ФЕДЕРАЛЬНОЕ АГЕНТСТВО ПО ОБРАЗОВАНИЮ
ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
«САМАРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Кафедра информатики и вычислительной математики

БАЗЫ ДАННЫХ.
OPEN OFFICE BASE. ORACLE

Утверждено редакционно-издательским советом университета в качестве практикума

Самара
Издательство «Самарский университет»
2009

УДК 004.65
ББК 73
Б17

Рецензент доцент кафедры геоинформатики
Самарского государственного аэрокосмического университета,
кандидат технических наук В.В. Пшеничников

Авторы: М. С. Русакова, Е. В. Рогачева, И. П. Яковлева, Е. А. Смагина

Б17 **Базы данных. Open Office Base. Oracle** : практикум / [М. С. Русакова, Е. В. Рогачева, И. П. Яковлева и др.]. - Самара : Изд-во «Самарский университет», 2009. - 148 с.

В предлагаемом практикуме приведены базовые сведения о работе в Open Office Base (версия 2.3), рассмотрены основы работы с сервером баз данных Oracle 9i/10g. Здесь также содержится ряд лабораторных работ для освоения наиболее часто используемых возможностей Open Office Base, сервера баз данных Oracle. Каждая лабораторная работа снабжена указаниями по выполнению и содержит набор учебных заданий разных уровней сложности. По каждому разделу предусмотрена разработка индивидуального задания.

Изучение данного практикума позволит студентам в дальнейшем эффективно использовать Open Office Base в работе с базами данных, успешно работать с сервером баз данных Oracle.

Предназначен для студентов специальностей «Математика», «Механика», «Прикладная математика и информатика», «Математическое обеспечение и администрирование информационных систем».

УДК 004.65
ББК 73

Введение

В практикуме содержатся основные сведения о разработке приложений баз данных средствами пакета Open Office.org Base и информационных систем с архитектурой клиент-сервер с использованием СУБД Oracle. Пакет Open Office.org Base обеспечивает как механизм подключения к внешним СУБД, так и работу со встроенной реляционной СУБД HSQLDB, входящей в состав Open Office.org. Небольшой размер и независимость от платформы последней делают ее удачным выбором, в частности, для создания небольших («настольных») приложений. СУБД Oracle является распространенной СУБД промышленного масштаба.

Практикум соответствует программе курсов «Практикум на ЭВМ», «Базы данных и экспертные системы», «Базы данных».

Часть 1 разработана Рогачевой Е.В. и адаптирована к Open Office Base версии 2.3 Русаковой М.С. Часть 2 (теоретическая часть) является совместной разработкой Русаковой М.С. и Смагиной Е.А.; все задания к лабораторным работам в СУБД Oracle составлены Русаковой М.С; разделы по оптимизации запросов в Oracle и по применению индексов при работе с СУБД Oracle - Яковлевой И.П.

В первой части практикума последовательно излагаются основные приемы работы с пакетом Open Office.org Base на примере создания базы данных «Фирма». Вторая часть содержит лабораторные работы, выполнение которых позволит студентам в дальнейшем уверенно работать с СУБД Oracle, пакетом Aqua Data Studio, овладеть языком SQL. Каждая лабораторная работа снабжена необходимым теоретическим материалом, примерами, иллюстрирующими и дополняющими его, контрольными вопросами и заданиями. И первая, и вторая части завершаются набором индивидуальных заданий, позволяющих студентам закрепить полученные навыки.

Авторы практикума выражают благодарность доценту кафедры ИиВМЛуканову А.С. за помощь в определении круга тем лабораторных работ в Open Office Base и СУБД Oracle.

Часть 1. Введение в СУБД Open Office Base

Лабораторная работа MI
Создание таблиц и установление связей

Целью этой лабораторной работы является изучение системы управления базами данных Open Office Base. Предполагается, что обучающиеся знакомы с основными понятиями теории баз данных.

Создание базы данных

Чтобы создать базу данных в Open Office Base (далее будем сокращенно обозначать программу ООВ), можно либо воспользоваться мастером создания базы данных, где выбрать пункт «Создать базу», по щелчку на кнопке «Далее» переходите к следующему диалоговому окну, в котором вы можете зарегистрировать базу данных, открыть ее в режиме редактирования и сразу же заказать создание таблиц. После проведенных манипуляций вам будет предложено сохранить базу, после чего можно приступать к созданию таблиц. Если же вам необходимо параллельно с вашей новой базой создать еще одну, то можно воспользоваться пунктом меню «Файл» -> «Создать» -> «Базу данных», после чего будет запущен уже знакомый вам мастер создания базы.

Теперь можно переходить к созданию таблиц. В левой части окна ООВ можно переключаться между режимами работы с таблицами, запросами, формами и отчетами. Нас интересуют на данном этапе таблицы. В правой верхней части окна перечисляются задачи, которые можно выполнять над таблицей - это создание таблицы в режиме дизайна, создание таблицы при помощи мастера и создание представления. Для начала предлагается выбрать создание таблицы в режиме дизайна. Учебная база данных, которую вы будете разрабатывать, - это фирма.

Фирма имеет несколько отделов (допустим, производственный отдел, рекламный отдел, бухгалтерию). В каждом из отделов есть несколько сотрудников, один из которых - начальник. Каждый сотрудник имеет уникальный табельный номер и (неуникальную) должность. Должности соответствует должностной оклад. В каждом отделе есть телефоны (возможно, несколько). Отделы имеют номера (по крайней мере, для внутреннего учета).

Создание таблиц

Итак, после того, как вы выберете «Создание таблиц в режиме дизайна», в открывшемся окне вы увидите таблицу из трех столбцов: «Имя поля», «Тип поля» и «Описание». Первые два столбца являются обязательными для заполнения. Столбец «Описание» предназначен для «внутренних» нужд пользователя: в нем вы можете поместить комментарий, описание поля, объяснение, почему оно нужно, и т. п. Теперь можно заполнять строки, чтобы получить макет таблицы.

Какие таблицы потребуются для описания фирмы? Во-первых, это таблица «Сотрудники». В ней будут содержаться данные о сотрудниках: табельный номер, личные данные (фамилия, имя, отчество; почтовый индекс, адрес, домашний телефон; мы ограничимся только ФИО), в каком отделе работает, является ли начальником. Во-вторых, это таблица «Отделы». В этой таблице можно сохранять данные об отделах: их названиях, их начальниках. В-третьих, в таблице «Должности» будут храниться сведения о должностных окладах. Наконец, должна быть таблица «Телефоны», в которой указаны данные о номерах телефонов и о том, к какому отделу они относятся.

Замечание 1. О нормализации таблиц говорилось в лекциях.

Создайте макет таблицы «Телефоны», имеющий следующие поля: «Номер телефона», тип - текстовый (допустим и числовой; см. замечание 1) и «Номер отдела», тип - числовой. Название поля набирается в столбце «Имя поля», а тип выбирается из выпадающего списка в столбце «Тип данных». Если хотите, добавьте описания. Сделайте поле «Номер телефона» ключевым. Вы можете воспользоваться контекстным меню, которое вызывается нажатием правой клавиши мыши. Проще всего сохранить сделанную работу, просто закрыв окно с таблицей. Назовите эту таблицу «Телефоны». Следующая таблица, которая будет разработана - «Должности». Эта таблица будет играть роль справочника. Ради экономии места в других таблицах лучше использовать номера должностей - разрядов, а не их текстовые названия (к тому же это ускоряет ввод и снижает вероятность ошибок). «Разряд» и будет ключевым полем в этой таблице. Кроме этого, нужны поля «Наименование должности» и «Должностной оклад». Поле «Разряд» будет иметь числовой тип, поле «Наименование должности» - текстовый, поле «Должностной оклад» - числовой, с денежным форматом.

Теперь таким же образом создайте таблицы «Отделы» и «Сотрудники».

Замечание 2. Тип поля «номер телефона» выбран текстовым, чтобы можно было использовать разделители (черточки) при записи номеров. Если просто выбрать числовой тип, придется записывать телефоны в таком виде: 332211.

Замечание 3. Если при создании макета таблицы у вас возникла потребность что-то изменить, исправить, дополнить, это легко сделать. Режим разработки макета таблицы называется режимом конструктора. Двойной щелчок поуже созданной таблице откроет ее для заполнения, а щелчок по кнопке «Редактирование» в окне документа базы данных приведет к открытию именно макета таблицы.

Замечание 4. В такой простой базе данных сложно запутаться в названиях полей. ООВ как раз и предполагает работу с достаточно простыми (класса «настольные») базами данных и приветствует одинаковое наименование в таблицах полей, описывающих одно и то же. Так, в нашем случае поле «Номер отдела» будет фигурировать в таблицах «Сотрудники», «Телефоны», «Отделы». Такой подход позволяет несколько автоматизировать создание базы данных

Когда вы закончите работу, должны получиться (примерно) следующие макеты.

Таблица 1. Сотрудники

Имя поля	Тип данных	Описание
Табельный номер	Числовой	Ключевое поле
ФИО	Текстовый	
Разряд	Числовой	
Номер отдела	Числовой	

Таблица 2. Отделы

Имя поля	Тип данных	Описание
Номер отдела	Числовой	Ключевое поле
Наименование	Текстовый	
Начальник	Числовой	

Таблица 3. Должности

Имя поля	Тип данных	Описание
Разряд	Числовой	Ключевое поле
Наименование	Текстовый	
Оклад	Денежный	

Таблица 4. Телефоны

Имя поля	Тип данных	Описание
Номер телефона	Текстовый	Ключевое поле
Номер отдела	Числовой	

Заполнение таблиц

Заполнить таблицу очень просто. Дважды щелкните на ярлыке (значке) таблицы в окне базы данных, и перед вами откроется окно заполняемой таблицы. Конечно, с точки зрения опытного разработчика, было бы правильно вначале установить связи между таблицами, разработать форму для ввода, что позволило бы проконтролировать ввод данных, однако учиться лучше на «живых» данных. Возможно, вам даже следует вооружиться- карандашом и бумагой, нарисовать таблицы и придумать данные, которыми они будут заполнены.

Итак, давайте пока ограничимся тремя описанными выше отделами: бухгалтерией, производственным и рекламным отделами. Присвоим им номера: 01, 02 и 03 соответственно. Предположим, что в бухгалтерии работает три сотрудника, в рекламном отделе - четверо, а в производственном - пятеро (итого - двенадцать человек), включая начальников этих отделов. Придумайте сами фамилии, имена и отчества сотрудников фирмы, а также их табельные номера и должности (например, начальник отдела, заместитель начальника отдела, бухгалтер, инженер, менеджер по рекламе, стажер). Теперь осталось разобраться с телефонами. Допустим, что в бухгалтерии и производственном отделе установлено по два телефона, а в рекламном отделе - четыре. Таким образом, нужно придумать восемь номеров телефонов.

Заполните каждую таблицу, следя за тем, чтобы данные в них были согласованы. Например, в таблице «Телефоны» не должны появиться несуществующие номера отделов, а в таблице «Сотрудники» - несуществующие должности.

Пример заполненных таблиц

Таблица 5. Отделы

Номер отдела	Наименование	Начальник
1	Бухгалтерия	1001
2	Производственный отдел	2006
3	Рекламный отдел	3021

Таблица 6. Должности

Разряд	Наименование	Оклад
6	Начальник отдела	15000 р.
5	Заместитель начальника отдела	14000 р.
4	бухгалтер	13500 р.
3	Инженер	13000 р.
2	Менеджер по рекламе	12500 р.
1	Стажер	11200 р.

Таблица 7. Сотрудники

Табельный номер	ФИО	Разряд	Номер отдела
1001	Иванов И.И.	6	1
1005	Петров П.П.	5	1
1012	Сидоров С.С.	4	1
2004	Федоров Ф.Ф.	5	2
2006	Владимиров В.В.	6	2
2009	Павлов П.П.	3	2
2013	Николаев Н.Н.	3	2
2019	Алексеев А.А.	1	2
ЗОН	Васильев В.В.	2	3
3015	Александров А.А.	5	3
3021	Романов Р.Р.	6	3
3023	Кириллов К.К.	1	3

Таблица 8. Телефоны

Номер телефона	Номер отдела
211-22-33	1
211-95-34	1
211-78-23	2

Установка связей между таблицами

Следующим шагом будет установка связей между таблицами. Выберите в меню пункт «Сервис» -> «Связи». В результате в главном окне появится еще одно окно - документа схемы данных. Добавьте по очереди все созданные вами таблицы в окно схемы данных.

Замечание 1. Обратите внимание: добавленная в схему данных таблица не исчезает из диалогового окна «Добавить»! Если вы поместили несколько экземпляров одной и той же таблицы, удалите лишние, чтобы не загромождать схему данных.

Установить связи между таблицами очень легко. Например, чтобы связать таблицы «Отделы» и «Телефоны», достаточно просто перетащить мышкой поле «Отделы: номер отдела» на поле «Телефоны: номер отдела». Когда вы отпустите клавишу мыши, появится новая связь. Можно вызвать диалоговое окно, в котором указаны таблицы и поля, между которыми устанавливается связь. Отметьте «Обновление каскада». Каскадное удаление связанных записей не менее полезно, однако использование его может привести к потере данных, и мы пока не будем им пользоваться.

Замечание 2. О типах связей вы можете прочитать самостоятельно. Если вам потребуется изменить связь, проще всего выделить ее, щелкнув по ней мышкой (черная линия станет жирной), и воспользоваться контекстным меню.

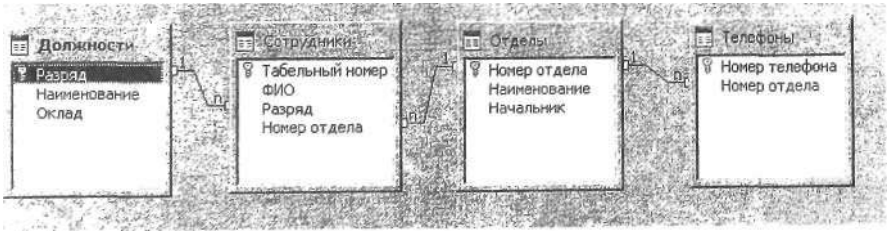

Рис. 1. Схема связей в БД «Фирма»

Теперь, когда таблицы связаны, можно позаботиться о внешнем представлении данных. С базой данных должно быть удобно работать. Одним из средств такого внешнего представления являются формы.

Вопросы и задания

1. Как добавить таблицу в схему данных?
2. В чем отличие между таблицей в режиме конструктора и таблицей в режиме заполнения?
3. Как сделать поле таблицы ключевым?
4. Что произойдет, если ни одно поле таблицы пользователь не сделал ключевым?
5. Обязательно ли добавлять описания в макет таблицы?
6. Опишите, как установить и изменить связи между таблицами.
7. Что такое целостность данных?
8. Что такое потенциальный ключ, внешний ключ?
9. Какие вы можете назвать виды отношений? Поясните на примере.
10. Где в базе нарушается целостность данных?
11. Где в базе присутствует логически отношение «многие-ко-многим»?
12. Что такое каскадное обновление и каскадное удаление?
13. Можно ли добавлять произвольные данные в родительскую таблицу? В дочернюю?
14. Добавьте в поле «Описание» макета каждой таблицы описание доменов, используемых в базе,

Лабораторная работа №2
Создание форм. Ввод данных с помощью форм

Создание форм (часть П

При создании форм лучше всего сначала воспользоваться услугами мастера форм, а затем отредактировать сделанную им заготовку. Переключитесь в окне базы данных на страницу «Формы».

Таблицы в базе данных связаны достаточно сложным образом. Поэтому мы не будем сразу строить форму, отображающую все данные. Начнем с простого: построим форму для таблицы «Сотрудники».

Щелкните мышкой на пункте «Использовать мастера для создания формы» и в появившемся диалоговом окне «Создание форм» выберите из выпадающего списка «Таблицы и запросы» пункт «Таблица: Сотрудники». Ваш выбор немедленно отразится на списке доступных полей: в нем будут показаны все поля таблицы «Сотрудники». Теперь следует переместить поля, которые будут отображаться в форме, в список выбранных полей. Чтобы переместить в этот список подсвеченное поле, нужно использовать кнопку «>». Кнопка «» » перемещает в список выбранных полей все поля, отображенные в списке доступных полей.

Какие поля нужно отображать? Очевидно, что отобразить придется поля «Табельный номер» и «ФИО»: эти поля присутствуют только в таблице «Сотрудники». Поля «Разряд» и «Номер отдела» используются для связи с таблицами «Должности» и «Отделы» соответственно. Поэтому лучше взять данные непосредственно из этих таблиц. Мастер форм прекрасно справляется с такой работой, и единственная проблема состоит в том, что созданная им форма не слишком удобна для ввода данных. Вот почему чуть позже нам придется призвать на помощь другого мастера. А пока переместите поля «Табельный номер» и «ФИО» в список выбранных полей и нажмите кнопку «Далее». В следующем диалоговом окне предлагается выбрать субформу. Пока нам не нужно использовать подчиненную форму, поэтому пропускаем этот шаг. На следующем шаге расположим элементы управления в соответствии с предлагаемым мастером вариантом их расположения. Выберите, к примеру, вариант «Столбцы - подписи сверху». На следующем этапе мастер предлагает обозначить режим источника данных, в котором можно запретить отображать данные, запретить изменение, добавление и удаление данных. Пока этот шаг тоже пропустим. Перейдя по кнопке «Далее», можно определить стиль оформления формы, а затем назвать форму и выбрать дальнейшие действия - модификацию или работу с формой.

По умолчанию имя формы совпадает с именем таблицы - «Сотрудники», и в дальнейшем мы будем так и называть эту форму. Вообще говоря, для наших целей следовало бы открыть форму в режиме конструктора, чтобы изменить макет. Однако сейчас подходящий момент посмотреть на результаты (хотя и промежуточные) вашей работы. Выберите пункт «Работа с формой» и нажмите кнопку «Готово». В результате будет открыто окно, в котором отобразится табельный номер и фамилия первого (по счету в таблице «Сотрудники») сотрудника. С помощью навигационной панели «Запись», расположенной внизу формы, можно перемещаться по записям.

Следующий шаг: отобразить отделы, в которых сотрудники работают. Нажмите на кнопку «Редактирование» в окне базы данных. На экране появится панель инструментов (если ее нет, то надо зайти в пункт меню «Вид» -> «Панели инструментов» и включить панель «Элементы управления») и макет формы. Принцип работы с объектами, размещенными на форме, интуитивно ясен: их можно растягивать или перемещать с помощью мыши, можно редактировать текст надписей. Вероятно, некоторое время вы потратите на то, чтобы расположить «красиво» уже имеющиеся поля. Когда вы закончите, обратитесь к панели инструментов. Она снабжена всплывающими подсказками: задержите мышку на пару секунд над кнопкой, и она сообщит вам о своем назначении. Из доступных элементов для наших целей (отобразить названия отделов) подходят два: «Поле» и «Список». Первый элемент является более подходящим, если форма используется в информационных целях. Если же разрабатывается форма для ввода данных, лучше применить список. Это позволит выбирать нужный отдел по названию из выпадающего списка. Щелкните по компоненту «Список» мышкой и переместите ее к окну макета формы. Указатель мыши примет форму компонента с плюсом в верхнем левом углу. Теперь растяните компонент в области формы - тут же появится первое диалоговое окно мастера создания полей со списком. В этом окне нужно выбрать таблицу, на основе которой будут переданы данные в форму. Наш случай очевиден: мы хотим использовать значения из таблицы «Отделы». Поэтому выбираем первый ответ и нажимаем на кнопку «Далее». На следующем шаге требуется обозначить поля, которые станут значениями поля со списком. Нам понадобится отобразить «Наименование», поэтому выберите его и нажмите кнопку «Далее».

Для связи данных на следующем шаге необходимо указать связанные поля из таблицы значений (поле «Номер отдела» таблицы «Сотрудники») и поля из таблицы списка (поле «Номер отдела» из таблицы «Отделы»). После этого можно нажимать на кнопку «Готово» и смотреть, что

получилось. Теперь на вашей форме уже не два, а три поля, и при перемещении по записям отображается отдел, в котором работает тот или иной сотрудник. Как вы заметите, список у нас никак не именуется, для того, чтобы его как-то назвать, можно опять перейти в режим конструктора формы, поставить метку (с той же самой панели элементов управления) и поэкспериментировать с цветами и выравниванием.

В точности так же можно добавить еще один список, который будет отображать должность сотрудника. В качестве выбранных полей из таблицы «Должности» следует использовать «Наименование», связанными полями будут «Разряд» из таблицы «Сотрудники» и «Разряд» из таблицы «Должности». Выполните это самостоятельно.

Создание форм (часть 21

Итак, построенная вами форма «Сотрудники» показывает данные о сотрудниках, используя для этого таблицы «Сотрудники», «Отделы» и «Должности». Обратите внимание, что таблицы «Отделы» и «Должности» связаны с таблицей «Сотрудники» отношением «один-ко-многим». Осталась одна незадействованная таблица - «Телефоны». Она напрямую не связана с таблицей «Сотрудники», а связана только с таблицей «Отделы». Вместе с тем при отображении данных о сотруднике было, бы удобно видеть номера телефонов, по которым с ним можно связаться. Будем предполагать, что сотрудника некоторого отдела могут пригласить к любому из телефонов этого отдела - и тогда это типичный пример связи «многие-ко-многим», которая вполне очевидно и естественно разбивается на две связи «один-ко-многим» (связи между таблицами «Отделы» и «Сотрудники», а также между таблицами «Отделы» и «Телефоны»).

