76. Ретроспективные запросы

Введение

В ORACLE есть возможность узнать, какие данные были в таблицах определенное время назад. Мы можем использовать эти данные, сохранить эти данные во временные таблицы.

Такие запросы называются FLASHBACK QUERY.

FLASHBACK QUERY – специальная возможность, которая позволяет видеть данные в базе таким образом, как будто это было сделано в заданный момент времени в прошлом.

Теория и практика

Механизм ретроспективных запросов (FlASHBACK QUERY) позволяет нам с помощью запросов посмотреть в прошлое.

Есть возможность использовать FLASHBACK QUERY двумя способами:

- специальная конструкция AS OF в SELECT-запросе;
- специальный встроенный пакет DBMS_FLASHBACK.

Конструкция:

```
select  *
 from tablename as of scn timestamp_to_scn(to_timestamp(time,'DD/MM/YYYY
HH24:MI:SS')) ;
```

Здесь

• **SELECT * FROM TABLENAME** – запрос к таблице, где необходимо посмотреть данные;

• of scn TIMESTAMP_TO_scn (TO_TIMESTAMP (time,«DD/MM/YYYY HH24:MI: SS»)) – специальная конструкция, где необходимо указать дату-время, на которую требуется посмотреть изменения, в соответствующем формате «DD/MM/YYYY HH24:MI: SS».

Для примера нам понадобится таблица, которую мы создадим на основе системного представления ALL_OBJECTS.

```
create table obj_t -- table
as select owner, object_type  from all_objects where rownum<101 group by owner,
object_type;
```

Запомним время создания таблицы.

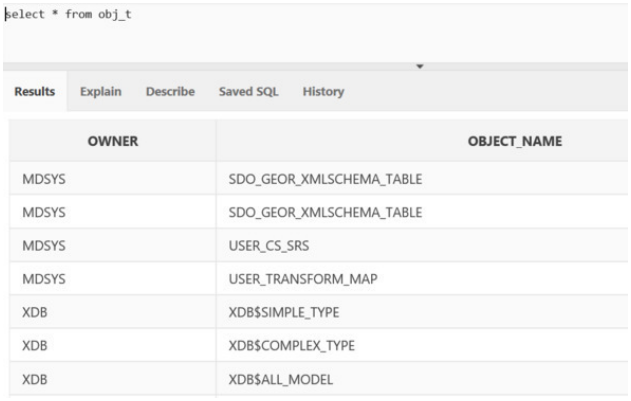
```
delete obj_t where object_type = 'TABLE';
```

Удалим записи с наименованиями таблиц и запомним время 17:10.

```
delete obj_t where owner in ('SYS', 'SYSTEM');
```

Удалим данные, где owner IN («SYS», «SYSTEM») и запомним время 17:11.

```
select * from obj_t;
```



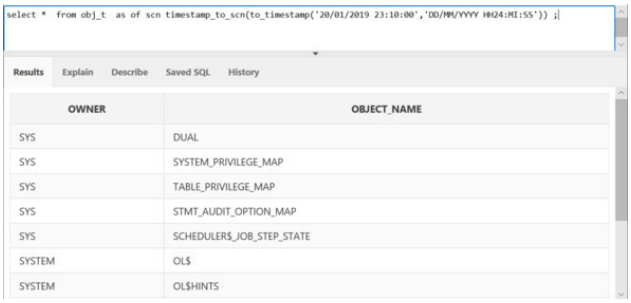
select * from obj_t

Results | Explain | Describe | Saved SQL | History

OWNER	OBJECT_NAME
MDSYS	SDO_GEOR_XMLSCHEMA_TABLE
MDSYS	SDO_GEOR_XMLSCHEMA_TABLE
MDSYS	USER_CS_SRS
MDSYS	USER_TRANSFORM_MAP
XDB	XDB$SIMPLE_TYPE
XDB	XDB$COMPLEX_TYPE
XDB	XDB$ALL_MODEL

Рисунок 213. Запрос по OBJ_T

```
select *  from obj_t  as of scn timestamp_to_scn(to_timestamp('19/04/2018
17:10:00','DD/MM/YYYY HH24:MI:SS')) ;
```



select * from obj_t as of scn timestamp_to_scn(to_timestamp('20/01/2019 23:10:00','DD/MM/YYYY HH24:MI:SS'))

Results | Explain | Describe | Saved SQL | History

OWNER	OBJECT_NAME
SYS	DUAL
SYS	SYSTEM_PRIVILEGE_MAP
SYS	TABLE_PRIVILEGE_MAP
SYS	STMT_AUDIT_OPTION_MAP
SYS	SCHEDULER$_JOB_STEP_STATE
SYSTEM	OL$
SYSTEM	OL$HINTS

Рисунок 214. Ретроспективный запрос

```
select *
 from obj_t as of scn timestamp_to_scn(to_timestamp('19/04/2018 17:11:00','DD/MM/YYYY
HH24:MI:SS')) ;
```

Для примера работы пакета FLASHBACK нам понадобится написать короткий код на PLSQL.
Пример DBMS_FLASHBACK:

```
DELETE obj_t WHERE owner='SYS'
```

Удалим данные из таблицы OBJ_T.

```
begin
        DBMS_FLASHBACK.ENABLE_AT_TIME('20-JAN-19 22:00:00');
End;
```

Эти несколько строчек кода переключат данные в нашей таблице на определенный момент в прошлом.
Запрос

```
SELECT * FROM obj_t;
```

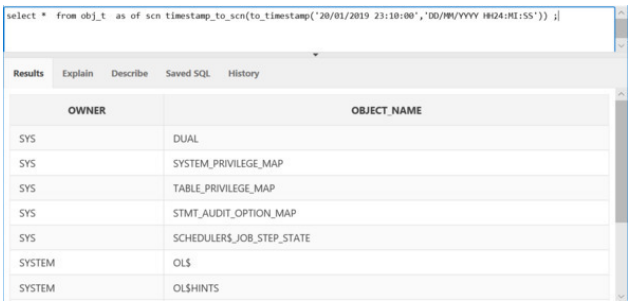


Рисунок 216. Ретроспективный запрос к OBJ_T

вернет данные на прошлый момент времени.

```
begin
 DBMS_FLASHBACK.Disable;
End;
```

Эти несколько строчек кода вернут данные к первоначальному состоянию.

Важные замечания

Для работы ретроспективных запросов (FLASHBACK QUERY) необходимо установить параметр инициализации UNDO_MANAGEMENT=AUTO.

На возможность отката изменений также влияют следующие параметры – размер табличного пространства UNDO и параметр UNDO_RETENTION (в секундах).

Вопросы учеников

Я не могу получить данные из таблицы указанным способом, почему?

Для этого может быть несколько причин, например очистка командой TRUNCATE TABLE, или критическая перезагрузка экземпляра базы, или переполнение UNDO-сегмента.

Контрольные вопросы и задания для самостоятельного выполнения

1. Повторите материалы данного шага.
2. Создайте таблицу AUTO1 как копию таблицы AUTO.
3. Удалите данные из таблицы AUTO1.
4. Восстановите данные из таблицы AUTO1 в новую таблицу AUTO3.

Шаг 77. ORACLE DATABASE LINK и соединение с другой базой данных

Введение

DATABASE LINK (связь) – это специальный механизм для связи локальной базы данных с удаленной базой данных. Связь устанавливается только с той стороны, где создан объект DB LINK.

При создании DATABASE LINK (связь) задаются параметры учетной записи удаленной базы данных, с которой устанавливается соединение. Соответственно, и права в этой базе данных после соединения будут идентичны привилегиям учетной записи, параметры которой были заданы при создании DATABASE LINK. DATABASE LINK могут применяться для выбора записей из таблиц распределенной базы данных, а также для вставки данных в собственную локальную базу из удаленной базы. Также DB LINK применяются для вставки, обновления записей непосредственно в удаленной базе данных.

Механизм ORACLE DATABASE LINK позволяет работать одновременно с множеством баз данных как с единой базой данных.

Для данного шага необходима установка дополнительного программного обеспечения ORACLEXE, SQLDEVELOPER; как это сделать, подробно описано в шаге 51.

Также необходимо создать еще один сервер ORACLE СУБД на другой, удаленной машине. Для этого я рекомендую воспользоваться виртуальной машиной VIRTUALBOX.

Информация о том, как работать с VIRTUALBOX, выходит за рамки этой книги, но ее можно легко найти в интернете, например на сайте ORACLE.com.

Теория и практика

Рассмотрим создание DATABASE LINK.
Синтаксис:

```
CREATE DATABASE LINK MTEST
connect to userdb1 identified by password1
USING 'sservise';
```

Здесь
• USERDB1 – имя пользователя на удаленной (другой) базе данных, с которой соединяемся;
• PASSWORD1 – пароль пользователя на базе, с которой соединяемся;
• 1521 – порт для соединения с другой базой;
• SSERVICE – имя соединения в TNSNAMES.

Также синтаксис может быть и таким:

```
Create Database Link MTEST   connect to userdb1 identified by password1   using
'(DESCRIPTION =
 (ADDRESS_LIST =  (ADDRESS = (PROTOCOL = TCP)(HOST = test.test.ru)(PORT = 1521))
 )  (CONNECT_DATA = (SERVER = DEDICATED) (SID = mtest1)  ))';
```

Здесь
• USERDB1 – имя пользователя на удаленной (другой) базе данных, с которой соединяемся;
• PASSWORD1 – пароль пользователя на базе, с которой соединяемся;
• DESCRIPTION – соединение;
• TEST.TEST.RU – имя хоста;
• 1521 – порт для соединения с другой базой.

DATABASE LINK может быть публичным, его будут видеть все пользователи базы данных, или локальным, только для текущего пользователя.

Для того чтобы создать DATABASE LINK публичный, требуется после команды create указать PUBLIC.

Пример создания публичного DATABASE LINK:

```
CREATE PUBLIC DATABASE LINK MTEST
connect to userdb1 identified by password1
USING 'sservise';
```

Важные замечания

Помимо операций вставки данных, выбора данных, обновления данных, через DATABASE LINK можно вызвать процедуры и функции на других серверах.

Пример

```
resval := stock.function1@remotedblink ( par1 => 'par1'    , parn => 'parn' , qty => 6 );
```

В одном запросе можно использовать несколько DB LINK-соединений.

Пример

Выбираем сведения об автомобилях в автосалонах в разных базах.

```
SELECT 'LOCAL', mark, color, phonenum,...
 FROM auto - из локальной базы данных
UNION ALL
 SELECT 'Moscow', mark, color, phonenum
  FROM auto@london
UNION ALL
 SELECT 'Kiev', mark, color, phonenum
  FROM auto@tokyo
UNION ALL
 SELECT 'Voronej', mark, color, phonenum
  FROM auto@sydney
UNION ALL
 SELECT 'Minsk', mark, color, phonenum
  FROM auto@la;
```

На основе запросов с использованием механизма DATABASE LINK можно создавать представления.

Пример представления на основе предыдущего запроса:

```
CREATE VIEW all_customers (mark, color, phonenum, ...) AS
 SELECT 'LOCAL', mark, color, phonenum,...
  FROM auto - из локальной базы данных
UNION ALL
 SELECT 'Voronej', mark, color, phonenum
  FROM auto@sydney
UNION ALL
 SELECT 'Minsk', mark, color, phonenum
  FROM auto@la;
```

При работе с такими представлениями есть сложность: если одно из соединений перестает работать по техническим причинам (обрыв связи, остановка сервера), то все представление перестает функционировать, следует учитывать эту проблему в работе.

Для создания DATABASE LINK пользователь должен обладать правами CREATE PRIVATE DATABASE LINK или CREATE PUBLIC DATABASE LINK в локальной базе данных. Эти права необходимо запросить у администратора базы данных.

Вопросы учеников

Когда мы обращаемся через DATABASE LINK к MS SQL Server или My SQl, существуют ли в этом случае какие-либо дополнительные ограничения и сложности?

Да, как правило, таких ограничений множество. Например, различие в типах данных не позволяет выбирать данные из некоторых колонок из таблиц в удаленной базе данных.

С какими правами будет работать DATABASE LINK?

DATABASE LINK работает с правами той учетной записи, которая прописана при создании DATABASE LINK.

Вы написали, что можно создать DB LINK, который использует соединение из TNSNAMES. Поясните подробнее.

В ORACLE есть специальный файл с описанием всех соединений

TNSNAMES. ora

Расположение файла
C:\oraclexe\app\oracle\product\11.2.0\server\network\ADMIN\
Путь необходимо скорректировать с учетом вашего каталога установки.

Пример записи из TNSNAMES:

```
test1 =
 (DESCRIPTION =
  (ADDRESS = (PROTOCOL = TCP)(HOST = host1)(PORT = 1521))
  (CONNECT_DATA =
   (SERVER = DEDICATED)
   (SERVICE_NAME = SN1)
  )
 )
```

Здесь
- TEST1 – наименование TNS-соединения;
- HOST1 – хост, где установлен удаленный ORACLE СУБД;
- PORT – порт на для соединения с другой ORACLE СУБД;
- SN1 – имя сервиса на удаленной машине.

Создание соединения:

```
CREATE DATABASE LINK MTEST
connect to userdb1 identified by password1
USING 'sservise';
```

На удаленной, другой базе данных есть таблица с полем CLOB.

```
create table t1(id number, c1 clob);
```

При попытке обращения к данной таблице TABLE_CLOB таблице через DB_LINK (конечно, я для этого создал DB_LINK на удаленную базу) при выполнении запроса

```
select * from Table_Clob@dl_test_load
```

получается ошибка.

ORA-22992: невозможно использовать указатели LOB, выбранные из удаленных таблиц.
ORA-22992: CANNOT USE LOB LOCATORS SELECTED FROM REMOTE TABLE.

Как решить эту проблему?

Можно воспользоваться специальной временной таблицей.

```
create global temporary table t_clob(id number, c1 clob);

-- напишем следующий запрос вставки
insert into t_clob(id number, c1 clob) select id,c1 from Table_Clob@test1
```

Данные находятся в нашей временной таблице, в чем легко убедиться, выполнив запрос перед выполнением COMMIT.

```
select * from  t_clob;
```

Ошибка не возникает и наши данные загрузились как нам нужно.

Контрольные вопросы и задания для самостоятельного выполнения

1. Повторите материалы данного шага.
2. Поясните назначение механизма DATABASE LINK в ORACLE.
3. Поясните способы создания DATABASE LINK.
4. Объясните возможные проблемы при использовании этого механизма.
5. Напишите запрос с таблице CITY на другой, удаленной базе данных.
6. Напишите запрос, который добавляет запись к таблице CITY на удаленной базе данных.

Шаг 78. Индексы сложные, индексы по функции

Введение

Мы уже проходили индексы в начале книги. Тогда мы рассматривали простые индексы, их назначение и использование при проектировании базы данных. На самом деле индексы в базе данных ORACLE имеют гораздо большую применимость.

Настало время раскрыть тему индексов базы данных подробнее.

Теория и практика

Использование индексов в базе данных ORACLE. Повторим, что мы уже знаем об индексах.

• Индексы предназначены для быстрого доступа к строкам таблиц, для быстрого нахождения ассоциированных строк таблицы.

• Индексы позволяют находить строку с определенным значением столбца, просматривая при этом лишь небольшую часть общего объема строк таблицы.

• Правильное использование индексов сокращает до минимума количество обращений к данным базы данных, существенно ускоряет выборку.

• Индексы в базе данных ORACLE бывают нескольких типов: простые, реверсивные, битмап-индексы, индексы по функции.

Создание простого индекса

Простые индексы используются достаточно часто и имеют реализацию на основе B-деревьев, где используется концепция сбалансированного дерева.

Пример

```
CREATE INDEX idx_auto_color_id ON auto(color);
```

Реверсивный индекс

Индексы с реверсивным ключом – это, по сути, то же самое, что и простые индексы, за исключением того, что при индексации данных порядок просмотра меняется на противоположный.

Синтаксис такой же, как при создании обычных индексов, но добавляется спецкоманда REVERSE.

```
CREATE INDEX idx_auto_color_id ON auto(color)  REVERSE;
```

Используются эффективно в случаях, где выборка осуществляется чаще в обратном порядке.

Например:

• выбираем курсы валют за последние несколько дней, используем реверсивный индекс по дате;

• выбираем чаще слова из словаря с буквы Я, используем реверсивный индекс по словам;

• выбираем чаще данные с большим идентификатором, используем реверсивный индекс по идентификатору.

Битмап-индекс

Битовые индексы используют битовые карты для указания значения индексированного столбца.

Синтаксис

```
CREATE BITMAP INDEX  color_bmp_idx ON auto(color);
```

Эффективно работают, когда в колонке, на которую устанавливается индекс, присутствует небольшое количество, заранее определенный перечень значений. Например, перечень цветов автомобиля, перечень месяцев года, перечень названий валют.

Индекс по функции

Индексы на основе функций предварительно вычисляют значения функций по заданному столбцу и сохраняют результат в индексе.

Синтаксис

```
CREATE INDEX autofunc_idx ON AUTO(SUBSTR(mark,1,1));
```

Индекс по первой букве марки авто.

Индекс по функции представляет собой как бы невидимую колонку со значениями заданной функции, по которой осуществляется поиск.

Важные замечания

Следует учитывать: индексы занимают достаточно большой объем на дисковом пространстве.

Битмап-индексы следует применять только когда в колонке есть небольшой перечень значений; при большом количестве значений битовая матрица теряет свою эффективность при поиске данных и занимает значительный объем дискового пространства.

Битмап-индексы во время пересчета блокируют операции изменения или удаления данных из сегмента таблицы, где происходит пересчет индексов.

Если в таблице есть простой индекс и индекс по функции, база данных будет предпочтительнее использовать простой индекс, в этом случае необходимо указать СУБД ORACLE принудительно использовать индекс по функции.

Вопросы учеников

Что означает сжатый индекс?

Такой тип индексов называется индекс со сжатым ключом, в сжатом виде индексы занимают меньший объем дискового пространства. Для сжатия индекса используется директива compress.

```
CREATE INDEX emp_indx1 ON employees(ename)
TABLESPACE MY_INDEXES
COMPRESS 1;
```

Что такое реверсивный индекс?

Это специальный тип индексов для запросов, где дерево индекса строится в обратном порядке, то есть данные в запросе чаще считываются в порядке убывания.

```
CREATE INDEX reverse_idx ON man(yearold) REVERSE;
```

Контрольные вопросы и задания для самостоятельного выполнения

1. Повторите материалы данного шага.
2. Для таблицы AUTO создайте функциональный индекс, который бы включал номер телефона без скобок (REPLACE).
3. Для таблицы CITY создайте обычный индекс для колонки PEOPLES.
4. Для таблицы AUTO создайте битпмап-индекс для колонки COLOR.

Шаг 79. Корзина в ORACLE

Введение

Начиная с версии ORACLE 10g, в ORACLE появился новый механизм корзины RECYCLE BIN, объекты после удаления теперь можно восстановить.

Теория и практика

Механизм RECYCLE BIN в ORACLE поддерживается начиная с версии 10g.

Если данный механизм включен (по умолчанию), то при удалении таблицы с помощью конструкции DROP объект не удаляется, а просто переименовывается, оставаясь в табличном пространстве пользователя.

Посмотреть содержимое корзины можно с помощью запроса:

```
SELECT * FROM recyclebin;
корзина текущей схемы

SELECT * FROM dba_recyclebin;
корзина для всех схем

очистка корзины:
-- чистим свою
PURGE RECYCLEBIN;
-- чистим все
PURGE DBA_RECYCLEBIN;
```

Отключение корзины:

```
ALTER SESSION SET recyclebin=OFF;
```

ВОССТАНОВЛЕНИЕ ТАБЛИЦЫ ИЗ КОРЗИНЫ:

```
FLASHBACK TABLE my_dropped_table TO BEFORE DROP;
```

Пример

```
create table t as select owner, count(object_name) cnt from all_objects group by owner;
drop table t;
FLASHBACK TABLE t TO BEFORE DROP;
select * from t;
```

В ORACLE все сведения хранятся в словаре данных.

Словарь данных представляет собой МНОЖЕСТВО таблиц и представлений.

Исходный код пакетов и процедур хранится в таблице **SYS.SOURCE$**, к которой, в свою очередь, применим РЕТРОСПЕКТИВНЫЙ запрос вида AS OF TIMESTAMP.

Сперва определим OBJECT_ID объекта, который нам предстоит спасти:

```
SELECT owner,
    object_name,
    object_type,
    object_id
FROM dba_objects
WHERE object_name = 'DAMAGED_PKG';
```

Если это пакет, то нам понадобятся два OBJECT_ID: один для тела, а другой для определения. Теперь, зная идентификаторы объектов, получаем код и кладем его в таблицу:

```
CREATE TABLE scott.source_pkg
AS
  SELECT *
   FROM sys.source$
      AS OF TIMESTAMP
                          TO_DATE ('20.04.2016 17:00', 'dd.mm.yyyy hh24:mi')
   WHERE obj# IN (4383720, 4385485);
```

Таким образом можно извлечь исходный код процедуры или пакеты.

Важные замечания

ORACLE DATABASE извлекает все индексы, определенные в таблице, полученные из корзины, за исключением индексов BITMAP (индексы BITMAP не помещаются в корзину во время операции DROP TABLE).

База данных также извлекает из корзины все триггеры и ограничения, определенные в таблице, за исключением ограничений ссылочной целостности, которые ссылаются на другие таблицы.

Извлеченные из корзины индексы, триггеры и ограничения имеют сгенерированные имена. Поэтому рекомендуется выполнить запрос представления USER_RECYCLEBIN перед выполнением оператора FLASHBACK TABLE… TO BEFORE DROP, чтобы можно было переименовать полученные триггеры и ограничения в более пригодные для использования имена.

При удалении таблицы все журналы материализованного представления, определенные в таблице, также удаляются, но не помещаются в корзину. Следовательно, журналы с материализованным представлением необходимо будет создавать заново.

При удалении таблицы все индексы в таблице удаляются и помещаются в корзину вместе с таблицей. Если возникает нехватка пространства, база данных освобождает пространство из корзины, сначала удаляя индексы. В этом случае, когда вы вернетесь назад к таблице, вы не сможете вернуть все индексы, которые были определены в таблице.

Вопросы учеников

В каком случае будет невозможно восстановить таблицу из корзины?

Невозможно вернуть данные из таблицы назад, если она была очищена пользователем или базой данных ORACLE в результате некоторой операции восстановления.

Контрольные вопросы и задания для самостоятельного выполнения

1. Повторите материалы данного урока.
2. Попробуйте создать таблицу t11 на основе системного представления ALL_TABLES.
3. Удалите таблицу AUTO1.
4. Просмотрите корзину с помощью запроса.
5. Восстановите таблицу AUTO1.
6. Напишите запрос, который выведет все записи этой таблицы.

Шаг 80. Массовая операция вставки данных

Введение

Есть возможность вставить одновременно в несколько таблиц данные на основе заданного запроса, в том числе и сложного.

Теория и практика

Для множественной вставки данных в SQL ORACLE существует команда INSERT ALL.

```
INSERT ALL
WHEN (<condition>) THEN
 INTO <table_name> (<column_list>)
 VALUES (<values_list>)
WHEN (<condition>) THEN
 INTO <table_name> (<column_list>)
 VALUES (<values_list>)
ELSE
 INTO <table_name> (<column_list>)
 VALUES (<values_list>)
SELECT <column_list> FROM <table_name>;
```

Здесь
• CONDITION – условие на вставку записей в заданную таблицу;
• TABLE_NAME – наименование таблицы;
• VALUES_LIST – перечень значений из основного запроса SELECT.

Чтобы понять лучше, создадим готовый пример для лучшего понимания принципов работы.

Создадим три разных таблицы на основе системного представления **ALL_OBJECTS**.

```
create table obj_t -- table
as select owner, object_name from all_objects where 1=0;
```

```
create table obj_i -- index
as select owner, object_name from all_objects where 1=0;
```

```
create table obj_c -- cluster
as select owner, object_name from all_objects where 1=0;
```

Мы создали три таблицы для наименований таблиц, индексов и кластеров.
Воспользуемся командой INSERT ALL, чтобы вставить данные в эти три таблицы.

```
INSERT ALL
  WHEN object_type = 'TABLE' THEN
     INTO obj_t values (owner, object_name)
  WHEN object_type = 'INDEX'  THEN
    INTO obj_i values (owner, object_name)
  WHEN object_type = 'CLUSTER' THEN
    INTO obj_c values (owner, object_name)
  select owner, object_type, object_name from all_objects where rownum<9900;
select * from obj_t;
```

В зависимости от типа объекта записи добавляются в одну из трех таблиц: OBJ_T, OBJ_I, OBJ_C.

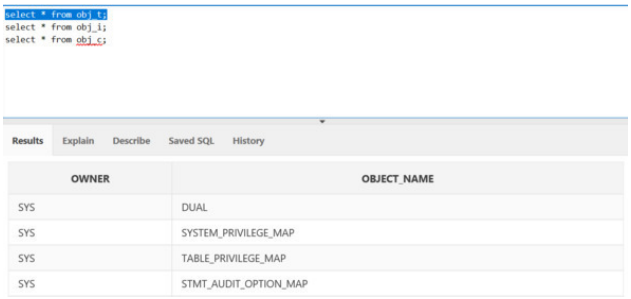


Рисунок 215. Использование INSERT ALL

```
select * from obj_i;
```

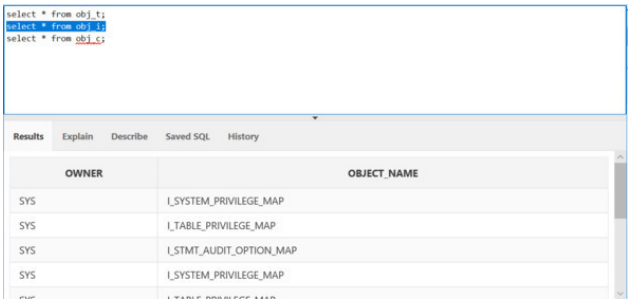


Рисунок 216. Использование INSERT ALL

```
select * from obj_c;
```

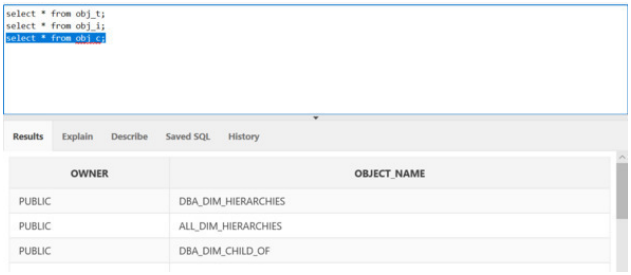


Рисунок 217. Использование INSERT ALL

В данном примере с помощью одного запроса:

- в таблицу OBJ_T были добавлены только наименования таблиц;
- в таблицу OBJ_T были добавлены только наименования индексов;
- в таблицу OBJ_T были добавлены только наименования кластеров.

Важные замечания

• INSERT ALL – операция модификации данных, после ее завершения необходимо выполнить COMMIT.
• INSERT ALL может использоваться для вставки данных только в таблицы, но не в представления или материализованные представления.
• Сумма столбцов во всех предложениях INSERT INTO не должна превышать 999.
• INSERT ALL нельзя использовать для таблиц с коллекциями.
• В INSERT ALL нельзя использовать последовательность SEQUENCE.

Вопросы учеников

Приведите, пожалуйста, пример на основе нашей учебной схемы.

Предположим, у нас есть таблицы CITYBIG, CITYSMALL такие же, как таблица CITY.

```
create table citybig as select * from city where 1=0;
create table citysmall as select * from city where 1=0;

INSERT ALL
  WHEN peoples > 1000000 THEN
    INTO citybig values (citycode, cityname, peoples)
  WHEN peoples <= 1000000 THEN
    INTO citysmall values (citycode, cityname, peoples)
  select citycode, cityname, peoples from city;
```

В таблице CITYBIG добавлены сведения о городах с населением больше 1 миллиона человек.

В таблице CITYSMALL добавлены сведения о городах с населением меньше 1 миллиона человек.

Контрольные вопросы и задания для самостоятельного выполнения

1. Повторить материал этого шага.
2. Создать три таблицы с полями, колонками «наименование объекта», «владелец».
3. В первую таблицу добавить из AUTO только машины владельца BMW.
4. Во вторую таблицу добавить из AUTO только машины владельца LADA.
5. В третью таблицу добавить из AUTO все остальные объекты, используя INSERT ALL.

День семнадцатый

Шаг 81. Массовое обновление данных

Введение

Существует интересная возможность обновления данных в связанных таблицах с помощью команды UPDATE. Это способ множественного обновления данных на основе запроса.

Теория и практика

Множественный UPDATE

Предположим, что нам необходимо заменить значения в таблице 1 данными из таблицы 2.

Можно, конечно, написать что-то вроде:

```
UPDATE t1 set t1.f1=select t2.f1 from t2 where t2.key1 = t1.key
```

В ORACLE SQL есть более легкий способ произвести подобную операцию.

```
UPDATE
(
SELECT
     t.c1, t.c2, s.c1 AS c1_new, s.c2 AS c2_new
FROM table1 t
INNER JOIN table2 s ON s.key=t.key
) tt  SET tt.c1=tt.c1_new, tt.c2=tt.c2_new

create table PHONES1
(
 PHONENUM VARCHAR2(48) not null primary key,
 NAME VARCHAR2(48)
);

create table PHONES
(
 PHONENUM VARCHAR2(48) not null primary key,
 NAME VARCHAR2(48)
);
```

Заполним данными и выполним UPDATE.

```
INSERT INTO PHONES1(PHONENUM, name) values ('495 1211133','Алексеев1');
INSERT INTO PHONES1(PHONENUM, name) values ('499 3311133','Михайлов1');
INSERT INTO PHONES1(PHONENUM, name) values ('917 12122sd3','Коробочкин1');
INSERT INTO PHONES1(PHONENUM, name) values ('499 33111133','Костин1');

INSERT INTO PHONES(PHONENUM, name) values ('495 1211133','Алексеев');
INSERT INTO PHONES(PHONENUM, name) values ('499 3311133','Михайлов');
INSERT INTO PHONES(PHONENUM, name) values ('917 12122sd3','Коробочкин');
INSERT INTO PHONES(PHONENUM, name) values ('499 33111133','Костин');
INSERT INTO PHONES(phonenum, name) values ('214 331das1133','Докучаев');
```

Обновление данных:

```
UPDATE (SELECT   t.phonenum AS phonenum1,      t.phonenum phonenum2,      s.name AS
name1, t.name AS name_new FROM phones1 t
INNER JOIN phones s ON s.phonenum=t.phonenum )  Tt SET tt.name1=tt.name_new
```

Данные были обновлены.

Важные замечания

Следует отметить, что для корректной работы такого способа UPDATE у обновляемой таблицы обязательно необходимо наличие первичного ключа, в нашем случае это PHONENUM.

Необходимо отметить, что такую команду обновления данных следует также завершать командой COMMIT.

Вопросы учеников

Как еще можно обновить данные таким способом?

В ORACLE SQL есть специальная команда MERGE, которая позволяет обновлять данные примерно так же. С этой командой мы познакомимся на следующем шаге.

Контрольные вопросы и задания для самостоятельного выполнения

1. Повторить материалы урока.
2. Выполнить самостоятельно упражнения из этого шага.

Шаг 82. Команда MERGE

Введение

Существует команда MERGE, которая одновременно выполняет вставку и обновление данных в одной таблице на основе данных из другой таблицы.

Теория и практика

MERGE – команда для обновления и вставки данных.

Позволяет дополнять и обновлять данные одной таблицы данными другой таблицы. При слиянии таблиц проверяется условие, и если оно истинно, то выполняется UPDATE, а если нет – INSERT. Причем нельзя изменять поля таблицы в секции UPDATE, по которым идет связывание двух таблиц.

Может работать также отдельно только как команда вставки или обновления.

Синтаксис

```
MERGE INTO таблица USING (select запрос) ON (condition)
 WHEN MATCHED THEN
 UPDATE SET column2  = value1 [, column2 = value2 ...]
 WHEN NOT MATCHED THEN
 INSERT (column1 [, column2 ...]) VALUES (value1 [, value2 ...]);
```

Здесь
- Таблица – таблица, для которой выполняется вставка-обновление;
- SELECT запрос – запрос, на основе которого обновляются, добавляются данные;
- Condition – условие связи двух таблиц.

Пример

Создадим следующую таблицу:

```
create table t_obj as
select object_type, count(1) objcount from all_objects where OWNER = 'SYS' group by
object_type
-- владелец, тип объекта, кол объектов данного типа
-- очистим поле
update t_obj  set objcount = 0;
```

Удалим из таблицы строки, где OBJECT_TYPE = TABLE или INDEX.

```
delete  t_obj where object_type in ('TABLE', 'INDEX');
```

Обновим данные в таблице T_OBJ на основе основного запроса.
Удалим из таблицы строки, где OBJECT_TYPE = TABLE или INDEX.

```
delete  t_obj where object_type in ('TABLE', 'INDEX');
```

Только вставка

```
merge into t_obj mt using (select owner, object_type, count(1) as objcount from all_objects
group by owner, object_type) t
on (mt.owner = t.owner and mt.object_type = t.object_type )
WHEN NOT MATCHED THEN
INSERT (mt.owner, mt.object_type, mt.objcount)
   VALUES (t.owner, t.object_type, t.objcount);
```

Только обновление

```
MERGE INTO t_obj mt using (select owner, object_type, count(1) as objcount from all_objects
group by owner, object_type) t
on (mt.owner = t.owner and mt.object_type = t.object_type )
WHEN MATCHED THEN
UPDATE SET mt.objcount = t.objcount;
```

Пример 2
Разберем работу оператора MERGE.

```
create table person(tabn number primary key, name varchar2(10), age number);
insert into person values (10 , 'Таня', 22); -- табельный номер , имя , возраст
insert into person values (11 , 'Саша', 9 );
insert into person values (12 , 'Вася', 30);
insert into person values (13 , 'Дима', 39);
insert into person values (14 , 'Олег', 51);
insert into person values (15 , 'Витя', 55);
insert into person values (16 , 'Лена', 67);
insert into person values (17 , 'Маня', 44);
insert into person values (18 , 'Даша', 12);
insert into person values (19 , 'Маша', 24);
insert into person values (20 , 'Миша', 10);
insert into person values (21 , 'Миша', 42)
```

Создадим таблицу PERSON1.

```
create table Person1 as select * from Person;
```

На основании PERSON обновим часть записей в PERSON и удалим часть из них для актуальности примера.

```
UPDATE person SET age = 55 where tabn in (10,11,12,13,14,15);
delete person where tabn in (15,18,20);
```

Выполним команду MERGE

```
MERGE INTO person p
  USING (   SELECT tabn, name, age FROM person1) p1
  ON (p.tabn = p1.tabn)
  WHEN MATCHED THEN UPDATE SET p.age = p1.age
  WHEN NOT MATCHED THEN INSERT (p.tabn, p.name, p.age)
   VALUES (p1.tabn, p1.name, p1.age)
```

Записи в PERSON будут обновлены и дополнены записями из PERSON1.

Пример 3

```
DELETE auto1;
```

Очистим таблицу AUTO1.

```
INSERT INTO auto1 SELECT * FROM auto WHERE rownum<5;
```

Добавим четыре записи из таблицы AUTO.

```
UPDATE auto1 SET mark = '*';
```

Обновим данные в поле MARK.

```
SELECT * FROM auto1;
```

REGNUM	MARK	COLOR	RELEASEDT	PHONENUM
111119	*	СИНИЙ	01/01/2017	9213333331
111115	*	КРАСНЫЙ	01/01/2013	9173333334
111121	*	СИНИЙ	01/01/2009	9173333332
111122	*	СИНИЙ	01/01/2011	9213333336

Рисунок 218. Выбор данных из AUTO1

```
MERGE INTO auto1 a1
  USING ( SELECT * FROM auto) a
  ON (a1.regnum = a.regnum)
  WHEN MATCHED THEN UPDATE SET a1.mark = a.mark
  WHEN NOT MATCHED THEN INSERT (a1.regnum, a1.mark, a1.color, a1.releasedt,
a1.phonenum)
   VALUES (a.regnum, a.mark, a.color, a.releasedt, a.phonenum);
```

Обновим данные

```
SELECT * FROM auto1 a1;
```

REGNUM	MARK	COLOR	RELEASEDT	PHONENUM
111119	LADA	СИНИЙ	01/01/2017	9213333331
111115	VOLVO	КРАСНЫЙ	01/01/2013	9173333334
111121	AUDI	СИНИЙ	01/01/2009	9173333332
111122	AUDI	СИНИЙ	01/01/2011	9213333336
1234	LADA	RED	-	-
111116	BMW	СИНИЙ	01/01/2015	9173333334
128877655	LADA	КРАСНЫЙ	01/01/2001	1231144444
111114	LADA	КРАСНЫЙ	01/01/2008	9152222221

Рисунок 219. Выбор данных из таблицы AUTO1

Выберем данные из таблицы AUTO1.

Важные замечания

MERGE является DML-командой, командой управления данными.
Эту команду следует завершать командой COMMIT.
Вместо таблицы INTO можно указать запрос.

Пример

```
MERGE INTO  ( SELECT * FROM auto1) a1
  USING ( SELECT * FROM auto) a
  ON (a1.regnum = a.regnum)
  WHEN MATCHED THEN UPDATE SET a1.mark = a.mark
  WHEN NOT MATCHED THEN INSERT (a1.regnum, a1.mark, a1.color, a1.releasedt,
a1.phonenum)
   VALUES (a.regnum, a.mark, a.color, a.releasedt, a.phonenum);
```

Вопросы учеников

Как добавить или обновить некоторое одиночное значение с помощью MERGE?

Нет ничего проще, можно использовать запрос к таблице DUAL.

Пример

```
merge into city1 using (
   select
      '2' citycode,
      'город 1' cityname
   from
      dual
) t1 on (
   city1.citycode=t1.citycode
)
when matched then
   update set city1.cityname= t1.cityname
when not matched then
   insert (citycode, cityname) values (t1.citycode, t1.cityname);
```

Контрольные вопросы и задания для самостоятельного выполнения

1. Повторите материалы данного шага.
2. Поясните работу оператора MERGE.
3. Обновите данные в таблице MAN1 на основе таблицы MAN с использованием MERGE.
4. Обновите данные в таблице CITY1 на основе таблицы CITY с использованием MERGE.

Шаг 83. Транзакции и блокировки

Введение

Для данного шага нам потребуется установить дополнительное программное обеспечение (ORACLEXE, PL SQL DEVELOPER). Подробнее о том, как устанавливать данные программы, описано в шаге 51 настоящей книги.

Данный шаг посвящен транзакционной модели и работе ORACLE SQL в многопользовательском режиме.

Теория и практика

В ORACLE СУБД используется транзакционная модель.

Изменения в данных, выполненные с помощью операторов INSERT, UPDATE, MERGE, DELETE, необходимо зафиксировать с помощью COMMIT или UPDATE.

Но работа в команде, работа одновременно нескольких пользователей подразумевает возможность одновременного редактирования данных.

Пример для демонстрации работы в многопользовательском режиме.

Создадим таблицу MAN5.

```
CREATE TABLE man5(id NUMBER PRIMARY KEY, name VARCHAR(30));
```

Откроем две программы SQLDEVELOPER независимо друг от друга в двух разных окнах. Либо можно открыть новый SQL WORKSHEET, воспользовавшись кнопкой NEW SHARED SQL WORKSHEET.

Подключимся к пользователю SYS в обоих окнах, как указано в шаге 51.

Заполним новую таблицу данными.

Выполним следующие команды в одном окне SQLDEVELOPER.

```
INSERT INTO  man5 VALUES (1, 'Макс');
INSERT INTO  man5 VALUES(2, 'Андрей');
INSERT INTO  man5 VALUES(3, 'Дима');
```

Переключим на другое окно программы SQLDEVELOPER.

Напишем запрос:

```
SELECT * FROM  man5
```

Запрос вернул пустое множество, переключим окно SQLDEVELOPER.

Уже в этом окне напишем запрос.

```
SELECT * FROM  man5
```

Запрос вернул три строки.
Выполним команду COMMIT;
Переключимся на второе окно SQL.

```
SELECT * FROM man5
```

Запрос вернул три строки.

Транзакция – неделимая последовательность из нескольких SQL-команд.
Пользователь делает изменения в своей транзакции, до выполнения команды COMMIT транзакция не фиксирует изменения в общей базе.

Таким образом, пока изменения не были зафиксированы командой COMMIT, только первый пользователь мог видеть изменения данных, которые он сделал; остальные пользователи базы данных этих изменений не видели.
Команда COMMIT зафиксировала изменения, и они стали доступны другим пользователям базы данных.
Предположим, что пользователь 1 отредактировал данные в некоторой таблице, но не завершил операцию редактирования командой COMMIT;

Одновременно другой пользователь также отредактировал данные и выполнил команду COMMIT; получается, что у пользователя 1 теперь будут некорректные данные?
Для того чтобы избежать подобной коллизии, ORACLE использует механизм блокировок.
Продемонстрируем работу данного механизма.
В первом окне обновим данные следующим запросом:

```
UPDATE man5 SET name = 'Олег';
```

Во втором окне так выполним соответствующую команду:

```
UPDATE man5 SET name = 'Олег' ;
Commit;
```

Это приводит к длительному зависанию программы и ошибке, данные в таблице не изменились.

Подобная ситуация произошла потому, что при выполнении команд обновления данных в первой сессии установлены блокировки на записи таблицы.

Для того чтобы избежать подобной ситуации, следует предварительно выполнить запрос со специальной директивой FOR UPDATE NO WAIT для таблицы, записи к которой следует обновить.

```
SELECT * FROM  tablename FOR UPDATE NO WAIT;
```

Повторим наш пример.
В первом окне обновим данные следующим запросом:

```
UPDATE man5 SET name = 'Олег' ;
```

Во втором окне так выполним соответствующую команду:

```
SELECT * FROM  man5 FOR UPDATE NOWAIT;
```

Результат:
ORA-00054: resource busy AND acquire with NOWAIT specified or timeout expired
00054. 00000 – «resource busy AND acquire with NOWAIT specified or timeout expired»
*Cause: INterested resource IS busy.
*Action: Retry if necessary or INcreASe timeout.

Важные замечания

Команды в языке SQL подразделяются на два типа: команды изменения данных и команды описания данных.

К первому типу команд (команды изменения данных) относятся операторы выбора (SELECT), вставки (INSERT, INSERT ALL), обновления данных (UPDATE, MERGE), удаления данных (DELETE).

К командам описания данных относятся команды создания структуры таблиц, изменения структуры таблиц (CREATE TABLE, MODIFY, ALTER TABLE), команды создания индексов и ограничений (CREATE INDEX, CHECK).

Транзакционная модель распространяется только на команды изменения данных, то есть команды вставки, обновления, удаления данных. То есть данные команды должны завершаться специальными операторами COMMIT или ROLLBACK для фиксации изменений в базе данных или отката изменений в базе данных.

Все изменения в базе данных, которые произведены с помощью операторов описания данных (CREATE TABLE, ALTER TABLE…), применяются немедленно.

Также к операторам описания данных можно отнести команду TRUNCATE, которая применяется для быстрой очистки таблицы.

Вопросы учеников

Как посмотреть блокировки с помощью запроса?

Вот пример такого запроса:

```
SELECT * FROM lock WHERE type = 'TX';
```

Как посмотреть запросом, кто кого блокирует?

Воспользуемся для этого специальным запросом:

```
select 'Session with sid '||a.sid||' blocked  sid '||b.sid
 from v$lock a, v$lock b
where a.BLOCK = 1   and b.REQUEST > 0  and a.ID1 = b.ID1  and a.ID2 = b.ID2;
```

Какие права требуются при выполнении данного запроса?

Для этого запроса требуются привилегии на системное представление V$LOCK, которые может выдать администратор базы данных.

Что такое DEADLOCK?

Это нестандартная ситуация в базе данных.

Такая ситуация возникает, когда одна сессия блокирует вторую сессию, а первая сессия, в свою очередь, блокирует первую, то есть это ситуация, когда две сессии взаимно блокируют друг друга.

Пример подобной ситуации:

```
В первом окне напишем

UPDATE man5 SET name = 'Олег' where id=1;
Во втором окне напишем две команды
UPDATE man5 SET name = 'Олег' where id=2;
Выполнено
UPDATE man5 SET name = 'Олег' where id=1;
Зависание блокировка

Теперь в первом окне напишем
UPDATE man5 SET name = 'Олег' where id=1;
UPDATE man5 SET name = 'Олег' where id=3;
Зависание, две сессии взаимно блокируют друг друга
Через некоторое время
Error report -
ORA-00060: deadlock detected while waiting for resource
```

Как разрешить ситуацию DEADLOCK?

Данную ситуацию сможет корректно разрешить администратор базы данных.

Контрольные вопросы и задания для самостоятельного выполнения

1. Что такое DEADLOCK?
2. Как разрешается ситуация DEADLOCK?
3. Как посмотреть блокировки, кто кого блокирует?
4. Какой избежать ситуации с блокировками?

Шаг 84. Режим SERIALIZABLE

Введение

Как мы убедились на предыдущем шаге, разные пользователи могут видеть разные данные. Однако в СУБД ORACLE есть возможность, чтобы пользователь всегда видел только те данные в таблицах, которые были с начала его сессии.

Теория и практика

Такой режим (уровень изоляции) называется SERIALIZABLE. Для того чтобы включить этот режим, используется команда:

```
Alter session set isolation_level=serializable;
```

Следующий пример показывает отличие режима SERIALIZABLE от стандартного режима эксплуатации СУБД уровня изоляции READ COMMITTED.

Откроем в двух разных окнах программу SQL DEVELOPER (или создадим новый WORKSHEET), подключимся к схеме SYS как администратор. Как создать такое подключение, подробно описано в шаге 51.

На прошлом шаге мы создали таблицу MAN5.

В одном окне выполним следующие команды:

```
INSERT INTO man5 VALUES(20, 'Олег');
INSERT INTO man5 VALUES(21, 'Влад');
INSERT INTO man5 VALUES(22, 'Саша');
Commit;
```

Напишем запрос:

```
SELECT * FROM man5 WHERE id>20;
```

Запрос вернул 3 строки.

Теперь перейдем во второе окно.
Напишем запрос:

```
SELECT * FROM man5 WHERE id>20;
```

Очистим нашу таблицу с помощью команды DELETE.
Рассмотрим, как работает режим SERIALIZABLE;
Во втором окне задействуем режим изоляции данных SERIALIZABLE.
ALTER SESSION SET ISOLATION_LEVEL=SERIALIZABLE;

В первом окне:

```
NSERT INTO man5 VALUES(30, 'Таня');
INSERT INTO man5 VALUES(31, 'Лена');
INSERT INTO man5 VALUES(32, 'Вадим');
INSERT INTO man5 VALUES(33, 'Ира');
Commit;
SELECT * FROM man5 WHERE id>20;
```

Запрос возвращает 7 строк.

Во втором окне, там, где включен режим (уровень изоляции) SERIALIZABLE, пишем запрос:

```
SELECT * FROM man5 WHERE id>20;
```

Запрос возвращает 3 строки.

В первом окне пишем команду удаления данных:

```
DELETE man5 WHERE id>20;
SELECT * FROM man5 WHERE id>20;
COMMIT;
```

Данный запрос возвращает нам пустое множество.

Переходим в окно, где используется уровень изоляции SERIALIZABLE.

```
SELECT * FROM man5 WHERE id>20;
```

По-прежнему возвращает 3 строки.

Данный режим обеспечивает изолированный собственный набор данных, на который не могут повлиять внешние сессии.

Важные замечания

Интересен и обратный пример. Закроем и заново откроем два окна SQL DEVELOPER.
Очистим таблицу MAN5:

```
Delete man5;
Commit;
```

В первом окне:

```
INSERT INTO man5 VALUES(20, 'Олег');
INSERT INTO man5 VALUES(21, 'Влад');
INSERT INTO man5 VALUES(22, 'Саша');
Commit;
```

Во втором окне перейдем в режим SERIALIZABLE.
Введем команду:

```
INSERT INTO man5 VALUES(32, 'Вадим');
INSERT INTO man5 VALUES(33, 'Ира');
Commit;
SELECT * FROM man5 WHERE id>20;
```

Вернет 5 записей.
Переходим в первое окно:

```
SELECT * FROM man5 WHERE id>20;
```

3 записи.
То есть при уровне изоляции SERIALIZABLE также справедливо утверждение, что данные во внешних сессиях так же изолированы от изменений данных в сессии SERIALIZABLE.

Вопросы учеников

Для чего используется на практике уровень изоляции SERIALIZABLE?

Обычно используется при выполнении сложных запросов, для построения отчетов, где необходима точность вычислений и данные в базе постоянно изменяются.

Например, отчет по платежам строится час, один из запросов вычисляет итоговую сумму, другие запросы вычисляют подытоги; если таблица платежей изменится, поступают новые платежи, тогда цифры итогов и подытогов разойдутся, не будут совпадать. В этом случае целесообразно использовать режим SERIALIZABLE, чтобы данные ко времени начала построения отчета не отличались от данных к окончанию построения отчета.

При работе с уровнем я получаю ошибку ORA-8177, почему так происходит и как избежать данной ошибки?

Данная ошибка возникает при изменении данных одной таблицы разными сессиями в режиме *SERIALIZABLE.*

Уровень изоляции SERIALIZABLE захватывает слот в списке заинтересованных транзакций. Если ORACLE-слот недоступен, то возникает ошибка ORA-8177. Доступные слоты ITL контролируются параметрами INITRANS и MAXTRANS, которые задаются при создании таблицы.

Попробуйте увеличить параметры INITRANS и MAXTRANS и пересоздать таблицы, с которыми работает сессия SERIALIZABLE.

Другие СУБД поддерживают уровень изоляции SERIALIZABLE?

MS SQL поддерживает данный уровень изоляции, MYSQL, PostgreSQL поддерживают в ограниченном режиме, этот момент необходимо уточнить в документации к этим СУБД.

Какие еще бывают уровни изоляции?

Использую материал из Википедии.

Read unCOMMITted (чтение незафиксированных данных) – низший (первый) уровень изоляции. Он гарантирует только отсутствие потерянных обновлений. Если несколько параллельных транзакций пытаются изменять одну и ту же строку таблицы, то в окончательном варианте строка будет иметь значение, определенное всем набором успешно выполненных транзакций. При этом возможно считывание не только логически несогласованных данных, но и данных, изменения которых еще не зафиксированы.

READ COMMITTED (чтение фиксированных данных). Большинство промышленных СУБД, в частности Microsoft SQL Server, PostgreSQL и ORACLE, по умолчанию используют именно этот уровень. На этом уровне обеспечивается защита от чернового, «грязного» чтения, тем не менее в процессе работы одной транзакции другая может быть успешно завершена и сделанные ею изменения зафиксированы. В итоге первая транзакция будет работать с другим набором данных.

REPEATABLE READ (повторяемость чтения) – уровень, при котором читающая транзакция «не видит» изменения данных, которые были ею ранее прочитаны. При этом никакая другая транзакция не может изменять данные, читаемые текущей транзакцией, пока та не окончена.

Контрольные вопросы и задания для самостоятельного выполнения

1. Повторите материалы данного шага.
2. Объясните, как работает сессия при уровне изоляции SERIALIZABLE.
3. Объясните, как используется режим SERIALIZABLE на практике.
4. Объясните причину ошибки *ORA-8177*.

Шаг 85. Материализованные представления

Введение

Материализованные представления – это объект базы данных, который содержит результат выполнения запроса. Отличие материализованных представлений от обычных представлений в том, что рассчитанные данные в материализованных представлениях статичны, физически располагаются в табличном пространстве и обновляются по заданному алгоритму.

Материализованные представления используются при построении отчетности и позволяют ускорить построение отчетов с помощью запросов SQL в несколько десятков раз, позволяя за считаные секунды оперировать миллионами записей.

Также использование материализованных представлений позволяет заранее вычислить подытоги финансовых операций при построении отчетности, что многократно увеличивает производительность.

Теория и практика

Материализованные представления появились в арсенале средств управления данными в базе данных ORACLE, начиная с ORACLE 8i. До этого подобные специальные конструкции назывались SNAPHSHOT. В материализованных представлениях предварительно вычисляются и сохраняются результаты запроса к базе данных, в этом запросе при желании могут быть использованы соединения, агрегирования, результаты сложных математических расчетов.

Создание материализованного представления.

Синтаксис:

```
CREATE MATERIALIZED mv_name BUILD OPTION
REFRESH OPTION
ENABLE QUERY REWRITE
AS SELECT * FROM table(SELECT Query )
```

Рассмотрим некоторые важные конструкции оператора CREATE MATERIALIZED VIEW.