Как представить данные, связанные отношением «многие-ко-многим», на форме? Одна из возможностей состоит в том, чтобы использовать так называемые подчиненные формы. Подчиненная форма - это форма, которая находится внутри другой формы, называемой главной или первичной. Такие формы, как правило, применяются для вывода данных из таблиц, связанных отношением «один-ко-многим». При этом в подчиненной форме выводятся записи, связанные с текущей записью в главной форме.

Оставим на время форму «Сотрудники» и создадим форму, выводящую данные об отделах и телефонах, установленных в них.

Вновь выберите пункт «Использовать мастера для создания формы» а из таблицы «Отделы» - поля «Название отдела» и «Номер отдела» для

отображения. На следующем шаге отметьте галочкой «Добавить субформу» и проверьте, чтобы по умолчанию была «Субформа, основанная на выбранных полях». В субформе будем отображать номера телефонов, поэтому выбираем таблицу «Телефоны» и перемещаем в форму все поля этой таблицы. Затем необходимо обозначить поля, которые будут объединять (согласовывать) данные в ваших формах. Ими будут поле «Номер отдела» таблицы «Телефоны» и «Номер отдела» таблицы «Отделы». После этого появится уже знакомое вам диалоговое окно, предлагающее расположить управляющие элементы, только теперь можно выбрать расположение головной формы блоками, а субформы - как лист данных. Следующие шаги, связанные с установкой источников данных, выбором стиля формы и имени формы, вам уже должны быть хорошо знакомы. Выполните их и просмотрите полученный результат. Обратите внимание, что при навигации по записям головной формы в субформе происходит автоматическая смена номеров телефонов. При этом вы можете добавлять новые номера (попробуйте это сделать, например, для второго отдела).

Однако подчиненную форму можно создать не только в режиме мастера при создании главной. Проиллюстрируем создание подчиненной формы для формы «Сотрудники» - разместим номера телефонов на главной форме и заставим их согласованно изменяться. Откройте форму «Сотрудники» в режиме редактирования. Нам понадобится для работы панель инструментов «Дизайн формы» - она должна быть отображена внизу окна (если ее нет, то включите ее в пункте меню «Вид» -> «Панели инструментов»). Щелчком по кнопке «Навигатор форм» откройте окно навигатора. В нем отображены все формы (в нашем случае пока это одна главная MainForm) и элементы управления, расположенные на форме. Выберите главную форму и из контекстного меню создайте новую форму, которую мы будем делать подчиненной. По умолчанию имя вновь созданной формы - Standart, но можете переименовать ее, например, в SubForm. Пока на ней еще ничего нет, но для размещения здесь номеров телефонов надо правильно прописать свойства формы. Для этого из контекстного меню надо открыть окно «Свойства формы» для нашей субформы и на закладке «Данные» прописать следующие свойства: «Тип содержимого источника» - «Таблица», «Содержимое» - выбираем таблицу «Телефоны», «Связь с главным полем» - делаем щелчок по кнопке с многоточием и в новом диалоговом окне Link Fields выбираем поле связи «Номер отдела» для таблиц «Телефоны» и «Сотрудники». Теперь на нашу субформу можно размещать номера телефонов. Но среди управляющих элементов нет способа отображения в виде листа данных, поэтому на панели инструментов «Элементы управления» надо сначала запустить

«Дополнительные элементы управления» (при этом проверить, чтобы был активен «Мастер» - иначе при добавлении нового компонента на форму не будет запускаться мастер обработки и придется вручную прописывать все свойства, для которых кое-где надо быть неплохо знакомым с языком запросов). На дополнительной панели управляющих элементов теперь можно выбрать «Таблицу» и поместить ее на субформу (проверьте, чтобы в навигаторе была активна субформа, а не главная форма). Мастер предложит вам выбрать поля таблицы «Телефоны» для отображения, нам нужны только их номера, поэтому выделяем одно поле и щелкаем по кнопке «Готово». Наша субформа создана. Посмотрите форму «Сотрудники» в режиме работы и попробуйте добавить в нее новые данные (например, поставить еще один телефон).

Итак, с помощью формы «Сотрудники» теперь можно получить полные сведения о любом из сотрудников фирмы.

Ввод данных с помощью форм

Вероятно, вы уже попробовали ввести какие-нибудь новые данные (или отредактировать старые) с помощью созданной формы. Если нет, то откройте форму «Сотрудники» в обычном режиме отображения и попробуйте добавить нового сотрудника (допустим, стажера Никитина Н.Н. с табельным номером 3078) в рекламный отдел, а также установить в рекламном отделе новый телефон (например, 211-70-98; вероятно, вы уже обратили внимание, что в рекламном отделе соблюдается правило: каждому сотруднику - по телефону). Чтобы это сделать, перейдите к первой пустой записи в таблице «Сотрудники». Заполните вручную пустые поля «Табельный номер» и «ФИО», а отдел и должность нового сотрудника выберите из выпадающих списков. Таблица телефонов в этот момент пуста, и добавить новый номер телефона для рекламного отдела пока нельзя. Чтобы это сделать, щелкните по кнопке «Сохранить запись», чтобы сведения о новом сотруднике были внесены в таблицы, а затем допишите в конец выводимого списка телефонов новый номер.

А что делать, если в фирме будет создан новый отдел или введена новая должность? Например, принято решение развернуть локальную компьютерную сеть, для обслуживания которой потребуется создать отдел автоматизации, в котором будут работать системный администратор, инженер по обслуживанию аппаратуры и программист? Форма «Сотрудники» пока не предоставляет возможности создавать новые отделы и новые должности. Впрочем, добиться этого не так уж сложно.

Вопросы и задания

1. Поясните, какие поля и почему были выбраны для отображения в форме. Как создать поле со списком и для чего оно нужно?
2. Назовите известные вам типы связей между таблицами. Почему возникла необходимость использовать подчиненную форму для отображения телефонов отделов?
3. Создайте формы для введения новых сведений об отделе и должности.
4. Модифицируйте форму «Сотрудники» так, чтобы на ней отображалась фамилия начальника отдела.

Лабораторная работа №3
Создание и обработка запросов

Разработка и использование запросов ("часть 1. вводная)

В разрабатываемой нами базе данных есть большой недостаток: при добавлении сотрудников через форму в таблице «Отделы» не обновляется поле «Начальник отдела». Почему - понять не так уж сложно. Действительно, не существует ни прямой, ни косвенной связи между данными о разряде сотрудника, номере отдела, в котором он работает, и полем «Начальник». Конечно, было бы удобно, если бы при выборе из выпадающего списка должности «Начальник отдела» табельный номер указанного сотрудника сразу бы заносился в таблицу «Отделы». Однако такая операция требует некоторых навыков программирования, а мы сознательно ограничиваем рассмотрение простейшими приемами работы с базами данных. Поэтому назначать начальника пришлось бы с помощью дополнительного запроса на обновление данных.

К сожалению, ООВ не поддерживает запросы на обновление, вставку и удаление данных. Несмотря на то что ООВ во многом является аналогом Access, она проигрывает последнему по части возможности работы с формой, широте формирования запросов, и совсем уж плохо дела обстоят с отчетами. ООВ предусматривает возможность составления запросов на специальном языке запросов SQL (Standard Query Language), который является общепринятым стандартом в мире баз данных. Поэтому можете самостоятельно попробовать написать вышеперечисленные виды запросов и убедиться в справедливости этого утверждения.

Пока же остановимся подробнее на средствах визуального проектирования запросов, не требующих знания какого-либо языка. Конечно, эти средства не отменяют необходимости четко понимать логику запроса.

Создание и использование запросов (часть 2, запрос-выборка)

Поставим задачу отобразить интересующие нас сведения в следующем формате: (наименование отдела |ФИО сотрудника - начальника отдела|.

Это можно сделать с помощью запроса-выборки. Переключитесь на страницу «Запросы» окна базы данных и выберите пункт «Создать запрос в режиме дизайна». Добавьте в новый запрос таблицы «Отделы» и «Сотрудники». В графу «Поле» первой колонки поместите поле «Отделы:

наименование» (для этого достаточно сделать двойной щелчок мышью по соответствующему полю в таблице), в графу «Имя таблицы» - «Отделы». Условие отбора должно логически выглядеть как

[Отделы]![Начальник] = [Сотрудник]![Табельный номер]

Чтобы это условие выполнилось, необходимо в окне конструктора запроса проставить связь между полями «Начальник отдела» таблицы «Отделы» и «Табельный номер» таблицы «Сотрудники» (аналогично тому, как вы это делали в окне схемы данных при проставлении связей между таблицами).

Закройте запрос и сохраните его как «Список начальников». Чтобы просмотреть результаты выполнения этого запроса, дважды щелкните по его значку в окне базы данных. Список начальников отделов будет выдан в виде простой таблицы.

Замечание 1. Просмотреть результаты запроса, не сохраняя его, можно с помощью кнопки «Выполнить запрос» на панели инструментов. Если вы выберете кнопку «Вкл./выкл. вид дизайна», то сможете посмотреть текст запроса на SQL.

Теперь создадим форму на основе только что созданного запроса. Переключитесь на страницу «Формы» окна базы данных и нажмите на кнопку «Создать». Поскольку форма будет создаваться на основе только одного запроса, воспользуйтесь мастером. Сохраните созданную форму под именем «Начальники отделов». Эта форма будет использоваться исключительно в информационных целях. Поэтому хорошим решением будет запретить в соответствующем диалоговом окне мастера как редактирование, так и добавление, удаление данных, отображаемых в этой форме.

Замечание 2. Если этого не сделать, то пользователь сможет ввести в форму новый отдел, но не сможет указать его начальника. Такой способ создания новых отделов нельзя признать удачным, потому его и следует исключить.

Создание и использование запросов (часть 3, запрос с параметрами)

Те запросы, которые были вами созданы, в некотором смысле «статичны». Чтобы изменить характер или отображение информации с помощью этих запросов, нужно редактировать их непосредственно. Это не всегда удобно. ООВ позволяет создавать и использовать запросы, в которых можно изменять некоторые параметры. Например, пользователь хочет узнать размер зарплаты какого-то конкретного служащего. Ему не нужны сведения обо всех, которые можно получить с помощью стандартного запроса-выборки.

Замечание 1. Да, конечно, в нашем случае оклад жестко привязан к разрядной сетке. Но со временем у сотрудников могут появиться персональные надбавки, льготы и т. п. Попробуйте дополнить базу данных такой информацией в качестве дополнительного упражнения.

Чтобы создать запрос с параметром, создайте сначала обычный запрос-выборку, содержащий следующие данные: ФИО сотрудника и его оклад. Затем в той колонке, в графе «Поле» которой размещено «ФИО», поместите в графе «Критерий» имя параметра, предваренное двоеточием, например, :Fam. Сохраните запрос под именем «Запрос об окладе» и выполните его, дважды щелкнув по значку запроса в окне базы данных. При этом появится диалоговое окно «Ввод параметра» с приглашением «FIO» и пустой строкой, в которой пользователь и должен напечатать данные сотрудника. Напечатайте в этой строке, например, «Филиппов Ф.Ф.», нажмите на OK - и вы сразу же увидите, что оклад этого сотрудника составляет 13700 р. в месяц.»

Замечание 2. Фамилию сотрудника, разумеется, надо печатать без кавычек.

У разработанного запроса есть следующий недостаток. Введенные фамилия и инициалы в точности должны совпадать с теми, что содержатся в базе данных. В противном случае никакой информации не выдается. Попробуйте, например, напечатать в строке запроса «Иванов» (без инициалов) - и, нажав на OK, вы увидите пустую таблицу. В принципе, избежать такой неприятности можно было бы, использовав оператор Like. Если в графе «Критерий» под полем «ФИО» прописать Like 'Ф*', то при выполнении запроса будут выведены данные обо всех сотрудниках, чья фамилия начинается с литеры «Ф». Попробуйте самостоятельно написать такой запрос.

Создание и использование запросов ("часть 4, групповые операции)

Достаточно часто возникает необходимость в вычислении каких-либо итоговых величин. Например, нужно узнать величину среднего оклада по отделу или же выяснить, каков фонд заработной платы данного отдела или фирмы в целом. Наконец, можно подсчитать, сколько телефонов приходится на одного сотрудника в данном отделе. В данном пункте мы ограничимся созданием запроса о фонде заработной платы по отделам. Фактически задача сводится к вычислению сумм по группам записей.

Выберите на странице «Запросы» окна базы данных пункт «Создание запроса в режиме дизайна» и добавьте в бланк нового запроса таблицы «Отделы», «Должности» и «Сотрудники». Теперь можно приступать к заполнению колонок. В графе «Поле» первой колонки поместите поле «Отделы: наименование». В графе «Имя таблицы» автоматически появится имя таблицы «Отделы», а в графе «Функция» нужно выбрать команду Group.

Вторая колонка будет содержать собственно вычисляемое выражение. Графу «Поле» нужно заполнить по следующему образцу: выбираем в нее поле «Оклад» таблицы должности, а в графе «Функция» выберите «Sum». Вот собственно и все - запрос готов.

Замечание 2. Помимо функции Sum существует еще ряд функций, наименования которых можно посмотреть в выпадающем списке графы «Функции».

Просмотрите результаты запроса, и если он корректно работает, сохраните его.

Вопросы и задания

1. Как можно увидеть текст SQL-запроса, который генерирует ООВ при создании запроса «по образцу» (т.е. с помощью бланка)? Прочитайте этот текст и прокомментируйте его.
2. Имеет ли значение регистр символов в запросе «Запрос об окладе»? Почему?
3. Как запретить ввод в поля формы?
4. Какие групповые операции возможны в ООВ? Опишите их.
5. Создайте запрос с параметром, вычисляющий среднюю величину оклада в данном отделе (название отдела вводит пользователь).

Лабораторная работа №4
Генерация отчетов

Генерация отчетов ("часть 1)

Создание отчетов в ООВ возможно с использованием мастера. После создания отчета доступна последующая коррекция макета, но возможные действия относятся более к оформлению макета, чем к определению его структуры и содержания. В данной работе будут создаваться только простые отчеты. ООВ позволяет создавать отчет либо на основе одного запроса, либо на основе одной таблицы. Если вам необходимо вывести данные из нескольких таблиц, то перед созданием отчета предварительно придется написать запрос, выводящий необходимые данные.

Выберите пункт «Использовать мастер для создания отчета» и в качестве основы для отчета запрос «Список начальников». В списке доступных полей вы увидите два поля: «Отделы: наименование» и «Сотрудники: ФИО». Переместите оба поля в список выбранных полей и нажмите на кнопку «Далее». Следующее диалоговое окно предлагает выбрать названия меток для выбранных полей. Их, естественно, вы пропечатываете сами, но по умолчанию названия меток совпадают с именами полей. Следующий шаг «Структура» необходим для создания отчета с группировкой данных. Пока нас не интересует эта возможность, поэтому переходим далее. Шаг «Выбор стиля» содержит варианты оформления разметки данных и разметки колонтитулов. Также, если это необходимо, есть возможность выбирать ориентацию страницы - книжную или альбомную. Наконец, на последнем шаге предлагается на выбор создание статического и динамического отчетов (нам нужен динамический, чтобы любое изменение в данных базы отображалось в отчете) и варианты работы: модификация отчета или его просмотр. Выберите просмотр отчета и оцените результаты своей работы и работы ООВ.

Переключитесь в режим редактирования макета отчета и попробуйте самостоятельно поварьировать его внешний вид. К примеру, измените названия меток, цвета шрифта, формат даты и номеров страниц.

Генерация отчетов ("часть 2)

Теперь, когда у вас появился некоторый опыт в создании отчетов, перейдем к разработке отчетов с группировкой данных. Для этого создадим отчет, содержащий список сотрудников фирмы, сгруппированный по отделам.

Напишите соответствующий запрос и начните создавать на его основе отчет. В диалоговом окне «Структура» следует добавить группировку по полю «Отделы: наименование». Для этого в списке в левой части диалогового окна нужно выделить это поле и нажать на кнопку «>». Завершите создание отчета самостоятельно.

Вопросы и задания

1. Разработайте усовершенствованную версию отчета «Сотрудники фирмы». В новом отчете дополнительно должны указываться должности сотрудников, а фамилии сотрудников должны быть расположены в порядке, соответствующем старшинству должностей (старшей считается должность начальника, далее - заместитель начальника и т. д. Подсказка: воспользуйтесь номерами разрядов). Если сотрудники занимают одинаковые должности, их фамилии должны располагаться в-алфавитном порядке (воспользуйтесь сортировкой данных).
2. Как можно изменить порядок сортировки значений того или иного поля в готовом отчете?

Лабораторная работа №5
Индивидуальные задания

Задания для самостоятельного выполнения

В индивидуальных заданиях предполагается самостоятельная разработка базы данных средствами Open Office Base. Должны быть созданы (и заполнены) таблицы, формы для ввода и редактирования данных, наиболее «вероятные» запросы **и** отчеты (с точки зрения пользователя). В самом задании указаны некоторые возможности, которые должна предоставлять разработанная база данных. Реализация этих возможностей обязательна. При этом подразумевается, что данные можно добавлять и редактировать. Удаление данных (разумеется, согласованное) не требуется в обязательном порядке ни в одном из вариантов; реализация такой возможности приветствуется.

При сдаче задания необходимо продемонстрировать схему данных (и прокомментировать ее), показать, как можно внести новую информацию в базу данных и отредактировать уже имеющиеся записи, представить имеющиеся запросы и отчеты и создать запрос (или отчет), предложенный преподавателем.

1. **Разработайте базу данных «Библиотека».** Она должна содержать данные о книгах, их авторах, месте и времени Издания. У книги может быть несколько авторов, автор же может написать несколько книг. В библиотеке может быть несколько экземпляров одной книги (с разными инвентарными номерами), при этом какие-то из них находятся «на стеллажах», а какие-то - «на руках у читателей». База данных позволяет узнать, есть ли такая книга в библиотеке вообще и может ли она быть выдана в данный момент.
2. **Разработайте базу данных «Склад».** На складе хранятся товары бытового назначения, которые поступают от поставщиков и распределяются по сети магазинов. Один поставщик может поставлять несколько товаров (допустим, утюги, электрические чайники и т. п.). В то же время товар одного наименования может поставляться разными поставщиками. (Предполагаем, что товары производятся малоизвестными фирмами и их цена определяется исключительно функциональностью, а не «громким именем»). База данных позволяет «заказать» товар у поставщика, «отгрузить» товар в магазин (в том случае, если товар в запрашиваемом количестве 'присутствует на складе).

3. **Разработайте базу данных «Издательство».** Издательство работает с авторами, заказывая им книги по разным темам. При этом книги по одной теме могут быть заказаны нескольким авторам. В то же время один автор может работать над книгами сразу по нескольким темам. Книги состоят из глав. Завершив главу, автор «отсылает» ее в издательство. Когда книга готова более чем на 70%, издательство помещает ее в свой тематический план.

4. **Разработайте базу данных «Магазин».** Магазин продает товары бытового назначения (впрочем, род товаров вы можете определить самостоятельно). Покупатель может приобрести несколько товаров одновременно, при этом «выбивается» единый чек. Поскольку на проданный товар существует гарантия, необходимо обеспечить хранение сведений о товаре на протяжении всего гарантийного срока. В течение этого срока покупатель может обратиться в магазин с просьбой о замене некачественного товара. После проверки, действительно ли товар куплен в этом магазине (по номеру чека) и гарантия на него действует, товар должен быть заменен. Напомним, что запас товаров в магазине должен пополняться по мере их продажи.

5. **Разработайте базу данных «Подписка».** Почтовое отделение проводит подписку на газеты и журналы. Каждый подписчик может оформить подписку на несколько изданий одновременно. Разумеется, одно издание может быть выписано несколькими подписчиками. Будем считать, что подписка может быть оформлена на срок от 2 до 12 месяцев. База данных позволяет узнать, сколько времени осталось до окончания действия подписки (и, быть может, напомнить о том, что ее следует продлить). Среди подписных изданий могут появляться новые, а некоторые существующие могут прекратить выпуск. В последнем случае об этом следует известить подписчиков.

6. **Разработайте базу данных «Фонотека».** В ней должны содержаться сведения об авторах и исполнителях музыкальных произведений, а также носителях, на которых они записаны (кассеты, диски различных форматов и т. п.). Одно и то же музыкальное произведение может быть записано в исполнении различных музыкантов. В то же время музыкант может исполнять различные произведения разных авторов. База данных позволяет узнать о наличии в фонотеке той или иной записи (поиск можно вести как по названию произведения, так и по фамилии автора или исполнителя), прослушать выбранную запись (мы не будем обсуждать техническую

реализацию этой функции), а также составить различные рейтинги (по числу обращений).