BUILD OPTION может принимать следующие значения:

• BUILD IMMEDIATE позволяет сразу заполнить материализованное представление данными из запроса; значение по умолчанию;

• BUILD DEFERRED разрешает загрузить данные в материализованное представление позднее, в указанное время.

REFRESH OPTION

REFRESG FAST для фиксации всех изменений главных таблиц – необходимы журналы материализованных представлений.

COMMIT конструкции REFRESH указывает на то, что все зафиксированные изменения главных таблиц распространялись на материализованное представление немедленно после фиксации этих изменений.

ENABLE QUERY REWRITE показывает оптимизатору ORACLE переписать все запросы с использованием материализованных представлений вместо лежащих в основе представления таблиц.

AS SELECT * FROM – запрос, на основании которого строится материализованное представление.

Для обновления данных в материализованном представлении может быть использована следующая команда:

```
begin
 dbms_mview.refresh(viewname);
end;
```

Примеры работы с материализованными представлениями.

Создадим две таблицы, заполним данными и создадим на основе этих таблиц материализованное представление.

```
create table tabm as select level * 10 as id, 'l'||to_char(mod(level, 30)) as txt from dual
connect by  level<10001 ;
create table tabp as select level as id , level * 10 as pid from dual connect by  level<100001 ;
```

Создание журналов для материализованного представления.

Журнал материализованного представления необходим для корректного быстрого обновления (FAST) для материализованного представления.

В нашем случае для этого нужно создание журнала для каждой из двух таблиц.

```
CREATE MATERIALIZED VIEW LOG
ON tabm WITH SEQUENCE, ROWID
(id,txt)
INCLUDING NEW VALUES;

CREATE MATERIALIZED VIEW LOG ON tabp
WITH SEQUENCE, ROWID
(id, pid)
INCLUDING NEW VALUES;
```

Создаем материализованное представление на основе этих двух таблиц.

```
CREATE MATERIALIZED VIEW test_mv
BUILD IMMEDIATE
REFRESH FAST
ENABLE QUERY REWRITE
AS
SELECT tabm.id, tabm.txt as txt, count(*) as cntm, count(tabm.txt) cnt FROM tabm INNER JOIN
tabp On tabm.id = tabp.pid GROUP bY tabm.id, tabm.txt;
```

Обратите внимание, что для корректного обновления материализованного представления методом FAST нам необходимо, при использовании операций GROUP BY в данном представлении, включать в запрос COUNT (*), COUNT (TABM. txt), то есть COUNT (*) и COUNT по всем колонкам, которые выводятся в материализованном представлении.

Выполним запрос к материализованному представлению.

```
select * from test_mv
```

ID	TXT	CNTM	CNT
270	I27	1	1
430	I13	1	1
450	I15	1	1
670	I7	1	1
720	I12	1	1

Рисунок 220. Результат выполнения запроса к материализованному представлению

Выполним обновление одной из таблиц.

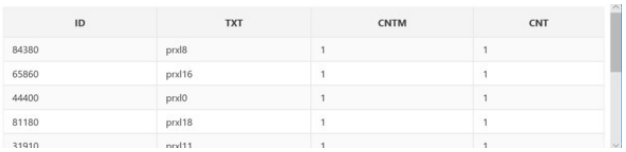
```
update tabm set txt='prx'||txt;
commit;
```

Выполним обновление материализованного представления.

```
begin
 dbms_mview.refresh('test_mv');
end;
```

Повторим запрос.

```
select * from test_mv
```



ID	TXT	CNTM	CNT
84380	prxl8	1	1
65860	prxl16	1	1
44400	prxl0	1	1
81180	prxl18	1	1
31910	prxl11	1	1

Рисунок 221. Результат выполнения запроса к материализованному представлению

Для материализованных представлений могут быть созданы индексы:

```
create index idx_mv on test_mv(txt);
```

Синтаксис создания индекса не отличается от создания индекса для таблицы.

Существует два основных способа обновления материализованных представлений:

• Полная перестройка (REFRESH FAST). В этом случае при каждом обновлении данных происходит выполнение запроса, на основе которого построено представление, и данные полностью перегружаются.

• Инкрементальное обновление (build immediate refresh complete). В этом случае, когда обновляется любая из таблиц, из запроса в материализованном представлении создается специальный журнал изменений и изменения представления распространяются только для данных, которые изменились.

Важные замечания

Для материализованных представлений можно использовать следующие возможности:
- создавать индексы;
- использовать секционирование в материализованных представлениях;
- создавать материализованные представления на основе секционированных таблиц.

В материализованном представлении допустимы различные типы агрегации вроде SUM, COUNT (*), AVG, MIN и MAX. В определении материализованного представления также можно использовать соединения множества таблиц.

При создании материализованных представлений существуют следующие ограничения.

Следующие конструкции SELECT не могут быть использованы при работе с материализованными представлениями: **DISTINCT, FIRST, HAVING, ORDER BY, UNION, UNION ALL, MINUS, INTERSECT, JOIN**.

Если вы считаете, что материализованное представление не нужно, можете уничтожить его с помощью оператора DROP MATERIALIZED VIEW:

```
DROP MATERIALIZED VIEW sales_sum_mv;
```

Вопросы учеников

Какие права необходимы для работы с материализованными представлениями?

Необходимые права:

```
grant create materialized view
grant query rewrite
```

Можно ли использовать в материализованных представлениях функции USER, CURRENT_USER, SESSION_USER?

Использование функций USER, CURRENT_USER, SESSION_USER в материализованных представлениях не разрешено.

Можно ли в материализованных представлениях использовать последовательности (SEQ)?

Нет, использовать NEXTVAL и CURRVAL в материализованных представлениях нельзя.

Приведите пример, для чего можно использовать материализованные представления на практике.

Например, ежедневная банковская отчетность: ежедневно обновляются материализованные представления, где считаются предварительные итоги, после чего с их использованием строится отчетность.

Контрольные вопросы и задания для самостоятельного выполнения

1. Повторите материалы этого шага.
2. Поясните назначение материализованных представлений.
3. Можно ли в материализованных представлениях использовать GROUP BY и SUM?
4. Объясните, для чего нужны журналы материализованных представлений.
5. Создайте материализованное представление (FAST) для таблиц MAN, AUTO, чтобы оно включало имя человека и количество автомобилей, которое купили люди с этим именем.

День восемнадцатый

Шаг 86. Контекст сеанса

Введение

В каждой сессии есть возможность использовать специальные структуры данных, именуемые контекстом.

Контексты – это набор данных вида «параметр – значение». Контекст предоставляет дополнительные возможности для приложений, использующих СУБД ORACLE.

Теория и практика

Контекст – это предопределенный набор параметров и значений, создаваемый в рамках определенной сессии. Несколько таких наборов «параметр – значение» объединяются с помощью структуры, называемой контекстом.

Задается значение контекста с помощью процедуры DBMS_SESSION.SET_CONTEXT.

Функция SYS_CONTEXT позволяет получить значение заданного контекста в рамках сессии.

Для создания контекста используется специальная команда CREATE CONTEXT.

Сначала следует создать специальную процедуру для управления контекстом.

```
CREATE OR REPLACE PROCEDURE set_mycontext_value ( par IN VARCHAR2, val IN VARCHAR2 )
AS BEGIN  DBMS_SESSION.SET_CONTEXT ( 'myctx', par, val ); END;

CREATE OR REPLACE CONTEXT myctx USING set_mycontext_value;


BEGIN
 set_mycontext_value ('paramuser', 'abc');
END;

SELECT SYS_CONTEXT ('myctx', 'paramuser' ) FROM dual;
```

Существует специальный системный контекст CLIENTCONTEXT.

CLIENTCONTEXT при открытии сеанса задает информацию, дополнительную к заданным сведениям (текущая схема, пользователь, тип авторизации, домен, хост, наименование DB, имя пользователя информационной системы…), контекста USERENV.

Пример

```
SELECT
 SYS_CONTEXT ( 'userenv', 'AUTHENTICATION_TYPE' ) authent
, SYS_CONTEXT ( 'userenv', 'CURRENT_SCHEMA' )     curr_schema
, SYS_CONTEXT ( 'userenv', 'CURRENT_USER' )       curr_user
, SYS_CONTEXT ( 'userenv', 'DB_NAME' )          db_name
, SYS_CONTEXT ( 'userenv', 'DB_DOMAIN' )        db_domain
, SYS_CONTEXT ( 'userenv', 'HOST' )           host
, SYS_CONTEXT ( 'userenv', 'IP_ADDRESS' )       ip_address
, SYS_CONTEXT ( 'userenv', 'OS_USER' )          os_user
FROM dual;
```

Результат для моего компьютера:

AUTHENT	CURR_SCHEMA	CURR_USER	DB_NAME	DB_DOMAIN	HOST	IP_ADDRESS	OS_USER
DATABASE	SQLADV	SQLADV	XE	(null)	DESKTOP-NEABUF1	127.0.0.1	XXX

Данная информация достаточно часто используется при разработке ORACLE-приложений.

Существует еще один специальный контекст – CLIENTCONTEXT, клиентский контекст.

Данный контекст не требует создания и позволяет сессии пользователя задавать собственные атрибуты и значения. Этот контекст разрешает устанавливать значения заранее, при открытии соединения с базой данных пользовательским приложением.

CLIENTCONTEXT можно считывать функцией SYS_CONTEXT, а также изменять и создавать новые атрибуты.

Пример

```
begin
 DBMS_SESSION.SET_CONTEXT ( 'CLIENTCONTEXT', 'paramuser', 'valuser' );
end;

SELECT SYS_CONTEXT ('CLIENTCONTEXT', 'paramuser' ) FROM dual;
```

– VALUEr

```
begin
 DBMS_SESSION.SET_CONTEXT ( 'CLIENTCONTEXT', 'paramuser', 'valuser1' );
end;

SELECT SYS_CONTEXT ('CLIENTCONTEXT', 'paramuser' ) FROM dual;
```

– valuser1

Важные замечания

DBMS_SESSION содержит ряд других подпрограмм для работы с контекстами.

Для создания контекста необходимы права CREATE ANY CONTEXT, они могут быть выданы администратором системы.

Вопросы учеников

Можно ли контекст использовать для других типов данных, а не только для строк?

Да, вот пример использования числа:

```
begin
 DBMS_SESSION.SET_CONTEXT( 'CLIENTCONTEXT', 'paramuser', 343435 );
end;

SELECT SYS_CONTEXT ('CLIENTCONTEXT', 'paramuser' ) FROM dual;
```

Какие еще данные можно извлечь из системного контекста?

Вот список информации, которую можно получить из системного контекста:

ACTION Возвращает позицию в модуле
AUDITED_CURSORID Возвращает идентификатор курсора SQL, который вызвал аудит
AUTHENTICATED_IDENTITY Возвращает идентификатор, использованный при аутентификации
AUTHENTICATION_DATA Данные аутентификации
AUTHENTICATION_METHOD Возвращает метод аутентификации
AUTHENTICATION_TYPE Описывает, как пользователь прошел проверку подлинности. Может принимать одно из следующих значений: База данных, ОС, Сеть или Прокси
BG_JOB_ID Если сеанс был установлен фоновым процессом Oracle, этот параметр вернет идентификатор задания. В противном случае он вернет NULL.
CLIENT_IDENTIFIER Возвращает идентификатор клиента (глобальный контекст)
CLIENT_INFO Информация о сеансе пользователя
CURRENT_BIND Связать переменные для детального аудита
CURRENT_SCHEMA Возвращает схему по умолчанию, используемую в текущей схеме
CURRENT_SCHEMAID Возвращает идентификатор схемы по умолчанию, используемой в текущей схеме
CURRENT_SQL Возвращает SQL, который вызвал событие аудита
CURRENT_SQL_LENGTH Возвращает длину текущего оператора SQL, который вызвал событие аудита
CURRENT_USER Имя текущего пользователя
CURRENT_USERID ИД пользователя текущего пользователя
DB_DOMAIN Домен базы данных из параметра инициализации DB_DOMAIN
DB_NAME Имя базы данных из параметра инициализации DB_NAME
DB_UNIQUE_NAME Имя базы данных из параметра инициализации DB_UNIQUE_NAME
ENTRYID Доступный идентификатор записи аудита
ENTERPRISE_IDENTITY Возвращает личность пользователя в масштабе предприятия
EXTERNAL_NAME Внешний пользователь базы данных
FG_JOB_ID Если сеанс был установлен клиентским процессом переднего плана, этот параметр вернет идентификатор задания. В противном случае он вернет NULL.
GLOBAL_CONTEXT_MEMORY Номер, используемый в глобальной области системы контекстом глобального доступа
GLOBAL_UID Глобальный идентификатор пользователя из Oracle Internet Directory для входа в систему безопасности предприятия. Возвращает NULL для всех остальных имен входа.
HOST Имя хост-машины, с которой клиент подключился
IDENTIFICATION_TYPE Возвращает способ создания схемы пользователя
INSTANCE Идентификационный номер текущего экземпляра
INSTANCE_NAME Имя текущего экземпляра
IP_ADDRESS IP-адрес компьютера, с которого клиент подключился
ISDBA Возвращает TRUE, если у пользователя есть привилегии DBA. В противном случае он вернет FALSE.
LANG Сокращение ISO для языка
ЯЗЫК Язык, территория и характер сеанса. В следующем формате: language_territory.characterset
MODULE Возвращает имя приложения, установленное через пакет DBMS_APPLICATION_INFO или OCI
NETWORK_PROTOCOL Используется сетевой протокол
NLS_CALENDAR Календарь текущей сессии
NLS_CURRENCY Валюта текущего сеанса
NLS_DATE_FORMAT Формат даты для текущего сеанса
NLS_DATE_LANGUAGE Язык, используемый для дат
NLS_SORT BINARY или основа лингвистической сортировки
NLS_TERRITORY Территория текущего
OS_USER Имя пользователя ОС для пользователя, вошедшего в систему

Информация о SYSCONTEXT

```
POLICY_INVOKER Призыватель функций политики безопасности на уровне строк
PROXY_ENTERPRISE_IDENTITY DN интернет-каталога Oracle, если прокси-пользователь
является корпоративным пользователем
PROXY_GLOBAL_UID Глобальный идентификатор пользователя из Oracle Internet Directory
для корпоративных пользователей-посредников. Возвращает NULL для всех других
прокси-пользователей.
PROXY_USER Имя пользователя, открывшего текущий сеанс от имени SESSION_USER
PROXY_USERID Идентификатор пользователя, открывшего текущий сеанс от имени
SESSION_USER
SERVER_HOST Имя хоста компьютера, на котором работает экземпляр
SERVICE_NAME Имя службы, к которой подключен сеанс
SESSION_USER Имя пользователя базы данных пользователя, вошедшего в систему
SESSION_USERID Идентификатор базы данных пользователя, вошедшего в систему
SESSIONID Идентификатор сеанса
SID Номер сеанса
STATEMENTID Идентификатор аудиторского заключения
TERMINAL Идентификатор ОС текущего сеанса
```

Информация о SYSCONTEXT

Пример:

```
SELECT SYS_CONTEXT('USERENV', 'LANG') AS RESULT FROM DUAL;
--RESULT RU

SELECT SYS_CONTEXT('USERENV', 'LANGUAGE') AS RESULT FROM DUAL;
--RESULT RUSSIAN_CIS.CL8MSWIN1251

SELECT SYS_CONTEXT('USERENV', 'NLS_CALENDAR') AS RESULT FROM DUAL;
--RESULT GREGORIAN

SELECT SYS_CONTEXT('USERENV', 'NLS_DATE_FORMAT') AS RESULT FROM DUAL;
--RESULT DD.MM.RR

SELECT SYS_CONTEXT('USERENV', 'NLS_TERRITORY') AS RESULT FROM DUAL;
--RESULT CIS
```

Контрольные вопросы и задания для самостоятельного выполнения

1. Выберите из системного контекста следующую информацию: имя схемы.
2. Выберите из системного контекста следующую информацию: имя пользователя операционной системы.
3. Создайте в клиентском контексте VAL1 со значением 123.
4. Выберите данные из клиентского контекста для VAL1.

Шаг 87. Планировщик JOB-заданий. Управление

Введение

В ORACLE есть механизм, позволяющий запланировать выполнение определенной программы на заданное время, такая программа может быть оператором SQL, программой на языке PLSQL либо даже внешней программой. Этот механизм называется механизмом заданий JOB.

Данный механизм может применяться, например, для планирования запуска тяжелых запросов в ночное время или на выходные дни, распределения задач построения отчетности.

Теория и практика

Для управления заданиями в ORACLE существует специальный пакет DBMS_JOB:

• создание заданий;

• можно использовать механизм DBMS_JOB или механизм DBMS_SCHEDULER – более современный способ.

```
DBMS_JOB.SUBMIT(
 JOB OUT BINARY_INTEGER,
 WHAT IN VARCHAR2,
 NEXT_DATE IN DATE DEFAULT SYSDATE,
 INTERVAL IN VARCHAR2 DEFAULT NULL,
 NO_PARSE IN BOOLEAN DEFAULT FALSE,
 INSTANCE IN BINARY_INTEGER DEFAULT any_instance,
 FORCE IN BOOLEAN DEFAULT FALSE
);
```

Здесь

JOB – входной параметр, уникальный идентификатор задания. Идентификатор генерируется специальной системной последовательностью;

WHAT – анонимный PL/SQL-блок, в данном блоке указывается последовательность команд, которая будет выполнена в процессе работы задания.

В этом же параметре можно также писать команды вставки, удаления, редактирования (INSER, UPDATE, DELETE), а также команды для создания индексов таблиц, создания индексов, ограничений;

NEXT_DATE – дата-время следующего выполнения задания. Если будет указана дата меньше, чем текущая дата, то выполнение задания будет начато немедленно;

INTERVAL – вычисляемая дата следующего выполнения задания в столбце NEXT_DATE.

Примеры интервала задания:

NULL
Задание выполнится однократно и удалится.

TRUNC (SYSDATE+1) +10/24
Задание будет выполняться ровно в 10 часов каждого дня.

TRUNC (SYSDATE+1) + (11+ (15/60)) /24
Задание будет выполняться ровно в 11 часов 15 минут каждого дня.

TRUNC (LAST_DAY (SYSDATE)) + (n+ (m/60)) /24
Задание будет выполняться ровно в n часов m минут последнего дня каждого месяца.

TRUNC (LAST_DAY (SYSDATE) +1) + (n+ (m/60)) /24

Задание будет выполняться ровно в n часов m минут первого дня каждого месяца;

NO_PARSE – флаг разбора PL/SQL-выражения. Если его значение равно FALSE, разбор происходит в момент установки задания. Иначе – в момент выполнения задания;

INSTANCE – какой экземпляр производит выполнение задания;

FORCE – если значение этого параметра истинно, тогда в качестве INSTANCE может выступать любое положительное целое число. В противном случае экземпляр, указанный в INSTANCE, должен быть обязательно запущен, иначе будет вызвано исключение.

Пример 1. Создание задания

```
DECLARE jobno NUMBER
BEGIN
 DBMS_JOB.SUBMIT(:jobno,
 'DELETE AUTO;',
 TO_DATE('01.01.2015 01:05','DD.MM.YYYY HH24:MI'), 'TRUNC(SYSDATE+1)+(1+(5/60))/24');
 COMMIT;
 END;
```

Таблица AUTO будет очищаться каждый день в 1 час 5 минут.

Пример 2. Создание задания

```
DECLARE jobno NUMBER
BEGIN
 DBMS_JOB.SUBMIT(:jobno,
 'DELETE AUTO; INSERT INTO AUTO SELECT * FROM AUTO1;',
 TO_DATE('01.01.2015 01:05','DD.MM.YYYY HH24:MI'), 'TRUNC(SYSDATE+1)+(3+(5/60))/24');
 COMMIT;
 END;
```

Таблица будет очищаться каждый день в 3 часа 5 минут и заполняться данными из AUTO1.

Пример 3

```
DECLARE jobno NUMBER
BEGIN
 DBMS_JOB.SUBMIT(:jobno,
 'DROP AUTO1; CREATE TABLE AUTO1 AS SELECT * FROM AUTO;',
 TO_DATE('01.01.2015 01:05','DD.MM.YYYY HH24:MI'), 'TRUNC(SYSDATE+1)+(3+(5/60))/24');
 COMMIT;
 END;
```

Таблица будет очищаться каждый день в 3 часа 5 минут и заполняться данными из AUTO1.

Удаление задания можно сделать следующей процедурой:

```
DBMS_JOB.REMOVE ( JOB IN BINARY_INTEGER );
```

Здесь JOB – идентификатор задачи.

Выключение задания

Бывают случаи, когда задание временно не должно выполняться. Для этого совсем необязательно его удалять. Достаточно его просто выключить. Выключение (включение) задания производится установкой специального флага состояния – BROKEN. Делается это с помощью следующей процедуры:

```
DBMS_JOB.BROKEN (
 JOB IN BINARY_INTEGER,
 BROKEN IN BOOLEAN,
 NEXT_DATE IN DATE DEFAULT SYSDATE);
```

Просмотр всех заданий.

Для просмотра всех заданий используются следующие таблицы: DBA_JOBS, ALL_JOBS и USER_JOBS.

- DBA_JOBS показывает все задания (JOB);
- ALL_JOBS показывает задания (JOB) текущего пользователя;
- USER_JOBS показывает задания (JOB) текущего пользователя.

Важные замечания

Вычисление NEXT_DATE с помощью формулы интервала происходит после выполнения задания. Поэтому всегда учитывайте это время и старайтесь не ставить NEXT_DATE на время, близкое к окончанию суток, из-за возможного неправильного расчета следующей даты выполнения.

При создании задания или изменения его параметров ORACLE записывает текущие параметры NLS владельца. Эти параметры каждый раз восстанавливаются при выполнении задания. Это может приводить к некоторым ошибкам в случае ожидания других значений. Поэтому, если необходимо, лучше производить установку нужных NLS-значений с помощью команды ALTER SESSION в параметре WHAT задания.

Задания в теле завершаются COMMIT;

Вопросы учеников

Есть ли еще способы создать задание, используя планировщик заданий?

Да, есть более современный метод – DBMS_SCHEDULLER.

Как создать задание, которое бы выполнялось каждые полдня?

Пример задания:

```
DECLARE jobno NUMBER
BEGIN
 DBMS_JOB.SUBMIT(:jobno,
 'DELETE AUTO; INSERT INTO AUTO SELECT * FROM AUTO1;',
 TO_DATE('01.01.2015 01:05','DD.MM.YYYY HH24:MI'), 'TRUNC(SYSDATE+1/2)');
 COMMIT;
 END;
```

Контрольные вопросы и задания для самостоятельного выполнения

1. Создайте задание, которое бы очищало таблицу MAN1 каждый день в 5 часов.
2. Создайте задание, которое бы запускало скрипт заполнения таблицы MAN1 таблицы MAN каждый день в 10 часов.
3. Просмотрите все задания, которые были запущены для текущего пользователя.

Шаг 88. Таблицы в ORACLE SQL. Дополнительные сведения

Введение

Мы с вами познакомились с обычными таблицами, существующими в ORACLE. Также в данной книге мы подробно разбираем работу с секционированными таблицами.

Но в ORACLE СУБД есть и другие виды таблиц.

Это таблицы, организованные по индексу, и кластеризованные таблицы.

Рассмотрим подробнее эти типы таблиц.

Теория и практика

Таблица, организованная по индексу – INDEX ORGANIZED TABLE, IOT

Индекс-таблицы (INDEX-ORGANIZED TABLE, IOT) – такие таблицы, в которых данные хранятся в виде индексной структуры, первичный ключ представляет собой индекс на основе B-дерева.

Индекс-таблицы отличаются от традиционных тем, что в индексных таблицах данные хранятся в упорядоченном виде. Данные сортируются по первичному ключу.

Индексные таблицы имеют уникальную идентификацию по первичному ключу, не могут быть с колонками типа CLOB, BLOB. Индексные таблицы не могут быть в кластерах.

Доступ к данным к индексных таблицах осуществляется быстрее.

Синтаксис создания индексной таблицы:

```
CREATE TABLE iot_tablename(
columnname_id NUMBER,
colname2 VARCHAR2(30),
colnameN VARCHAR2(120),
CONSTRAINT pk_ columnname _new PRIMARY KEY (columnname_id))
ORGANIZATION INDEX;
```

Здесь

• IOT_TABLENAME – имя создаваемой таблицы;

• COLNAME2…COLNAMEN – наименование колонки;

• PK_COLUMNNAME_NEW – название ключа.

ORGANIZATION INDEX означает что таблица организована по индексу.

Пример создания таблицы, организованной по индексу:

```
CREATE TABLE auto_iot(
regnum NUMBER,
mark VARCHAR2(30),
color VARCHAR2(120),
CONSTRAINT pk_auto_iot PRIMARY KEY (regnum)) ORGANIZATION INDEX;
```

Таблицы в общем кластере

Кластер – это несколько таблиц, которые физически хранятся вместе. Обычно используется в таблицах, участвующих в запросах с объединениями JOIN.

Целью кластеров таблиц является повышение производительности запросов с объединениями для этих таблиц.

При создании кластеризованных таблиц необходимо создать кластер, где будут располагаться эти таблицы.

Создадим кластер для двух таблиц MANc, CITYc, являющихся копиями таблиц MAN, CITY.

```
CREATE CLUSTER man_city(citycode NUMBER);
```

Далее создаем две таблицы MANc, CITYc.

Создаем две таблицы MANc, CITYc для кластера MAN_CITY:

```
CREATE TABLE MANc
  (       PHONENUM VARCHAR2(15),
          FIRSTNAME VARCHAR2(50),
          LASTNAME VARCHAR2(50),
          CITYCODE NUMBER REFERENCES CITY,
          YEAROLD NUMBER,
      PRIMARY KEY ("PHONENUM")
   ) CLUSTER man_city(citycode);

CREATE TABLE CITYc
  (       CITYCODE NUMBER,
          CITYNAME VARCHAR2(50),
          PEOPLES NUMBER,
           PRIMARY KEY ("CITYCODE")    ) CLUSTER man_city(citycode)
```

Дополнительные действия для таблиц

Сжатие таблиц в ORACLE

Для ускорения производительности запросов, для экономии дискового пространства, экономии оперативной памяти используется сжатие таблиц.

В сжатых таблицах выполняются запросы вставки и обновления данных, но на сжатых таблицах эти операции требуют больше ресурсов.

Сжатие таблиц не рекомендуется применять в таблицах при интенсивной вставке и обновлении данных. Наиболее эффективно использовать сжатые таблицы при работе в хранилищах данных.

Лучше всего сжатие работает на таблицах, где изменения данных редки или касаются небольшого числа записей.

Материализованные представления также могут быть сжаты.

Создание сжатой таблицы либо изменение статуса таблицы на сжатый можно осуществить с помощью специальной инструкции COMPRESS.

COMPRESS используется в операторе CREATE TABLE или ALTER TABLE… COMPRESS. Если вы изменяете таблицу, то только новые данные будут после этого подвергаться сжатию. Таким образом, таблица может в одно и то же время содержать в себе как сжатые, так и несжатые данные. При использовании директивы TABLE… UNCOMPRESS новые данные, вносимые в таблицу, не будут сжатыми.

Примеры сжатия таблиц

```
CREATE TABLE testtable(
Test1 varchar2(20)
Test2  varchar2(50))
COMPRESS FOR ALL OPERATIONS;
```

Создается сжатая таблица для всех операторов вставки и обновления.

Данный запрос покажет все сжатые таблицы в базе данных:

```
SELECT table_name, compression, compress_for FROM dba_tables where compress_for <> '-';
```

COMPESS_FOR показывает тип сжатия таблицы (для всех операций или только для загрузки в прямом режиме).

Оценка физического размера таблиц, объема дискового пространства

Для оценки дискового пространства, занимаемого таблицей, можно использовать следующие скрипты:

```
Узнаем сколько занимает каждая таблица - Дисковое пространство:
(Размер в мегабайтах)
        select a.tablespace_name, totalspace, nvl(freespace,0) freespace,
          (totalspace-nvl(freespace,0)) used,
          ((totalspace-nvl(freespace,0))/totalspace)*100 "%USED"
        from
          (select tablespace_name, sum(bytes)/1048576 totalspace
           from dba_data_files
           group by tablespace_name) a,
          (select tablespace_name, sum(Bytes)/1048576 freespace
           from dba_free_space
           group by tablespace_name) b
        where a.tablespace_name = b.tablespace_name (+)
          and ((totalspace-nvl(freespace,0))/totalspace)*100 > 90
         order by 5 desc


Размер таблиц в указанном Дисковом пространстве:

        select de.owner, de.segment_name, sum(de.bytes)/1048576, de.segment_type from
        dba_extents de
         where de.owner in ('нужный tablespace')
        group by de.owner, de.segment_name, de.segment_type
```

Переименование колонки таблицы.
Для переименования колонки таблицы используется команда:

```
ALTER TABLE tablename RENAME COLUMN oldcolumnname TO newcolumnname;
```

где
• TABLENAME – имя таблицы;
• OLDCOLUMNNAME – прежнее имя колонки таблицы;
• NEWCOLUMNNAME – новое имя колонки таблицы.

Переименование таблицы

Иногда разработчику приложений может понадобиться переименовать таблицу.

```
ALTER TABLE oldtablename RENAME TO newtablename;
```

• OldTABLENAME – прежнее имя таблицы;
• NewTABlNAME – новое имя таблицы.

Удаление таблиц

Для удаления таблицы используется оператор

```
DROP TABLE tablename;
```

где TABLENAME – имя таблицы.

Можно удалить таблицу, которая принадлежит к схеме текущего пользователя, или же администратор данных должен дать пользователю права на удаление таблиц DROP ANY TABLE.

После выполнения DROP TABLE ORACLE сохраняет ее в корзине (RECYCLE BIN).

Есть возможность восстановить удаленную таблицу с помощью команды:

```
FLASHBACK TABLE table_name TO BEFORE DROP;
```

где TABLENAME – имя таблицы.

Удалить таблицу без возможности восстановления можно с помощью следующей команды:

DROP TABLE TABLENAME PURGE;

где TABLENAME – имя таблицы.

Когда таблица имеет связи с другими таблицами, в данной таблице используются ограничения вторичных ключей и при удалении с помощью данной команды возникает ошибка ограничения, следует использовать следующую команду:

```
DROP TABLE tablename CASCADE CONSTRAINTS;
```

где TABLENAME – имя таблицы

Данная команда игнорирует целостность базы и удалит таблицу.

Важные замечания

Команда DROP TABLE имя_таблицы в версии до ORACLE DATABASE 10g удаляет таблицу без возможности восстановления.

При удалении таблицы с помощью команды DROP также удаляются связанные с ней объекты, индексы, триггеры. Исключение составляют представления и синонимы, они не удаляются из базы данных, но становятся нерабочими.

Вопросы учеников

В работе часто встречается ситуация, когда необходимо обновить довольно большой объем данных в таблице. Какие методы можно применить?

Можно вставить необходимые рассчитанные данные в таблицу с другим именем, удалить таблицу, переименовать таблицу с другим именем в таблицу, которую мы удалили.

Можно ли использовать сжатые таблицы в ситуации, когда в таблицу довольно часто добавляются данные?

Да, иногда сжатые таблицы могут применяться в таких ситуациях, требуется анализировать производительность.

Можно ли узнать прогноз, сколько места будет занимать таблица?

Да, для этого есть специальный пакет, который рассчитывает прогноз роста объема, основанный на статистике дискового пространства, – DBMS_SPACE.

В каких ситуациях лучше использовать таблицы, организованные по индексу?

Когда вам необходимо, чтобы ваши данные хранились в упорядоченном виде, например курс валют, сортировка по дате, проводки с сортировкой по дате, перечень покупок.

Контрольные вопросы и задания для самостоятельного выполнения

1. Повторите материалы данного шага.
2. Поясните преимущества использования таблиц, организованных по индексу.
3. С какой целью используются таблицы в общем кластере?
4. Что означает CASCADE CONSTRAINTS и в какой команде используется данная инструкция?
5. Создайте сжатую таблицу AUTO_COMPRESS, копию таблицы AUTO.

Шаг 89. Быстрая очистка таблиц и EXECUTE IMMEDIATE

Введение

Для быстрой очистки таблиц в ORACLE применяется специальный оператор TRUNCATE TABLE.

Данный оператор является оператором DDL, оператором мгновенного выполнения, и поэтому для его вызова нам потребуется оператор динамического SQL.

Динамический SQL используется в ORACLE с помощью команды EXECUTE IMMEDIATE.

Теория и практика

Оператор EXECUTE IMMEDIATE выполняет динамический оператор SQL или анонимный PL/SQL-блок. В нашем случае мы будем использовать EXECUTE IMMEDIATE для работы с TRUNCATE TABLE.

Текст динамического SQL заключается в кавычки.

Оператор TRUNCATE TABLE используется для быстрого удаления всех записей из таблицы в ORACLE. По результату аналогичен DELETE, без условий WHERE, но выполняется гораздо быстрее.

Синтаксис

```
TRUNCATE TABLE table_name
 [ PRESERVE MATERIALIZED VIEW LOG | PURGE MATERIALIZED VIEW LOG ]
 [ DROP STORAGE | REUSE STORAGE ] ;
```

Здесь TABLE_NAME – таблица, которую вы хотите очистить.

PRESERVE MATERIALIZED VIEW LOG

Необязательный параметр. Если он задан, то MATERIALIZED VIEW LOG будет сохранен, когда таблица очищается. Это значение по умолчанию.

PURGE MATERIALIZED VIEW LOG

Необязательный параметр. Если он задан, то MATERIALIZED VIEW LOG будет очищен, когда таблица очищается.

DROP STORAGE

Необязательный параметр. Показывает, что все хранилище очищающихся высвобождено, за исключением пространства, которое было выделено в MINEXTENTS.

REUSE STORAGE

Необязательный параметр. Если он задан, строки останутся распределенными в таблице.

```
TRUNCATE TABLE test1;
```

Этот пример очистит таблицу TEST1 и удалит все записи из этой таблицы.

Было бы равносильно следующему предложению DELETE в ORACLE:

```
DELETE FROM test1;
```

Оба эти примера приведут к тому, что все данные из таблицы TEST1 удаляются. Иногда этот оператор не может быть выполнен без применения EXCUTE IMMEDIATE. Поэтому следует выполнять следующую команду:

```
EXECUTE IMMEDIATE 'TRUNCATE TABLE test1'
```

Этот пример очистит таблицу с именем TEST1.

Важные замечания

TRUNCATE TABLE выполняется сразу, изменения сразу будут в базе, выполнения команды COMMIT не потребуется.

При выполнении операции TRUNCATE TABLE откат этой операции будет невозможен.

Также при выполнении операции TRUNCATE TABLE нельзя будет извлечь данные из таблицы с помощью ретроспективного запроса.

Перед именем таблицы можно указать имя схемы, где создана данная таблица.

Пример

```
TRUNCATE TABLE myscheme.test1;
```

Для выполнения оператора в другой схеме системный администратор должен выдать вам права DROP ANY TABLE.

Вопросы учеников

Зачем применять TRUNCATE TABLE, если есть оператор DELETE?

Оператор TRUNCATE TABLE, как правило, выполняется в несколько раз быстрее, чем оператор DELETE, особенно эта разница во времени отработки заметна на больших таблицах (более 1 миллиона записей).

Контрольные вопросы и задания для самостоятельного выполнения

1. Повторите материалы данного шага.
2. Для чего используется оператор TRUNCATE TABLE?
3. Для чего используется EXECUTE IMMEDIATE?
4. Чем отличается TRUNCATE TABLE от команды DELETE?

Шаг 90. Объекты базы данных

Введение

Небольшое отступление. Разберемся, какие типы объектов существуют в базе данных ORACLE.

Небольшой обзор существующих в базе данных объектов.

Теория и практика

Разберем основные типы объектов в базе данных, с которыми вам, скорее всего, придется столкнуться в работе.

Таблицы

Таблицы в базе данных представляют собой такие же таблицы, как мы привыкли видеть в Word и Excel, используются для сохранения различной типизированной информации.

Таблицы в базе данных могут быть различных типов: обычные, EXTERNAL, INDEX ORGANIZED. Таблицы создаются в базе данных с помощью команды CREATE TABLE.

Извлечение данных из таблиц, изменение данных в таблицах производится с помощью соответствующих SQL-операторов.

Пример создания таблицы:

```
CREATE TABLE test_tbl(
  test_id INT NOT NULL PRIMARY KEY,
  test_title VARCHAR2(100) NOT NULL,
  test_author VARCHAR(40) NOT NULL,
  dt_date DATE
);
```

Индексы

Индекс (INDEX) – это специальный объект базы данных. Индекс нужен для повышения производительности поиска данных в базе. Таблицы в базе данных могут иметь большое количество строк, которые хранятся в произвольном порядке, и их поиск по заданному критерию путем последовательного просмотра таблицы строка за строкой может занимать много времени. Ускорение работы с использованием индексов достигается в первую очередь за счет того, что индекс имеет структуру, оптимизированную под поиск, – например, структуру сбалансированного дерева.

Пример создания индекса

```
CREATE INDEX tbl_idx_name ON table (column_name);
```

Ограничения

В базе данных существуют ограничения; ограничения применяются, чтобы предотвратить ввод недопустимых данных в таблицы базы данных.

Типы ограничений в базе данных ORACLE:

- NOT NULL
- UNIQUE
- PRIMARY KEY
- FOREIGN KEY
- CHECK

Пример создания ограничения в таблице

```
ALTER TABLE table_name ADD CONSTRAINT constraint_name CHECK (column_name
condition)
```

Представления

Представление базы данных – это объект базы данных, который представляет собой результат запроса к данным в таблицам базы данных. Представления заново формируют данные из запроса. Если данные в таблицах базы данных, на основе которых построен запрос представления, изменяются, данные в представлении также изменятся.

Представления могут также содержать агрегированные данные.

Представления могут быть построены на основе запроса к нескольким таблицам.

Пример создания представления:

```
CREATE VIEW view1 (id1,idn1) AS SELECT a.id, b.id FROM a,b where a.id=b.id;
```

Пример обращения к представлению:

```
SELECT * FROM view1
```

Триггеры

Триггеры являются одной из разновидностей хранимых процедур именованного кода на PL SQL. Их исполнение происходит при возникновении для таблицы какого-либо события, например вставки данных, изменения данных в таблицах. Триггеры используются для проверки целостности данных, а также для некорректных данных.

Триггеры бывают различных типов: могут быть триггеры уровня строки, уровня таблицы, а также триггеры на системные события в базе данных.

Триггеры довольно часто используются для формирования первичных ключей в таблице.

Пример создания триггера:

```
CREATE OR REPLACE TRIGGER man_after_insert
AFTER INSERT
  ON man
  FOR EACH ROW
BEGIN
INSERT INTO man_audit
  ( phone,
    firstname,
    lastname )
  VALUES
  ( :new. phone,
    :new. firstname,
    :new. Lastname );
END;
```

При добавлении записи в таблицу MAN вызывается триггер MAN_AFTER_INSERT.

Также триггеры могут вызываться при возникновении системных событий.

Функции

Функция – это именованная подпрограмма, которая возвращает определенное значение.

У функции есть наименование, функция возвращает значение заданного типа и может использоваться в языке SQL. Функции могут иметь параметры, которые используются при вычислении значений. В функциях можно использовать данные из таблиц базы.

Пример создания функции:

```
CREATE OR REPLACE Function SQRTF
  ( x_in IN NUMBER )
  RETURN number
IS
  cnumber number;
 BEGIN
 Cnumber := x_in*x_in
RETURN cnumber;
 EXCEPTION
WHEN OTHERS THEN
 raise_application_error(-20001,'error - '||SQLCODE||' -ERROR- '||SQLERRM);
END;
```

Пример вызова функции из SQL:

```
SELECT cityname, SQRTF (peoples) FROM city;
```

Процедуры

Процедура – это именованная подпрограмма, которая выполняет некоторое заданное действие.

Процедура может иметь параметры, которые используются при вычислении значений. В процедурах можно использовать данные из таблиц базы данных.

Пример создания процедуры:

```
CREATE OR REPLACE procedure SQRTF
  ( x_in IN NUMBER )
  RETURN number
IS
  cnumber number;
 BEGIN
 Cnumber := x_in*x_in
 Insert into abl(c) values (cnumber);
EXCEPTION
WHEN OTHERS THEN
 raise_application_error(-20001,' error - '||SQLCODE||' -ERROR- '||SQLERRM);
END;
```

Пример вызова процедуры:

```
Begin
  Sortf(10);
End;
```

Пакеты

Пакет – это специальный объект базы данных, объединяющий несколько функций и процедур, состоит из тела пакета и заголовка пакета.

То есть пакет представляет собой контейнер для процедур и функций.

Пример создания пакета:

```
CREATE PACKAGE pkg_rt AS
PROCEDURE proc1 ( ename  VARCHAR2);
PROCEDURE proc2 (v_id NUMBER);
END emp_actions;

CREATE PACKAGE BODY pkg_rt AS
PROCEDURE proc1 ( ename  VARCHAR2)
IS
BEGIN
 Null;
END;
PROCEDURE fire_employee (v_id NUMBER) IS
BEGIN
    DELETE FROM tbl1 WHERE tid = v_id;
END fire_employee;
END pkg_rt;
```

Синонимы

Тему синонимов мы рассматривали на страницах этой книги.

Синонимы (synonyms) – специальные псевдонимы объектов базы данных, применяются для удобства доступа к объектам, другим схемам базы данных, могут использоваться для распределения прав и безопасности доступа к данным.

Пример создания синонима:

```
CREATE SYNONYM a9 FOR auto;
```

Пример использования синонима:

```
SELECT * FROM a9;
```

DATABASE LINK

DATABASE LINK – это специальный объект в ORACLE для соединения с другой (удаленной) базой данных, разрешающий доступ к объектам другой базы данных.

Вы также можете создать DATABASE LINK, чтобы присоединить ORACLE к базе данных MySQL, SQL Server, для этого требуется установка дополнительного ПО.

Пример создания:

```
CREATE public database LINK merge.link CONNECT TO qwr identified by test using 'TEST';
```

Материализованные представления

Материализованные представления – это специализированные представления, которые в отличие от обычных представлений занимают место и в целом похожи на обычные таблицы. Материализованные представления можно секционировать и создавать индексы.

Чаще всего используются для подсчета итогов и создания отчетов.

Пример создания материализованного представления:

```
CREATE MATERIALIZED VIEW prod_mv
PCTFREE 0 STORAGE (initial 8k next 8k pctincrease 0)
BUILD IMMEDIATE
REFRESH FAST ON DEMAND
ENABLE QUERY REWRITE
AS SELECT
        Prod_name
FROM  products p
GROUP BY  prod_name;
```

Пример использования материализованного представления:

```
SELECT * FROM prod_mv
```

Важные замечания

Здесь перечислены далеко не все объекты, а только наиболее часто используемые на практике.

Для получения информации по всем объектам базы данных ORACLE следует обратиться к документации.

Вопросы учеников

Для каких системных событий могут использоваться триггеры?

Например, следующие события: вход в систему, компиляция процедуры, создание процедуры, создание таблицы.

Что такое пакет в ORACLE?

Пакет – это набор процедур, набор функций, связанных переменных и встроенных типов.

Какой язык используется для создания процедур или функций?

Это специальный язык PL SQL, также можно создавать процедуры и функции на языке JAVA.

Контрольные вопросы и задания для самостоятельного выполнения

1. Для чего нужны триггеры базе данных?
2. Что такое представления в базе данных?
3. Чем отличаются материализованные представления от обычных представлений?
4. Объясните назначение процедур и функций, назначение пакетов в базе.

День девятнадцатый

Шаг 91. Последовательности и формирование первичного ключа

Введение

Последовательность SEQUENCE в ORACLE SQL – это объект базы данных, который генерирует целые числа в соответствии с правилами, установленными во время его создания.

Пример создания последовательности:

```
CREATE SEQUENCE seq_3
START WITH 20
INCREMENT BY -1
MAXVALUE 20
MINVALUE 0
CYCLE
CACHE 10;
```

Пример использования последовательности:

```
SELECT seq_3.naxt_val FROM dual;
```

Теория и практика

Чаще всего этот объект базы данных – последовательность SEQUENCE – используется при формировании первичного ключа таблицы.

Синтаксис создания объекта последовательности:

```
CREATE SEQUENCE sequence_name
MINVALUE value
MAXVALUE value
START WITH value
INCREMENT BY value
CYRCLE
CACHE value;
```

Основные параметры

• START WITH позволяет создателю последовательности указать первое генерируемое ей значение. После создания последовательность генерирует указанное в START WITH значение при первой ссылке на ее виртуальный столбец NEXTVAL

• INCREMENT BY n определяет приращение последовательности при каждой ссылке на виртуальный столбец NEXVAL. Если значение не указано явно, по умолчанию устанавливается 1. Для возрастающих последовательностей устанавливается положительное n, для убывающих или последовательностей с обратным отсчетом – отрицательное.

• MINVALUE определяет минимальное значение, создаваемое последовательностью. Если оно не указано, ORACLE применяет значение по умолчанию NOMINVALUE.

• MAXVALUE определяет максимальное значение, создаваемое последовательностью. Если оно не указано, ORACLE применяет значение по умолчанию NOMAXVALUE.

• CYCLE позволяет последовательности повторно использовать созданные значения при достижении MAXVALUE или MINVALUE. То есть последовательность будет продолжать генерировать значения после достижения своего максимума или минимума. Возрастающая последовательность после достижения своего максимума генерирует свой минимум. Убывающая последовательность после достижения своего минимума генерирует свой максимум. Если циклический режим нежелателен или не установлен явным образом, ORACLE применяет значение по умолчанию NOCYCLE. Указывать CYCLE вместе с NOMAXVALUE или NOMINVALUE нельзя. Если нужна циклическая последовательность, необходимо указать MAXVALUE для возрастающей последовательности или MINVALUE для убывающей.

• CACHE n указывает, сколько значений последовательности ORACLE распределяет заранее и поддерживает в памяти для быстрого доступа. Минимальное значение этого параметра равно 2. Для циклических последовательностей это значение должно быть меньше, чем количество значений в цикле. Если кэширование нежелательно или не установлено явным образом, ORACLE применяет значение по умолчанию – 20 значений.

Пример создания последовательности от 1 с шагом 1:

```
CREATE SEQUENCE seq_1
START WITH 1
INCREMENT BY 1
NOCYCLE
CACHE 10;
```

Пример создания последовательности, начало 10 000, с шагом 10:

```
CREATE SEQUENCE seq_2
START WITH 10000
INCREMENT BY 10
NOCYCLE
CACHE 10;
```

Пример создания последовательности, начало 10 000, с шагом -1, обратная последовательность:

```
CREATE SEQUENCE seq_3
START WITH 20
INCREMENT BY -1
MAXVALUE 20
MINVALUE 0
CYCLE
CACHE 10;
```

Обращение к последовательности из SQL:
Seq.CURRVAL – текущее значение последовательности;
Seq.NEXTVAL – приращение последовательности (возвращает ее следующее значение);
первое обращение Seq.NEXTVAL инициирует начальное значение последовательности;
примеры в SQL;
оператор SELECT.

```
SELECT
Select seq_1.nextval from dual;
Select seq_2.nextval, o.object_name from all_objects o;
```

Вставка INSERT:

```
INSERT into t (n1,n2) values (seq_1.nextval,2);
INSERT into t (n1,s1) Select seq_2.nextval, o.object_name from all_objects o;
```

Обновление UPDATE:

```
Update t set n1 = seq_3.currval
```

Важные замечания

Для создания первичного ключа из последовательности поможет следующий пример:

```
CREATE TABLE city1 AS SELECT * FROM city WHERE 1=0;
INSERT INTO city1(citycode, cityname, peoples) SELECT seq_1.nextval, cityname, peoples
FROM city;
```

Создание таблицы CITY1 и далее заполнение таблицы записями из CITY, первичный ключ CITYCODE, заполняется из seq_1.

Команда DROP SEQUENCE SEQ удаляет последовательность из базы, SEQ – имя последовательности.

Нельзя использовать значения CURRVAL и NEXTVAL в следующих ситуациях:

- в подзапросе;
- предложении SELECT с оператором DISTINCT;
- предложении SELECT с фразой GROUP BY или ORDER BY;
- предложении SELECT, объединенном с другим предложением SELECT оператором множеств UNION;
- фразе WHERE предложения SELECT;
- умалчиваемом (DEFAULT) значении столбца в предложении CREATE TABLE или ALTER TABLE;
- условии ограничения CHECK.

Нельзя использовать CURR_VAL, если вы еще ни разу не использовали команду NEXT_VAL для заданной последовательности, поскольку последовательность еще не была инициирована командой NEXT_VAL.

Вопросы учеников

Чем отличаются CURR_VAL, NEXT_VAL при обращении к SEQUENCE?

CURR_VAL выводит текущее значение, NEXT_VAL выводит следующее значение.

При использовании NEXT_VAL что происходит?

SEQUENCE. NEXT_VAL выводит следующий номер последовательности, и последовательность будет равняться следующему номеру из этой последовательности, происходит переключение последовательности.

Контрольные вопросы и задания для самостоятельного выполнения

1. Создайте последовательность SQ1, которая бы генерировала числа от 1 до 5.
2. Создайте последовательность SQ10, которая бы генерировала числа от 10 до 50 000 с шагом 10.
3. Напишите запрос, который бы выводил на экран строки из таблицы AUTO и значения из последовательности SQ1.
4. Напишите запрос, который бы выводил на экран строки из таблицы AUTO и значения из последовательности SQ10.

Шаг 92. Пользователь и схема. Разграничение прав, роли

Введение

Рассмотрим такие важные понятия, как схема и пользователь в СУБД ORACLE. Понятия схемы и пользователя необходимо знать, поскольку, скорее всего, вы столкнетесь с ними в дальнейшей работе.

Пользователи и схемы существуют в базе данных ORACLE для безопасности информационной системы, а также для разграничения доступа к объектам на чтение, запись и просмотр информации в соответствии с заданными бизнес-правилами.

Для изучения данного шага, выполнения практических примеров требуется установка дополнительного программного обеспечения: ORACLEXE, SQLDEVELOPER.

Как установить данное программное обеспечение, подробно описано в шаге 51.

Теория и практика

У каждого пользователя базы данных ORACLE есть свое уникальное имя пользователя (логин) и пароль. Логин и пароль пользователя называются учетной записью пользователя.

Зная свой логин и свой пароль, параметры своей учетной записи, пользователь осуществляет соединение с базой данных.

Все объекты базы данных: таблицы, представления, последовательности и т. д. – принадлежат определенным пользователям.

Множество объектов (таблиц, представлений…) заданного пользователя называется схемой пользователя.

То есть когда определенный пользователь создает какую-либо таблицу или другой объект базы данных, этот объект создается в схеме данного пользователя.

При обращении к объекту пользователя из другой учетной записи (когда мы соединяемся с базой данных под другим логином и паролем) необходимо перед наименованием объекта, к которому идет обращение (таблице, функции, представлению), добавлять имя пользователя – схему и через точку указывать сам объект. Такое обращение называется квалифицированным обращением.

Для обращения к объекту не из своей схемы пользователь, владелец объекта, должен предоставить соответствующие права на чтение данных соответствующего Имя_пользователя. имя_таблицы1.

Пример квалифицированного обращения:

```
SELECT * FROM hr.STAFF
SELECT * FROM user3.STAFF
DELETE user3.TABLE
```

Для создания пользователя применяется команда
CREATE USER USERNAME IDENTIFIED BY PASSWORS,
где USERNAME – имя пользователя;

PASswors – пароль пользователя.

Есть несколько дополнительных параметров: DEFAULT TABLESPACE USERS QUOTA 100M ON USERS – TEMPORARY TABLESPACE TEMP QUOTA 10M ON TEMP, которые позволяют задать дисковую квоту для заданного пользователя.