7. **Разработайте базу данных «Отдел кадров предприятия».** В ней должны храниться сведения о сотрудниках, их перемещениях по службе, изменениях в оплате труда (в том числе доплатах за стаж), об отпусках. Структурной единицей предприятия считается отдел. Сотрудник может работать только в одном отделе. Одновременно в отпуске может находиться не более 30% списочного состава отдела, при этом «руководящий состав» должен уходить в отпуска строго по очереди (заместитель начальника отдела не может уйти в отпуск, пока из отпуска не вернется начальник). Сотрудник может переходить из одного отдела в другой, при этом стаж должен сохраняться. База данных позволяет узнать различные сведения о сотруднике, в том числе - может ли он уйти в отпуск в данные сроки.
8. **Разработайте базу данных «Турнир».** В турнире участвует несколько (спортивных) команд, причем каждая команда в течение турнира должна сыграть со всеми остальными не менее 2 раз. Турнир проводится в несколько туров (посчитайте, сколько). Игры одного тура должны проводиться в один день. За победу команде присуждается 3 очка, за ничью - 1 очко, за поражение - 0 очков. База данных позволяет узнать статистику (число результативных игр и ничьих) для каждого тура или для каждой команды (число очков, побед, поражений, ничьих). Будем считать, что расписание турнира составляется вручную; база данных должна дать ответ: сколько игр данная команда провела с тем или иным соперником. (Возможно, вам будет удобно разделить турнир на циклы, полагая, что в течение цикла каждая команда должна ровно один раз сыграть со всеми остальными.)
9. **Разработайте базу данных «Хобби».** База данных должна содержать сведения о людях и их увлечениях. Каждый человек может иметь несколько увлечений. Несколько людей могут увлекаться одним и тем же. Список увлечений должен быть расширяемым. Следует хранить информацию об образовании и / или профессии каждого человека, его возрасте, степени увлеченности предметом (глубине знаний в этой области). База данных позволяет найти человека, который наиболее подробно может ответить на некоторый вопрос. Вопрос определяется предметными областями (и может, вообще говоря, относиться к нескольким предметным областям сразу).
10. **Разработайте базу данных «Меню».** База данных должна содержать кулинарные рецепты различных блюд в разных

категориях, например, «Закуска», «Горячее блюдо», «Десерт» и т. п. Рецепт каждого блюда приводится в расчете на одну порцию, указывается также длительность изготовления. База данных позволяет узнать, какие продукты и в каком количестве потребуются, согласно выбранному меню, на день (завтрак, обед, ужин) или на праздничный ужин, а также то, сколько времени потребуется на приготовление (возможно, временем приготовления, допустим, обеда можно считать время приготовления того блюда, у которого оно наибольшее).

11. **Справочник туриста.** Турагентства и предлагаемые услуги: страна, город (или маршрут круиза), условия проживания и проезда, экскурсионное обслуживание, сервис принимающей стороны, стоимость путевки.

12. **Салон видеопроката.** База видеофильмов: название, студия, жанр, год выпуска, режиссер, исполнители главных ролей, краткое содержание, субъективная оценка фильма. Факт наличия фильма в видеотеке. Оформление выдачи и возврата кассеты.

13. Ежедневник. База намечаемых мероприятий - дата, время и протяженность, место проведения. Автоматическое напоминание ближайшего дела: по текущей дате и времени; удаление вчерашних дел либо перенос на будущее. Анализ «накладок» - пересечений планируемых дел. Просмотр дел на завтра, послезавтра и так далее.

14. Бюро знакомств. База потенциальных женихов и невест: пол, регистрационный номер, дата регистрации, сведения о себе, требования к партнеру. Выбор подмножества подходящих кандидатур, подготовка встреч (формирование приглашения для знакомства). Перенос в архив пар, решивших свои семейные проблемы, удаление клиентов, отказавшихся от услуг.

15. **Записная книжка.** Анкетные данные, адреса, телефоны, место работы или учебы, должность знакомых, коллег и родственников, характер знакомства, деловые качества и так далее. Автоматическое формирование поздравления с днем рождения (по текущей дате). Упорядочение по алфавиту и по дате последней корректировки. Поиск по произвольному шаблону.

16. **Справочник коммерческих банков.** Наименование, адрес, статус (форма собственности), условия хранения средств на лицевом счете (годовые проценты на различных видах вкладов). Выбор банка с наибольшим процентом для заданного типа вклада.

17. Автосалон. База новых и подержанных отечественных и зарубежных автомобилей: марка, год выпуска, технические характеристики, особенности исполнения, техническое состояние, запрашиваемая цена. База покупателей: контактные координаты, требования к марке, техническим характеристикам и техническому состоянию, финансовые возможности. Автоматизация подбора вариантов для покупателя, формирование заявки для поставщиков и перегонщиков.

18. **Крылатые фразы.** Справочник пословиц, поговорок, афоризмов, каламбуров, других словесных курьезов. Классификация по авторам и источникам, поиск по темам и ключевым словам.

19. **Справочник астронома.** Для каждой из зарегистрированных звезд известны: название, созвездие, видимая звездная величина, расстояние, координаты на небосклоне. Поиск звезд заданного созвездия, самых ярких звезд, видимых звезд и созвездий в заданной точке земного шара в заданное время.

20. **Ломбард.** База хранимых товаров и недвижимости: анкетные данные клиента; наименование товара; оценочная стоимость; сумма, выданная под залог; дата сдачи; срок хранения. Операции приема товара, возврата, продажи по истечении срока хранения.

Часть 2. Oracle

Лабораторная работа №1
Подключение к Oracle через AquaDataStudio.
Создание и заполнение таблиц. Выборка данных

Подключение к Oracle через AquaDataStudio

1. Запустите AquaData через Пуск -> Программы -> Средства разработки -> AquaDataStudio.
2. При первом запуске программа должна спросить вас, как вы намерены ее использовать. Вы выбираете Usage -> Personal Use, в Personal License отмечаете галочку и печатаете YES, нажмите на OK. На следующем шаге мастера отметьте радиокнопку. Do not send и выберите Continue.
3. После этого программа спросит, надо ли регистрировать сервер БД. Вы соглашаетесь.
4. На вкладке RDBMS выберите Oracle 9i/10g.
5. Name напишете study.
6. Type выберите test.
7. Login name - ваш логин на Oracle (он совпадает с вашим логином в сети).
8. Password - ваш пароль на Oracle.
9. Connect as оставьте Normal.
10. Host укажите callisto.uni-smr.ac.ru (или 195.209.66.1).
11. Port задайте 1521.
12. SID задайте ssu9.
13. Folder выберите диск W (а лучше, если вы создадите на нем папку для лабораторных и пропишете путь к ней).
14. Нажмите OK - сервер зарегистрирован.
15. После этого в программе появится сервер study, если вы раскроете плюсик рядом с ним, то увидите список схем. Раскройте его и найдите свой логин - это ваша схема, в которой вы и будете работать. Раскройте плюсик рядом с ней. Вы сможете увидеть, какие объекты БД могут храниться на сервере. Пока что они все пусты. Для создания объектов и написания запросов нам понадобится в дальнейшем окно SQL-редактора. Оно запускается комбинацией клавиш Ctrl + Q. В нем можно писать запросы и запускать их на выполнение командой Ctrl + E.

Создание таблиц можно проводить в окне редактора SQL-запросов, а также можно запустить мастер создания таблиц через контекстное меню по объектам «Таблицы» пункт Create Table.

Создавать таблицы можно и в SQL*PLUS, который можно запустить через Пуск -> Программы -> Средства разработки -> Oracle-OraClient 10g_homel -> Application Development -> SQL*Plus. Остановимся пока подробнее на проектировании базы и командах SQL для создания и заполнения таблиц, а потом обсудим соответствующие возможности Aqua Data Studio.

Примечание. Настройка подключения к СУБД Oracle из пакета SQL Developer во многом идентична приведенным выше настройкам для Aqua Data Studio. Основные возможности пакетов аналогичны, однако, на взгляд авторов, пакет Aqua Data Studio обладает большей гибкостью инструментов и большей прозрачностью выполняемых действий.

Будем создавать базу данных «Аренда недвижимости». В этой базе, очевидно, будет несколько таблиц, взаимосвязанных между собой. Нам необходимо хранить информацию об арендаторах и владельцах недвижимости, о самой недвижимости и цене на нее, а также о заключенных договорах. Знания терминологии для создания таблиц недостаточно. Дело в том, что среды разработки редко поддерживают русскую кодировку в обозначениях. Поэтому введем англоязычные обозначения таблиц базы данных «Аренда».

Таблица 9. Обозначения сущностей

Русскоязычные обозначения	**Англоязычные обозначения**
Арендатор (№Ар , Ар, АдрАр)	Tenant (NTn, Tn, AdT)
Владелец (№Вл, Вл, АдрЦл)	Owner (NOn, Ow, AdO)
Договор (№Дог, №Ар, №Ёл, АдрНд, Дата)	Lease (NLease, NTn, NOn, AdR, LDate)
Недвижимость (АдрНд, Тип)	Realty f AdR, Type)
Плата (Тип, Пл)	Rent (Type, Rn)

Теперь посмотрим, как можно реализовать таблицу Owner (NOn, Ow, AdO), в которой хранятся данные о владельцах собственности, с нуля.

Создание таблиц в Oracle производится командой **create table.** Например, чтобы создать таблицу «Владелец», структура которой приведена выше, необходимо выполнить следующие команды:

```
CREATE TABLE Owner
(NOn NUMBER (5, 0) PRIMARY KEY,
Ow VARCHAR2(25) NOT NULL,
AdO VARCHAR2(50) NOT NULL)
/
```

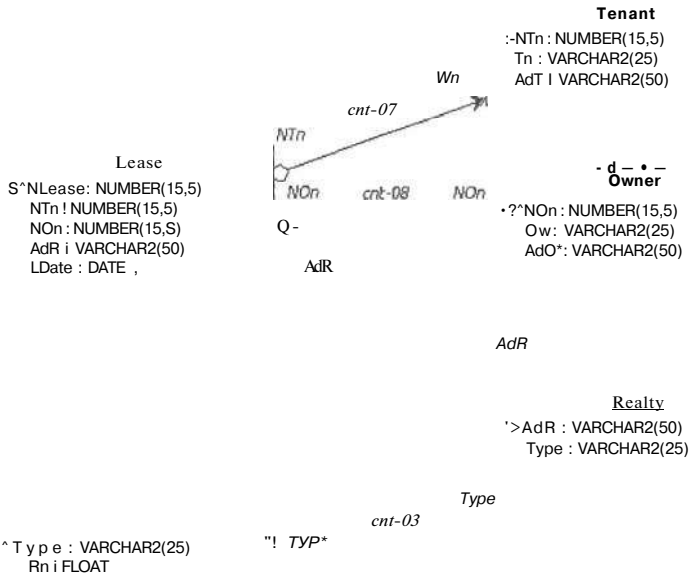

Рис. 2. Диаграмма базы данных «Аренда»

Примечание. В зависимости от версии Oracle, с которой вам приходится работать, в таблицу можно внести следующие типы данных:

NUMBER(p, s) Числовое значение, максимальное количество цифр равнор, а количество десятичных знаков - s

VARCHAR(s) Символьная строка переменной длины, максимальный размер равен s. Сейчас используют varchar2(s)

DATE Значение даты и времени между 1 января 4712 г. до н.э. и 31 декабря 4712 г. н. э.

CHAR(s) Символьное значение постоянной длины s

Чтобы заполнить таблицу данными, можно записать, например, вот такой код:

```
INSERT INTO OWNER
VALUES (1,'ИВАНОВ А.В.', 'КИРОВА, 1Г)

WERT INTO OWNER
VALUES (2,'ПЕТРОВА А.Н.'/САМАРСКАЯ, 17')
/
COMMIT
/
```

Примечание. Чтобы сохранить внесенные изменения, воспользуйтесь командой COMMIT.

Связи между таблицами устанавливаются уже при создании самих таблиц. Для этого необходимо указать внешний ключ и установить ссылку на родительскую таблицу. Например, когда будете создавать таблицу «Договор», не забудьте проставить связи:

```
CREATE TABLE Lease
(NLease NUMBER(5,0) PRIMARY KEY,
NQnNUMBER(5,0),
NTnNUMBER(5,0),
AdRVARCHAR2(25),
LDate DATE,
FOREIGN KEY (NOri) REFERENCES Owner,
FOREIGN KEY (NTn) REFERENCES Tenant,
FOREIGN KEY (AdR) REFERENCES Realty)
/
```

Aqua Data Studio предоставляет возможность создания базы без специальных знаний языка запросов. Создать таблицы можно, запустив мастера создания таблиц через контекстное меню по объектам «Таблицы», пункт Create Table. В мастере задаете имя таблицы, имена полей. Тип, длину (если предусмотрено типом), разрешено ли нулевое значение (для первичных ключей не разрешается) для поля - выбираете в диалоговом окне мастера.

Для проставления первичных ключей и связей надо в этом же диалоговом окне перейти на вкладку Constraints и в имени ограничения написать какой-либо уникальный для всей базы идентификатор (если он не

сгенерирован системно), в типе - выбрать Primary key. В нижней табличке Constraint Definition выбираете имя поля, которое должно быть первичным ключом.

Связи проставляются на той же вкладке. Назовите ограничение уникальным для базы именем (на рис. 2 ограничения названы cnt-07, cnt-08 и т. п.), тип - Foreign key (внешний ключ), Ref Schema - выберите свой логин, Ref Table - на какую таблицу должен ссылаться. В нижней таблице Constraint Definition выберите имена полей, по которым идет связь. На вкладке Preview SQL можно посмотреть, как будет выглядеть SQL-запрос.

Чтобы построить диаграмму вашей базы данных, воспользуйтесь командой ER Diagram Generator пункта меню Tools. В окне мастера надо выбрать сервер study, подключение Connect as default, на следующем шаге работы мастера необходимо выбрать свою рабочую схему и отметить галочкой пункт Tables. После чего в окне выбора отображаемых объектов появится список созданных вами таблиц, отмеченных галочками. Нажмите OK, и мастер сгенерирует вам диаграмму базы, аналогичную той, что приведена на рис. 2. Для заполнения таблицы достаточно вызвать контекстное меню для конкретной таблицы и выбрать там команду Edit Table Data.

После того как вы создадите и заполните таблицы, потренируйтесь и вспомните простые команды языка и выполните следующие задания.

Задания

1. Вывести информацию об арендаторах и владельцах 3-комнатных квартир. Для столбцов создать псевдонимы. Упорядочить в обратном алфавитном порядке по фамилиям арендаторов.
2. Вывести информацию об адресах домов (имеется в виду конкретный тип недвижимости) и дате начала их аренды. Упорядочить по дате, начиная с самой поздней аренды.
3. Вывести информацию о фамилиях и адресах владельцев, которые сдают 1-комнатные квартиры.
4. Вывести информацию о владельцах, их адреса, арендаторов, их адреса, если они заключили договор ранее 01.01.05 и позже 01.01.04. Упорядочить информацию по алфавиту по адресам владельцев, создать псевдонимы столбцов.
5. Вывести информацию о владельцах, их адреса, арендаторов, их адреса, если они заключили договор на 2-комнатную квартиру ранее 01.01.05 и позже 01.01.04. Упорядочить информацию по алфавиту по адресам владельцев, создать псевдонимы столбцов.

Лабораторная работа №2
Однострочные и групповые функции

Однострочные функции

В Oracle предусмотрен ряд встроенных функций (так называемые однострочные функции), при помощи которых можно управлять отображением данных и их преобразованием. Однострочные функции в Oracle можно подразделить на:

- символьные;
- числовые;
- функции даты;
- функции преобразования.

Синтаксис:

FUNCTION_NAME(COLUMN|EXPRESSION,[ARG1,ARG2,...])

Таблица 10. Символьные функции

Функция	Назначение
ЬО^¥ЕЩстолбец|выражение)	Преобразование алфавитных символов в нижний регистр
ЦРРЕЩстолбец|выражение)	Преобразование алфавитных символов в верхний регистр
1№ТСАР(столбец[выражение)	Преобразование начальных символов в верхний регистр, остальные преобразуются в нижний регистр
СО>1САТ(столбец 1|выражение 1, столбец2|выражение2)	Конкатенация первого символьного значения со вторым. Эквивалентно оператору конкатенации (||)
8ЦВ8Т11(столбец(выражение, m[,n])	Возвращает п символов из символьного значения, начиная с позиции т. Если число т отрицательно, то отсчет начинается от конца символьного значения
ЬЕМСТН(столбец|выражение)	Возвращает количество символов в значении
МУЦстолбец|выражение 1, столбец|выражение2)	Возвращает второе значение, если первое является неопределенным

Таблица 11. Числовые функции

Функция	Назначение
ЯОиЖ)(столбец|выражение, п)	Округляет столбец, выражение или значение до п десятичных знаков. Если п опущено, то до целого. Если п отрицательно, округляется целая часть числа
ТКиНС(столбец|выражение, п)	Усекает столбец, выражение или значение до п десятичных „„знаков. Если п опущено, то до целого. Если п отрицательно, обнуляются разряды целой части числа
MOD(m,n)	Возвращает остаток от деления га на п

Даты в системе Oracle хранятся во внутреннем числовом формате, где представлены столетие, год, месяц, день, часы, минуты, секунды. SYSDATE - функция даты, возвращает текущие дату и время. Обычно выборка SYSDATE производится из фиктивной таблицы DUAL.

Таблица 12. Функции даты

Функция	**Назначение**
MONTHSJBETWEENOiaTal, дата2)	Определяет количество месяцев между датами 1 и 2
ADD_MONTHSOa$_т$a,n)	К дате прибавляет п календарных месяцев. N может быть отрицательным, но должно быть целым
NEXT_DAY(flara, 'символ')	Определение даты ближайшего дня недели, заданного «символом» после указанной даты. Символ может задавать порядковый номер, название дня недели
LAST_DAY(flaTa)	Определение последнего дня месяца, содержащего заданную дату
ROUND0iaTa[,'fmt'])	При отсутствии аргумента 'fmt' округляет до даты на момент полуночи (до целого числа суток)
TRUNCC4aTa[,'fint'])	Если модель 'fmt' не задана, то возвращает первый день месяца, указанного в аргументе «дата». Если fmt = YEAR, то возвращает дату первого дня года, содержащего указанную дату

Арифметические операции с датами

1. Результатом прибавления числа к дате и вычитания числа из даты является дата.
2. Результатом вычитания одной даты из другой будет количество дней, разделяющее эти даты.
3. Прибавление часов к дате осуществляется путем деления количества часов на 24.

Таблица 13. Функции преобразования

Функция	**Назначение**
TO_CHAK(число|дата,['ппГ])	Преобразует число или дату в строку с заданной моделью формата
TO_NUMBER(CHMB)	Преобразует строку, содержащую цифры, в число
TO_DATE(cHMB,['fmt'])	Преобразует стоку символов с датой в дату с заданным форматом

Таблица 14. Функция TO_CHAR с датами

Элемент	**Описание**
YYY или YY или Y	Последние 3, 2 или 1 цифра года
Y,YYY	Год с запятой в указанной позиции
IYYY, IYY, IY, I	4,3,2 или 1 цифра года в соответствии со стандартом ISO
SYEAR или YEAR	Год словами. S означает, что даты до н.э. получают префикс «-»
BC или AD	Индикатор «до н.э./н.э.»
B.C. или A.D.	Индикатор «до н.э./н.э.» с точками
Q	Квартал
MM	Месяц в виде двузначного числа
MONTH	Название месяца, дополненное конечными пробелами до 9 символов
MON	Трехбуквенное сокращенное название месяца
RM	Номер месяца римскими цифрами
WW или W	Неделя года или месяца
DDD или DD или D	День года, месяца или недели
DAY	Название дня, дополненное конечными пробелами до 9 символов
DY	Трехбуквенное название дня

Таблица 15. Форматы времени

Элемент	**Описание**
AM или PM	Индикатор «до полудня / после полудня»
A.M. или P.M.	Индикатор «до полудня / после полудня» с точками
HH или HH12 или HH24	Время суток, час в 12-часовом или в 24-часовом диапазоне
MI	Минуты (0-59)
SS	Секунды (0-59)
SSSSS	Количество секунд после полуночи (0-86399)

Таблица 16. Другие форматы

Элемент	**Описание**
A,	Знаки пунктуации включаются в результат
«из множества»	Строка, заключенная в кавычки, включается в результат

Таблица 17. Суффиксы, влияющие на формат вывода чисел

Суффикс	**Описание**
TH	Порядковый номер (например, DDTH для вывода в формате «4th»)
SP	Вывод числа словами
SPTH или **THSP**	Вывод порядкового числительного словами

Примечание. Это еще далеко не полный набор однострочных функций. Например, постарайтесь самостоятельно найти, чем отличается формат даты RR от формата YY. Также на самостоятельное изучение остается функция TOCHAR с числами.

Групповые функции

В отличие от однострочных функций групповые функции работают над множествами строк и выдают один результат на группу. Групповые функции используются и в списке SELECT, и в предложении HAVING.

Групповые функции:

- AVG;
- COUNT;
- MAX;
- MIN;
- STDDEV;
- SUM;
- VARIANCE.

Предложения GROUP BY и HAVING в команде SELECT

По умолчанию все строки таблицы рассматриваются как одна группа. Для разбиения таблицы на меньшие группы строк используется предложение GROUP BY команды SELECT. Кроме того, для отбора возвращаемых групп используется предложение HAVING.

Синтаксис запроса, содержащего групповую функцию:

```
SELECT    столбец, групповая функция
FROM      таблица
[WHERE    условие]
[GROUP    BY выражение_ группирования]
[HAVING   условие_группы]
[ORDER    BY столбец]
I
```

Здесь: *выражение группирования* задает столбец, на основе значения которого группируются строки;

условие группы включает в выходной результат только те группы, для которых заданное условие будет истинно (TRUE).

Каждая функция допускает один аргумент. В следующей таблице показаны возможные варианты синтаксиса.

Таблица 18. Варианты синтаксиса групповых функций

Функция	Описание
AVG(DISTINCT|ALL|n)	Среднее значение п без учета неопределенных значений
COUNT(DISTINCT|ALL| *выражение]*)*	Количество строк только с определенными результатами вычисления *выражения*. По "*" подсчитываются все строки, включая повторяющиеся и строки с неопределенными значениями
MAX(DISTINCT|ALL| *выражение)*	Максимальное значения *выражения*
MIN(DISTINCT|ALL| *выражение)*	Минимальное значения *выражения*
STDDEV(DISTINCT|ALL|n)	Стандартное отклонение п без учета неопределенных значений
SUM(DISTINCT|ALL|n)	Сумма значений п без учета неопределенных значений
VARIANCE(DISTINCT|ALL|n)	Дисперсия п без учета неопределенных значений

Указания

- Если задано слово DISTINCT, функция учитывает лишь неповторяющиеся значения; при наличии слова ALL рассматриваются все значения. Вариант ALL принимается по умолчанию, поэтому задавать его нет необходимости.
- Если задано выражение, допустимыми типами данных для аргументов являются CHAR, VARCHAR2, N U M B E R . H DATE.
- Все групповые функции, кроме COUNT(*), игнорируют неопределенные значения. Для подстановки значения вместо неопределенного используется функция NVL.

Пример

Вывод средней, самой высокой и самой низкой арендной платы.

```
SELECT AVG (Rn) as "AVG", MAX (Rn) as "MAX",
MIN (Rn) as "MIN"
FROM Rent
/
```

Примечание. Функции AVG и SUM применяются к столбцам с числовыми данными, функции MAX и MIN применяются к данным любого типа.

Функция COUNT имеет два варианта формата: COUNT(*) и СОиЪЩвыражение). Функция COUNT(*) возвращает количество всех строк в таблице, включая повторяющиеся строки и строки с неопределенными значениями.

Пример

Вывод количества всех договоров.

```
SELECT COUNT (*) FROM Lease
/
```

В отличие от COUNT(*) функция СОТЖГ(выражение) подсчитывает только строки с определенными значениями в столбце, описанном выражением.

Предложение GROUP BY

Предложение GROUP BY используется для разбиения строк таблицы на группы. Затем для получения сводной информации по каждой группе можно применять групповые функции.

Указания

- Если в предложение SELECT включена групповая функция, получить одновременно отдельные результаты можно только в случае, если в предложении GROUP BY задан отдельный столбец. Если список столбцов отсутствует, выдается сообщение об ошибке.
- Предложение WHERE позволяет исключить некоторые строки до начала разбиения на группы.
- В предложении GROUP BY должны быть заданы столбцы.
- В предложении GROUP BY нельзя использовать позиционные обозначения или псевдонимы столбцов.
- По умолчанию строки сортируются в порядке возрастания в соответствии со списком GROUP BY. Изменить порядок сортировки можно с помощью предложения ORDER BY.
- Сводные результаты по группам и подгруппам можно получить путем указания более чем одного столбца в предложении GROUP BY.

Пример

Вывод суммарной годовой стоимости аренды объектов Ивановым.

```
SELECT "Tenanf'.'Tn", (SUM (12* "Rent"."Rn")) AS
ИТОГ
FROM "Realty", "Lease", "Tenant", "Rent"
WHERE  "Lease'V'AdR" = "Realty'V'AdR"
AND  "Lease'V'NTn" = "Realty"."NTn"
AND  "Realty"."Type" = "Renf'."Type"
AND "Tenanf'.'Tn" LIKE '%Иванов%'
GROUP BY "Tenanf'.'Tn"
/
```

Примечание. Обрамление имен таблиц и полей двойными кавычками не обязательно и может потребоваться только при работе в пакете Aqua Data Studio. При обращении к базе из командной строки либо из пакета SQL Developer двойные кавычки чаще всего не требуются.

Предложение HAVING

Предложение HAVING задает условие отбора групп для вывода. Следовательно, на группы накладывается дальнейшее ограничение, основанное на сводной информации.

Если используется предложение HAVING, сервер Oracle выполняет следующие действия:

1) группирует строки;
2) применяет групповую функцию;
3) производит вывод групп, удовлетворяющих условию предложения HAVING.

Предложение HAVING может предшествовать предложению GROUP BY, но более логично ставить предложение GROUP BY первым. Образование групп и вычисление групповых функций происходит до того, как к группам из списка SELECT применяется ограничение, заданное в предложении HAVING.

Пример

Определить, средняя стоимость каких арендуемых объектов превышает 5000 руб.

```
SELECT  Realty.AdR, Realty.Type
FROM Realty, Rent,
WHERE  (Rent.Type=Realty.Type)
GROUP BY Realty.AdR, Realty.Type
HAVING  AVG(Rent.Rn)>5000
/
```

Задания

Часть 1. Однострочные функции

1. Выведите фамилии владельцев недвижимости с заглавной буквы, а типы недвижимости - через запятую и заглавными буквами.
2. Установите сначала значение арендной платы за аренду гаража неопределенным, а потом сделайте ее равной 3000 руб.
3. Определите дату окончания аренды для всех арендаторов, если дом сдается на год, квартира - на полгода, гараж - на квартал. Выведите дату в формате «27 of October, 2004». Упорядочьте по дате, начиная с самой ранней.
4. Определите дату следующего платежа для каждого арендатора, если платежи производятся ежемесячно по вторникам. Выведите дату в формате «27/X/04». Упорядочьте по датам, начиная с самой ранней.
5. Определите, какое время прошло от момента аренды. Выведите адрес недвижимости, фамилию арендатора, дату начала аренды в формате «27th Oct 004» и количество месяцев, прошедшее с момента начала аренды, а также количество кварталов, прошедшее с момента начала аренды (значение округлите - подумайте, чем здесь лучше воспользоваться - round или trunk?). Для столбцов создайте псевдонимы, данные упорядочьте по количеству кварталов по возрастанию.
6. Придумайте самостоятельно (и реализуйте) еще 2 запроса, позволяющие продемонстрировать возможности однострочных функций.

Часть 2. Групповые функции

1. Определите количество договоров, заключенных до 01.01.2004 г. Результат выведите с логичной значимой подписью.
2. Определите количество владельцев недвижимости, со дня заключения договора с которыми прошло более 2 лет. Результат выведите с логичной значимой подписью.

3. Получите для каждого арендатора суммарную квартальную стоимость аренды, для каждого владельца - суммарную выручку за квартал.
4. Придумайте и реализуйте собственный запрос на групповые функции.

Требования к сдаче лабораторной работы

1. Выполнить запросы с использованием однострочных функций. Каждый запрос должен быть сохранен в отдельный файл.
2. Выполнить запросы на групповые функции. Каждый запрос должен быть сохранен в отдельный файл.
3. Придумать и соответствующим образом оформить ваши собственные запросы. Естественно, они должны быть выполнены, сохранены в отдельные файлы и не должны совпадать с запросами ваших одногруппников.

Лабораторная работа №3
Подзапросы

Общие сведения

Подзапрос - это команда SELECT, вложенная в предложение другой команды SQL. С помощью подзапросов можно создавать из простых команд гораздо более мощные. Это может быть удобно для выборки строк таблицы по условию, зависящему от данных в самой таблице.

Подзапросы можно использовать в разных предложениях команд SQL:

- Предложение WHERE.
- Предложение HAVING.
- Предложение FROM команды SELECT или DELETE.

Подзапросы очень полезны при написании команд SELECT для выборки значений по неизвестному условному значению. С помощью подзапроса можно находить неизвестные значения.

Синтаксис:

```
SELECT   список_выбора
FROM     таблица
WHERE    выражение оператор
         (SELECT   список_выбора
         FROM      таблица)
I
```

Здесь: *оператор* оператор сравнения (например, >, = или IN).

Примечание. Операторы сравнения делятся на два класса: однострочные (>, = >=, <, <>, <=) и многострочные (IN, NOTIN).

Подзапросы иногда называют вложенной командой SELECT, подкомандой SELECT или внутренней командой SELECT. Как правило, подзапрос выполняется первым, а его результат используется для полного определения условия во внешнем запросе.

Указания

- Подзапрос должен быть заключен в скобки.
- Подзапрос должен находиться после оператора сравнения.
- В подзапросе нельзя использовать предложение ORDER BY. На каждую команду SELECT разрешается только одно предложение ORDER BY, и если оно используется, то должно быть последним в главной команде SELECT.

Команда SELECT может рассматриваться как блок запроса. Этот пример состоит из двух блоков запроса: главный запрос и внутренний запрос.

Однострочные подзапросы

Однострочный подзапрос возвращает из вложенной команды SELECT только одну строку. В подзапросах этого типа используется однострочный оператор сравнения или логический оператор: =, >, <, >= или <=.

Пример

Выборка номеров договоров, которые были заключены в тот же день, что и у владельца Иванова.

```
SELECT NLease
FROM Lease
WHERE LDate = (SELECT Lease.LDate
FROM Lease, Owner
WHERE (Owner.Ow = 'Иванов') AND
(Owner .NOn=Lease.NOn))
/
```

Первой выполняется команда SELECT вложенного блока запроса. Пусть результат запроса - Ol-Feb-05. Далее обрабатывается главный блок запроса. Результат подзапроса используется при этом для вычисления условия поиска. Фактически для сервера Oracle главный запрос будет выглядеть следующим образом:

```
SELECT NLease
FROM Lease
WHERE LDate = 'Ol-Feb-05'
/
```

Можно выводить данные из основного запроса, используя групповую функцию в подзапросе для возврата одной строки. Подзапрос заключается в скобки и помещается после оператора сравнения.

Пример

Вывод адресов объектов с арендной платой ниже среднего.

```
SELECT Realty.AdR
FROM Realty, Rent
WHERE (Realty.Type=Rent.Type)
AND (Rent.Rn<(SELECT AVG(Rent.Rn)
FROM Rent))
/
```

Многострочные подзапросы

Подзапросы, возвращающие более одной строки, называются многострочными. В них следует использовать не однострочный, а многострочный оператор - например, IN. Оператор IN ожидает одного или нескольких значений.

Предложение HAVING с подзапросами

Подзапросы можно использовать не только в предложении WHERE, но и в предложении HAVING. Сервер Oracle выполняет подзапрос первым и возвращает результаты в предложение HAVING главного запроса.

Задания

1. Определить, для каких типов недвижимости минимальная арендная плата не превышает 10000 руб. Посчитать количество арендаторов, готовых платить за недвижимость данных типов. Посчитать количество владельцев недвижимости, готовых предоставлять эту недвижимость в аренду. Посчитать количество договоров, заключенных по недвижимости данных типов.
2. Определить адреса недвижимости и номера договоров, заключенных с одним и тем же владельцем недвижимости. Определить адреса недвижимости и номера договоров, заключенных с одним и тем же арендатором недвижимости.
3. Вывести тип и адрес недвижимости, для которых было заключено 2 и более договоров аренды.
4. Подсчитать количество владельцев недвижимости, сумма выручки которых превысила среднюю арендную стоимость по всем заключенным договорам.
5. Придумать и реализовать 2 собственных запроса, в которых необходимо использовать подзапрос (или подзапросы).

Лабораторная работа №4
Определение переменных во время выполнения

Для ограничения количества выходных строк можно создать командный файл с предложением WHERE. Для изменения условия при каждом выполнении командного файла используются так называемые переменные подстановки. Значения переменных подстановки могут заменять значения в предложении WHERE, текстовую строку и даже имя столбца таблицы.

Отчеты, которые вы получали до сих пор, не были интерактивными. В готовом приложении пользователь запрашивал отчет, который создавался без каких-либо приглашений ввести данные. Объем данных в отчете был предопределен постоянным предложением WHERE. Но SQL*Plus позволяет создавать и интерактивные отчеты, когда пользователя приглашают ввести значения, ограничивающие объем выходных данных.

Для создания отчета в командный файл или в отдельные команды SQL включаются переменные подстановки. Переменная выступает в роли контейнера, в котором временно хранятся значения.

В SQL*Plus можно использовать переменную подстановки с одним амперсандом для временного хранения значений. SQL*Plus позволяет также определять переменные заранее с помощью команд ACCEPT и DEFINE. Команда ACCEPT считывает строку данных, введенную пользователем, и сохраняет ее в переменной. Команда DEFINE создает переменную и присваивает ей значение.

Примеры ограничения интервалов данных

- Отчет за текущий квартал или период времени.
- Отчет, содержащий данные только по пользователю, который его запросил.
- Получение списка служащих только одного отдела.

Другие интерактивные возможности

- Динамическое изменение верхних и нижних колонтитулов страниц.
- Получение входных параметров из файла, а не от оператора.
- Передача значений между командами SQL.

Переменные подстановки с одним амперсандом

При создании отчетов пользователь часто заинтересован в динамическом ограничении объема выходных данных. SQL*Plus обеспечивает такую гибкость через переменные пользователя. Задать переменные в своих командах SQL можно с помощью амперсанда (&). Присваивать значение каждой переменной при этом не требуется.

Таблица 19. Переменная подстановки с одним амперсандом

Пример	Описание
&переменная	Переменная в команде SQL; если переменная не существует, SQL*Plus просит пользователя ввести значение. SQL*Plus не сохраняет переменную после ее использования

Пример

```
SELECT AdR, LDate FROM lease
WHERE NLease = &n_lease;
```

Текстовые строки и даты в предложении WHERE должны быть заключены в апострофы. Это же правило распространяется и на переменные подстановки. Чтобы не вводить апострофы во время выполнения команды SQL, рекомендуется заключать в них переменную в самой команде SQL.

Пример

```
SELECT AdT, NTn FROM tenant
WHERE Tn = '&tn_name';
```

Подстановка переменных может использоваться не только в предложении WHERE, но и для подстановки имен столбцов, выражений и текстовых строк.

Пример

```
SELECT NLease, &column_name FROM lease
WHERE &column = &condition;
```

Определение переменных пользователя

Задать переменные можно до выполнения команды SELECT. Для определения и установки значений переменных SQL*Plus предлагает две команды - DEFINE и ACCEPT.

Таблица 20. Команды определения переменных пользователя

Команда	Описание
DEFINE переменная = значение	Создание переменной типа CFIAR и присвоение ей значения
DEFINE переменная	Вывод переменной, ее значения и типа данных
DEFINE	Вывод всех переменных пользователя, их значений и типа данных
ACCEPT	Чтение строки, введенной пользователем, и сохранение ее в переменной

Сокращенный синтаксис:

ACCEPT *переменная [тип_данных]* [FORMAT] [PROMPT *текст]* [HIDE]

Здесь:

переменная — имя переменной, хранящей значение. Если переменная не существует, SQL* Plus ее создаст;

тип_данных — NUMBER, CHAR или DATE. Максимальная длина для типа CHAR - 240 байтов. DATE сверяется с моделью формата;

FORMAT — модель формата, например, A10 или 9.999;

PROMPT *текст* — текст, который выдается прежде, чем пользователь может ввести значение;

HIDE — предотвращает отображение данных, введенных пользователем.

Примечание. Если для определения переменной используется команда ACCEPT, амперсанд перед параметром подстановки не ставится.

Указания

- Команды ACCEPT и DEFINE создают переменную, если она не существует, а существующую переменную переопределяют.
- Пользуясь командой DEFINE, не забывайте заключать в апострофы строки, содержащие пробелы.
- Команда ACCEPT используется в следующих целях:
  - o для создания приглашения к вводу данных с учетом пожеланий пользователя. В противном случае выдается стандартное приглашение, используемое по умолчанию: «Enter value for variable»;
  - o для явного определения переменных типа NUMBER и DATE;
  - o для ввода данных без их отображения в целях секретности.

Пример

```
ACCEPT TypeName PROMPT 'Задайте тип
недвижимости:'

SELECT d.Rn, dl.AdR, d2.LDate
FROM rent d, realty d1, lease d2
WHERE d.type = dl.type
AND dl.AdR = d2.AdR
AND d.Type = '&Type_Name'
/
```

Команда UNDEFINE

Переменная сохраняет свое значение до:

- очистки командой UNDEFINE;
- выхода из SQL*Plus.

Отменив определение переменной, можно проверить произведенное изменение с помощью команды DEFINE. При выходе из SQL*Plus переменные, заданные во время сеанса, теряются. Для сохранения переменных от сеанса к сеансу измените свой файл login.sql так, чтобы эти переменные инициализировались сразу после загрузки системы.

Передача значений переменных в командный файл

Параметр - это значение, которое можно передать в отчет прямо из командной строки. Для создания получения отчета с учетом параметров выполните следующие шаги.

1. Создайте командный файл с командой SELECT.
2. В команде SELECT для каждой переменной используйте ссылку типа «&номер».
3. В командной строке задайте значение переменной прямо за именем командного файла. Значение переменной должно отделяться от имени файла пробелом.

Пример

```
SELECT NLease, AdR, LDate
FROM Lease
WHERE NTn = &l
/
```

Указания

- Присвоить параметрам значимые имена можно с помощью команды DEFINE.
- Используйте префикс для различных названий столбцов, обычных переменных и переменных с параметрами.
- Последовательность значений параметров в командной строке очень важна. Первое значение соответствует переменной &1, второе - переменной &2 и т. д.
- Запрос может содержать до девяти параметров с именами от &1 до &9.
- SQL*Plus сохраняет параметры и их значения до следующего переопределения или выхода из SQL*Plus.

Задания

1. Создайте командный файл для выборки информации о недвижимости, дата начала аренды которой заключена в пределах определенного диапазона. Выходные данные должны включать фамилии владельцев и арендаторов, адрес недвижимости, ее тип и арендную плату. Даты, определяющие временной диапазон, запрашиваются у пользователя. Используйте формат MM/DD/YY.
2. Создайте командный файл для получения списка имен, адресов проживания и номеров арендаторов определенного типа недвижимости. Тип недвижимости задается с клавиатуры. Поиск должен осуществляться независимо от регистра символов.
3. То же, что и в задании 2, но только для владельцев недвижимости.
4. Создайте отчет, содержащий фамилии владельцев недвижимости, фамилии всех арендаторов, заключивших договор с данным вла-

дельцем, выручку владельцев с каждого своего арендатора. Арендная плата начисляется ежемесячно с момента заключения договора.

5. Придумайте и реализуйте еще 2 запроса с использованием переменных подстановки.

Требования к сдаче лабораторной работы

1. Каждое задание должно быть сохранено в отдельный командный файл.
2. Ваши 2 запроса должны быть оригинальными (т. е. не совпадать с запросами из книги, заданиями и запросами ваших одногруппников).

Лабораторная работа MS
Представления

Представление - это логическая таблица, созданная на основе реальной таблицы или другого представления. Представление не содержит собственных данных, а скорее является «окном», через которое можно просматривать или изменять данные из таблиц. Представление хранится в словаре данных как команда SELECT.

Преимущества представлений:

- Ограничивается доступ к базе данных вследствие того, что представления могут отображать определенное подмножество базы данных;
- С помощью простых запросов пользователь может получить такие же результаты, как с помощью сложных. Например, использование представлений позволяет осуществить выборку данных из нескольких таблиц без каких-либо знаний об операторе соединения таблиц (join);
- Обеспечивается независимость данных для пользователей, посылающих случайные незапрограммированные запросы, и прикладных программ. Одно и то же представление может использоваться для выборки данных из нескольких таблиц;
- Использование представлений позволяет обеспечить доступ к данным по различным критериям для различных групп пользователей.

Создание представления

Представление создается путем включения подзапроса в команду CREATE VIEW.

Сокращенный синтаксис:

CREATE [OR REPLACE] [FORCE | NOFORCE] VIEW
представление [(псевдоним [, *псевдоним]...)]*
AS *подзапрос*
[WITH CHECK OPTION [CONSTRAINT *ограничение]]*
[WITH READ ONLY]

Здесь: FORCE — создание представления независимо от того, существуют ли базовые таблицы;

NOFORCE	создание представления только при условии существования базовых таблиц. Значение принято по умолчанию;
представление	имя представления;
псевдоним	имена выражений, выбранных в запросе для представления. Количество псевдонимов должно быть равным количеству выражений, выбранных представлением;
подзапрос	полная команда SELECT. Для столбцов в списке SELECT можно использовать псевдонимы;
WITH CHECK OPTION	режим, при котором добавлять или обновлять можно только строки, доступные в представлении;
ограничение	имя, присвоенное ограничению CHECK OPTION;
WITH READ ONLY	запрет применения к данному представлению операций DML.

Указания

- Запрос, который определяет представление, может содержать команду SELECT со сложным синтаксисом, включая соединения, группы и подзапросы.
- Запрос, который определяет представление, не может содержать предложение ORDER BY.
- Если вы не указываете имя ограничения сами, система присвоит его по умолчанию в формате SYS_Cn.
- Для изменения определения представления без его удаления и создания заново, а также для изменения предоставленных привилегий можно использовать режим OR REPLACE.

Существуют два вида представлений: простые и сложные. Основное отличие связано с операциями DML.

Таблица 21. Сравнительные характеристики простых и сложных представлений

Характеристика	Простые представления	Сложные представления
Количество таблиц	Только одна	Одна или более
Содержит функции	Нет	Да
Содержит группы данных (DISTINCT или групповые функции)	Нет	Да
Операции DML над представлением	Да	Нет

Пример

```
CREATE VIEW leasevw
AS SELECT NLease, AdR, LDate FROM lease
/
SELECT 2 FROM lease_vw
I
```

Указать имена столбцов можно путем включения псевдонимов столбцов в подзапрос. Еще один способ указания имен столбцов - это включить псевдонимы столбцов в предложение CREATE VIEW. Для изменения определения представления используйте предложение CREATE OR REPLACE.