После создания пользователя необходимо выполнить команду

```
GRANT CREATE SESSION TO USERNAME;
```

а также

```
GRANT CREATE TABLE TO USERNAME
GRANT CREATE PROCEDURE TO USERNAME
GRANT CREATE TRIGGER TO USERNAME
GRANT CREATE VIEW TO USERNAME
GRANT CREATE SEQUENCE TO USERNAME
```

права для создания таблиц, процедур и функций.
Эта команда дает созданному пользователю права на создание сессии.
Ее выполнение необходимо для того, чтобы разрешить присоединение пользователя к базе данных.
Приведем пример.
Зайдем в программу SQLDEVELOPER, откроем сессию SYS с паролем, который вы ввели при установке.
Выполнить команды:

CREATE USER USER1 IDENTIFIED BY PASS1;

Создание пользователя USER1 с паролем PASS1:

GRANT CREATE SESSION TO USER1;
GRANT CREATE TABLE TO USER1;

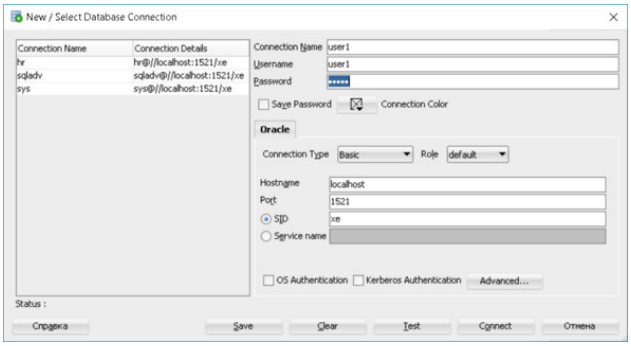
Даем право на создание сессии и создание таблиц созданному пользователю:

CREATE USER USER2 IDENTIFIED BY PASS2;

Создание пользователя USER1 с паролем PASS1:

GRANT CREATE SESSION TO USER2;

Даем право на создание сессии созданному пользователю.
Перезайдем под пользователем USER1, в выпадающем списке необходимо отметить DEFAULT.



CONNECTION NAME: USER1
USER NAME: USER1
PASSWORD: USER1

CREATE TABLE T1 AS SELECT SYSDATE SDT FROM DUAL;

Зайдем под учетной записью USER2.
CONNECTION NAME: USER2
USER NAME: USER2
PASSWORD: USER2
Выполним запрос:

SELECT * FROM USER1.t1

Ошибка ORA-00942: TABLE OR VIEW DOES NOT EXIST.
Перейдем во вкладку USER1.

GRANT SELECT, INSERT ON T1 TO USER2;

Добавим права на выбор и вставку SELECT, INSERT.
Соединимся снова с базой под пользователем USER2 и выполним запрос:

SELECT * FROM USER1.t1

– 01.01.2011

Таким образом, мы получили данные из таблицы в другой схеме – USER1.t1.

Системные административные пользователи (SYS SYSTEM)

Есть специальные системные пользователи с максимальными правами на объекты.

Системный пользователь SYS владеет таблицами словаря данных, содержащими информацию обо всех остальных структурах базы данных.

Системному пользователю SYSTEM принадлежат представления, полученные на основе таблиц пользователя SYS. Все объекты базы данных создаются с учетными записями их владельцев.

Под учетными записями SYS, SYSTEM реализован максимальный набор прав: права создавать объекты, изменять права для объектов любого пользователя, назначать любому пользователю роли и привилегии.

Для входа под данными учетными записями используется специальный тип входа SYS AS SYSDBA, SYSTEM AS SYSDBA. При входе через интерфейс программы SQLDEVELOPER для входа под этими пользователя необходимо выбрать в выпадающем списке ROLE – SYSDBA.

Система прав и ролей

В ORACLE есть системные привилегии, объектные привилегии и роли.

• Системные привилегии – специальный тип привилегий для выполнения конкретного действия в базе данных либо действия с любым объектом схемы конкретного типа.

• Объектная привилегия назначается для конкретного объекта таблицы.

• Роль – это предопределенный набор прав и привилегий, назначается заданным пользователям.

Для назначения ролей и привилегий используется оператор GRANT.

Синтаксис:

GRANT тип операции или роль ON объект TO пользователь;

где тип операции – SELECT, INSERT, EXECUTE, UPDATE;
объект – параметр может быть пропущен;
пользователь – учетная запись системы.

Таблицы с информацией о ролях пользователей

Для того чтобы посмотреть информацию о правах, ролях и пользователях, существуют следующие системные таблицы.

DBA_USERS хранит информацию обо всех, кто имеет учетную запись в базе данных ORACLE. Вместе с именем и хешированным паролем пользователя хранится имя назначенного ему пользователя.

DBA_PROFILE для каждого профиля хранит информацию о ресурсах и их лимитах.

DBA_ROLES детализирует все роли, содержащиеся в базе данных.

DBA_ROLE_PRIVS – роли, которые были назначены конкретным пользователям и другим ролям.

DBA_SYS_PRIVS – системные привилегии, которые были выданы конкретным пользователям или ролям.

DBA_TAB_PRIVS – привилегии SELECT, INSERT и UPDATE, которые были выданы конкретным пользователям или ролям.

DBA_COL_PRIVS – привилегии SELECT, INSERT и UPDATE, которые были выданы конкретным пользователям или ролям.

ROLE_ROLE_PRIVS – роли, назначенные другим ролям.

ROLE_SYS_PRIVS – системные привилегии, выданные ролям.

ROLE_TAB_PRIVS – привилегии доступа к таблицам, выданные ролям.

ROLE_COL_PRIVS – привилегии доступа к столбцам таблиц, выданные ролям.

USER_ROLE_PRIVS – роли, назначенные текущему пользователю.

USER_SYS_PRIVS – системные привилегии, выданные текущему пользователю.

USER_TAB_PRIVS – привилегии доступа к таблицам, выданные текущему пользователю.

USER_COL_PRIVS – привилегии доступа к столбцам таблиц, выданные текущему пользователю.

Пример

```
SELECT * FROM USER_ROLE_PRIVS WHERE USERNAME =USER
```

Просмотр всех системных привилегий заданного пользователя user.

Объектные привилегии

Для каждого из объектов базы данных можно назначать привилегии, которые необходимы заданным пользователям.

Синтаксис

Предоставление объектных привилегий:

GRANT тип привилегий ON объект TO пользователь WITH GRANT OPTION;

Тип привилегий может быть на выборку, на вставку, на обновление данных, на запуск процедуры или функции SELECT, INSERT, UPDATE, EXCEUTE.

GRANT OPTION – возможность передавать данные права.

Примеры

Права на выбор данных из таблицы t2:

GRANT SELECT ON USER2.t2 TO USER1;

На оператор UPDATE для USER1 для таблицы USER2.t2:

GRANT UPDATE ON USER2.t2 TO USER;

На INSERT с возможностью пользователя передавать другим эту привилегию для таблицы USER2.t2:

GRANT INSERT, UPDATE, SELECT ON USER2.t2 TO USER1 WITH GRANT OPTION;

Системные привилегии

Системные права позволяют пользователю выполнить конкретное действие в базе данных либо действие с любым объектом схемы конкретного типа.

Перечень системных привилегий:
CREATE TABLESPACE – права на создание табличного пространства.
CREATE USER – права на создание пользователя.
DROP USER – права на удаление пользователя.
ALTER USER – права на изменение пользователя.
GRANT ANY PRIVILEGE – права на назначение любой привилегии.
INSERT ANY TABLE – права на вставку в любую таблицу.
DELETE ANY TABLE – права на удаление.
CREATE PROCEDURE – права на создание в схеме пользователя хранимой процедуры, функции или пакета.
CREATE VIEW – права на создание в схеме пользователя представления.
CREATE SEQUENCE – права на создание в схеме пользователя последовательности.
CREATE TABLE – права на создание пользователя таблицы.
CREATE SESSION – права на создание соединения с базой данных.
CREATE DATABASE LINK – права на создание в схеме пользователя таблицы.

Пример

GRANT CREATE TABLE TO USER1; – права на создание таблицы USER1

GRANT CREATE SEQUENCE TO USER1; – права на создание последовательности USER1

GRANT CREATE VIEW TO USER1; – права на создание представления USER1

GRANT CREATE USER TO USER1; – права на создание пользователя USER1

Роли

Роль включает набор прав и системных привилегий. Роль может быть назначена указанным пользователям.

Создание ролей: CREATE ROLE NEWROLE;

Предоставление привилегий роли: GRANT CREATE TABLE, CREATE PROCEDURE, CREATE TRIGGER, CREATE VIEW, CREATE SEQUENCE.

TO NEWROLE;

Связь роли с пользователем: GRANT NEWROLE TO USER1;

Вопросы учеников

Для чего все же нужны роли?

Роль позволяет предварительно задать наборы прав для определенных групп бизнес-пользователей. После чего назначать этим пользователям не права по отдельности, а готовые бизнес-роли.

Можно ли создать схему без создания пользователя?

Нельзя, но можно создать пользователя без прав соединения, в этом случае будет создана схема.

Контрольные вопросы и задания для самостоятельного выполнения

1. Объясните отличия системных привилегий от объектных привилегий.
2. Напишите команду, которая бы предоставила пользователю USER2 права на создание таблиц и представлений.
3. Создайте новую роль с правами на создание последовательности (SEQUENCE) и создание таблиц, назначьте эту роль пользователю USER2.
4. Дайте права на чтение из таблицы USER1.t1 пользователю t2.

Шаг 93. Системные представления. Сведения об объектах схемы

Введение

В ORACLE существуют специальные таблицы и представления, из которых можно получить сведения о структуре базы данных и структуре объектов базы данных. Эти сущности являются системными представлениями и системными таблицами словаря данных.

Теория и практика

Основные системные таблицы и представления, доступные пользователю без специальных разрешений:

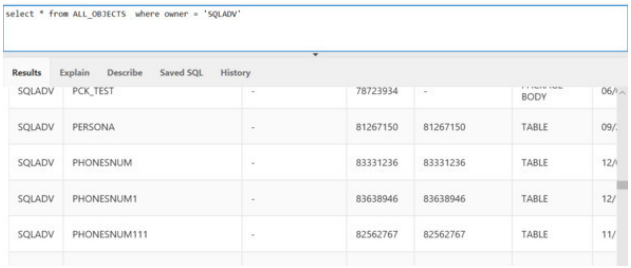
ALL_OBJECTS – сведения об основных объектах, доступных этому пользователю; это таблицы, представления, функции, процедуры, и это наиболее важное из всех системных представлений.

Системное представление ALL_OBJECTS содержит следующую значимую информацию:

Колонка	Описание колонки
OWNER	Владелец объекта или схема,
OBJECT_NAME	Наименование объекта
SUBOBJECT_NAME	Наименование субъекта
OBJECT_ID	Идентификатор объекта из словаря данных
OBJECT_TYPE	Тип объекта (talble, index)
CREATED	Дата время создания объекта
LAST_DDL_TIME	Последнее время изменяя объекта
STATUS	Статус объекта VALID, INVALID, or N/A

Запрос к системному представлению ALL_OBJECTS:

```
select * from ALL_OBJECTS  where owner = 'SQLADV'
```



ALL_SEQUENCES – сведения об описании последовательностей, доступных пользователю.

Содержит следующую важную информацию:

Колонка	Описание колонки
SEQUENCE_OWNER	Владелец последовательности или схема,
SEQUENCE_NAME	Наименование последовательности
MIN_VALUE	Минимальное значение последовательности
MAX_VALUE	Максимальное значение последовательности
INCREMENT_BY	Инкремент приращения последовательности
CYCLE_FLAG	Цикличность последовательности (Y/N)
LAST_DDL_TIME	Последнее время изменения объекта
STATUS	Статус объекта VALID, INVALID, or N/A
CACHE_SIZE	Размер кэша
LAST_NUMBER	Последний номер последовательности

Запрос к системному представлению ALL_SEQUENCES:

```
select * from ALL_SEQUENCES where SEQUENCE_owner = 'SQLADV'
```

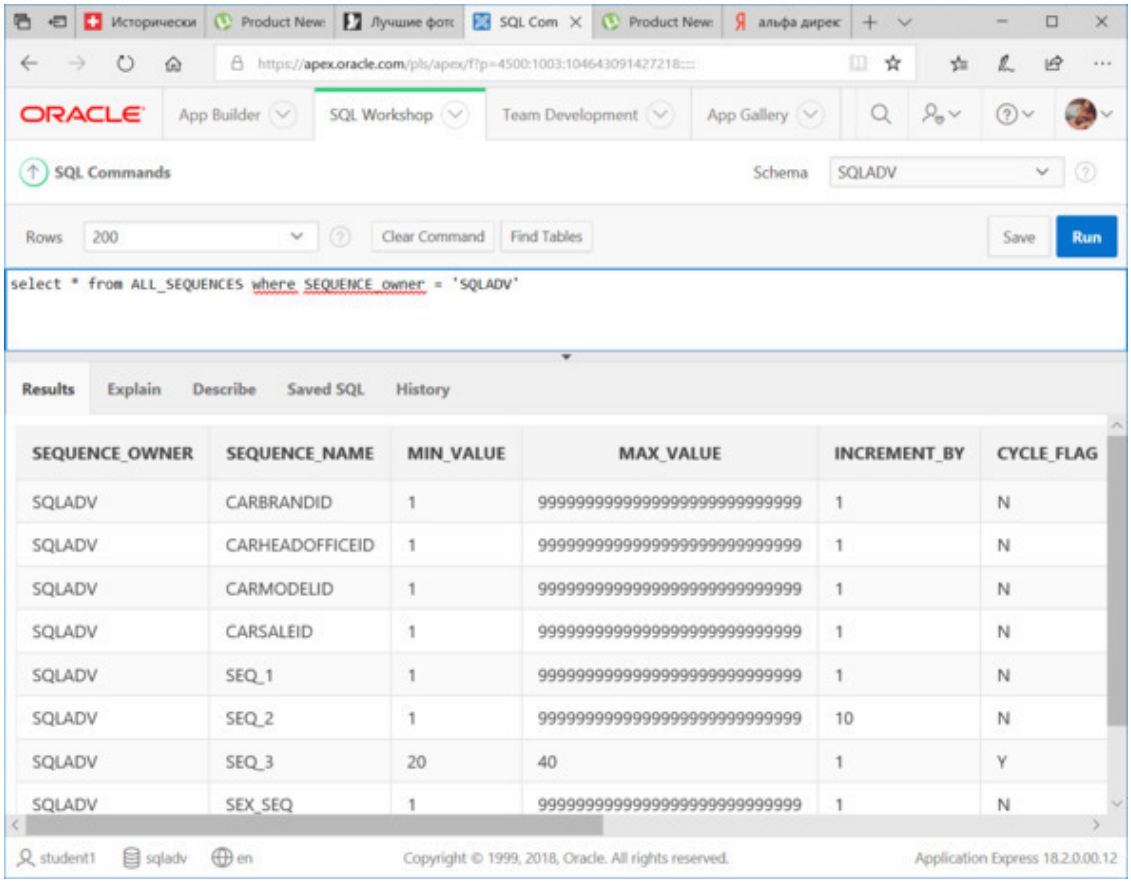


Рисунок 222. Все последовательности SQ

ALL_SOURCE – исходный текст кода объектов, доступных пользователю. Это может быть текст функций, процедур, триггеров.
ALL_SOURCE содержит следующую важную информацию:

Колонка	Описание колонки
OWNER	Владелец объекта или схема,
NAME	Наименование объекта
TYPE	Тип объекта FUNCTION, JAVA SOURCE, PACKAGE, PACKAGE BODY, PROCEDURE, TRIGGER, TYPE, TYPE BODY
LINE	Номер строи в исходном коде
TEXT	Текст исходного кода
LAST_DDL_TIME	Последнее время изменения объекта
STATUS	Статус объекта VALID, INVALID, or N/A
CACHE_SIZE	Размер кэша

Запрос к ALL_SOURCE:

```
select * from ALL_SOURCE where owner = 'SQLADV'
```

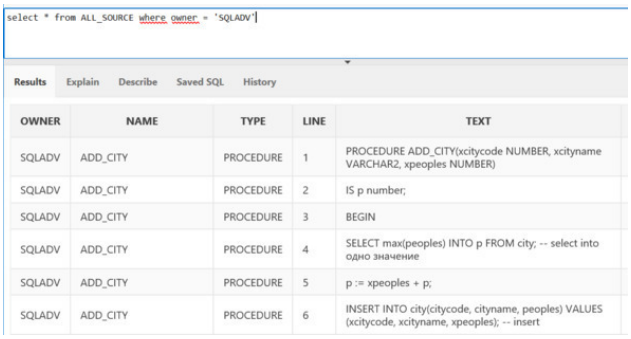


Рисунок 223. Запрос к ALL_SOURCE

ALL_SYNONYMS – сведения о синонимах, доступных пользователю.
ALL_SYNONYMS содержит следующую важную информацию:

Колонка	Описание колонки
OWNER	Владелец объекта или схема,
SYNONYM_NAME	Наименование синонима
TABLE_OWNER	Таблица владелец, если синоним создан для таблицы
DB_LINK	Дата линк , если синоним создан для даталика
TEXT	Текст исходного кода
CYCLE_FLAG	Цикличность последовательности (Y/N)
LAST_DDL_TIME	Последнее время изменяя обьекта
STATUS	Статус обьекта VALID, INVALID, or N/A
CACHE_SIZE	Размер кэша
LAST_NUMBER	Последний номер последовательности

Запрос к ALL_SYNONYMS:

```
select * from ALL_SYNONYMS  where owner = 'SQLADV'
```

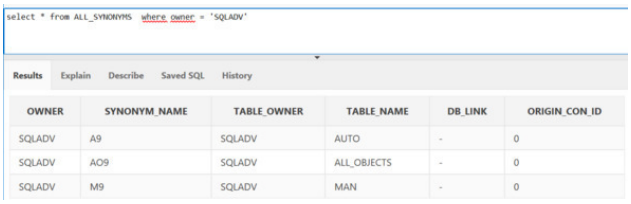


Рисунок 224. Запрос к ALL_SYNONIMS

USER_TABLES (ALL_TABLE) – подробные данные о каждой таблице пользователя.
USER_TABLES содержит следующую важную информацию:

Колонка	Описание колонки
OWNER	Владелец таблицы или схема,
TABLE_NAME	Наименование таблицы
TABLESPACE_NAME	Наименование табличного пространства TABLESPACE , включающего таблицу
CLUSTER_NAME	Наименование кластера, если таблица принадлежит к кластеру
IOT_NAME	Наименование таблицы организованной по индексу, если колонка не заполнена то таблица является обычной таблицей
STATUS	Флаг(UNUSABLE.VALID) ели команда DROP TABLE не была успешна
LAST_DDL_TIME	Последнее время изменения объекта
STATUS	Статус объекта VALID, INVALID, or N/A
CACHE_SIZE	Размер кэша
LAST_NUMBER	Последний номер последовательности
PCT_FREE	Минимальный процент свободного места в блоке
PCT_USED	Минимальный процент используемого места в блоке
INI_TRANS	Инит номер транзакции
MAX_TRANS	Макс номер транзакции
LOGGING	Признак логирования
NUM_ROWS	Количество строк
EMPTY_BLOCKS	Количество свободных блоков
TABLE_LOCK	Признак блока на таблице
LAST_ANALYZED	Дата последнего сбора аналитики
TEMPORARY	Признак временной таблицы
NESTED	Признак вложенной таблицы
PARTITIONED	Признак партицируемости
SAMPLE_SIZE	Примерный размер использования в анализируемой таблице
ROW_MOVEMENT	Допуск переноса строк между секциями в партицируемой таблице
COMPRESSION	Признак сжатой таблицы
DROPPED	Признак что таблица была удалена в корзину

Запрос к USER_TABLES:

```
select * from USER_TABLES
```

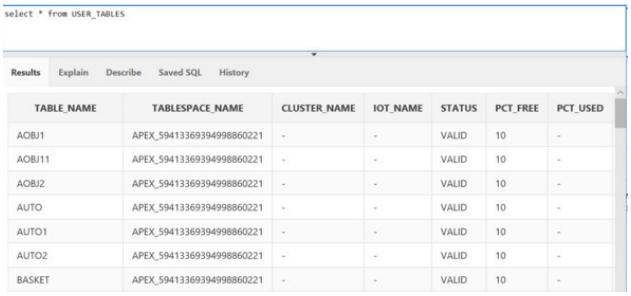


Рисунок 225. Запрос к USER_TABLES

USER_TAB_COLUMNS – описание колонок всех таблиц пользователя, подробное описание.
USER_TAB_COLUMNS содержит следующую важную информацию:

Колонка	Описание колонки
OWNER	Владелец таблицы или схема,
TABLE_NAME	Наименование таблицы
COLUMN_NAME	Наименование колонки таблицы
DATA_TYPE	Тип данных колонки
DATA_LENGTH	Длина колонки в байтах
DATA_PRECISION	Количество разрядов запятой
DATA_SCALE	Количество разрядов до запятой
NULLABLE	Колонка NULL или not NULL
COLUMN_ID	Последовательный номер колонки при создании
DEFAULT_LENGTH	Значение длины колонки по умолчанию
DATA_DEFAULT	Значение по умолчанию для данной колонки таблиц

Запрос к USER_TAB_COLUMNS для таблицы AUTO:

```
select * from USER_TAB_COLUMNS where table_name = 'AUTO'
```

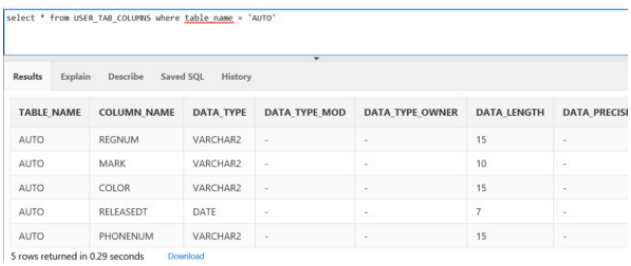


Рисунок 226. Запрос к USER_TAB_COLUMNS

USER_TAB_COMMENTS – комментарии к таблицам и представлениям, принадлежащим пользователю.

USER_TAB_COMMENTS содержит следующую важную информацию:

Колонка	Описание колонки
TABLE_NAME	Наименование таблицы
TABLE_TYPE	Тип таблицы
COMMENTS	Текс комментария к таблице

Важные замечания

USER_TABLES является синонимом для ALL_TABLES.
USER_ COLUMN является синонимом для ALL_COLUMN.

Вопросы учеников

Как посмотреть состояние всех последовательностей для текущего пользователя?

Попробуйте выполнить запрос:

```
select * from ALL_SEQUENCES where SEQUENCE_owner = 'SQLADV'
```

Колонка LAST_NUMBER – это последнее число последовательности.

Контрольные вопросы и задания для самостоятельного выполнения

1. Повторите материалы данного шага.
2. Каким запросом найти последнее время изменения объекта?
3. Каким запросом найти все представления данной схемы, данного пользователя?
4. Каким запросом найти тип колонки MARK для таблицы AUTO?

Шаг 94. Системные представления. Сведения об объектах базы данных

Введение

Для данного шага требуется установка дополнительного программного обеспечения ORACLEExpress, SQLDEVELOPER. Как это сделать, подробно описано в шаге 51.

Существуют специальные системные таблицы и представления, из которых можно получить сведения о структуре базы данных и структуре объектов базы данных, текущих сессиях и процессах в базе данных. На работу с данными представлениями необходимо специальное разрешение.

Для просмотра информации из представлений необходимо соединиться с базой данных под пользователем SYS с ролью SYSDBA.

Разберем каждое из этих представлений подробнее.

Теория и практика

Ниже рассматриваются наиболее важные системные представления, наиболее часто используемые во время практической работы.

V$DATABASE показывает информацию о базе данных из CONTROL-файла. Для того чтобы посмотреть информацию о базе данных из CONTROL-файла, основные параметры базы данных, используется представление V$DATABASE.

Колонка	Описание колонки
DBID	Уникальный идентификатор базы данных вычисляете при создании базы данных восстановлении базы данных
NAME	Наименование базы данных
CREATED	Текс комментария к таблице
RESETLOGS_CHANGE	Номер системного изменения
RESETLOGS_TIME	Время дата открытия resetlog
PRIOR_RESETLOGS_CHANGE	SCN предыдущий resetlogs
PRIOR_RESETLOGS_TIME	Дата время предыдущего resetlog
LOG_MODE	Режим archivelog NOARCHIVELOG ARCHIVELOG

Запрос к V$DATABASE на просмотр информации о базе:

```
select * from V$DATABASE
```

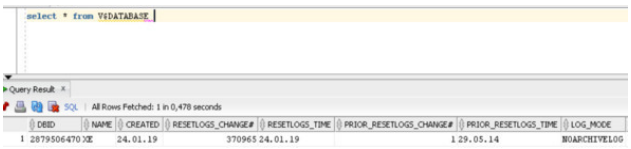


Рисунок 227. Просмотр информации о базе

V$VERSION показывает номер версии ядра библиотеки в ORACLE DATABASE.

Это представление используется для получения исчерпывающей подробной информации о базе данных и экземпляре.

Колонка	Описание колонки
BANNER	Наименование компонента и номер версии

Запрос к V$VERSION на просмотр информации о версии:

```
select * from V$VERSION
```

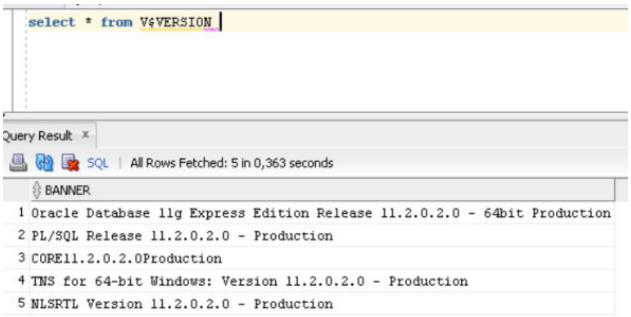


Рисунок 228. Запрос информации о версии базы данных

V$INSTANCE показывает статус текущего экземпляра. Данное представление часто используется в работе для получения исчерпывающей информации о текущем экземпляре базы.

Колонка	Описание колонки
INSTANCE_NUMBER	Номер используемый для регистрации экземпляра базы данных
INSTANCE_NAME	Наименование экземпляра
HOST_NAME	Наименование хоста машины
VERSION	Версия базы данных
STARTUP_TIME DATE	Время, когда экземляр был запущен
STATUS	Статус экземпляра базы STARTED MOUNTED OPEN OPEN MIGRATE
ARCHIVER	STOPPED | STARTED | FAILED возможная ошибки ARCHIVER
ACTIVE_STATE	NORMAL|QUIESCING|QUIESCED Normal – Экземпляр базы работает нормально
INSTANCE_ROLE	Описывает роль экземпляра базы данных

Запрос к V$INSTANCE на просмотр информации о версии экземпляра:

```
Select * from V$INSTANCE
```

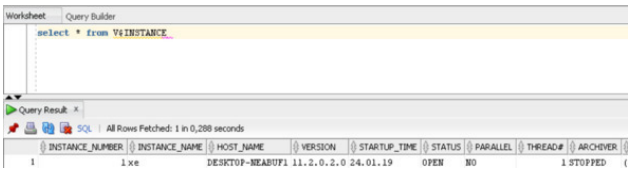


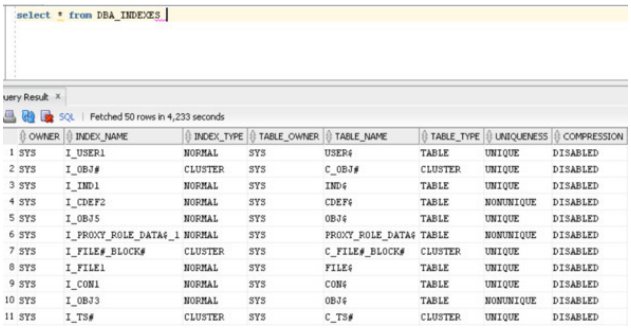
Рисунок 229. Информация об экземпляре

DBA_INDEXES описывает все индексы в базе данных. Данное системное представление содержит подробную информацию по индексам базы данных. Для подробной информации по конкретному индексу необходимо выбрать соответствующую запись по наименованию заданного индекса.