Пример

```
CREATE VIEW leasejvw
AS SELECT NLease Number, AdR Address, LDate
Date_Dogovor FROM lease;
```

Пример

```
CREATE OR REPLACE VIEW lease_vw
(NLease, AdR, LDate)
AS SELECT NLease, Non, NTn FROM lease
/
```

Для вывода данных из двух и более таблиц используется сложное представление.

Пример

```
CREATE VIEW renta_vw
(RentJType, Rent_Address, Tn_Name, Min_Rn)
AS SELECT realty.Type, leaseAdR, tenant.Tn, rent.Rn
FROM realty, lease, tenant, rent
WHERE lease.NTn = tenant.NTn
AND lease Ad R = realty Ad R
AND realty.Type = rent.Type
/
```

Выполнение операции DML над представлением

Операции DML могут выполняться над представлениями в соответствии со следующими правилами.

- Удаление строки из представления возможно, если представление не содержит ничего из следующего:
  - условие соединения;
  - групповые функции;
  - предложение GROUP BY;
  - команда DISTINCT.
- Данные в представлении могут быть изменены, если представление не содержит ничего из вышеперечисленного и ничего из следующего списка:
  - столбцы, описанные как выражения, например, Rn* 1.5;
  - псевдостолбец ROWNUM.
- Добавление данных через представление возможно, если оно не содержит ничего из вышеперечисленного и если в базовой таблице нет столбцов типа NOT NULL, не включенных в представление. Все необходимые значения должны присутствовать в представлении. Помните, что через представление вы добавляете значения прямо в основную таблицу.

Можно сделать так, чтобы при добавлении или обновлении данных через простое представление можно было запрашивать добавленные или обновленные данные через представление. Можно сделать так, чтобы выполнение операции DML через представление было невозможно. Для этого представление создается с режимом WITH READ ONLY.

Пример

```
CREATE OR REPLACE VIEW lease_vw
AS SELECT *FROM Lease
WHERE Non = 2
WITH CHECK OPTION CONSTRAINT lease_vw_ck
/
UPDATE  lease_vw
SET NOn= 10000
WHERE NLease = 3
/
```

Пример

```
CREATE OR REPLACE VIEW lease_vw
(NLease, AdR, LDate)
AS SELECT NLease, Non, NTn FROM Lease
WHERE Non = 2
WITH READ ONLY
/
DELETE FROM lease_vw
WHERE NLease = 3
/
```

Вывод имен и структур представления

Увидеть имена представлений и предложения SELECT, определяющие представления, можно в таблице словаря данных USER_VIEWS.

Пример

```
DESC  userjviews;

SELECT * FROM user_views
/
```

Для удаления представления используйте команду DROP VIEW. Эта команда удаляет определение представления из базы данных. Удаление представления не изменяет таблиц, на основе которых оно было создано. Другие представления или приложения, созданные на базе удаленных представлений, становятся недействительными и не могут далее использоваться. Удалить представление может только тот, кто его создал, или пользователь с привилегией DROP ANY VIEW.

Синтаксис:

DROP VIEW *представление;*

Здесь: *представление* имя представления.

Задания

1. На основе таблицы owner создайте простое представление, в котором хранятся фамилии и адреса владельцев. Присвойте столбцам представления осмысленные названия. Выведите на экран содержимое представления. Напишите скрипт-файл для вывода на экран определения представления. Передайте скрипт-файлу имя представления. Сохраните этот скрипт-файл и выполните его. Измените при помощи представления адрес одного из владельцев недвижимости.
2. На основе таблиц tenant, owner, lease и rent создайте представление для вывода полных данных **о** каждом объекте недвижимости: тип, адрес, стоимость, дата договора, фамилии владельцев и арендаторов. Выведите структуру и содержимое представления, а также определение представления. Получите при помощи представления фамилии всех владельцев определенного типа недвижимости. Тип недвижимости задается с клавиатуры. Получите при помощи представления все адреса определенного типа недвижимости, потом фамилии всех арендаторов определенного типа недвижимости.
3. Измените предыдущее представление так, чтобы оно содержало только данные по одному самому распространенному типу недвижимости. Добавьте ограничение, запрещающее изменять тип недвижимости. Выведите содержимое нового представления. Попробуйте изменить тип недвижимости.
4. Измените представление в задании 1 так, чтобы строки в нем можно было просматривать только между 11:00 и 17:00.

Требования к сдаче **лабораторной работы**

1. Каждое задание должно быть сохранено в отдельный командный файл.
2. После проверки преподавателем ваших заданий удалите все представления из словаря данных.

Лабораторная работа №6
Управление доступом пользователей

В многопользовательской среде необходимо заботиться о безопасности базы данных. Система безопасности сервера Oracle позволяет:

- управлять доступом к базе данных;
- предоставлять доступ к определенным объектам базы данных;
- просматривать выданные и полученные привилегии в словаре данных Oracle;
- создавать синонимы для объектов базы данных.

Безопасность базы данных можно разделить на 2 части: безопасность системы и безопасность данных. Безопасность системы охватывает доступ к базе данных и пользование базой данных на системном уровне, т. е. имя и пароль пользователя, дисковое пространство, выделяемое пользователю, и системные операции, разрешенные пользователю. Безопасность данных включает доступ к объектам базы данных, их использование и действия, которые может выполнять пользователь с этими объектами.

Привилегии

Привилегии - это права на выполнение определенных команд SQL. Самый высокий уровень пользователя - это администратор базы данных. Он имеет право разрешать другим пользователям доступ к базе данных и ее объектам. Пользователям необходимы системные привилегии для получения доступа к базе данных вообще и привилегии на объекты дня манипулирования содержимым объектов в базе данных. Пользователям может быть также предоставлена привилегия на выдачу дополнительных привилегий другим пользователям или ролям, которые являются именованными группами связанных привилегий.

Схема

Схема - это совокупность таких объектов, как таблицы, представления и последовательности. Владельцем схемы является пользователь базы данных. Схема имеет такое же имя, как пользователь.

Создание **пользователя**

Администратор базы данных регистрирует нового пользователя сервера и предоставляет ему ряд системных привилегий. От этих привилегий зависит, что может делать данный пользователь на уровне базы данных. Для регистрации пользователя администратор базы данных использует команду CREATE USER. После выполнения просто этой команды пользователь системных привилегий не имеет.

Сокращенный синтаксис:

CREATE USER *пользователь* IDENTIFIED BY *пароль;*

Здесь: *пользователь* имя регистрируемого пользователя;

пароль пароль, с которым пользователь должен входить в систему.

Системные привилегии

Существует более 80 видов привилегий, предоставляемых пользователям и ролям. Системные привилегии обычно предоставляются администратором базы данных. Зарегистрировав пользователя, администратор базы данных может предоставить ему привилегии. Администратор базы данных предоставляет системные привилегии пользователям с помощью команды GRANT. Пользователь может пользоваться предоставленными привилегиями сразу после их получения.

Таблица 22. Обычные привилегии администратора базы данных

Системная привилегия	Разрешенные операции
CREATE USER	Позволяет регистрировать других пользователей Oracle (необходимая для роли администратора базы данных)
DROP USER	Позволяет удалять других пользователей
DROP ANY TABLE	Позволяет удалять таблицы из любой схемы
BACKUP ANY TABLE	Позволяет копировать любую таблицу в любой схеме с помощью утилиты экспорта

Таблица 23. Основные привилегии пользователя

Системная привилегия	Авторизованные операции
CREATE SESSION	Позволяет начать сеанс работы с базой данных
CREATE TABLE	Позволяет создавать таблицы в пользовательской схеме
CREATE SEQUENCE	Позволяет создавать последовательности в пользовательской схеме
CREATE VIEW	Позволяет создавать представления в пользовательской схеме
CREATE PROCEDURE	Позволяет создавать хранимые процедуры, функции или пакеты в пользовательской схеме

Сокращенный синтаксис:

GRANT *привилегия* [, *привилегия...]*
TO *пользователь* [, *пользователь...];*

Здесь: *привилегия* предоставляемая системная привилегия;

пользователь имя пользователя.

Роли

Роль - это именованная группа взаимосвязанных привилегий, которая может быть предоставлена пользователю. Этот метод упрощает процесс предоставления и отмены привилегий.

Пользователь может иметь доступ к нескольким ролям, а одна и та же роль может быть предоставлена многим пользователям. Обычно роли создаются для приложений базы данных.

Создание и присвоение роли:

CREATE ROLE *роль;*

Здесь: *роль* имя создаваемой роли.

Теперь администратор может назначить данной роли системные привилегии, а также назначить пользователям данную роль.

Пример

```
CREATE ROLE manager
/

GRANT create table, create view TO manager
/

GRANT manager TO las, sirvp, volena
/
```

Изменение своего пароля

Каждый пользователь имеет пароль, присвоенный ему администратором баз данных при регистрации. Изменить свой пароль можно при помощи команды ALTER USER.

Синтаксис:

ALTER USER *пользователь* IDENTIFIED BY *пароль;*

Здесь: *пользователь* имя пользователя;

пароль новый пароль.

Примечание. Кроме изменения своего пароля эта команда предоставляет и другие возможности, но для их использования необходима привилегия ALTER USER.

Привилегии на объекты

Таблица 24. Привилегии на объекты

Привилегии на объект	Таблица	Представление	Последовательность	Процедура	Снимок
ALTER	X		X		
DELETE	X	X			
EXECUTE				X	
INDEX	X				
INSERT	X	X			
REFERENCES	X				
SELECT	X	X	X		X
UPDATE	X	X			

Администратор баз данных может разрешать пользователям выполнять конкретные действия над определенными таблицами, представлениями, последовательностями, процедурами. Владелец объекта имеет на него все привилегии. Чтобы предоставить доступ к' своим объектам другим пользователям, необходимо использовать команду GRANT.

Синтаксис:

GRANT *{привилегия* (, *привилегия...) \ ALL}[(столбцы)]*
ON *объект*

TO *{пользователь* (, *пользователь....*) |*роль* | PUBLIC}
[WITH GRANT OPTION]

Здесь: *привилегия*	предоставляемая привилегия на объект;
ALL	все привилегии на объект;
столбцы	столбец таблицы или представления, на который даются привилегии;
ON *объект*	объект, на который предоставляются привилегии;
TO	кому предоставляются привилегии;
PUBLIC	привилегии на объект предоставляются всем пользователям;
WITH GRANT OPTION	позволяет пользователю, получившему привилегии на объект, передавать их другим пользователям и ролям.

Примечание. Что касается процедур, то речь идет об одиночных процедурах, функциях и конструкциях в общедоступных пакетах. Привилегии INDEX и REFERENCES ролям не предоставляются.

Указания

- Чтобы предоставлять привилегии на объект, объект должен находиться в вашей схеме или вы сами должны были получить эти привилегии с правом передачи (WITH GRANT OPTION).
- Владелец объекта может предоставить любую привилегию на свой объект любому пользователю или роли.
- Владелец объекта автоматически получает все привилегии на свой объект.

Пример

```
GRANT select ON lease TO las, volena
/

GRANT UPDATE (NTn, Tn, AdT) ON tenant TO las,
manager
/
```

Предложение WITH GRANT OPTION

Пользователь, получивший привилегию с опцией WITH GRANT OPTION, может предоставлять эту привилегию другим пользователям. Привилегия на объект, предоставленная другому пользователю в режиме WITH GRANT OPTION, отменяется тогда, когда она отменяется для того, кто ее выдал.

Пример

```
GRANT select, insert ON lease TO las, volena
WITH GRANT OPTION
/
```

Ключевое **слово PUBLIC**

С помощью ключевого слова PUBLIC владелец таблицы может предоставить доступ к ней всем другим пользователям.

Пример

```
GRANT select ON lease TO public
/
```

Просмотр предоставленных привилегий, отмена привилегий

Если вы попытаетесь выполнить неразрешенную операцию, например, удалить строку из таблицы, на которую у вас нет привилегии DELETE, то сервер Oracle 7 не позволит вам это сделать.

Сообщение от сервера «таблица или представление не существует» может означать следующее:

- таблицы или представления с таким именем не существует;
- вы пытались обратиться к таблице или представлению, на которые у вас нет соответствующей привилегии.

Свои привилегии можно посмотреть в словаре данных.

Таблица 25. Таблицы словаря данных, хранящие сведения о привилегиях

Таблица словаря данных	Описание
ROLE_SYS_PRIVS	Системные привилегии, предоставленные ролям
ROLE_TAB_PRIVS	Привилегии на таблицы, предоставленные ролям
USER ROLE PRIVS	Роли, доступные пользователю
USERJTABPRIVSMADE	Привилегии, предоставленные пользователем на его объекты
USERTABPRIVSJRECD	Привилегии на чужие объекты, предоставленные пользователю
USER_COL_PRIVS_MADE	Привилегии, предоставленные пользователем на определенные столбцы его объектов
USER_COL_PRIVS_RECD	Привилегии, предоставленные пользователю на определенные столбцы чужих объектов

Отмена привилегий, предоставленных другим пользователям, производится при помощи команды REVOKE, отменяющей указанные вами привилегии для всех указанных пользователей, которым они были предоставлены.

Синтаксис:

REVOKE *{привилегия [, привилегия...]* | ALL}
ON *объект*
FROM *{пользователь [, пользователь...] \роль* | PUBLIC}
[CASCADE CONSTRAINTS]

Здесь: CASCADE CONSTRAINTS используется для отмены ограничений, наложенных на объект с помощью привилегии REFERENCES для сохранения ссылочной целостности.

Пример

REVOKE select ON lease FROM las, volena
/

Создание синонима объекта

Для ссылки на таблицу, принадлежащую другому пользователю, необходимо добавить к имени таблицы префикс в виде имени пользователя, создавшего таблицу, и точки. Создание синонима устраняет необходимость указания схемы в имени объекта и обеспечивает альтернативное имя для таблицы, представления, последовательности, процедуры или других объектов. Метод особенно полезен для длинных имен объектов - например, имен представлений.

Синтаксис:

CREATE [PUBLIC] SYNONYM *синоним* FOR *объект;*

Здесь:	PUBLIC	создает синоним, доступный всем пользователям;
	синоним	имя создаваемого синонима;
	объект	объект, для которого создается синоним.

Указания

- Объект не может быть частью пакета.
- Имя личного синонима должно отличаться от имен всех других объектов данного пользователя.

Пример

```
CREATE SYNONYM myjease FOR volena.lease
/

CREATE PUBLIC SYNONYM myjease
FOR volena. lease
/
```

Для удаления синонима используется команда DROP SYNONYM. Синоним PUBLIC может удалить только администратор базы данных.

Пример

```
DROP SYNONYM myjease
/
```

Вопросы

1. Какая привилегия требуется пользователю для входа на сервер? Системная это привилегия или привилегия на объект?
2. Что такое системная привилегия? Какие они бывают?
3. Какая привилегия требуется пользователю для создания, удаления таблиц?
4. Вы - администратор базы данных и регистрируете много пользователей, которым требуются одни и те же системные привилегии. Что вы можете сделать для облегчения своей работы?
5. Как изменить свой пароль?
6. Что значит опция WITH GRANT OPTION?
7. Кто может передавать привилегии на ваши объекты другим пользователям?

Задания

1. Предоставьте другим пользователям право доступа к своей таблице tenant (разумеется, без права DELETE, подумайте сами, какие привилегии стоит предоставлять, а какие нет). Выберите все строки из чужой таблицы tenant. Добавьте туда 2 новые записи. Сделайте внесенные изменения постоянными. Напишите запрос для чужой таблицы tenant и выполните его.
2. Создайте синоним чужой таблицы tenant. С его помощью выведите на экран данные из чужой таблицы.
3. Просмотрите привилегии на таблицы, предоставленные вам. Отмените привилегию SELECT, которую вы предоставили другим. После этого попробуйте выполнить команду SELECT над чужой таблицей. Отмените все привилегии, которые вы предоставили другим пользователям, удалите синоним.

Требования к сдаче лабораторной работы

1. Продемонстрировать задания 1-3.
2. Ответить на вопросы.

Лабораторная работа №7
Обработка данных. Управление транзакциями

Совокупность команд языка манипулирования данными (DML), результаты действия которых еще не стали постоянными, называется *транзакцией* или логической единицей работы.

Таблица 26. Команды обработки данных и управления транзакциями

Команда	Описание
INSERT	Добавляет новую строку в таблицу
UPDATE	Изменяет существующие строки таблицы
DELETE	Удаляет строки из таблицы
COMMIT	Делает все незафиксированные изменения постоянными
SAVEPOINT	Позволяет произвести откат до определенной точки сохранения
ROLLBACK	Отменяет все незафиксированные изменения данных

Вставка строк в таблицу

Добавить новую строку в таблицу можно при помощи команды INSERT.

Синтаксис:

INSERT INTO *таблица {(столбец^, столбец..^)]* VALUES *(значение* [, *значение*...]);

Здесь: *таблица* имя таблицы;

столбец имена столбцов таблицы, в которые вносятся значения;

значение соответствующее значение столбцов.

Примечание. Эта команда с предложением VALVES добавляет только одну строку.

Таблица 27. Способы вставки неопределенных значений

Метод	Описание
Неявный	Опустить столбец в списке столбцов
Явный	Задать ключевое слово NULL в списке VALUES
	Задать пустую строку в списке VALUES (только для символьных строк и дат)

Для вставки в таблицу специальных значений можно применять псевдостолбцы. Используйте USERID для ввода имени текущего пользователя и SYSDATE для ввода текущей даты и времени. Проверку успешной вставки значений можно осуществить при помощи команды SELECT.

Пример

```
INSERT INTO Rent (Type, Rn)
VALUES ('коттедж', NULL)
/

INSERT INTO owner (Non, Ow, AdO)
VALUES (10, USERID,
'Ставропольская, 35-43')
/

INSERT INTO lease (NLease, NTn, Non, AdR, LDate)
VALUES (5,4,7, 'Ставропольская IV, SYSDATE)
/
```

При вставке любого значения даты обычно используют формат DD-MON-YY. В этом формате стандартным значением столетия считается текущее столетие; времени - полночь. Если требуется задать другой век и конкретное время, применяется функция TO_DATE.

Пример

```
INSERT INTO lease (NLease, NTn, Non, AdR, LDate)
VALUES (6,4,2, 'Самарская Г,
TO_DATE ('01-JAN-967DD-MON-YY'))
/
```

Команда INSERT позволяет пользователю вводить значения в интерактивном режиме при помощи переменных подстановки SQL*Plus.

Пример

```
INSERT INTO owner (Non, Ow, AdO) VALUES
(«fcownernumber,  '&owner_name',  '&owner_address')
/
```

Примечание. Для дат и символьных значений амперсанд и имя переменной должны быть заключены в апострофы.

Свою команду с переменными подстановки можно сохранить в файле для последующего использования. При каждом выполнении такой команды будет предложено ввести новые значения переменных. Команда ACCEPT запоминает введенное значение в переменной. Команда PROMPT позволяет выводить на экран текст, удобный для пользователя.

Пример

```
ACCEPT ownerjunb PROMPT 'Enter owner number:'

ACCEPT ownerjname PROMPT 'Enter owner name:'

ACCEPT owner_adr PROMPT 'Enter owner address:'

INSERT INTO owner (Non, Ow, AdO) VALUES
(&owner_nmb,  '&owner_name',  '&owner_adr')
/
```

Команду INSERT можно использовать для вставки в таблицу новых строк, которые копируются из уже существующих таблиц. Для этого вместо VALUES в строке помещается подзапрос.

Обновление строк

Существующие строки таблицы можно обновить командой UPDATE.

Синтаксис:

UPDATE *таблица* SET *столбец = значение* [, *столбец = значение]* [WHERE *условие];*

Здесь: *таблица* имя таблицы;

, *столбец* имя обновляемого столбца таблицы;

значение соответствующее значение или подзапрос для этого столбца;

условие задает строки, которые необходимо изменить, и состоит из имен столбцов, выражений, констант, подзапросов и операторов сравнения.

Пример

```
UPDATE owner SET AdO = 'Самарская 111-112'
WHERE Non = 3
/
```

Если в команде UPDATE отсутствует предложение WHERE, изменяются все строки таблицы. Если при попытке обновления новое значение столбца противоречит ограничению, выдается сообщение об ошибке.

Удаление строк

Существующие строки удаляются командой DELETE.

Синтаксис:

DELETE [FROM] *таблица* [WHERE*условие};*

Здесь: *таблица* имя таблицы;

условие задает строки, которые необходимо удалить, и состоит из имен столбцов, выражений, констант, подзапросов и операторов сравнения.

Если WHERE отсутствует, то удаляются все строки таблицы. Если вы пытаетесь удалить запись, на значение которой имеется ограничение, выдается сообщение об ошибке. Так, при попытке удалить строку с первичным ключом, который одновременно является внешним ключом в другой таблице, выдается сообщение об ошибке в связи с нарушением ограничения.

Транзакции базы данных:

- содержат что-либо из следующего: g|J
  - o команды DML, выполняющие единое согласованное изменение данных;
  - o одна команда DDL;
  - o одна команда DCL;
- начинаются с выполнения первой исполняемой команды SQL;
- заканчиваются одним из следующих событий:
  - o выполнение команды COMMIT или ROLLBACK;
  - o выполнение команды DDL или DCL (автоматическая фиксация);
  - o ошибка, завершение сеанса работы или аварийный останов системы.

Обработка транзакций

Транзакции обеспечивают большую гибкость, более широкий спектр средств управления при изменении данных в случае ошибки в пользовательском процессе или сбоя системы.

Транзакции состоят из команд DML, которые вносят в данные одно согласованное изменение. Например, перевод средств между двумя счетами включает дебетование одного счета и кредитование другого на одну и ту же сумму. Успешно или неудачно выполненными должны быть одновременно признаны оба эти действия. Оставлять в базе данные о кредитовании без данных о дебетование нельзя.

Таблица 28. Типы транзакций

Тип	Описание
Манипулирование данными (DML)	Состоит из произвольного количества предложений DML, воспринимаемых сервером Oracle как одно целое или как логическая единица таблицы
Определение данных (DDL)	Состоит из одного предложения DDL
Управление данными (DCL)	Состоит из одного предложения DCL

Таблица 29. Команды управления транзакциями

Команда	Описание
COMMIT	Завершает текущую транзакцию, делая постоянными все произведенные изменения данных
SAVEPOINT	Устанавливает в текущей транзакции маркер точки сохранения (savepoint)
ROLLBACK [TO SAVEPOINT имя!	Прекращает текущую транзакцию, отменяя все произведенные изменения в данных

Каждое изменение данных, выполненное в ходе транзакции, является временным до'тех пор, пока транзакция не будет зафиксирована.

Состояние данных, предшествующее выполнению команд COMMIT или ROLLBACK:

- операции по обработке данных изменяют только буфер базы данных. Следовательно, предыдущее состояние данных может быть восстановлено;
- текущий пользователь может просмотреть результаты своих операции по обработке данных, запрашивая данные из таблиц;
- другие пользователи не могут видеть результаты операций обработки данных, выполняемых текущим пользователем;
- изменяемые строки блокируются, и другие пользователи не имеют возможности менять их содержимое.

Все внесенные изменения фиксируются с помощью COMMIT, в результате чего:

- измененные данные записываются в базу данных. Предшествующее состояние базы данных теряется;
- все пользователи могут видеть результаты выполненной операции;
- блокировка с измененных строк снимается, другие пользователи получают возможность вносить в них изменения;
- все точки сохранения удаляются.

Пример

```
INSERT INTO owner (Non, Ow, AdO)
VALUES (15, 'Федоров Ф.Ф.', 'Фадеева 45-54')
/
UPDATE lease SET Non=54 WHERE NLease = 3
/
COMMIT
/
```

Все незафиксированные изменения отменяются командой ROLLBACK, в результате чего:

- сделанные изменения теряются;
- разблокируются строки, с которыми производились операции.

Пример

```
DELETE FROM rent
/

ROLLBACK
/
```

Команда SAVEPOINT позволяет создавать в текущей транзакции маркеры, разбивающие ее на единицы меньшего размера. После этого с помощью команды ROLLBACK TO SAVEPOINT можно отменять незафиксированные изменения до этого маркера.

Примечание. Создание второй точки сохранения с тем. же именем, что и первая, вызывает удаление первой.

Пример

```
UPDATE rent SET Rn = Rn* 1.5 WHERE Type = 'дом'
/

SAVEPOINT  Raising
/

INSERT INTO realty (AdR, Type)
VALUES  ('Проспект Ленина,  13','дом')
/

SELECT * FROM realty WHERE Type = 'дом'
/

ROLLBACK TO Raising
/

SELECT * FROM realty WHERE Type = 'дом'
/
```

Результаты работы запросов до и после отката будут различными.

Если при выполнении команды обнаружена ошибка, то часть транзакции можно отменить при помощи неявного отката. Если в ходе транзакции возникла ошибка в одной-единственной команде DML, отменяется только результат этой команды, т.е. производится откат на уровне этой команды. Но изменения, внесенные во время этой транзакции предыдущими командами DML, сохраняются. Завершать транзакции рекомендуется явно.

Задания

1. Добавьте по одной новой записи в каждую таблицу. В таблице Rent для значения последнего типа недвижимости не устанавливайте пока плату.
2. Обновите данные таблиц Lease и Rent - в таблице Lease установите дату договора на 01.01.05 для всех договоров, срок начала которых раньше 01.01.05. Установите плату за каждый вид недвижимости на 50 % выше.
3. Убедитесь, что внесенные вами изменения сохранены в таблице. Сделайте эти изменения постоянными.
4. Удалите все данные из таблицы Lease для арендатора с номером 4. Создайте перед этим точку отката. Обновите данные таблицы Lease, потом сделайте откат. Убедитесь, что восстановлены исходные данные. Добавьте туда еще 2 строки, сделайте эти изменения постоянными.

Требования к сдаче лабораторной работы

1. Продемонстрировать измененные данные.
2. Продемонстрировать умение обрабатывать данные.

Лабораторная работа №8
Создание последовательностей

Многим приложениям требуются уникальные числа в качестве первичных ключей. Для генерации уникальных чисел можно включить необходимый код в само приложение или воспользоваться последовательностями.

Последовательность:

- автоматически-генерщэует уникальные числа:
- является совместно используемым объектом;
- обычно применяется для получения значении первичного ключа;
- заменяет кОдтзтпгакладной программе;
- ускоряет доступ к числам последовательности, если они находятся в сверхоперативной памяти (кэш-памяти^.

Числа последовательности хранятся и_генерируются независимо от таблиц. Следовательно; одна и та же последовательность может быть применена одновременно к нескольким таблицам.

Создание последовательности

Последовательность для автоматической генерации чисел создается командой CREATE SEQUENCE.

Синтаксис:

```
CREATE SEQUENCE последовательность
[INCREMENT BY n]
[START WITH n]
[{MAXVALUE n | NOMAXVALUE}]
[{MINVALUE n | NOMINVALUE)!
[ICYCLEINOCYCLE11
[{CACHE n | NOCACHE}]
```

Здесь: *последовательность* имя генератора последовательности;

INCREMENT BY п интервал между двумя последовательными номерами; п - целое. По умолчанию п = 1;

START WITH n	первое генерируемое число в последовательности. По умолчанию это 1;
MAXVALUE n	максимальное значение, которое может генерировать последовательность;
NOMAXVALUE	максимальное значение по умолчанию, равное 1027;
MINVALUE n	минимальное значение;
NOMINVALUE	задает минимальное значение, равное 1;
CYCLE | NOCYCLE	продолжается ли циклическая генерация чисел после достижения максимального или минимального значения. По умолчанию - NOCYCLE;
CACHE | NOCACHE	количество чисел, которое сервер Oracle распределяет предварительно и хранит в памяти. По умолчанию сервер хранит в кэш-памяти 20 значений.

Пример

```
CREATE SEQUENCE N_OW_ID
      INCREMENT BY 1
      START WITH 10
      MAXVALUE 1000
      NOCACHE
      NOCYCLE
      /
```

Созданная нами последовательность документируется в словаре данных. Поскольку последовательность является объектом базы данных, информацию о ней можно найти в таблице словаря данных USER_OBJECTS.

Проверить параметры последовательности можно также путем выборки данных из таблицы словаря данных USER_SEQUENCES.

Пример

```
SELECT sequence_name, min_value, max_value,
increment_by, last_number FROM usersequences
/
```

Числа созданной последовательности можно использовать в качестве порядковых номеров для таолиц. Обращение к числам последовательности ^производится с помощью Нсейдостолбцов NEXTVAL и CURVAL.

Псевдостолбцы NEXTVAL и CURVAL

Псевдостолбец NEXTVAL используется для выборки следующего свободного числа последовательности. Имя NEXTVAL необходимо дополнить именем последовательности. Если вы ссылаетесь на *последовательность.NEXTVAL,* генерируется следующее число, и текущее число помещается в CURVAL.

Псевдостолбец CURVAL используется для ссылки на число, только что сгенерированное текущим пользователем. Прежде, чем обращаться к CURVAL, необходимо использовать NEXTVAL для генерации числа в текущем сеансе пользователя. Имя CURVAL необходимо дополнить именем последовательности. При ссылке на *последовательность.*CURVAL выдается последнее значение, возвращенное процессу этого пользователя.

Команды и предложения, где используются NEXTVAL и CURVAL:

- список SELECT команды SELECT, которая не является частью подзапроса;
- список SELECT подзапроса в команде INSERT;
- предложение VALUES команды INSERT;
- предложение SET команды UPDATE.

Команды и предложения, где НЕ используются NEXTVAL и CURVAL:

- список SELECT представления (view);
- команда SELECT с ключевым словом DISTINCT;
- команда SELECT с предложениями GROUP BY, HAVING, ORDER BY;
- подзапрос в команде SELECT, UPDATE или DELETE;
- предложение DEFAULT команды CREATE TABLE или ALTER TABLE.

Кэширование последовательностей в памяти ускоряет доступ к их значениям. Кэш-память заполняется при первой ссылке на последовательность. В ответ на каждый последующий запрос выдается одно из чисел, находящихся в кэш-памяти. Если последнее из этих значений использовано, запрос следующего значения приводит к записи в память очередной партии значений.

Хотя генераторы последовательностей выдают числа без пропусков, это действие выполняется независимо от операций COMMIT и ROLLBACK. Следовательно, при откате команды, содержащей ссылку на последовательность, это число теряется. Еще одна возможная причина пропусков в последовательности - это сбой системы. Значения последовательности, находящиеся в кэш-памяти, при сбоях теряются.

Поскольку последовательности не связаны прямо с таблицами, одна и та же последовательность может использоваться в нескольких таблицах. В этом случае пропуски в последовательности чисел могут быть в каждой таблице.

Увидеть следующее свободное значение последовательности, не увеличив его, можно только в случае, если последовательность создана с параметром NOCACHE. Для этого выполняется запрос к таблице USERSEQUENCES.

Изменение параметров последовательности

Если последовательность достигла верхнего предела (MAXVALUE), дополнительные значения не предоставляются, и возникнет ошибка. Чтобы продолжать пользоваться последовательностью, можно изменить ее параметры с помощью команды ALTER SEQUENCE.

Синтаксис:

ALTER SEQUENCE *последовательность*
[INCREMENT BY n]
[{MAXVALUE n | NOMAXVALUE}]
[{MINVALUE n | NOMINVALUE}]
[{CYCLE | NOCYCLE}] [{CACHE n | NOCACHE}]

Указания

- Для изменения параметров необходимо быть владельцем последовательности или иметь для нее привилегию ALTER.

- Команда ALTER SEQUENCE влияет только на числа, генерируемые после введения изменения.
- Выполняются некоторые проверки. Например, новое значение MAXVALUE не может быть меньше текущего числа в последовательности.
- Изменить параметр START WITH командой ALTER SEQUENCE нельзя. Чтобы начать последовательность с другого числа, необходимо ее удалить (drop) и создать заново.

Для удаления последовательности из словаря данных используется команда DROP SEQUENCE. Чтобы удалить последовательность, необходимо быть ее владельцем или иметь привилегию DROP ANY SEQUENCE.

Синтаксис:

DROP SEQUENCE *последовательность;*

Здесь: *последовательность* имя генератора последовательности.

Вопросы

1. Что такое последовательность? Когда она используется?
2. Какие у последовательности есть параметры?
3. Что такое кэширование последовательности? Зачем оно нужно?
4. Как, не увеличивая текущее значение, просмотреть следующее свободное число последовательности?
5. С какими командами и предложениями используются псевдостолбцы NEXTVAL и CURVAL? С какими предложениями эти псевдостолбцы не используются? Приведите примеры.
6. Как изменить параметры последовательности?
7. Влияет ли откат транзакции на последовательность?

Задание

В таблицах, где используется счетчик в качестве первичного ключа, сгенерируйте последовательности с различными начальными значениями и шагом. Попробуйте применить транзакции с откатом. Объясните результат. Просмотрите сведения о последовательности из словаря данных. Не сохраняйте внесенные изменения, а последовательности удалите.

Лабораторная работа №9
Триггеры базы данных

Триггеры базы данных - процедуры, которые хранятся в базе данных и неявно исполняются («возбуждаются»), когда модифицируется ассоциированная таблица.

Создание **триггеров**

Создаются триггеры командой CREATE TRIGGER, синтаксис которой имеет вид:

```
CREATE [OR REPLACE] TRIGGER шля триггера
(BEFORE | AFTER} {UPDATE | DELETE | INSERT} ON
имя_таблщы
[FOR EACH ROW]
[DECLARE
{секция объявления переменных}]
BEGIN
{тело триггера}
END
/
```

Здесь должны быть выполнены следующие условия.

1. Имена триггеров должны быть уникальными среди всех триггеров в той же схеме. Имена триггеров не обязаны быть уникальными по отношению к другим объектам схемы (таким как таблицы, обзоры, процедуры); например, таблица и триггер могут иметь одно и то же имя (хотя, во избежание путаницы, это не рекомендуется).

2. Либо опция BEFORE, либо опция AFTER должна быть указана в предложении CREATE TRIGGER, чтобы точно специфицировать, когда должно исполняться тело триггера по отношению к исполнению предложения триггера. В предложении CREATE TRIGGER опция BEFORE или AFTER задается непосредственно перед ключевым словом, обозначающим предложение триггера.

Примечание. Триггеры строк AFTER несколько более эффективны, чем триггеры строк BEFORE.

3. Предложение триггера специфицирует:
    а) Тип предложения SQL, которое возбуждает тело триггера. Допустимыми возможностями являются DELETE, INSERT и UPDATE. В спецификацию предложения триггера могут быть включены одна, две или все три эти опции;
    б) Таблицу, ассоциированную с триггером. Заметьте, что в предложении триггера может быть специфицирована ровно одна таблица.

Примечание. Если предложение триггера специфицирует UPDATE, то в эту спецификацию может быть включен необязательный список столбцов. Если вы включаете список столбцов, то данный триггер возбуждается по предложению UPDATE лишь тогда, когда это предложение обновляет один из перечисленных столбцов. Если вы опускаете список столбцов, то триггер возбуждается при обновлении любого столбца ассоциированной таблицы. Список столбцов не может быть специфицирован для предложений триггера INSERT или DELETE.

4. Присутствие или отсутствие опции FOR EACH ROW определяет, является ли этот триггер триггером предложения или триггером строки. Если эта опция включена, она указывает, что тело триггера возбуждается отдельно для каждой строки таблицы, затрагиваемой предложением триггера. Отсутствие опции FOR EACH ROW означает, что данный триггер должен возбуждаться лишь один раз для предложения триггера.
5. Тело триггера - это блок PL/SQL, который может содержать предложения SQL и PL/SQL. Эти* предложения исполняются тогда, когда выдано предложение триггера и ограничение триггера (если оно есть) вычислено как TRUE. Для триггеров строк тело триггера имеет некоторые специальные конструкты, которые могут быть включены в код этого блока PL/SQL: корреляционные имена, опцию REFERENCING, а также условные предикаты INSERTING, DELETING и UPDATING.

Доступ к значениям столбцов в триггерах строки

Внутри тела триггера строк код PL/SQL и предложения SQL имеют доступ как к старым, так и к новым значениям столбцов текущей строки, затрагиваемой предложением триггера. Для каждого столбца модифицируемой таблицы определены два КОРРЕЛЯЦИОННЫХ ИМЕНИ: одно - для старого **(old),** другое - для нового значения столбца **(new).** В зависимости от типа предложения триггера, то или иное корреляционное имя может быть лишено смысла.

- Триггер, возбужденный предложением INSERT, имеет осмысленный доступ лишь к новым значениям столбцов. Поскольку строка создается предложением INSERT, старые значения столбцов пусты (NULL).
- Триггер, возбужденный предложением UPDATE, имеет доступ как к старым, так и к новым значениям столбцов для обоих возможных типов триггера (BEFORE или AFTER).
- Триггер, возбужденный предложением DELETE, имеет осмысленный доступ лишь к старым значениям столбцов. Поскольку строка перестает существовать после ее удаления, новые значения столбцов пусты (NULL).

Новые значения столбцов адресуются квалификатором NEW перед именем столбца, старые - квалификатором **OLD.**

Примечание. Старые и новые значения доступны как в триггерах BEFORE, так и в триггерах AFTER. Назначать новое значение столбца можно в триггере строк BEFORE, но не в триггере строк AFTER (потому что предложение триггера уже выполнено, прежде чем триггер AFTER получает управление). Если триггер строк BEFORE изменяет значение NEW для столбца, то триггер AFTER, возбужденный тем же самым предложением, видит значение, которое было назначено триггером BEFORE.

Пример

Триггер на каскадное обновление таблиц Rent-Realty

```
CREATE OR REPLACE TRIGGER
CURENT_REALTY
AFTER UPDATE ON RENT
FOR EACH ROW
BEGIN
IF (:OLD.Type o :NEW.Type) THEN
UPDATE Realty SET Realty.Type = :NEW.Type
WHERE Realty.Type = :OLD.Type;
END IF;
END
/
```

Триггер на каскадное удаление из таблиц Realty-Lease

```
CREATE OR REPLACE TRIGGER
CD  REALTY  LEASE
BEFORE DELETE ON REALTY
FOR EACH ROW
BEGIN
DELETE Lease WHERE Lease.AdR = :OLD.AdR;
END
/
```

Триггер на генерацию значения первичного ключа для таблицы Lease

```
CREATE OR REPLACE TRIGGER INS  LEASE
BEFORE INSERT ON LEASE
FOR EACH ROW
BEGIN
SELECT lease  sq.NEXTVAL  INTO  :NEW.NLease
FROM DUAL;
END
/
```

Условные предикаты

Если триггер может быть возбужден более чем одним типом предложения DML (например, «INSERT OR DELETE OR UPDATE ON Lease»), то в теле триггера можно использовать условные предикаты INSERTING, DELETING и UPDATING, для того чтобы выполнять различные участки кода в зависимости от типа предложения, возбудившего триггер. Предположим, что предложение триггера определено следующим образом:

INSERT OR UPDATE ON Lease

В коде внутри тела триггера вы можете использовать следующие условия:

IF INSERTING THEN . . . END IF;

IF UPDATING THEN . .. END IF;

Первое условие будет вычисляться как TRUE лишь в тех случаях, когда триггер был возбужден предложением INSERT; второе условие

будет вычисляться как TRUE лишь в тех случаях, когда триггер был возбужден предложением UPDATE.

Примечание. Лишь один триггер каждого типа может существовать на таблицу. Это позволяет иметь для таблицы двенадцать возможных триггеров:

BEFORE UPDATE строк
AFTER UPDATE строк
BEFORE DELETE строк
AFTER DELETE строк
BEFORE INSERT предложения
AFTER INSERT предложения
BEFORE INSERT строк
AFTER INSERT строк
BEFORE UPDATE предложения
AFTER UPDATE предложения
BEFORE DELETE предложения
AFTER DELETE предложения

Каждая таблица может иметь до четырех триггеров UPDATE (BEFORE/AFTER, предложения/строки), независимо от того, возбуждаются ли эти триггеры при обновлении специфических столбцов.

Чтобы создать триггер в своей схеме, вы должны иметь системную привилегию CREATE TRIGGER, а также либо:

- владеть таблицей, специфицированной в предложении триггера;
- иметь привилегию ALTER для таблицы, специфицированной в предложении триггера;
- иметь системную привилегию ALTER ANY TABLE.

Чтобы создать триггер в схеме другого пользователя, вы должны иметь системную привилегию CREATE ANY TRIGGER. Эта привилегия позволять создать триггер в любой схеме и ассоциировать его с таблицей любого пользователя.

Удаление триггеров

Для удаления триггера из базы данных используйте команду DROP TRIGGER. Например, чтобы удалить триггер с именем REORDER, введите следующее предложение:

```
DROP TRIGGER reorder
/
```

Чтобы удалить триггер, вы должны иметь его в своей схеме либо иметь системную привилегию DROP ANY TRIGGER.

Включение и выключение триггеров

Триггер может находиться в одном из двух различных режимов:

1) включен - выполняет свое тело, если выдано предложение триггера и если ограничение триггера (если есть) вычисляется как TRUE;
2) выключен - не выполняет свое тело, даже если выдано предложение триггера и ограничение триггера (если есть) вычисляется как TRUE.

Вы можете временно выключить триггер, если имеет место одно из следующих условий:

- объект, к которому обращается триггер, недоступен;
- вы должны выполнить массовую загрузку данных и хотите осуществить ее быстро, не возбуждая триггеров;
- вы загружаете данные в таблицу, к которой применяется триггер.

По умолчанию, триггер включается в момент его создания. Чтобы отключить триггер, используйте команду ALTER TRIGGER с опцией DISABLE.

Пример

Отключение триггера CU_RENTJREALTY:

```
ALTER TRIGGER CUJRENTJREALTY DISABLE
/
```

Вы можете одновременно отключить все триггеры, ассоциированные с таблицей, с помощью команды ALTER TABLE с опциями DISABLE и ALL TRIGGERS.

Пример

Отключение всех триггеров, определенных для таблицы Realty:

```
ALTER TABLE Realty DISABLE ALL TRIGGERS
```

По умолчанию триггер автоматически включается в момент его создания, однако позже он может быть выключен. Закончив задачу, для которой потребовалось выключать триггер, вы можете снова включить его.

Чтобы включить триггер, используйте команду ALTER TRIGGER с опцией ENABLE.

Пример

Включение триггера CU_RENT_REALTY:

```
ALTER TRIGGER CUJRENTJREALTY ENABLE
/
```

Вы можете одновременно включить все триггеры, ассоциированные с таблицей, с помощью команды ALTER TABLE с опциями ENABLE и ALL TRIGGERS.

Пример

Включение всех триггеров, определенных для таблицы Realty:

```
ALTER TABLE Realty ENABLE ALL TRIGGERS
/
```

Для включения и выключения триггеров с помощью команды ALTER TABLE вы должны либо владеть таблицей, либо иметь объектную привилегию ALTER TABLE для таблицы или системную привилегию ALTER ANY TABLE. Для включения или выключения индивидуального триггера с помощью команды ALTER TRIGGER вы должны либо владеть триггером, либо иметь системную привилегию ALTER ANY TRIGGER.