Колонка	Описание колонки
OWNER	Владелец , схема
INDEX_NAME	Наименование индекса
INDEX_TYPE	Тип индекса • NORMAL • BITMAP • FUNCTION-BASED NORMAL • FUNCTION-BASED BITMAP • DOMAIN
TABLE_OWNER	Владелец схема владельца таблицы индекса
TABLE_NAME	Наименование таблицы
TABLE_TYPE	Тип индексируемого объекта TABLE, CLUSTER
UNIQUENESS	Индекс is UNIQUE or NONUNIQUE
COMPRESSION	Если включено сжатие индекса то значениеENABLE
TABLESPACE_NAME	Наименование табличного пространства
INIT_TRANS	Начальный номер транзакции
MAX_TRANS	Максимальный номер транзакции
INCLUDE_COLUMN	Column ID последней колонки включенной в таблицу организованною по индексу
LOGGING	Инфрмация о логировании
STATUS	Статус индекса VALID или UNUSABLE
PARTITIONED	Индикатор если индекс сегментирован(YES) or not (NO)
TEMPORARY	Indicates whether the index is on a temporary table
GENERATED	Индикатор что наименование индекса сгенерировано системой
GLOBAL_STATS	Для сегментированных индексов указывает, была ли статистика собрана путем анализа индекса в целом (Yes) или была оценена на основе статистики по основным разделам и подразделам индекса (No)
DOMIDX_STATUS	Статус для domain index: NULL - индекс не является индексом домена VALID - индекс является допустимым доменным индексом IDXTYP_INVLD - недопустимый тип индекса индекса домена
FUNCIDX_STATUS	Состояние индекса на основе функции: NULL - индекс не является индексом на основе функций ENABLED - индекс на основе функций включен DISABLED - индекс на основе функций отключен
JOIN_INDEX	Указывает, является ли индекс индексом соединения таблиц (Yes) или нет (NO)

Запрос к DBA_INDEXES на просмотр информации об индексах базы данных:

```
select * from DBA_INDEXES
```



```
select * from DBA_INDEXES
```

Query Result | SQL | Fetched 50 rows in 4,233 seconds

	OWNER	INDEX_NAME	INDEX_TYPE	TABLE_OWNER	TABLE_NAME	TABLE_TYPE	UNIQUENESS	COMPRESSION
1	SYS	I_USER1	NORMAL	SYS	USER$	TABLE	UNIQUE	DISABLED
2	SYS	I_OBJ#	CLUSTER	SYS	C_OBJ#	CLUSTER	UNIQUE	DISABLED
3	SYS	I_IND1	NORMAL	SYS	IND$	TABLE	UNIQUE	DISABLED
4	SYS	I_CDEF2	NORMAL	SYS	CDEF$	TABLE	NONUNIQUE	DISABLED
5	SYS	I_OBJ5	NORMAL	SYS	OBJ$	TABLE	UNIQUE	DISABLED
6	SYS	I_PROXY_ROLE_DATA$_1	NORMAL	SYS	PROXY_ROLE_DATA$	TABLE	NONUNIQUE	DISABLED
7	SYS	I_FILE#_BLOCK#	CLUSTER	SYS	C_FILE#_BLOCK#	CLUSTER	UNIQUE	DISABLED
8	SYS	I_FILE1	NORMAL	SYS	FILE$	TABLE	UNIQUE	DISABLED
9	SYS	I_CON1	NORMAL	SYS	CON$	TABLE	UNIQUE	DISABLED
10	SYS	I_OBJ3	NORMAL	SYS	OBJ$	TABLE	NONUNIQUE	DISABLED
11	SYS	I_TS#	CLUSTER	SYS	C_TS#	CLUSTER	UNIQUE	DISABLED

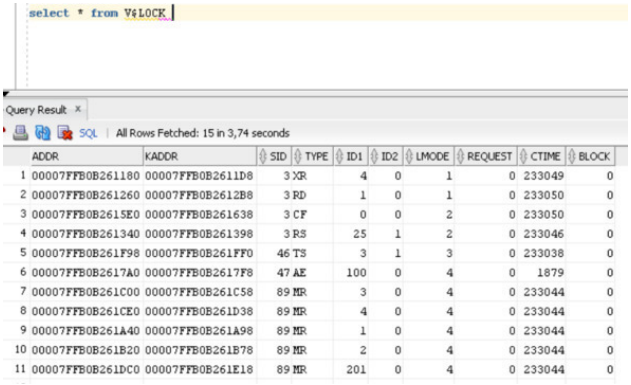
Рисунок 230. Информация о версии базы данных

V$LOCK – в этом представлении перечислены блокировки, которые в настоящее время хранятся в базе данных ORACLE, и запросы, ожидающие блокировку или защелку. Представление используется для получения информации о блокировках и сессиях.

Колонка	Описание колонки
ADDR	Адрес объекта состояния блокировки
KADDR	Адрес блокировки
SID	Идентификатор для проведения сеанса или получения блокировки
TYPE	Тип блокировки пользователя или системы Блокировки пользовательских типов получаются пользовательскими приложениями. Любой процесс, который блокирует другие, вероятно, будет удерживать одну из этих блокировок. Пользовательские блокировки типа: • TM - DML enqueue • TX - очередь транзакций • UL - пользователь предоставил Блокировки типов систем удерживаются в течение очень коротких периодов времени. Тип системы замков приведен в таблице 4-1 .
ID1	Идентификатор блокировки № 1 (зависит от типа)
ID2	Идентификатор блокировки № 2 (зависит от типа)
LMODE	Режим блокировки, в котором сеанс удерживает блокировку
REQUEST	Режим блокировки, в котором процесс запрашивает блокировку: • 0 - нет • 1 - ноль (NULL) • 2 - ряд-S (CC) • 3 - ряд-X (SX)4 - доля (S) • 5 - S / Row-X (SSX) • 6 - эксклюзив (X)
CTIME	Время с момента предоставления текущего режима
BLOCK	Блокировка блокирует другую блокировку

Запрос к V$LOCK на просмотр информации о блокировках базы данных:

```
select * from V$LOCK
```



```
select * from V$LOCK
```

Query Result

SQL | All Rows Fetched: 15 in 3,74 seconds

	ADDR	KADDR	SID	TYPE	ID1	ID2	LMODE	REQUEST	CTIME	BLOCK
1	00007FFB0B261180	00007FFB0B2611D8	3	XR	4	0	1	0	233049	0
2	00007FFB0B261260	00007FFB0B2612B8	3	RD	1	0	1	0	233050	0
3	00007FFB0B2615E0	00007FFB0B261638	3	CF	0	0	2	0	233050	0
4	00007FFB0B261340	00007FFB0B261398	3	RS	25	1	2	0	233046	0
5	00007FFB0B261F98	00007FFB0B261FF0	46	TS	3	1	3	0	233038	0
6	00007FFB0B2617A0	00007FFB0B2617F8	47	AE	100	0	4	0	1879	0
7	00007FFB0B261C00	00007FFB0B261C58	89	MR	3	0	4	0	233044	0
8	00007FFB0B261CE0	00007FFB0B261D38	89	MR	4	0	4	0	233044	0
9	00007FFB0B261A40	00007FFB0B261A98	89	MR	1	0	4	0	233044	0
10	00007FFB0B261B20	00007FFB0B261B78	89	MR	2	0	4	0	233044	0
11	00007FFB0B261DC0	00007FFB0B261E18	89	MR	201	0	4	0	233044	0

Рисунок 231. Запрос информации о блокировках

DBA_LOCK показывает блокировки или защелки DBA_LOCK в базе данных и все невыполненные запросы на блокировку или защелку.

Колонка	Описание колонки
SESSION_ID	Идентификатор сессии удерживающей блокировку
LOCK_TYPE	Тип блокировки
MODE HELD	Режим блокировки
MODE REQUESTED	Запрашиваемый режим блокировки
LOCK_ID1	типа идентификатор типа блокировки 1
LOCK_ID2	типа идентификатор типа блокировки 2
BLOCKING_OTHERS	Блокирует ли блокировка другие обьекты

Запрос к DBA_LOCK на просмотр информации обо всех блокировках базы данных:

```
Select * from DBA_lock
```

Рисунок 232. Запрос информации обо всех блокировках системы

V$SESSION_WAIT отображает ресурсы или события, которые ожидают активные сессии. Используется для просмотра подвисших сессий.

Колонка	Описание колонки
SID	Идентификатор сессии
SEQ#	Порядковый номер, который однозначно определяет это ожидание. Увеличивается за каждое ожидание.
EVENT	Ресурс или событие, которое ожидает сеанс
P1TEXT	Описание первого дополнительного параметра
P1	Первый дополнительный параметр
P1RAW	Первый дополнительный параметр RAW
P2TEXT	Описание второго дополнительного параметра
P2	Второй дополнительный параметр
P2RAW	Второй дополнительный параметр RAW
P3TEXT	Описание третьего дополнительного параметра
P3	Третий дополнительный параметр
P3RAW	Третий дополнительный параметр RAW
WAIT_CLASS_ID	Идентификатор класса ожидания
WAIT_CLASS#	Номер класса ожидания
WAIT_CLASS	Название класса ожидания
WAIT_TIME	Ненулевое значение - это время ожидания последнего сеанса
SECONDS_IN_WAIT	Время ожидания в секундах ожидания. Если WAIT_TIME > 0 , то SECONDS_IN_WAIT - это секунды с начала последнего ожидания, а SECONDS_IN_WAIT - WAIT_TIME / 100 - активные секунды с момента окончания последнего ожидания. STATE VARCHAR2(19) Состояние ожидания: 0 - WAITING (сессия в данный момент ожидает) -2 - WAITED UNKNOWN TIME (продолжительность последнего ожидания неизвестна) -1 - WAITED SHORT TIME (последнее ожидание < -1 - WAITED SHORT TIME секунды) >0 - WAITED KNOWN TIME (WAIT_TIME = продолжительность последнего ожидания)

Запрос к V$SESSION_WAIT о сессиях в режиме ожидания:

```
select * from V$SESSION_WAIT
```

V$SESSION – в этом представлении перечислены сведения о сессиях для каждой сессии. Используется для просмотра подробной информации о сессиях пользователей.

Колонка	Описание колонки
SADDR	Адрес сессии
SID	Идентификатор сессии
SERIAL#	Серийный номер сессии. Используется для уникальной идентификации объектов сессии. Гарантирует, что команды уровня сессии применяются к правильным объектам сеанса, если сеанс заканчивается и другой сессии начинается с тем же идентификатором сессии.
AUDSID	Идентификатор сеанса аудита
PADDR	Адрес процесса, которому принадлежит сессия
USER	Идентификатор пользователя Oracle
USERNAME	Имя пользователя Oracle
COMMAND	Выполняется команда
STATUS	Статус сеанса: ACTIVE – сессия активна, в данный момент выполняется SQL INACTIVE – сессия неактивна KILLED - Сессия отмечена как убитая SNIPED - Сессия неактивна, ожидание на клиенте
SERVER SQL_ADDRESS SQL_HASH_VALUE Используется для уникальной идентификации текущего выполняемого выражения SQL.	Тип подключения к серверу (DEDICATED, SHARED, PSEUDO, NONE)
OSUSER	Имя Пользователь в OS
PROGRAM	Программа OS, которая выполнила подключение к базе данных.
TERMINAL	Имя или тип терминала с которого выполнено подключение
TYPE	Тип терминала (BACKGROUND или USER)
SQL_ADDRESS SQL_HASH_VALUE	Используется для уникальной идентификации текущего выполняемого выражения SQL.
P2RAW	Второй дополнительный параметр RAW
P3TEXT	Описание третьего дополнительного параметра
P3	Третий дополнительный параметр
P3RAW	Третий дополнительный параметр RAW
WAIT_CLASS_ID	Идентификатор класса ожидания
WAIT_CLASS#	Номер класса ожидания
WAIT_CLASS	Название класса ожидания

Запрос к V$SESSION о сессиях экземпляра базы данных:

```
select * from V$SESSION
```

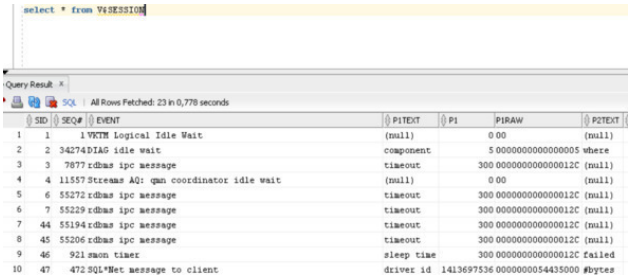


Рисунок 234. Запрос информации о сессиях

DBA_DB_LINKS описывает все DB_LiNK, существующие в базе данных, используется для просмотра подробной информации о заданном DB_LiNK.

Колонка		Описание колонки
OWNER		Владелец или схема DB LINK
DB_LINK		Наименование DB LINK
USERNAME		Имя пользователя базы куда указывает DB LINK
HOST		Хост соединения DB LINK
CREATED		Время дата создания DB LINK
LOCK_ID2		типа идентификатор типа блокировки 2

Запрос к DB_LiNK для информации о DB_LiNK:

```
select * from DBA_DB_LINKS
```

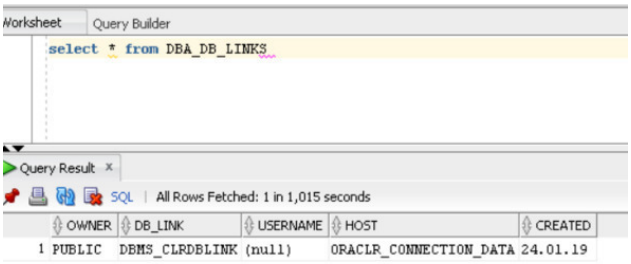


Рисунок 235. Запрос информации db lINk

DBA_OBJECTS – все объекты в базе. Для просмотра информации по заданным объектам, статусам объектов правильно использовать именно это представление.

Колонка	Описание колонки
OWNER	Владелец объекта или схема,
OBJECT_NAME	Наименование объекта
SUBOBJECT_NAME	Наименование субъекта
OBJECT_ID	Идентификатор объекта из словаря данных
OBJECT_TYPE	Тип объекта (talble, index)
CREATED	Дата время создания объекта
LAST_DDL_TIME	Последнее время изменяя объекта
STATUS	Статус объекта VALID, INVALID, or N/A

Запрос к DBA_OBJECTS на просмотр сведений обо всех объектах базы данных:

```
select * from DBA_OBJECTS
```

DBA_ROLES – список всех ролей, которые присутствуют в базе данных.

Колонка	Описание колонки
ROLE	Наименование роли
PASSWORD_REQUIRED	Признак требуется ли пароль

Запрос к DBA_ROLES на просмотр сведений обо всех ролях безопасности базы данных:

```
select * from DBA_ROLES
```

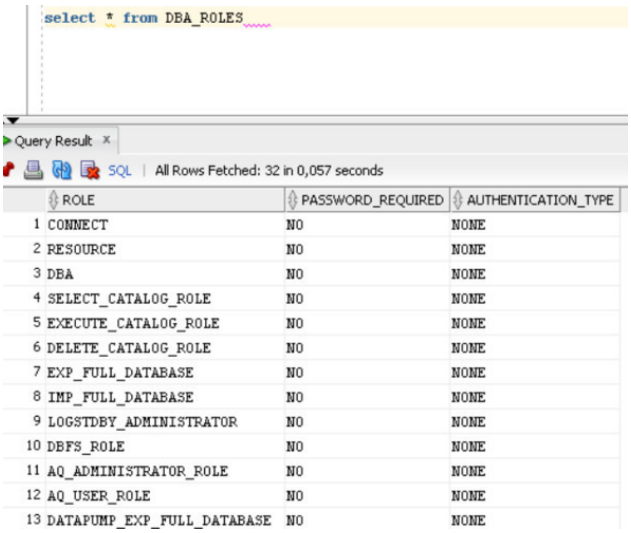


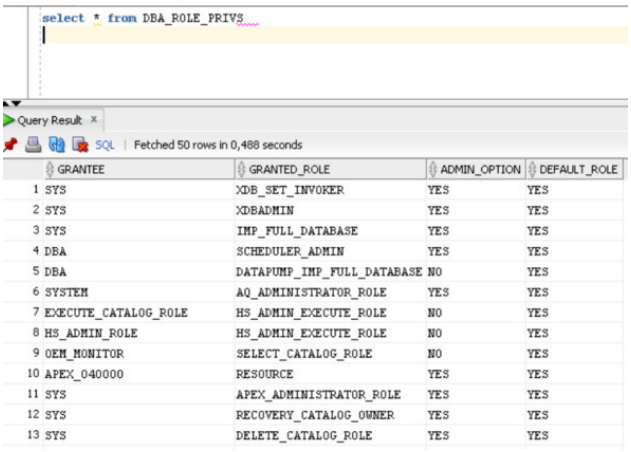
Рисунок 237. Запрос информации по ролям в базе данных

DBA_ROLE_PRIVS описывает права, предоставленные всем пользователям и ролям в базе данных.

Колонка	Описание колонки
GRANTEE	Имя пользователя или роли, получающей грант
GRANTED_ROLE	Имя предоставленной роли
ADMIN_OPTION	Указывает, был ли грант с ADMIN OPTION (YES) или нет (NO)
DEFAULT_ROLE	Указывает, обозначена ли роль как DEFAULT ROLE для пользователя (YES) или нет (NO)

Запрос к DBA_ROLE_PRIVS на просмотр сведений обо всех ролях пользователей базы данных:

```
select * from DBA_ROLE_PRIVS
```



```
select * from DBA_ROLE_PRIVS
```

Query Result | SQL | Fetched 50 rows in 0,488 seconds

	GRANTEE	GRANTED_ROLE	ADMIN_OPTION	DEFAULT_ROLE
1	SYS	XDB_SET_INVOKER	YES	YES
2	SYS	XDBADMIN	YES	YES
3	SYS	IMP_FULL_DATABASE	YES	YES
4	DBA	SCHEDULER_ADMIN	YES	YES
5	DBA	DATAPUMP_IMP_FULL_DATABASE	NO	YES
6	SYSTEM	AQ_ADMINISTRATOR_ROLE	YES	YES
7	EXECUTE_CATALOG_ROLE	HS_ADMIN_EXECUTE_ROLE	NO	YES
8	HS_ADMIN_ROLE	HS_ADMIN_EXECUTE_ROLE	NO	YES
9	OEM_MONITOR	SELECT_CATALOG_ROLE	NO	YES
10	APEX_040000	RESOURCE	YES	YES
11	SYS	APEX_ADMINISTRATOR_ROLE	YES	YES
12	SYS	RECOVERY_CATALOG_OWNER	YES	YES
13	SYS	DELETE_CATALOG_ROLE	YES	YES

Рисунок 238. Информация о привилегиях

DBA_SOURCE – исходные коды всех хранимых объектов. Используется для просмотра исходных кодов процедур.

Колонка	Описание колонки
OWNER	Владелец объекта или схема,
NAME	Наименование объекта
TYPE	Тип объекта FUNCTION, JAVA SOURCE, PACKAGE, PACKAGE BODY, PROCEDURE, TRIGGER, TYPE, TYPE BODY
LINE	Номер строи в исходном коде
TEXT	Текст исходного кода
CYCLE_FLAG	Цикличность последовательности (Y/N)
LAST_DDL_TIME	Последнее время изменения объекта
STATUS	Статус объекта VALID, INVALID, or N/A
CACHE_SIZE	Размер кэша
LAST_NUMBER	Последний номер последовательности

Запрос к DBA_SOURCE на исходные коды всех объектов базы данных:

```
select * from DBA_SOURCE
```

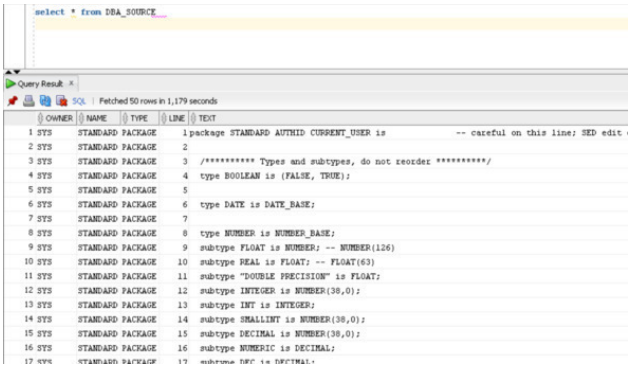


Рисунок 239. Информация об исходных кодах

DBA_TABLESPACES описывает табличные пространства, присутствующие в базе данных. Подробная информация о табличных пространствах. DBA_TABLESPACES используется для того, чтобы посмотреть информацию по заданному табличному пространству.

Колонка	Описание колонки
TABLESPACE_NAME	Наименование табличного пространства
BLOCK_SIZE	Размер блока табличного пространства (в байтах)
INITIAL_EXTENT	Начальный размер экстента по умолчанию (в байтах)
NEXT_EXTENT	Размер добавочного экстента по умолчанию (в байтах)
MIN_EXTENTS	Минимальное количество экстентов по умолчанию
MAX_EXTENTS	Максимальное количество экстентов по умолчанию
MAX_SIZE	Максимальный размер сегментов по умолчанию (в блоках Oracle)
PCT_INCREASE	Процент увеличения по умолчанию для размера экстента
STATUS	Состояние табличного пространства: • ONLINE • OFFLINE • READ ONLY
CONTENTS	Содержимое табличного пространства: UNDO PERMANENT TEMPORARY
LOGGING	Атрибут регистрации по умолчанию: LOGGING NOLOGGING
FORCE_LOGGING	Указывает, находится ли табличное пространство в режиме принудительного ведения журнала (YES) или нет (NO)
EXTENT_MANAGEMENT	Указывает, являются ли экстенты в табличном пространстве управляемыми словарем (DICTIONARY) или локально управляемыми (LOCAL)
ALLOCATION_TYPE	Тип действующего распределения экстента для табличного пространства: SYSTEM UNIFORM USER
PLUGGED_IN	Указывает, подключено ли табличное пространство (YES) или нет (NO)
BIGFILE	Указывает, является ли табличное пространство табличным пространством с большим файлом (YES) или табличным пространством с небольшим файлом (NO)
DEF_TAB_COMPRESSION	Указывает, включено ли сжатие таблицы по умолчанию (ENABLED) или нет (DISABLED) Примечание. Включение сжатия таблиц по умолчанию означает, что все таблицы в табличном пространстве будут созданы с включенным сжатием таблиц, если не указано иное.
DEF_TAB_COMPRESSION	Указывает, включено ли сжатие таблицы по умолчанию (ENABLED) или нет (DISABLED) Примечание. Включение сжатия таблиц по умолчанию означает, что все таблицы в

	табличном пространстве будут созданы с включенным сжатием таблиц, если не указано иное
RETENTION	Отменить сохранение табличного пространства:
GUARANTEE	Табличное пространство является табличным пространством отмены с RETENTION указанным как GUARANTEE
NOT APPLY	табличное пространство не является отмененным табличным пространством
ENCRYPTED	Указывает, зашифровано ли табличное пространство (YES) или нет (NO)
COMPRESS_FOR	Сжатие по умолчанию для каких операций BASIC OLTP QUERY LOW Foot 1 QUERY HIGH Footref 1 ARCHIVE LOW Footref 1 ARCHIVE HIGH ВЫСОКОЙ Footref 1

Запрос к DBA_TABLESPACES на просмотр информации обо всех табличных пространствах базы данных:

```
select * from DBA_TABLESPACES
```

	TABLESPACE_NAME	BLOCK_SIZE	INITIAL_EXTENT	NEXT_EXTENT	MIN_EXTENTS	MAX_EXTENTS	MAX_SIZE	PCT_INCREASE	MIN_EXTLEN	STATUS
1	SYSTEM	8192	65536	(null)	1	2147483645	2147483645	(null)	65536	ONLINE
2	SYSAUX	8192	65536	(null)	1	2147483645	2147483645	(null)	65536	ONLINE
3	UNDOTBS1	8192	65536	(null)	1	2147483645	2147483645	(null)	65536	ONLINE
4	TEMP	8192	1048576	1048576	1	(null)	2147483645	0	1048576	ONLINE
5	USERS	8192	65536	(null)	1	2147483645	2147483645	(null)	65536	ONLINE

Рисунок 240. Информация о табличных пространствах

DBA_TAB_PRIVS описывает все объектные привилегии в базе данных.

USER_TAB_PRIVS описывает привилегии объекта, для которого текущий пользователь является владельцем объекта, лицом, предоставляющим право, или получателем привилегии.