Вывод информации о триггерах

Следующие обзоры словаря данных раскрывают информацию о триггерах:

. USERJTRIGGERS;
. ALLJTRIGGERS;
. DBAJTRIGGERS.

Примеры применения триггеров

Вы можете использовать триггеры разнообразными способами, чтобы привязать к своим нуждам управление данными в базе данных Oracle. Например, триггеры обычно используются для:

- предотвращения незаконных транзакций;
- обеспечения ссылочной целостности между узлами в распределенной базе данных;

- реализации сложных организационных правил;
- ввода в действие комплексных правил защиты;
- прозрачной регистрации событий;
- автоматической генерации значений вычисляемых столбцов;
- поддержания синхронных дублирований таблиц.

Вопросы

1. Что такое триггер? Для чего применяются триггеры?
2. Какие ограничения накладываются на пространство имен триггеров?
3. Какие вы знаете опции триггера? За что они отвечают?
4. Сколько и каких триггеров можно создать для одной таблицы?
5. Какими привилегиями необходимо обладать для создания триггера? Использования триггера? Удаления триггера? Изменения триггера?

Задания

1. Напишите триггеры, обеспечивающие каскадное обновление и каскадное удаление из таблиц. Продемонстрируйте их работу. Используйте хотя бы один триггер с условным предикатом.
2. Напишите триггеры, которые при добавлении новой записи автоматически генерируют значение первичного ключа (подсказка: при написании триггера можно воспользоваться последовательностью). Обеспечьте проверку правильности вводимых данных (желательно обработкой исключительной ситуации). Продемонстрируйте их работу.
3. Модифицируйте триггер на вставку новой записи в таблицу Lease так, чтобы он автоматически записывал текущую дату заключения договора в формате 'DD.MM.YYYY HH24:МГ. Продемонстрируйте его работу.
4. Выведите из словаря данных информацию о пользовательских триггерах. Отключите триггер на каскадное обновление для таблицы Realty. Отключите все триггеры для таблицы Owner. Выведите информацию о пользовательских триггерах: имя триггера, имя таблицы из предложения триггера, его статус. Включите все триггеры. Еще раз просмотрите информацию из словаря данных о триггерах.

Требования к сдаче лабораторной работы

1. Продемонстрировать наличие и работу триггеров базы данных.
2. Уметь включать/отключать триггеры.
3. Уметь выводить информацию из словаря данных о триггерах.

Лабораторная работа Ml O
Создание индексов.
Оптимизация работы запросов при использовании индексов

Индекс сервера Oracle - это объект базы данных, с помощью которого можно ускорить поиск строк за счет использования указателя. Индексы создаются вручную или автоматически. Они прозрачны для пользователя. Если вы не создали индекс по столбцу, таблица будет просматриваться полностью.

Индекс - это объект базы данных, обеспечивающий прямой и быстрый доступ к строкам в таблице. Индексы создаются для уменьшения потребности в дисковых операциях ввода-вывода за счет использования B*-дерева для быстрого поиска данных. Индексы автоматически используются и поддерживаются сервером Oracle. Если индекс создан, от пользователя в дальнейшем никаких прямых действий не требуется. Индексы физически и логически независимы от индексируемой таблицы. Это означает, что индексы могут быть созданы или удалены в любое время и это не повлияет на базовые таблицы и другие индексы.

Имеется 2 типа индексов. Первый - это уникальный индекс. Сервер Oracle создает его автоматически, если для столбца в таблице задано ограничение PRIMARY KEY или UNIQUE. Имя индекса - это имя, присвоенное ограничению. Пользователь может создавать индексы еще одного типа - неуникальные индексы. Например, для увеличения скорости обработки запроса с соединением таблиц можно создать индекс по столбцу с ограничением FOREIGN KEY.

Если индекс создан, то сервер обращается к нему всякий раз, когда можно ускорить доступ к данным.. Использование индексов происходит автоматически. Использование индексов частично зависит от того, какой оптимизатор Oracle задействован в данный момент. Сервер поддерживает 2 метода оптимизации: оптимизацию на основе правил и оптимизацию по стоимости (более подробно об оптимизаторе и способах оптимизации будет говориться в следующей лабораторной работе).

При оптимизации на основе правил сервер решает, пользоваться ли индексом, руководствуясь своими внутренними правилами. При этом сервер определяет индексированные столбцы и типы индексов. Метод оптимизации по стоимости использует для выбора плана выполнения команды SQL статические данные о таблицах и информацию о доступных индексах.

Создание индекса

Индекс по одному или нескольким столбцам создается с помощью команды CREATE INDEX.

Сокращенный синтаксис:

CREATE INDEX *индекс* ON *таблица {столбец [, столбец]...);*

Здесь: *индекс* имя индекса;

таблица имя таблицы;

столбец имя столбца в индексируемой таблице.

Пример

```
CREATE INDEX lease_ind ON lease (AdR)
/
```

Структура индекса и типы индексов

Таблица 30. Типы индексов

Тип	Описание
Уникальный	Обеспечивает уникальность значений для указанных столбцов
Неуникальный	Ускоряет запросы
Одностолбцовый	В индексе использован только один столбец
Составной или сложный	Индекс может содержать до 16 столбцов для ускорения запросов или проверки уникальности. Столбцы необязательно должны быть смежными

Примечание. Типы индексов не исключают друг друга. Например, можно создать уникальный составной индекс.

СУБД Oracle предлагает много различных типов индексов.

1. **Индексы на основе B*-дерева.** Эти индексы называют «обычными». Они, несомненно, чаще всего используются в СУБД Oracle, да и в других СУБД. Аналогичные по конструкции двоичному дереву, они обеспечивают быстрый доступ по ключу к отдельной строке или диапазону строк, требуя обычно очень немного чтений для поиска соответствующей строки. Индекс на основе B*-дерева имеет несколько подтипов.

а) **Таблицы, организованные по индексу.** Это таблицы, хранящиеся в структуре B*-дерева.

б) **Индексы кластера на основе B*-дерева.** Они немного отличаются от обычных, используются для индексации ключей кластера и отдельно рассматриваться не будут. Они используются не для перехода от ключа к строке, а для перехода от ключа кластера к блоку, содержащему строки, связанные с этим ключом.

в) **Индексы с обращенным ключом.** Это индексы на основе B*-дерева, байты ключа в которых инвертированы. Это используется для более равномерного распределения записей по индексу при вводе возрастающих значений ключей. Предположим, при использовании последовательности для генерации первичного ключа генерируются значения 987500, 987501, 987502 и т. д. Поскольку это последовательные значения, они будут попадать в один и тот же блок индекса, конкурируя за него. В индексе **с** обращенным ключом сервер Oracle будет индексировать значения 205789, 105789, 005789. Эти значения обычно будут далеко отстоять друг от друга в индексе, и вставки в индекс будут распределены по нескольким блокам.

г) **Индексы по убыванию.** Индексы по убыванию позволяют отсортировать данные в структуре индекса от «больших» к «меньшим» (по убыванию), а не от «меньших» к «большим» (по возрастанию).

2. **Индексы на основе битовых карт.** Обычно в B*-дереве имеется однозначное соответствие между записью индекса и строкой - запись индекса указывает на строку. В индексе на основе битовых карт запись использует битовую карту для ссылки на большое количество строк одновременно. Такие индексы подходят для данных с небольшим количеством различных значений, которые обычно только читаются. Столбец, имеющий всего три значения - Y, N и NULL, - в таблице с миллионом строк очень хорошо подходит для создания индекса на основе битовых карт.
3. **Индексы по функции.** Эти индексы на основе B*-дерева или битовых карт хранят вычисленный результат применения функции к столбцу или столбцам строки, а не сами данные строки. Это можно использовать для ускорения выполнения запросов вида:

```
SELECT * FROM T
WHERE ФУНКЦИЯ(СТОЛБЕЦ) = НЕКОТОРОЕ_ЗНАЧЕНИЕ,
```

поскольку значение ФУНКЦИЯ(СТОЛБЕЦ) уже вычислено и хранится в индексе.

4. **Прикладные (application domain) индексы.** Это индексы, которые строит и хранит приложение, будь то в базе данных Oracle или даже вне базы данных Oracle. Надо сообщить оптимизатору, насколько избирателен индекс, насколько «дорогостояще» его использование, а оптимизатор решает на основе этой информации, использовать этот индекс или нет. Текстовый индекс **interMedia** - пример прикладного индекса; он построен с помощью тех же средств, которые можно использовать для создания собственных прикладных индексов.
5. **Текстовые индексы interMedia.** Это встроенные в сервер Oracle специализированные индексы для обеспечения поиска ключевых слов в текстах большого объема.

Индексы с обращенным ключом

Индексы с обращенным ключом создаются, если в конце предложения CREATE INDEX указана опция REVERSE. В индексе с обращенным ключом просто обращается порядок байтов в каждом столбце ключа индекса, в результате обращенные ключи окажутся «далеко» друг от друга. При этом сокращается количество экземпляров, обращающихся к одному и тому же блоку, и, следовательно, количество выполняемых тестовых опросов.

Один из недостатков индекса с обращенными ключами - то, что его нельзя использовать в некоторых случаях, когда обычный индекс вполне применим. Например, при поиске по следующему критерию индекс с обращенным ключом по столбцу x не поможет: **where** x > **5.** Данные в индексе не отсортированы, поэтому просмотреть диапазон нельзя. С другой стороны, некоторые просмотры диапазонов в индексе с обращенным ключом вполне выполнимы. Если имеется составной индекс по столбцам X, Y, при поиске по следующему условию можно будет использовать индекс с обращенным ключом и «просматривать диапазон» в нем: **where x = 5.**

Дело в том, что байты в столбцах X и Y обращены. Сервер Oracle не обращает байты значения X || Y, а сохраняет в записи индекса результат выполнения reverse(X) || reverse(Y). Это означает, что все значения X = 5 будут храниться вместе, так что сервер Oracle может просматривать последовательно листовые блоки индекса для поиска всех таких строк.

Индексы по убыванию

Индексы по убыванию - новое средство сервера Oracle, начиная с версии 8i, расширяющее функциональные возможности индекса на основе

B -дерева. Они позволяют хранить значения столбца в индексе от «большего» к «меньшему», а не по возрастанию.

Возможность создавать индекс по убыванию имеет значение только для составного индекса, в котором некоторые столбцы упорядочены по возрастанию (ASC), а некоторые - по убыванию (DESC). Чтобы создать индекс, упорядоченный по убыванию, достаточно в предложении CREATE INDEX указать ключевое слово DESC после имени индексируемого столбца:

```
CREATE INDEX DESCJRNJDX ON RENT
(RN DESC, TYPE ASC)
/
```

Если в запросе присутствует условие where с проверкой проиндексированного столбца и сортировка по этому столбцу в порядке убывания, то в плане выполнения можно увидеть дополнительный шаг сортировки (если не создан индекс по убыванию). В нашем случае с использованием индекса, упорядоченного по убыванию, дополнительного шага сортировки в конце плана нет.

Индексы на основе битовых карт

Индексы на основе битовых карт появились, начиная с версии 7.3 сервера Oracle. Индексы на основе битовых карт - это структуры, в которых хранятся указатели на множество строк, соответствующих одному значению ключа индекса, тогда как в структуре B*-дерева количество ключей индекса обычно примерно соответствует количеству строк. В индексе на основе битовых карт записей очень мало, и каждая из них указывает на множество строк. В индексе на основе B*-дерева обычно имеется однозначное соответствие - запись индекса ссылается на одну строку.

Предположим, создается индекс на основе битовых карт по столбцу Type в таблице Realty:

```
CREATE BITMAP INDEX TYPEJDX ON Realty (Type)
/
```

Сервер Oracle будет хранить в индексе примерно следующее.

Таблица 31. Пример индекса на основе битовой карты

Значение/ Строка	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Комната	1	0	0	0	1	0	0	0	0	1	1	0	0	0
Квартира 1 коми	0	0	0	1	0	1	0	0	0	0	0	0	0	1
Квартира 2 коми	0	1	1	0	0	0	0	0	0	0	0	0	0	0
Квартира 3 коми	0	0	0	0	0	0	1	0	0	0	0	0	0	0
Гараж	0	0	0	0	0	0	0	0	0	0	0	0	1	0
Офисное помещение	0	0	0	0	0	0	0	0	0	0	0	1	0	0
Дача	0	0	0	0	0	0	0	1	1	0	0	0	0	0

Это показывает, что в строках 1, 5, 10 и 11 находится значение «Комната», тогда как в строках 4, 6 и 14 - значение «Квартира 1 коми». Также понятно, что пустых строк нет (индексы на основе битовых карт содержат записи для пустых значений — отсутствие такой записи в индексе означает, что пустых строк нет). Если необходимо посчитать, в скольких строках хранится значение «Квартира 1 коми», индекс на основе битовых карт позволит сделать это очень быстро, без обращения к таблице. Если необходимо найти все строки, в которых в столбце Type хранится значение «Квартира 1 коми» или «Квартира 2 крмн», достаточно просто скомбинировать соответствующие битовые карты из индекса.

Таблица 32. Пример комбинирования значений в индексе на основе битовой карты

Значение/ Строка	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Квартира 1 коми	0	0	0	1	0	1	0	0	0	0	0	0	0	1
Квартира 2 коми	0	1	1	0	0	0	0	0	0	0	0	0	0	0
Квартира 1 коми или Квартира 2 коми	0	1	1	1	0	1	0	0	0	0	0	0	0	1

Это позволяет быстро понять, что критериям поиска удовлетворяют строки 1, 2, 3, 4, 6, 14. Битовая карта, которую сервер Oracle хранит для каждого значения ключа, устроена так, что каждая позиция представляет идентификатор строки базовой таблицы на случай, если понадобится выбрать для дальнейшей обработки соответствующую строку. На запросы вида:

```
SELECT COUNT (*) FROM Realty
WHERE Type = 'Квартира 1 коми'
OR Type = 'Квартира 2 коми'
/
```

можно ответить непосредственно по индексу на основе битовых карт. Для ответа на запрос вида:

```
SELECT * FROM Realty
WHERE Type = 'Квартира 1 коми'
OR Type = 'Квартира 2 коми'
/
```

придется обратиться к таблице. Сервер Oracle применит функцию, преобразующую установленный бит i в битовой карте в идентификатор строки, по которому можно обратиться к таблице.

Индексы по функциям

Индексы по функциям были добавлены в версии сервера Oracle 8.1.5. Они позволяют индексировать вычисляемые столбцы и эффективно использовать их в запросах, т. е. они позволяют реализовывать не зависящий от регистра символов поиск или сортировку, искать результаты вычисления сложных выражений и эффективно расширять возможности языка SQL, добавляя собственные функции, а затем эффективно осуществляя по ним поиск.

Индексы по функциям имеет смысл использовать, т. к.:

1) их легко добавить, и они дают немедленный результат;
2) они могут ускорить работу существующих приложений, не изменяя логику их работы и запросы.

Пример

```
CREATE INDEX OWN_UPPERIDX ON Owner
(UPPER(Ow))
/
```

Теперь имеется индекс по функции UPPER от столбца. Любое приложение, использующее не зависящие от регистра запросы, будет обращаться к этому индексу, что значительно повысит производительность. До появления подобных индексов нужно было просматривать каждую строку в таблице Owner, переводить значение столбца в верхний регистр и сравнивать с литералом. Теперь, при наличии индекса по UPPER(Ow), сервер ищет заданный литерал по индексу, просматривая несколько блоков данных, а затем обращается к таблице по идентификатору строки для получения соответствующих данных. Это делается очень быстро.

Рост производительности особенно заметен при индексировании по заданным пользователем функциям от столбцов. Начиная с версии Oracle 7.1, появилась возможность использовать в операторах SQL функции, задаваемые пользователем. Обратите внимание, что в пользовательской функции должно присутствовать новое ключевое слово, DETERMINISTIC. Оно означает, что данная функция при одних и тех же входных данных всегда даст одинаковый результат. Это обязательно следует указать при создании индекса по функции, заданной пользователем. Необходимо сообщить серверу Oracle, что результат выполнения функции предопределен (DETERMINISTIC) и она будет всегда давать одинаковые результаты при одинаковых входных данных.

При использовании индексов по функции существует проблема: не удается создать такой индекс по встроенной функции TO_DATE. Для этого придется создавать собственную интерфейсную функцию для встроенной функции TO_DATE и индексировать ее.

Пример

```
CREATE OR REPLACE
FUNCTION MY_TO_DATE (P_STR IN VARCHAR2,
PJFMT IN VARCHAR2) RETURN DATE
DETERMINISTIC
IS
BEGIN
RETURN TO_DATE (P_STR, PJFMT);
END;
/
CREATE INDEX FUNDATE ON
Lease (MY_TO_DATE (LDate, 'Mon-yyyy'))
/
```

Увеличение количества индексов для таблицы не всегда ускоряет запросы. Каждая операция DML, выполняемая над таблицей с индексами, требует обновления индекса. Чем больше индексов связано с таблицей, тем больше усилий требуется от сервера на обновление всех индексов после операции DML.

Когда создавать индекс:

- столбец часто используется в предложении WHERE или условиях соединения;
- столбец имеет широкий диапазон значений;
- столбец содержит большое количество неопределенных значений;
- два или более столбцов часто используются вместе в предложении WHERE или условии соединения;
- таблица большого размера, и предполагается, что большая часть запросов будет выбирать менее 10-15% строк.

Помните, что если вы хотите обеспечить уникальность, то следует задать ограничение UNIQUE в определении таблицы. Тогда уникальный индекс будет создан автоматически.

Когда не создавать индекс:

- таблица небольшого размера;
- столбцы не очень часто используются как условие в запросе;
- большая часть запросов будет выбирать более 10-15% строк;
- таблица часто обновляется.

Просмотр индексов, удаление индексов

Существование индексов можно проверить с помощью представления словаря данных USER_INDEXES. Столбцы, включенные в индекс, можно проверить с помощью представления USERINDCOLUMNS.

Изменять индексы нельзя. Чтобы изменить индекс, его нужно сначала удалить, потом создать заново. Для удаления индекса из словаря используется команда DROP INDEX. Чтобы удалить индекс, необходимо быть его владельцем или иметь привилегию DROP ANY INDEX.

Синтаксис:

DROP INDEX *индекс;*

Здесь: *индекс* имя индекса.

Вопросы

1. Что такое индекс? Какие бывают виды индексов?
2. Что такое индекс на основе битовых карт?
3. Что такое прикладной индекс?
4. Когда необходимо использовать индексы?
5. Можно ли проиндексировать представление? Зачем нужны индексы по реверсированным ключам?
6. В каких случаях используется функциональный индекс? Можно ли его применять для пользовательской функции? Если да, то в каком случае?

Задания

1. Создайте составной индекс по всем внешним ключам таблицы Lease. Напишите запрос, который будет использовать ваш индекс для доступа к таблице: например, выведите фамилии владельцев и адреса сдаваемой ими недвижимости, если номер владельца находится в первой пятерке записей, а номер арендатора - во второй. Убедитесь, что используется ваш индекс. Напишите запрос, который будет обращаться только к индексу, например, пересчитайте количество договоров, удовлетворяющих условиям из первого запроса. Удалите индекс.
2. Создайте составной индекс по столбцам NOn и NTn таблицы Lease, который будет упорядочен по убыванию по одному полю и по возрастанию по другому. Напишите запрос, который будет выводить данные из таблицы (таблиц), используя созданный индекс для доступа к таблице. Убедитесь, что используется ваш индекс. Напишите запрос, который будет обращаться только к индексу. Удалите индекс.
3. Создайте реверсный составной индекс по столбцам NOn и NTn таблицы Lease. Напишите запрос, который будет выводить данные из таблицы (таблиц), используя созданный индекс для доступа к таблице. Убедитесь, что используется ваш индекс. Напишите запрос, который будет обращаться только к индексу.
4. Создайте индекс на основе встроенной однострочной функции Oracle. Напишите запрос, который будет выводить данные из

таблицы (таблиц), используя созданный индекс для доступа к таблице. Убедитесь, что используется ваш индекс. Напишите запрос, который будет обращаться только к индексу. Создайте индекс, работающий с функцией to_date (в формате которой для даты удерживается только год). Подсказка: возьмите пользовательскую функцию для работы с датами, приведенную в книге Тома Кайта. Аналогично предыдущим случаям, напишите запросы, использующие индекс для доступа к таблице и для вывода данных.

5. Просмотрите информацию о пользовательских индексах в словаре данных. Удалите индексы. Снова создайте индекс из задания 1, просмотрите информацию о нем. Удалите его и пересоздайте, скомпрессировав по первому столбцу. Просмотрите изменения.

Лабораторная работа №11
Оптимизация кода приложения в Oracle.
Подсказки оптимизатору, команды Explain Plan иAutotrace

Данный материал посвящен ряду вопросов, связанных с проблемами оптимизации запросов SQL. Рассматривается процесс, с помощью которого Oracle создает план выполнения команды, что влияет на выбор плана, как его посмотреть, как на него повлиять.