Колонка	Описание колонки
GRANTEE	Имя пользователя, которому был предоставлен доступ
OWNER	Владелец объекта
TABLE_NAME	Наименование объекта. Объект может быть любым объектом, включая таблицы, пакеты, индексы, последовательности и т.
GRANTOR	Имя пользователя, выполнившего грант
PRIVILEGE	Привилегия на объект
GRANTABLE	Указывает, была ли предоставлена привилегия с GRANT OPTION (YES) или нет (NO)
HIERARCHY	Указывает, была ли предоставлена привилегия с помощью HIERARCHY OPTION (YES) или нет (NO)

Запрос к DBA_TAB_PRIVS на просмотр информации обо всех объектных привилегиях базы данных:

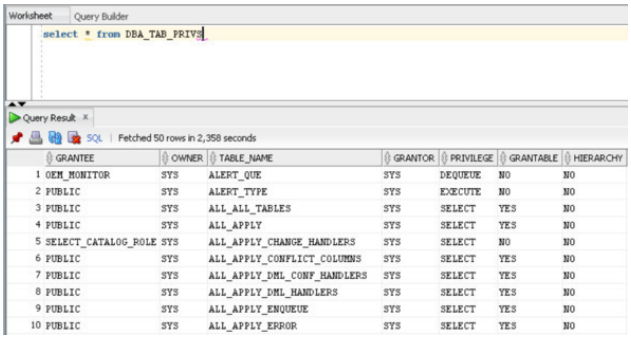


Рисунок 241. Информация об объектных привилегиях базы данных

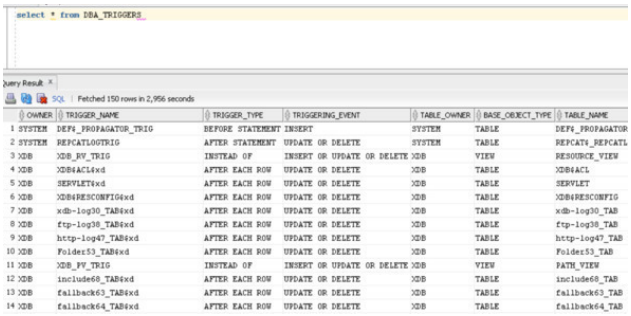
DBA_TRIGGERS описывает все триггеры в базе данных.

USER_TRIGGERS описывает все триггеры, принадлежащие текущему пользователю. Это представление не отображает столбец OWNER.

Колонка	Описание колонки
OWNER	Владелец объекта триггера
TRIGGER_NAME	Название триггера
TRIGGER_TYPE	Когда срабатывает триггер: BEFORE STATEMENT , BEFORE EACH ROW , BEFORE EVENT , AFTER STATEMEN T, AFTER EACH ROW и AFTER EVENT
TRIGGERING_EVENT	DML, DDL или событие базы данных, которое запускает триггер. Для получения списка инициирующих событи
TABLE_OWNER	Владелец таблицы, на которой определяется триггер
BASE_OBJECT_TYPE	Базовый объект, для которого определен триггер: TABLE, VIEW , SCHEMA или DATABASE
TABLE_NAME	Если базовый тип объекта триггера SCHEMA или DATABASE , то этот столбец NULL ; если базовый тип объекта триггера - TABLE или VIEW , в этих столбцах указывается имя таблицы / представления, для которого определен триггер
COLUMN_NAME	Имя столбца вложенной таблицы (если триггер вложенной таблицы), иначе null
REFERENCING_NAMES	Имена, используемые для ссылки на значения столбцов OLD и NEW изнутри триггер
WHEN_CLAUSE	TRUE для выполнения в теле триггера
STATUS	Указывает, включен ли триггер (ENABLED) или отключен (DISABLED)
DESCRIPTION	Описание триггера; полезно для повторного создания оператора создания триггера
ACTION_TYPE	Тип действия триггера (CALL или PL/SQL)
TRIGGER_BODY	Программа, выполняемая триггером при его срабатывании

Запрос к DBA_TRIGGERS на просмотр информации обо всех триггерах базы данных:

```
select * from DBA_TRIGGERS
```



select * from DBA_TRIGGERS

Query Result | SQL | Fetched 150 rows in 2,956 seconds

	OWNER	TRIGGER_NAME	TRIGGER_TYPE	TRIGGERING_EVENT	TABLE_OWNER	BASE_OBJECT_TYPE	TABLE_NAME
1	SYSTEM	DEF$_PROPAGATOR_TRIG	BEFORE STATEMENT	INSERT	SYSTEM	TABLE	DEF$_PROPAGATOR
2	SYSTEM	REPCATLOGTRIG	AFTER STATEMENT	UPDATE OR DELETE	SYSTEM	TABLE	REPCAT$_REPCATL
3	XDB	XDB_PV_TRIG	INSTEAD OF	INSERT OR UPDATE OR DELETE	XDB	VIEW	RESOURCE_VIEW
4	XDB	XDB$ACL#xd	AFTER EACH ROW	UPDATE OR DELETE	XDB	TABLE	XDB$ACL
5	XDB	SERVLET#xd	AFTER EACH ROW	UPDATE OR DELETE	XDB	TABLE	SERVLET
6	XDB	XDB$RESCONFIG#xd	AFTER EACH ROW	UPDATE OR DELETE	XDB	TABLE	XDB$RESCONFIG
7	XDB	xdb-log30_TAB#xd	AFTER EACH ROW	UPDATE OR DELETE	XDB	TABLE	xdb-log30_TAB
8	XDB	ftp-log38_TAB#xd	AFTER EACH ROW	UPDATE OR DELETE	XDB	TABLE	ftp-log38_TAB
9	XDB	http-log47_TAB#xd	AFTER EACH ROW	UPDATE OR DELETE	XDB	TABLE	http-log47_TAB
10	XDB	Folder53_TAB#xd	AFTER EACH ROW	UPDATE OR DELETE	XDB	TABLE	Folder53_TAB
11	XDB	XDB_PV_TRIG	INSTEAD OF	INSERT OR UPDATE OR DELETE	XDB	VIEW	PATH_VIEW
12	XDB	include68_TAB#xd	AFTER EACH ROW	UPDATE OR DELETE	XDB	TABLE	include68_TAB
13	XDB	fallback63_TAB#xd	AFTER EACH ROW	UPDATE OR DELETE	XDB	TABLE	fallback63_TAB
14	XDB	fallback64_TAB#xd	AFTER EACH ROW	UPDATE OR DELETE	XDB	TABLE	fallback64_TAB

Рисунок 242. Информация о триггерах базы данных

DBA_USERS описывает всех пользователей базы данных.

Колонка	Описание колонки
USERNAME	Имя пользователя
USER_ID	Идентификационный номер пользователя
PASSWORD	Зашифрованный пароль
ACCOUNT_STATUS	Статус аккаунта: • OPEN • EXPIRED • EXPIRED(GRACE) • LOCKED(TIMED) • LOCKED • EXPIRED & LOCKED(TIMED) • EXPIRED(GRACE) & LOCKED(TIMED) • EXPIRED & LOCKED • EXPIRED(GRACE) & LOCKED
LOCK_DATE	Дата, когда аккаунт был заблокирован, если статус аккаунта был заблокирован
EXPIRY_DATE	Дата истечения срока действия аккаунта
DEFAULT_TABLESPACE	Табличное пространство по умолчанию для данных
TEMPORARY_TABLESPACE	Имя табличного пространства по умолчанию для временных таблиц или имя группы табличных пространств
CREATED	Дата создания пользователя
PROFILE	Имя профиля ресурса пользователя
INITIAL_RSRC_CONSUMER_GROUP	Исходная группа ресурсов для пользователя
EXTERNAL_NAME	Внешнее имя пользователя

Запрос к DBA_USERS на просмотр информации обо всех пользователях базы данных:

```
select * from DBA_USERS
```

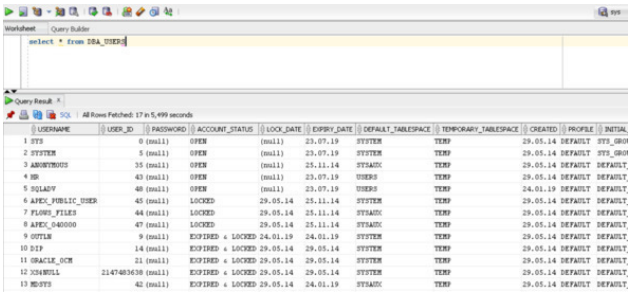


Рисунок 243. Информация о пользователях базы данных

USER_USERS описывает текущего пользователя. В этом представлении не отображаются столбцы PASSWORD или PROFILE.

DBA_VIEWS описывает все представления в базе данных.

Колонка	Описание колонки
OWNER	Владелец представления
VIEW_NAME	Наименование представления
TEXT_LENGTH	Длина текста представления
TEXT	Текст представления
SUPERVIEW_NAME	Наименование супер представления

USER_VIEWS описывает представления, принадлежащие текущему пользователю. Это представление не отображает столбец OWNER.

Запрос к USER_VIEWS на просмотр информации обо всех представлениях базы данных:

```
select * from DBA_VIEWS
```

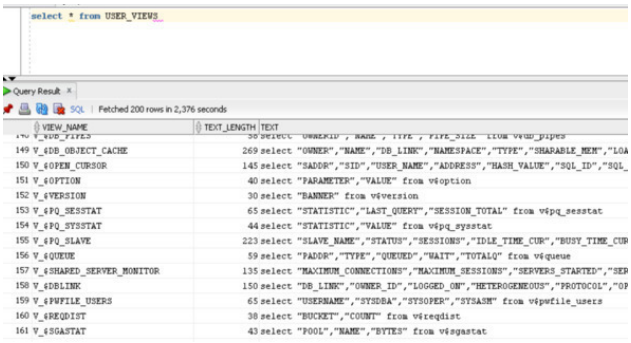


Рисунок 244. Информация о пользователях базы

Важные замечания

Для работы с этими системными представлениями необходимы специальные права.

Вопросы учеников

Для каких конкретных задач используются системные представления?

Например, чтобы посмотреть колонки в заданной таблице, для просмотра индексов, которые есть в заданной схеме.

Пример – просмотр всех нерабочих объектов в базе данных.

```
SELECT * FROM ALL_OBJECTS WHERE STATUS <> 'VALID'
```

Контрольные вопросы и задания для самостоятельного выполнения

1. Каким запросом можно посмотреть все табличные пространства в базе данных?
2. Найти запросом все триггеры для таблицы AUTO.
3. Посмотреть список всех DB_LINK в базе.
4. Посмотреть все представления, где участвует таблица CITY.
5. Найти все имена заблокированных (LOCKED) пользователей.

Шаг 95. Внешние таблицы EXTERNAL TABLE

Введение

Внешние таблицы – специальный механизм ORACLE СУБД, с помощью которого можно обращаться к данным, хранящимся в файлах вне базы данных, как к обычным таблицам.

Для загрузки данных могут использоваться команды драйвера ORACLELoader. К ExternalTABLE не могут применяться операторы изменения данных (DELETE, INSERT, UPDATE, MERGE).

Но к таким таблицам вполне могут применяться стандартные запросы SELECT с использованием групповых операций, агрегатных функций, аналитического SQL.

Все это делает механизм внешних таблиц особенно эффективным для проектов DWH (хранилищ данных), при формировании ETL (процедур загрузки) для хранилищ данных.

Теория и практика

Предположим, у нас есть несколько файлов заданного формата, файлы расположены в каталоге TEMP на диске c:

Необходимо подключить эти файлы как внешние таблицы к экземпляру нашей базы данных.

Название файлов CITY. csv и MAN. csv, кодировка UTF8. Ниже представлено содержание этих файлов текстовом виде.

CITY. csv

1,Москва,10000000
2,Владимир,500000
3,Орел,300000
4,Курск,200000
5,Казань,2000000
7,Котлас,110000
8,Мурманск,400000
9,Ярославль,500000

MAN. csv

9152222221,Андрей, Николаев,1,22
9152222222,Максим, Москитов,1,31
9153333333,Олег, Денисов,3,34
9173333334,Алиса, Никифорова,4,31
9173333335,Таня, Иванова,4,31
9213333336,Алексей, Иванов,7,25
9213333331,Андрей, Некрасов,2,27
9213333332,Миша, Рогозин,2,21
9214444444,Алексей, Галкин,1,38

Вы можете создать эти файлы сами с помощью любого текстового редактора. Напоминаю, что кодировка файлов UTF8.

Перед использованием внешних таблиц необходимо создать специальный объект DIRECTORY, указывающий на каталог, где расположены файлы для внешних таблиц.

CREATE OR REPLACE DIRECTORY EXT_TAB_DATA AS 'c:/temp';

Здесь ext_TAB_DATA – название объекта DIRECTORY;
AS 'c:/TEMP» – каталог, где расположены файлы для формирования внешних таблиц.

Далее формируем временные таблицы.

Создаем таблицу для файла CITY. csv:

```
CREATE TABLE city_ext (
 CITYCODE     NUMBER,
 CITYNAME     VARCHAR2(50),
 PEOPLES  NUMBER
)
ORGANIZATION EXTERNAL (
 TYPE ORACLE_LOADER
 DEFAULT DIRECTORY ext_tab_data
 ACCESS PARAMETERS (
  RECORDS DELIMITED BY NEWLINE
  FIELDS TERMINATED BY ','
  MISSING FIELD VALUES ARE NULL
 )
 LOCATION ('city.csv')
)
PARALLEL 5
REJECT LIMIT UNLIMITED;
```

Здесь CITY_EXT – название таблицы;
• TYPE – драйвер загрузки;
• DEFAULT DIRECTORY – объект-директория, которую мы создали;
• ACCESS PARAMETERS – параметры загрузки файла, см. документацию к LOADER;
• LOCATION – название файла, на основе которого создается внешняя таблица.

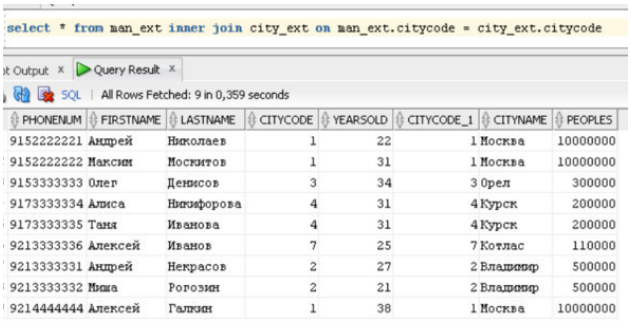
Создаем таблицу для файла MAN. csv:

```
CREATE TABLE man_ext (
 PHONENUM     VARCHAR2(50),
 FIRSTNAME    VARCHAR2(50),
 LASTNAME     VARCHAR2(50),
 CITYCODE     NUMBER,
 YEARSOLD  NUMBER
)
ORGANIZATION EXTERNAL (
 TYPE ORACLE_LOADER
 DEFAULT DIRECTORY ext_tab_data
 ACCESS PARAMETERS (
  RECORDS DELIMITED BY NEWLINE
  FIELDS TERMINATED BY ','
  MISSING FIELD VALUES ARE NULL
 )
 LOCATION ('man.csv')
)
PARALLEL 5
REJECT LIMIT UNLIMITED;
```

Здесь CITY_EXT – название таблицы;
• TYPE – драйвер загрузки;
• DEFAULT DIRECTORY – объект-директория, который мы создали;
• ACCESS PARAMETERS – параметры загрузки файла, см. документацию к LOADER;
• LOCATION – название файла, на основе которого создается внешняя таблица.

Обращаемся к таблицам, которые мы создали, с помощью запроса:

```
select * from man_ext inner join city_ext on man_ext.citycode = city_ext.citycode
```



```
select * from man_ext inner join city_ext on man_ext.citycode = city_ext.citycode
```

Output | Query Result | SQL | All Rows Fetched: 9 in 0,359 seconds

PHONENUM	FIRSTNAME	LASTNAME	CITYCODE	YEARSOLD	CITYCODE_1	CITYNAME	PEOPLES
9152222221	Андрей	Николаев	1	22	1	Москва	10000000
9152222222	Максим	Москитов	1	31	1	Москва	10000000
9153333333	Олег	Денисов	3	34	3	Орел	300000
9173333334	Алиса	Никифорова	4	31	4	Курск	200000
9173333335	Таня	Иванова	4	31	4	Курск	200000
9213333336	Алексей	Иванов	7	25	7	Котлас	110000
9213333331	Андрей	Некрасов	2	27	2	Владимир	500000
9213333332	Миша	Рогозин	2	21	2	Владимир	500000
9214444444	Алексей	Галкин	1	38	1	Москва	10000000

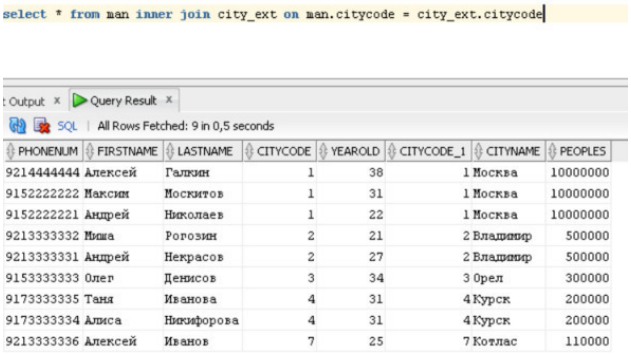
Используем агрегатную функцию и группировку данных.

```
select cityname, count(man_ext.phonenum)
from man_ext inner join city_ext on city_ext.citycode = city_ext.citycode
group by cityname
```

CITYNAME	COUNT(MAN_EXT.PHONENUM)
Владимир	9
Мурманск	9
Казань	9
Курск	9
Москва	9
Котлас	9
Ярославль	9
Орел	9

Соединяем внешнюю таблицу и обычную таблицу в запросе.

```
select * from man inner join city_ext on man.citycode = city_ext.citycode
```



```
select * from man inner join city_ext on man.citycode = city_ext.citycode
```

Output | Query Result

SQL | All Rows Fetched: 9 in 0,5 seconds

PHONENUM	FIRSTNAME	LASTNAME	CITYCODE	YEAROLD	CITYCODE_1	CITYNAME	PEOPLES
9214444444	Алексей	Галкин	1	38	1	Москва	10000000
9152222222	Максим	Москитов	1	31	1	Москва	10000000
9152222221	Андрей	Николаев	1	22	1	Москва	10000000
9213333332	Миша	Рогозин	2	21	2	Владимир	500000
9213333331	Андрей	Некрасов	2	27	2	Владимир	500000
9153333333	Олег	Денисов	3	34	3	Орел	300000
9173333335	Таня	Иванова	4	31	4	Курск	200000
9173333334	Алиса	Никифорова	4	31	4	Курск	200000
9213333336	Алексей	Иванов	7	25	7	Котлас	110000

Используем аналитический SQL.

```
select firstname, lastname, row_number() over (partition by firstname order by firstname) as
rn from man_ext select firstname, lastname, row_number() over (partition by firstname order
by firstname) as rn from man_ext
```

FIRSTNAME	LASTNAME	RN
Алексей	Галкин	1
Алексей	Иванов	2
Алиса	Никифорова	1
Андрей	Николаев	1
Андрей	Некрасов	2
Максим	Москитов	1
Миша	Рогозин	1
Олег	Денисов	1
Таня	Иванова	1

Важные замечания

• Для внешних таблиц могут создаваться представления и синонимы.

• Кодировка файлов во внешних таблицах должна совпадать с кодировкой базы данных.

• На папки, в которых собираются файлы для подготовки внешних таблиц, администратор системы должен выдать специальное разрешение на чтение пользователю, из-под которого устанавливалась ORACLE СУБД.

• Вешние таблицы неэффективны при частом обращении к этим таблицам в высоконагруженных проектах; для таких проектов следует искать другое решение.

Вопросы учеников

Где еще используются на практике внешние таблицы?

Довольно часто внешние таблицы используются при обмене данными между системами.

Где можно посмотреть материалы по ACCESS PARAMETERS?

В описании SQLLoader на сайте ORACLE.

В разобранном примере используется драйвер SQLLoader. Можно ли использовать другие драйвера?

Да, например можно для загрузки данных воспользоваться драйвером DATAPUMP.

Можно ли загрузить данные из внешней таблицы в обычную таблицу?

Да, и на этой возможности основано довольно много ETL (процедур загрузки). Пример – следующий запрос создаст таблицу CITYcCODE:

```
create table cityccode as
select cityname, count(man_ext.citycode) as ctcode from man_ext inner join city_ext on
city_ext.citycode = city_ext.citycode
group by cityname
```

Запрос добавляет в эту таблицу данные, не забудьте завершить запрос операцией COMMIT.

```
insert into cityccode
select cityname, count(man_ext.citycode) as ctcode from man_ext inner join city_ext on
city_ext.citycode = city_ext.citycode
group by cityname
```

Контрольные вопросы и задания для самостоятельного выполнения

1. Изучите материалы данного шага.
2. Выгрузите таблицу AUTO в CSV-файл, перенесите получившийся файл в каталог TEMP.
3. Создайте внешнюю таблицу для созданного файла.
4. Напишите запрос к созданной таблице, группирующий авто по марке и считающий количество цветов заданной марки авто.
5. Создайте на основе этого запроса таблицу.
6. Перегрузите данные из внешнего файла в таблицу, используя SQL-запрос.

День двадцатый

Шаг 96. Оптимизатор запросов, чтение плана запроса

Введение

Любой SQL-запрос выполняется по заданному плану. В плане выполнения запроса описывается последовательность операций для доступа к данным, особенности использования, последовательность операций сортировок и объединения строк таблиц. Профессиональный SQL-разработчик должен уметь читать план запроса, чтобы оценить эффективность выполнения запроса.

Теория и практика

План запроса строит оптимизатор ORACLE, задача разработчика – анализируя план запроса, найти проблемные места в производительности и постараться решить проблемы.

Для того чтобы посмотреть план запроса, необходимо выполнить команду:

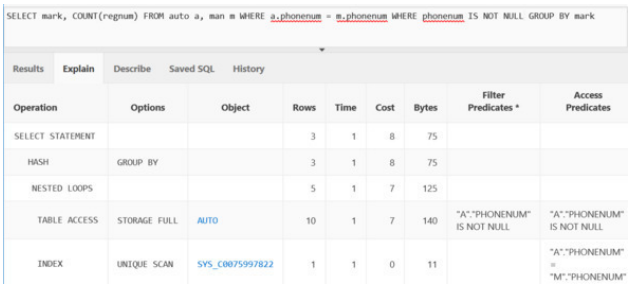
```
EXPLAIN PLAN FOR SELECT FIRSTNAME, LASTNAME FROM MAN WHERE CITYCODE> 1; – или любой другой запрос
```

Здесь SELECT FIRSTNAME, LASTNAME FROM MAN WHERE CITYCODE> 1; может быть изменен на любой другой запрос.

После выполнения данной команды можно посмотреть план запроса с помощью команды:

```
SELECT * FROM TABLE (DBMS_XPLAN. DISPLAY (NULL, NULL,«BASIC»));
```

Во многих редакторах есть специальные средства для просмотра плана запроса, подобный инструмент есть и в ORACLE Appex.



```
SELECT mark, COUNT(regnum) FROM auto a, man m WHERE a.phonenum = m.phonenum WHERE phonenum IS NOT NULL GROUP BY mark
```

Results | Explain | Describe | Saved SQL | History

Operation	Options	Object	Rows	Time	Cost	Bytes	Filter Predicates *	Access Predicates
SELECT STATEMENT			3	1	8	75		
HASH	GROUP BY		3	1	8	75		
NESTED LOOPS			5	1	7	125		
TABLE ACCESS	STORAGE FULL	AUTO	10	1	7	140	"A"."PHONENUM" IS NOT NULL	"A"."PHONENUM" IS NOT NULL
INDEX	UNIQUE SCAN	SYS_C0075997822	1	1	0	11		"A"."PHONENUM" = "M"."PHONENUM"

Рисунок 245. План запроса: сервис APEX

В плане запроса указываются таблицы, используемые в запросе, стоимость и время выполнения каждой операции в запросе.

Как читать план запроса?

План запроса читается с самого дальнего уровня: читается операция самого максимального уровня вложения.

На представленном плане запроса:

- Операция – операция;
- OPTION – доступ к данным;
- Объект – таблица или представление, из которого выбираются данные;
- ROWS – количество строк, которое извлекается операцией;
- TIME – условное время, за которое выполняется та или иная операция;
- COST – стоимость операций для оптимизатора.

Наибольшее внимание следует обратить на параметр COST – это стоимость той или иной операции, стоимость выполнения данной операции и совокупности операций извлечения данных.

Следует стремиться к тому, чтобы минимизировать параметр Cost при выполнении данного запроса. Для этого следует обратить внимание на использование индексов в таблицах

запроса, правильное построение связей. Разберем несколько планов запросов: во-первых, план с рисунка.

– Таблица AUTO: извлекаются все строки из таблицы, стоимость операции 7 (COST), извлекается 10 строк, стоимость операции 7.

– Осуществляется обращение к таблице MAN, которое происходит по UNIQUE INDEX SCAN, извлекается 1 строка (ROW), стоимость операций 0 для каждой строки таблицы AUTO.

– Для объединения двух таблиц (операции 1, 2) используются вложенные циклы NESTED LOOP, в результате операции получаем 5 строк, стоимость данной операции 7 складывается из стоимости операций 1 плюс стоимость операций 2.

– Для связанных данных используется операция группировки, 3 строки, общая стоимость 8.

– Результат помещается в запрос (3 строки).

Далее осуществляется операция группировки со стоимостью cost 8.

Ниже представлен список операций, которые используются, и опций доступа к данным.

Разбор запроса на связь двух таблиц AUTO, MAN.

```
SELECT * FROM sqladv.AUTO, sqladv.man WHERE sqladv.AUTO.phonenum=sqladv.man.phonenum
```

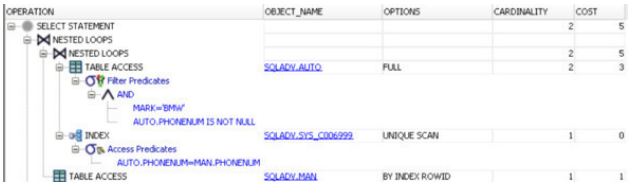
Рисунок 246. План запроса: AUTO, MAN

1) Из таблицы AUTO извлекаются 7 строк – стоимость 3.
2) Из таблицы по идексу извлекаются 9 строк – стоимость 2+1.
3) Обьединение извлеченных строк MERGE JOIN – общая стоимость 6.

Общий результат – стоимость 6, извлечено 7 строк.

Пример 2. Разбор плана запроса на объединение двух таблиц с ограничением MARK = «BMW».

```
SELECT * FROM sqladv.AUTO, sqladv.man WHERE
sqladv.AUTO.phonenum=sqladv.man.phonenum AND mark = 'BMW'
```



1) Из таблицы AUTO извлекаются 2 строки – стоимость 3 (ограничение BMW).
2) Из таблицы по индексу извлекается 1 строка – стоимость 1.
3) Объединение извлеченных строк с помощью вложенных циклов – общая стоимость 6.
Объединенный результат – стоимость 5, извлечено 2 строки.

Простой пример демонстрации работы индексов:

```
CREATE TABLE arr AS SELECT level id, trunc(DBMS_RANDOM.VALUE*10000) as val FROM dual
CONNECT BY level < 1000001
SELECT * FROM arr WHERE val= 11 and id=1670
```

Рисунок 247. План запроса без индекса

Полное сканирование таблицы (FULL), стоимость 530.

```
CREATE INDEX idxtr ON arr(val);
CREATE unique INDEX idxtr1 ON arr(id);
SELECT * FROM arr WHERE val >9000
```

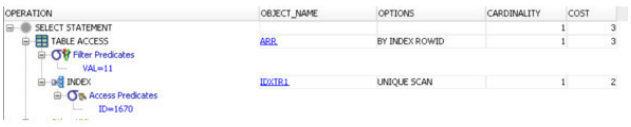


Рисунок 248. План запроса с использованием индекса

Сканирование по индексам – стоимость 3, почти в 150 раз меньше. Демонстрация эффективного использования индексов.

Доступ к данным

FULL TABLE SCAN (FTS)

Полное сканирование таблицы при поиске. Просматривается последовательно каждая строка таблицы.

INDEX LOOKUP

Просмотр значения по заданному индексу. Возвращается уникальный идентификатор ROWID, по ROWID осуществляется доступ к нужной строке таблицы.

INDEX UNIQUE SCAN

Для поиска значения используется уникальный индекс. Находится только одно значение.

INDEX RANGE SCAN

Используется, когда необходимо выбрать по индексу диапазон значений, операции BETWEEN,>, <, <>,> =, <=. Или для условий равенств, например LASTDOC=10.

INDEX FULL SCAN

Полное сканирование индекса. Применяется, если все данные могут быть извлечены только с применением индекса.

FAST FULL SCAN

Быстрое сканирование. Используется при параллельном доступе. Не осуществляет сортировку считанных данных. Может применяться только тогда, когда все столбцы, участвующие в запросе, присутствуют в индексе и имеют ограничение NOT NULL.

INDEX SKIP SCAN

Используется при доступе по составному индексу.

ROWID

Самый быстрый метод доступа к данным. Используется внутренний системный идентификатор записи. Обычно применяется после индексного поиска или при явном указании в SQL-запросе значения ROWID строки данных таблицы.

JOIN

Соединение – это предикат, объединяет данные из нескольких источников данных (таблиц и представлений). Используется при извлечении данных из нескольких таблиц.

СУБД ORACLE использует три различных вида соединений:

NESTED LOOPS (NL)

Соединение таблиц с помощью вложенных циклов. Для связанной таблицы данные перебираются построчно в цикле; если таблиц несколько, то применяются вложенные циклы.

HASH JOIN (HJ)

Соединение таблиц на основе хэш-массива. Предварительно строится хэш-массив, на основании которого связываются таблицы.

CARTESIAN

Соединение с помощью Декартова произведения.

В итоге каждая строка из таблицы 1 соединяется со всеми строками таблицы 2, затем то же самое для каждой строки таблицы 2 по отношению к таблице 3, и так далее.

Операции

SORT

Если есть операции сортировки данных, которые предполагается выполнить:

ORDER BY – GROUP BY – SORT MERGE JOIN

FILTER

Отражает условия WHERE и HAVING.

VIEW

Использование в SQL-запросе представления ORACLE.

MATERIALIZED VIEW

Показывает использование в SQL-запросе материализованного представления ORACLE.

MERGE JOIN

Объединение нескольких таблиц слиянием.

JOIN

Объединение нескольких таблиц.

Важные замечания

HASH JOIN

Самый быстрый алгоритм соединения данных из нескольких источников, появился начиная с версии ORACLE 7.3, однако не является универсальным, то есть не может быть использован для любых условий соединений.

CARTESIAN

Соединение чаще всего генерирует значительное количество информации и является ошибкой кодирования, когда, например, не указаны никакие условия соединения.

NESTED LOOP

В случае отсутствия индексов стоимость выполнения соединения может оказаться неприемлемо высокой.

INDEX SKIP SCAN

Существует начиная с ORACLE 9. В более ранних версиях ORACLE не использовался.

INDEX FULL SCAN

Возвращает считанные данные в отсортированном виде. Может быть неэффективен в ряде случаев.

Как правило, решение об эффективности использования принимает оптимизатор на основе собранной ранее статистики.

FULL SCAN

Если вы видите в запросе FULL SCAN с высокой стоимостью COST, следует рассмотреть план запроса, и возможно – это показание к необходимости для оптимизации запроса.

Вопросы учеников

Как составляется план запроса?

План запроса, как выполнять данный запрос, составляет оптимизатор запросов.

В чем состоит стратегия оптимизации запросов?

В том, чтобы снизить стоимость COST наиболее ресурсоемких операций.

Всегда ли FULL SCAN значит снижение производительности?

Нет, в небольших таблицах FULL SCAN как раз является оптимальным решением.

Контрольные вопросы и задания для самостоятельного выполнения

1. На какие операции следует обратить внимание при чтении плана запроса?
2. Какие операции указывают на объединение нескольких таблиц?
3. Что обозначает INDEX LOOKUP?
4. Опишите способ доступа к строке таблицы ROWID.
5. Опишите соединение HASH JOIN.

Шаг 97. Подсказки оптимизатору

Введение

Планом запроса можно управлять, то есть менять план запроса для более эффективного выполнения запроса. Процесс изменения плана запроса называется стоимостной оптимизацией плана запроса.

Теория и практика

Для изменения плана запроса используется специальный механизм подсказок.

Подсказка пишется в тексте запроса, начинается /*+ и заканчивается */. Подсказка применяется только в запросе, в котором она находится, вложенные запросы требуют своих собственных подсказок.

С помощью подсказок есть возможность оптимизировать запрос с целью увеличения скорости выполнения запроса, производительности запроса.

Подсказки оптимизатора:

FULL

Подсказка FULL управляет запросом, задает оптимизатору режим полного сканирования таблиц.

```
Select /*+ FULL(table_name) */ col1, col2 ...
```

ORDERED

Подсказка ORDERED подсказывает оптимизатору обращаться к таблицам в заданном порядке, в том порядке, который используется после инструкции FROM.

```
select/*+ ORDERED */ column1, column2 ... from table1, table2
```

INDEX

Подсказка INDEX сообщает оптимизатору, что для выполнения данного запроса необходимо использовать один или несколько индексов.

```
select /*+ INDEX(table_name index_name1 index_name2 ...) */ col1, coln  from table1
select /*+ INDEX(sinonm_name index_name1 index_name2 ...) */ col1, coln  from table1
```

ALL_ROWS

Данная подсказка позволяет настроить запрос с максимальной эффективностью.

```
Select /*+ ALL_ROWS */ c1, cn... From Table
```

FIRST_ROWS (n)

Подсказка говорит оптимизатору, что необходимо выбирать заданное количество первых строк.

```
select /*+ FIRST_ROWS */ c1, c2 from Table
```

USE_NL

Подсказка для оптимизатора задает использование вложенных циклов при объединении заданных таблиц.

```
select /*+ USE_NL (tableA tableB) */ column1, column2 ...
```

USE_HASH

Подсказка для оптимизатора задает использование HASH, хэш-таблицы, тип объединения при объединении заданных таблиц.

```
select /*+ USE_hash (tableA tableB) */ column1, column2 ...
```

Примеры

Использование подсказки USE_NL:

```
select /*+ USE_NL(a m) */* from auto a, man m where a.phonenum = m.phonenum
```

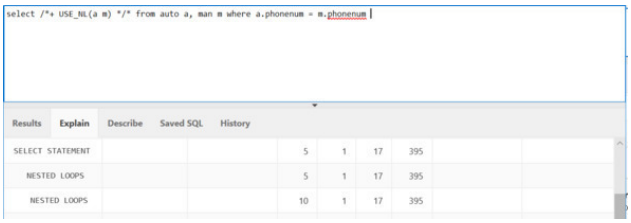


Рисунок 249. План запроса с USE_NL

Использование подсказки USE_HASH:

```
select /*+ USE_HASH(a m) */* from auto a, man m where a.phonenum = m.phonenum
```

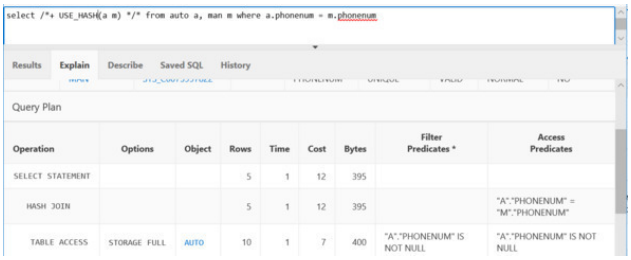


Рисунок 250. План запроса с USE_HASH

Использование подсказки USE_MERGE:

```
select /*+ USE_ MERGE (a m) */* from auto a, man m where a.phonenum = m.phonenum
```

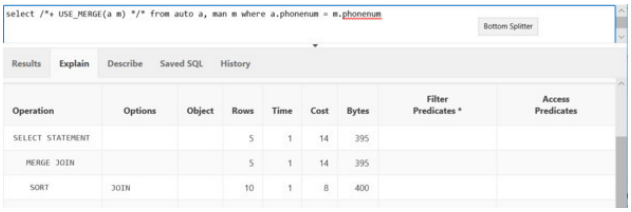


Рисунок 251. План запроса с USE_MERGE

Использование ORDERED:

```
SELECT /*+ ORDERED */* from man m, auto a WHERE a.phonenum = m.phonenum
```

```
SELECT /*+ ORDERED */* from man m, auto a WHERE a.phonenum = m.phonenum
```

Results | Explain | Describe | Saved SQL | History

Operation	Options	Object	Rows	Time	Cost	Bytes
SELECT STATEMENT			5	1	12	395
HASH JOIN			5	1	12	395
TABLE ACCESS	STORAGE FULL	MAN	13	1	5	507
TABLE ACCESS	STORAGE FULL	AUTO	10	1	7	400

Рисунок 252. План запроса ORDERED

Использование нескольких подсказок оптимизаторов одновременно:

```
SELECT /*+ ORDERED USE_NL(a m) */* from man m, auto a WHERE a.phonenum = m.phonenum
```

Results | Explain | Describe | Saved SQL | History

Operation	Options	Object	Rows	Time	Cost	Bytes	Filter Predicates *	Access Predicates
SELECT STATEMENT			5	1	73	395		
NESTED LOOPS			5	1	73	395		
TABLE ACCESS	STORAGE FULL	MAN	13	1	5	507		
TABLE ACCESS	STORAGE FULL	AUTO	1	1	5	40	"A"."PHONENUM" = "M"."PHONENUM" AND "A"."PHONENUM" IS NOT	"A"."PHONENUM" = "M"."PHONENUM" AND "A"."PHONENUM" IS NOT

Рисунок 253. План запроса USE_NL

Важные замечания

• Подсказка по размеру имеет ограничение не более 255 символов.

• Если по базе данных не собрана статистика, то некоторые подсказки могут работать неэффективно.

• Для корректного использования подсказок необходимо, чтобы в базе данных был установлен режим (CHOOSE).

Вопросы учеников

Какие подсказки лучше использовать в запросах с объединением таблиц?

В зависимости от ситуации можно использовать USE_HASH, USE_NL.

Какие подсказки лучше использовать, когда важно сначала получить первые строки запроса?

Для этого есть специальная подсказка FIRST_ROWS, FIRST_ROWS (1).

Какие подсказки лучше использовать, когда важны все строки запроса?

Для этого есть специальная подсказка ALL_ROWS.

Контрольные вопросы и задания для самостоятельного выполнения

1. Повторите материалы данной главы.
2. Поясните, для чего используется подсказка INDEX.
3. Поясните, для чего используется подсказка ORDERED.
4. Поясните, в каких случаях лучше использовать подсказку USE_MERGE.

Шаг 98. Задачи с собеседований в крупные компании и фирмы

Практика

Используя регулярные выражения, подсчитать количество букв О в слове МОЛОКО.

Решение:

```
select regexp_count('МОЛОКО','О') from dual
```

Используя регулярные выражения, найти в таблице **t** такие записи, где в колонке n только цифры.

Решение:

```
select  * from t where regexp_count(n,'[^0-9]{1}')=0
```

Есть таблица логов OPLOG (op VARCHAR2 (50), status VARCHAR2 (50), opdt DATE). Примеры заполнения:

```
Op    status    opdt
Оп1   начало    01.01.2018  10:00
Оп1   заверш    01.01.2018  10:12
Оп2   начало    01.01.2018  11:00
Оп2   заверш    01.01.2018  10:11
Оп3   начало    01.01.2018  12:00
Оп3   заверш    01.01.2018  10:22
```

Найти, сколько времени выполнялась каждая операция.

Решение:

```
select opdt, op, lag(opdt,1,null) OVER (partition by op order by opdt),
opdt-lag(opdt,1,null) OVER (partition by op order by opdt)
from oplog
```

Даны две таблицы, созданные следующим образом:

```
create table test_a (id number, data varchar2(1));
create table test_b (id number);
insert into test_a(id, data) values (10, 'A');
insert into test_a(id, data) values (20, 'A');
insert into test_a(id, data) values (30, 'F');
insert into test_a(id, data) values (40, 'D');
insert into test_a(id, data) values(50, 'C');

insert into test_b(id) values(10);
insert into test_b(id) values(30);
insert into test_b(id) values(50);
```

Напишите запрос, который вернет строки из таблицы TEST_A, ID которых нет в таблице TEST_B, НЕ используя ключевого слова NOT.

Решение:

```
SELECT * FROM test_a T WHERE  T.ID IN (select ID FROM TEST_A MINUS select ID FROM
TEST_B)
еще решения
select * from test_A LEFT JOIN test_b ON test_A.ID = test_B.ID WHERE  test_B.ID IS NULL
select * from test_A where test_A.ID <> all(SELECT test_B.ID From test_b)
select * from test_A minus select * from test_A where test_A.ID IN (SELECT (ID) from TEST_B)
```

Вывести рабочие дни в следующем месяце, выходные не учитываются.

Решение:

```
select dt , to_char(dt,'dy') as dn from (
 select trunc(last_day(sysdate))+level as dt from dual
 connect by level <= (trunc(last_day(trunc(last_day(sysdate)+1))-trunc(last_day(sysdate))))
 )
WHERE to_char(dt,'dy') not in ('сб','вс')
```

Есть таблица:

```
t111(a number not null, b number not null, c number not null);
```

В таблице есть дублирующиеся записи, задача – удалить эти дубли, используя аналитические функции.

Решение:

```
DELETE FROM t111 WHERE rowid in (
select rowid FROM (
 select rowid, row_number() over (partition by a,b,c order by a) rn from t111) where rn>1)
 )
```

Есть таблица:

```
mrgres(
mrg_id number not null, -- менеджер
dept_id number not null, -- отдел
mrg_res number -- продажи
)
```

Найти пять менеджеров с лучшими продажами в каждом отделе, учесть, что у менеджеров могут быть равные продажи.

Решение:

```
SELECT * FROM (
SELECT mrg_id, dept_id, mrg_res, rank() over(partition by dept_id order by mrg_res) rn FROM
mrgres) WHERE rn<=5
```

Есть таблицы:

```
t_t(name varchar2(50)) t_i(name varchar2(50))
```

Необходимо с помощью одного запроса внести названия таблиц в t_t, названия индексов в таблицу t_i из представления ALL_OBJECTS, где OWNER = SYS.

Решение:

```
INSERT ALL
  WHEN object_type = 'TABLE' THEN
    INTO t_t values (object_name)
  WHEN object_type = 'INDEX'  THEN
    INTO t_i values (object_name)
  select  object_name from all_objects where owner = 'SYS'
```

Есть таблицы TEST_A и TEST_B:

```
create table test_a (id number, data varchar2(1));
create table test_b (id number, data varchar2(1));
insert into test_a(id, data) values
(10, 'A'),(20, 'A'),(30, 'F'),(40, 'D'),(50, 'C');
insert into test_b(id) values
(10,'A'),(30, 'M'),(50, 'F');
```

Необходимо обновить данные в таблице TEST_B из TEST_A по идентификатору ID и добавить недостающие записи, сделать это с помощью одной команды.

Решение:

```
MERGE INTO test_b b
   USING (select * from test_a) a
   ON (a.ID = b.ID)
   WHEN MATCHED THEN
    UPDATE   SET b.data  = a.data
   WHEN NOT MATCHED THEN
    INSERT (ID, DATa)  VALUES (a.ID, a.data);
```

Необходимо размножить записи в таблице одним запросом.

```
create table t123(n number);
insert into t123 values (10);insert into t123 values (15);insert into t123 values (20);
надо чтобы было пять значений 10, пять значений 15, пять значений 20
```

Решение:

```
select * from (select level lnl from dual connect by level < 5+1)
cross join (select n from t123)
```

Шаг 99. Задачи для самостоятельного выполнения

Практика

• Выведите календарь с помощью запроса за 2020 год в формате «день, месяц, год»; исключите из календаря выходные и праздники.

• Выведите с помощью запроса таблицу синусов от 1 до 100, исключите из данной таблицы синусы нечетных чисел.

• Выведите данные из таблицы MAN в следующем формате: Имя, Фамилия; Фамилия; Фамилия3:

Максим Иванов; Алексеев; Максимов;

Андрей Фролов; Ростовский, то есть имя, а далее фамилии с этим именем через точку с запятой.

• Из таблицы «город» (CITY) подсчитайте количество городов с количеством населения> 1 000 000 и населением <1 000 000 одним запросом, общее количество городов.

• Есть таблица: валюта, курс, дата курса валюты. Найти среднее по курсам заданной валюты за 20 дней.

• Найти дату с максимальным курсом валюты, вывести в формате «курс, дата, цена».

• Найти первые три курса каждой валюты по убыванию.

• Объясните, когда необходимо использовать индекс по функции.

• Создайте представление, выбирающее данные из таблиц CITY, MAN, AUTO. Напишите запрос к этому представлению.

• Поясните назначение внешних таблиц EXTERNAL TABLE.

• Есть три таблицы AUTOBMW, AUTOLADA, AUTOOTHER. Напишите запрос, который бы добавлял данные одновременно в каждую из таблиц в зависимости от марки автомобиля.

• Напишите запрос, который бы считал данные в таблице AUTO по количеству автомобилей каждого цвета, общие итоги по всем автомобилям.

• Удалите данные из таблицы AUTO1. Восстановите данные с помощью ретроспективного запроса.

• Создайте копию таблицы AUTO с помощью запроса.

• Транспонируйте запрос к таблице MAN, где в заголовках столбцов – имена людей, строчки – фамилии, а в пересечении – возраст этих людей.

• Повторите операцию PIVOT, объясните назначение данного оператора.

• Выберите данные из таблицы AUTO. Выведите данные в следующем виде: марка, цвет и дополнительная колонка. Если марка автомобиля LADA, тогда выведите: это автомобиль Лада, если BMW, выведите: это авто BMW, иначе: это другой автомобиль.

• Соедините две таблицы AUTO, MAN, выведите данные из этих таблиц, используйте WITH.

• Объясните разницу между TIMESTAMP и DATE.

• Объясните, что обозначает NESTED LOOP в плане запроса.

Шаг 100. Подводим итоги. Задачи. Что изучать и читать дальше?

Проверьте ваши знания, попробуйте найти ответы на следующие вопросы по основным темам.

Несколько теоретических вопросов для закрепления учебного материала из книги, постарайтесь ответить на них самостоятельно.

- Поясните назначение предиката ANY.
- Объясните, чем отличаются RIGHT JOIN, LEFT JOIN, INNER JOIN.
- Для чего нужен оператор MINUS в SQL?
- Поясните назначение функции SUBSTR.
- Расскажите, как добавить к текущей дате 2 дня, какие функции для этого следует использовать.
- Для чего в SQL применяется DECODE и CASE? Поясните различие этих операторов.
- Для каких задач чаще всего применяется аналитический SQL?
- Для чего используются аналитические функции LAG, LEAD?
- Поясните назначение команды MODEL.
- Какой командой SQL можно добавить данные одновременно в три таблицы?
- Какой командой можно подвести итог в запросе?
- Что такое режим SERIALIZABLE, когда он используется?
- Если в плане запроса вы видите FULL SCAN, что это может значить?
- С помощью какой подсказки можно указать оптимизатору использовать индекс?

Дополнительные материалы

Рекомендую вам изучить эти темы, чтобы продолжить ваш профессиональный рост.

Остановимся на материалах, которые в данной книге не отражены, но я их настоятельно рекомендую к изучению.

Эти материалы могут быть по следующим темам:

• Распределенные базы данных – в книге была затронута тема распределенных вычислений и удаленных баз данных, тем не менее распределенные вычисления в базах данных как никогда актуальны сегодня. Для дальнейшего профессионального роста эту тему следует изучить подробнее.

• Архитектура – вопросы архитектуры практически не были отражены здесь, тем не менее значение архитектуры базы данных, строения ее структурных компонентов необходимо каждому профессиональному разработчику.

• Сложные типы данных – в одном из предыдущих шагов мы уже рассматривали тип XMLType, есть и другие типы данных: CLOB, BLOB, BFILE или же объектный тип данных. Эти типы используются достаточно часто на практике, и их изучение необходимо для дальнейшего развития разработчика.

• Параллельные вычисления – в ORACLE существуют специальные средства для параллельных вычислений, это когда запрос вычисляется сразу несколькими процессами одновременно. Такой способ достаточно часто используется на практике, и поэтому я рекомендую всем ученикам обязательно изучить тему параллельных вычислений.

• Загрузка данных – обязательный элемент, с которым вы непременно столкнетесь, если будете профессиональным разработчиком. Изучите материалы по импорту, экспорту данных, изучите утилиты IMP EXP, DATAPAMP, DATAEXPLORER.

• Права и привилегии – разграничение прав и привилегий, доступ и безопасность необходимо знать профессиональному разработчику базы данных. Данную информацию можно получить из документации к СУБД.

• Резервное копирование – администратор должен делать резервные копии базы данных, если случится аварийная ситуация, поэтому необходим практический опыт создания резервных копий, управления резервными копиями базы данных.

• Восстановление базы данных. Восстановление работы базы данных после сбоев – одна из самых важных тем в администрировании баз данных. Одна из прямых обязанностей администратора базы данных – в кратчайшие сроки восстановить работоспособность базы.

• Вопросы оптимизации и производительности – в данной книге были затронуты вопросы оптимизации и производительности базы данных, но на самом деле это достаточно обширная и сложная тема, ей посвящаются целые книги; изучение оптимизации и производительности базы данных – дополнительный плюс к вашему профессиональному росту.

Литература к прочтению

Я также бы хотел посоветовать к изучению некоторую литературу. В этих книгах отражены практически все вопросы из раздела «Дополнительные материалы».

Сам я изучил эти книги и считаю, что их можно порекомендовать. Каждая из этих книг позволит более существенно продвинуться в следующих профессиональных областях: изучение SQL, администрирование серверов баз данных, изучение архитектуры различных СУБД.

Чтение и уверенное владение практическими приемами, изложенными в данных книгах, позволит вам достичь высочайшего профессионального уровня в работе с базами данных.

Одна из перечисленных книг – об MS SQL Server (большинство посвящены ORACLE СУБД). Книга Энтони Моллинаро раскрывает особенности запросов SQL в различных системах управления базами данных – СУБД MS SQL, ORACLE, PostgreeSQL.

Перечислю эти интереснейшие книги, прокомментировав самые важные моменты.

Том Кайт. ORACLE для профессионалов

Некоторые специалисты называют эту книгу библией ORACLE. Лично я считаю, что данную книгу должен изучить любой, кто собирается работать с серьезно ORACLE СУБД. Книга из двух томов.

Автор – один из самых известных в мире специалистов по IT-технологиям, эксперт по разработке решений для ORACLE СУБД, является ведущим одного из самых популярных блогов по ORACLE СУБД в мире.

Первый том посвящен правильной разработке приложений для ORACLE СУБД, подробному обзору архитектуры ORACLE, разбору проблем одновременного доступа. Также рассматриваются подробно сущности ORACLE СУБД – транзакции, таблицы, индексы.

Во второй части рассматриваются загрузка данных, материализованные представления, стабилизация плана оптимизатора, стратегии и настройка планов производительности, механизм использования внешних таблиц.

Энтони Моллинаро. SQL: сборник рецептов. Издательств O'Relly

В этой книге автор рассматривает решение задач, наиболее часто возникающих в практической работе на языке SQL для различных баз данных: MS SQL, ORACLE, PostrgeSQL.

Книгу можно рассматривать как практическое пособие, учебник и как справочник.

Автор разобрал наиболее сложные задачи и варианты их решения.

Душан Петкович. MS SQL Server: руководство для начинающих

Прекрасная книга по MS SQL Server. В книге раскрыты определенные особенности архитектуры MS Server, тонкости процесса установки MS SQL SERVER, среда управления SQL Server MANagement Studio.

Описывается устройство индексов и таблиц в MS SQL, типы индексов, типы таблиц, вопросы администрирования MS SQL, а также отладка приложений MS SQL Server.

В книге раскрываются основы Transact SQL, написание процедур, функций. Подробно описана система прав и объектов – система безопасности.

Сэм Апалти. ORACLE DATAbASe 11g: руководство администратора баз данных

Прекрасное руководство для администратора базы данных ORACLE. Автор рассказывает о своем личном опыте работы. Рассматриваются наиболее сложные проблемы и их решения в работе администратора базы данных: настройка инструментов резервного копирования, установка и модернизация ORACLE, управление и мониторинг работающей базы данных, экспорт и импорт баз данных.

Книга включает подробный обзор основных инструментов, используемых для администрирования крупных серверов ORACLE.

Джонатан Льюис. Ядро ORACLE. Внутреннее устройство для администраторов и разработчиков баз данных

Замечательный учебник по архитектуре базы данных ORACLE. Автор – прекрасный профессионал, который несколько десятилетий работал с ORACLE СУБД. В книге подробно описаны ядро базы данных, процессы, потоки, механизм транзакций, оптимизация, блокировки и защелки, отладка и просмотр журналов, восстановление системы после сбоев.

Одна из самых лучших книг, которые мне попадались.

Кэрри Милсан, Джефф Холт. ORACLE: оптимизация производительности

Авторы предлагают свою собственную уникальную методику оптимизации ORACLE-приложений. Методику, которая подразумевает анализ данных в системах представлений, разработку технологии измерения производительности, сбор трассировочных данных, работу с профилем ресурсов. Книга содержит массу полезных сведений по архитектуре базы данных ORACLE.

Желаю вам успеха в вашей профессиональной карьере и надеюсь, что моя книга вам поможет достичь профессиональных высот.

Удачи вам в дальнейшем профессиональном росте.

Чалышев Максим Михайлович
SQL Advanced School
SQLADV.ru