Фазы обработки SQL-предложения

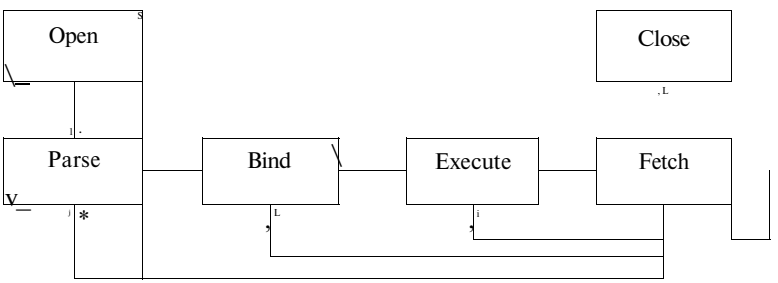

Рис. 3. Стадии обработки SQL-предложения

Четырьмя наиболее важными стадиями обработки SQL-запроса являются анализ (Parse), связывание (Bind), выполнение (Execute) и выборка (Fetch).

Обратные стрелки показывают сценарии обработки (например, Выборка -> Связывание (или пересвязывание) -> Исполнение -> Выборка (Fetch -> (Re)Bind -> Execute -> Fetch)). Стадия выборки применяется только к запросам и DML-предложениям, возвращающим выражение.

Примечание. Детальное описание обработки SQL-предложения можно найти в Oracle9i Application Developers Guide: Fundamentals & Concepts.

При обработке SQL-команды некоторые стадии (например, анализ) могут быть пропущены при условии, что идентичный SQL-запрос уже выполнялся ранее. Такой пропуск стадии анализа приводит к выигрышу в производительности. Т. е. при написании запросов надо стремиться к совместному использованию (разделению) курсора. Курсор может быть совместно использован (разделен) только теми SQL-командами, которые содержат в себе следующие идентичные элементы:

- текст:
  - верхний и нижний регистры;
  - пробельные символы (пробелы, символы табуляции, возврат каретки);
  - комментарии;
- объекты БД, к которым обращается запрос;
- типы данных переменных для указания диапазона.

Таким образом:

- только идентичные SQL-команды могут использовать один и тот же курсор;
- текст SQL-выражений должен быть в точности таким же, включая регистр, пробелы, возвраты каретки и комментарии;
- объекты, к которым обращается SQL-предложение, должны быть идентичны;
- типы переменных диапазона, используемых в SQL-запросах, должны быть одинаковыми.

Примечание.

- *Перед отправкой SQL-предложения серверу Oracle9i Server большинство инструментов Oracle предварительно обрабатывает SQL-предложения, чтобы сделать их как можно более идентичными, удаляя комментарии, сжимая до минимума пробельные символы и конвертируя текст в верхний или нижний регистры. SQL *Plus, однако, отсылает SQL-команды серверу Oracle9i Server именно в том виде, в котором они были введены пользователем.*
- *В Oracle9i PL/SQL прекомпилятор передает SQL-команды в точности так, как они были введены. Таким образом, SQL-предложения в блоке должны быть идентичны, чтобы воспользоваться преимуществами разделения курсора.*
- *Сервер Oracle проверяет, существуют ли идентичные предложения в совместно используемой области, прежде чем начать анализ вновь поступившего запроса. Если предложения различаются хотя бы регистром, они не рассматриваются как идеентичные. Избегая стадии анализа, можно обеспечить значительное повышение производительности.*

Разделяемые курсоры

- Если регистр или количество пробелов не совпадают, предложения не идентичны.

```
Select * from sh.customers where cust_id=180;

Select * from sh.CUSTOMERS where CUSTJEN180;
```

- Если объекты принадлежат различным пользователям, предложения не идентичны. SQL-команды должны быть идентичны, чтобы совместно использовать курсор. Отметим, что разделение предложений не играет роли в окружении системы поддержки решений (decision support system (DSS) environment), т.к. большинство предложений так или иначе все равно будут отличаться.

Первые два примера запросов не идентичны. Заметим, что регистр различается для имени таблицы и имени столбцов. Из-за этого различия регистров предложения не идентичны и не могут совместно использовать SQL-область. Из-за этого различия в регистрах предложений анализатор для них выдает различные внутренние коды (HASH VALUES), что можно проверить просмотром представления V$SQL.

Два предложения могут выглядеть как идентичные, но если объекты, на которые они ссылаются, на самом деле относятся к различным объектам базы, то предложения будут разными.

Как написать SQL для совместного использования курсора? Пишите настраиваемый код, использующий:

- хранимые процедуры и пакеты;
- триггеры базы данных;
- Referenced Oracle Forms Services процедуры;
- любые другие стандартные библиотеки и процедуры.

Команда EXPLAIN PLAN

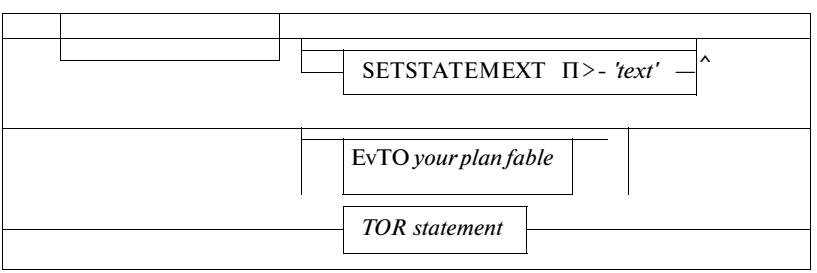

Рис. 4. Синтаксическая диаграмма команды Explain Plan

При помощи команды EXPLAIN PLAN вы можете поместить план выполнения той или иной инструкции в таблицу. Затем вы можете отобразить план, выполнив в SQL*Plus запрос к этой таблице.

На синтаксической диаграмме курсивные поля означают:

text	необязательный идентификатор предложения; задавать значение нужно для того, чтобы вы могли идентифицировать каждое предложение при обращении к таблице плана выполнения, если в ней хранится несколько планов;
schema table	необязательное имя выходной таблицы; по умолчанию задано PLAN_TABLE;
statement	текст SQL-предложения.

Вы можете просмотреть план выполнения, отобразив содержимое таблицы в том виде, какой он есть. Но обычно план выполнения отображают, используя иерархический запрос (результат его работы вы могли уже неоднократно видеть в окне Execution Plan программы Aqua Data Studio). Oracle разделяет выполнение запроса на серию вложенных шагов, каждый из которых поставляет данные родительскому шагу. Исходным родителем является сам запрос, результаты которого возвращаются приложению. Возможный запрос для отображения плана выполнения в иерархическом виде может выглядеть так:

```
SELECT LPAD('  '• 2*(level-l)) ‖ operation ||'  ' ‖
options ‖
'  ' ‖ objectjname ||'  ' ‖
DECODE (id.0.'Cost=' ‖ position) "Query plan"
FROM   PLANTABLE
START WITH id = 0 AND statement_id =
' my_statement_id'
CONNECT BY prior id=parent_id
AND statement_id = 'my_statement_id'
```

I

SQL*Plus AUTOTRACE

Команда EXPLAIN PLAN является мощным средством для сбора информации о плане выполнения запросов, но при работе не всегда удобно часто обращаться к таблице плана, чтобы посмотреть, как разбирался тот или иной запрос. Гораздо удобнее информацию о плане выполнения, а иногда и о статистике получать сразу же при работе запроса. Для этого используются настройки AUTOTRACE.

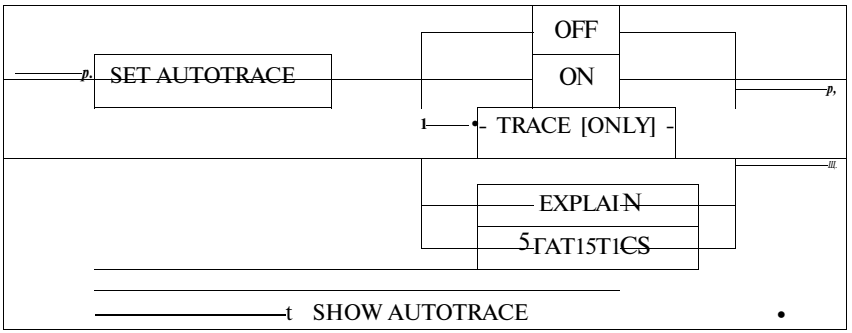

Рис. 5. Синтаксическая диаграмма команды Autotrace

В SQL* Plus можно автоматически после выполнения запроса получить план выполнения и некоторую дополнительную статистику обработанной SQL-команды, используя настройки AUTOTRACE. В отличие от команды EXPLAIN PLAN, предложение выполняется и выдает результат на самом деле (в EXPLAIN PLAN только собирается информация о плане выполнения, и, чтобы ее посмотреть, потом надо сделать выборку из таблицы плана). Однако вы можете отключить выполнение запроса, задав AUTOTRACE TRACEONLY EXPLAIN.

AUTOTRACE является удобным средством диагностики для последующей настройки SQL-предложения. Т. к. команда чисто декларативная, ее проще использовать, чем EXPLAIN PLAN.

Командные опции:

OFF	выключает автотрассировку SQL-предложения;
ON	включает автотрассировку SQL-предложения;
TRACEONLY	включает автотрассировку SQL-предложения и подавляет вывод результата запроса;
EXPLAIN	выводит план выполнения без статистики;
STATISTICS	выводит статистику без плана выполнения.

Примечание. Если обе командные опции EXPLAIN и STATISTICS опущены, по умолчанию выводятся и план выполнения, и статистика.

Oracle не всегда выбирает оптимальный план выполнения запроса.

Например, может отсутствовать статистика по некоторым таблицам, или она может быть устаревшей. Чтобы изменить план выполнения, вы можете управлять режимами оптимизатора, задавая их на. уровне сессии либо на уровне запроса (выдавая подсказки - hints - оптимизатору).

Режимы оптимизатора

Оптимизация по синтаксису

В этом режиме процесс сервера выбирает путь доступа к данным, анализируя запрос. Оптимизатор имеет полный набор правил для ранжирования путей доступа. Опытные разработчики, знакомые с этими правилами, учитывают их при настройке команд SQL. Оптимизатор *по синтаксису (rule-based optimizer* - RBO) является синтаксически управляемым, что означает, что для определения используемого плана выполнения он анализирует синтаксис команды вместе с информацией из словаря данных. Этот режим оптимизатора поддерживается для обратной совместимости с более ранними версиями сервера Oracle (версия Oracle 10g официально не поддерживает RBO).

Оптимизация по стоимости

В этом режиме оптимизатор анализирует каждую команду и определяет все возможные пути доступа к данным. Затем подсчитывается стоимость каждого пути и выбирается самый дешевый. Стоимость в основном базируется на количестве логических чтений. *Оптимизатор по стоимости (cost-based optimizer* - CBO) является статистически управляемым, это означает, что для получения наиболее эффективного плана выполнения он использует статистики, собранные для объектов, присутствующих в команде SQL. Если для какого-либо объекта в команде существуют статистические данные, будет задействован оптимизатор по стоимости. Если статистика не существует, то, в зависимости от параметров экземпляра (optimizerjnode, optimizer_dynamic_sampling и optimizer_features_enabled), CBO ее соберет для выполнения правильной оценки стоимости шагов.

Что такое стоимость?

Это один из самых популярных вопросов, который возникает/задается при просмотре плана выполнения, например, командой explain plan в следующем контексте - стоимость объединения *hash join* составляет 7 миллионов, а *nested loops* всего 42 - так почему *hash join* выполняется за 3 секунды, а *nested loops* за 14 часов?

Ответ прост: стоимость всегда представляет (и представляла) лучшее *расчетное* время, необходимое для выполнения команды. Однако поскольку это расчетное время, то возможны следующие основные причины некорректного определения стоимости:

- необходимая статистика о распределении данных есть, но она некорректна (неактуальна);
- нет необходимой статистики распределения данных;
- неизвестна истинная производительность оборудования сервера;
- неизвестна текущая рабочая загрузка (запросы, переменные привязки и их значения);
- ошибка (bug) в коде Oracle.

В Oracle 8i оптимизатор просто подсчитывал количество обращений к подсистеме ввода/вывода. План с наименьшим числом операций считался самым выгодным. Здесь не учитывался тот факт, что сканирование таблицы (tablescan) может потребовать гораздо больше времени ЦПУ, чем доступ с помощью индекса. Не учитывалось, например, что теоретически 128-блочная операция чтения может, на самом деле, превратиться в 25 отдельных операций чтения, поскольку некий случайный поднабор нужных блоков уже находится в кэше буферов. Не учитывался тот факт, что запрос операции В/В может быть ограничен обращением в кэш буферов (т. е. в память) вместо физического обращения к диску.

В Oracle 9i оптимизатор вводит новый элемент оценки, известный как стоимость ЦПУ (CPU cost). Сохраняя сведения о времени выполнения одноблочного и многоблочного чтения вместе с информацией о размере многоблочного чтения, оптимизатор учитывает это в формуле расчета стоимости. Кроме того, оптимизатор рассчитывает время ЦПУ, затрачиваемое на выполнение SQL-функций и операторов сравнения. Все это позволяет более точно определить стоимость выполнения каждого шага плана.

В Oracle 10g появился автономный оптимизатор *(offline optimizer).* Вы можете создать и сохранить ключевую статистическую информацию (в форме **профиля SQL),** которая поможет оперативному оптимизатору *(online optimizer)* корректно решать проблемы с неравномерным распределением данных. В отличие от хранимых шаблонов, профиль SQL - это не набор подсказок, которые говорят, что оптимизатору делать, а дополнительные статистические данные. Можно представить их как подсказки вида «На данном шаге вы получите в 15 раз больше данных, чем предположил оптимизатор».

В 9i и 10g собирается статистика использования кэша (cache statistics) на уровне объекта, т. е. частота работы с объектом и его объем в кэше. Цель - правильно рассчитывать стоимость шагов для новых команд. Кроме того, 9i и 10g собирает фактическую статистику выполнения шагов команды в представлениях v$sql_plan_statistics и v$sql_plan_statistics_all. Эти статистики, в теории, дают оптимизатору второй шанс по оптимизации запроса, если настоящая статистика слишком сильно отличается от предположений оптимизатора.

Как говорилось ранее, под стоимостью подразумевается время. В документации *Oracle 9.2 Performance Guide and Reference* вводится следующее определение:

```
Cost = (
#SRds * sreadtim +#MRds * mreadtim +
#CPUCycles / cpuspeed
) / sreadtim
```

где

SRds — число одноблочных чтений;

#MRds - число многоблочных чтений;

#CPUCycles - число циклов ЦПУ;

sreadtim - время чтения одного блока;

mreadtim - время многоблочного чтения;

cpuspeed - число циклов ЦПУ в секунду.

Таким образом, *стоимость* - это время выполнения команды, выраженное в единицах времени одиночного чтения.

Функции оптимизатора

Задача оптимизатора - установить связь между тем, *что* требуется команде SQL и *как* это будет выполнено машиной SQL. Для одной команды SQL может быть найдено огромное количество различных планов выполнения, и все из них возвращают правильный результат, но имеют разное время выполнения. Поэтому задача оптимизатора быстро найти самый дешевый план выполнения, учитывая время, затрачиваемое на поиск оптимального плана и время выполнения самого плана.

Для расчета стоимости плана выполнения оптимизатор использует стоимостную модель и статистику. Стоимостная модель состоит из стоимостных функций (cost function) для каждого возможного оператора в плане и из механизма расчета объема промежуточных результатов.

Таким образом, для каждой команды SQL оптимизатор выполняет следующие шаги:

- расчет выражений и условий с константами;
- трансформацию запроса. Для запросов с коррелированными подзапросами или представлениями оптимизатор может преобразовать исходный запрос в эквивалентный вариант с операцией объединения;
- выбор цели оптимизатора (optimizer approaches). Цель определяется значением параметра OPTIMIZERMODE. По умолчанию это CHOOSE. Т. е. если есть статистика хотя бы по одному объекту запроса, то CBO в режиме ALL_ROWS, иначе RBO;
- выбор метода доступа (access path). Для каждой таблицы в команде оптимизатор выбирает один или несколько возможных вариантов доступа к данным таблицы;
- выбор порядка объединения (Join order). В команде, где объединяется более двух таблиц, оптимизатор выбирает, какая пара таблиц должна объединяться первой, какая таблица затем объединяется с результатом и т. д.;
- выбор метода объединения (Join method).

Оптимизация команды **SQL**

Цель оптимизатора зависит от задач вашего приложения.

- Для пакетных приложений (например, Oracle Reports) необходима оптимизация пропускной *способности (best throutput)*. Пропускная способность для такого рода приложений является ключевой, поскольку основная цель приложения - закончить *всю* свою работу как можно за меньшее время. Время отклика является менее критичным, поскольку пользователи ждут полного окончания работы приложения.
- Для интерактивных приложений (например, Oracle Forms или запросы в SQL*Plus) цель оптимизации - наилучшее *время отклика (response time)*. Это связано с тем, что пользователь ждет не окончания выполнения всего задания, а возврата нескольких первых строк.

Поэтому необходимо правильно установить параметр OPTIMIZER_MODE, определяющий цель оптимизации запроса. Данный параметр может быть определен на уровне БД или на уровне сеанса. Он может принимать следующие значения:

```
OPTIMIZERMODE = {CHOOSE | RULE I FIRST ROWS [_w] |
ALLROWS}
/
```

где п - 1, 10,100, 1000.

CHOOSE использует оптимизацию по стоимости, если имеется статистика по крайней мере хотя бы для одной таблицы запроса.

Режим RULE - применяется оптимизация по синтаксису.

Режим FIRST_ROWS[_n] - оптимизатор выбирает такой план выполнения, который максимально быстро возвращает первые п строк. Хотя весь запрос может отработать и дольше. При небольших значениях п CBO пытается создать план, состоящий из вложенных циклов (nested loops) и поиска по индексу (index lookup). При больших значения п CBO использует хеш-объединение (hash join) и полное сканирование таблицы (full table scan).

Режим ALL_ROWS - выбор плана, который максимально быстро отрабатывает весь запрос. На уровне сессии параметр изменяется командой ALTER SESSION.

Синтаксис:

```
ALTER SESSION SET optimizermode = {choose | rule
| firstjrows [_n] | alljrows};
```

Вы можете изменить цель оптимизатора на уровне команды, используя подсказку (hint).

Синтаксис:

```
SELECT /*+ {choose | rule | first_rows(n) | alljrows} */
имена_полей FROM имя_таблицы;
```

где п - любое положительное число.

Параметр FIRST_ROWS остался для обратной совместимости и не рекомендуется в использовании.

В Oracle 10g устарели значения RULE, использующие оптимизацию на основе правил. Таким образом, Oracle 10g официально не поддерживает RBO (а неофициально работает).

Пути доступа к данным (access paths)

Путь доступа - это способ, с помощью которого данные считываются из БД. Любая строка данных может быть получена одним из следующих методов доступа.

Полное сканирование таблицы ***(Full Table Scans)***

В этом случае считываются все строки таблицы и отфильтровываются в соответствии с критериями запроса. Полное сканирование означает чтение всех блоков таблицы до отметки максимального заполнения (High-water mark).

Полное сканирование выгоднее сканирования по индексу, когда предполагается обращение к большему числу блоков таблицы. Стоимость подобной операции дешевле еще и потому, что она выполняется за несколько больших операций чтения, в то время как работа с индексом требует много маленьких операций чтения.

Оптимизатор использует данный способ доступа в следующих случаях.

- Отсутствует индекс. Нет индекса по полю, фигурирующему в предложении WHERE, или там используется функция по индексному столбцу, например

  Upper (last_name) =' ИВАНОВ'.

- Большое количество данных. Если оптимизатор считает, что запрос будет требовать обращения к большинству блоков таблицы, тогда применяется метод полного сканирования таблицы, даже если существует необходимый индекс.
- Маленькая таблица. Если таблица очень маленькая, тогда полное сканирование может быть дешевле, чем сканирование индекса по диапазону, независимо от того, какая часть таблицы требуется.

Принудительно полное сканирование таблицы можно задать подсказкой оптимизатору FULL (имя_таблицы), тогда даже при наличии индекса Oracle выполнит полное сканирование.

Сканирование по ROWID

Это наиболее быстрый способ доступа к любой строке БД. Первым шагом Oracle получает список необходимых ему ROWID в соответствии с критериями предложений WHERE, используя один или несколько индексов, затем обращается к таблице.

Доступ к таблице требуется в любом случае, если в запросе есть столбцы, которые не представлены в ключе индекса. Однако если есть несколько индексов, которые вместе охватывают все необходимые столбцы, то доступ по ROWID может и не потребоваться.

Сканирование по индексу

При использовании данного метода Oracle обращается к блокам индекса для проверки или поиска значений, необходимых для данного запроса. Если необходимо, формируется список ROWID для целевого обращения к строкам таблицы. Индекс может использоваться не только для получения списка необходимых строк, но и для исключения операции сортировки.

Принудительное сканирование по индексу можно задать подсказкой оптимизатору INDEX (имя_таблицы [имяиндекса]).

Уникальное сканирование по индексу (Index Unique Scans)

Данный метод возвращает максимально только одну строку. Oracle выполняет уникальное сканирование, если команда содержит условие *на равенство* по столбцам ограничения UNIQUE или PRIMARY КЕУ, которые гарантируют доступ только к одной строке.

Сканирование индекса по диапазону (Index Range Scans)

Это наиболее распространенный метод доступа по индексу. Он может быть ограничен (с обеих сторон) или быть не ограничен (с одной или обеих сторон). Данные возвращаются в возрастающем порядке значений ключа индекса. Несколько строк с одинаковыми значениями сортируются по возрастанию ROWID. Если в команде есть предложение ORDER ВУ, то, если возможно, применяется сканирование по индексу для исключения сортировки. Оптимизатор использует данный метод, когда находит один или более лидирующих столбцов индекса в условии запроса типа coll = :Ы, co12 < :Ы, coll > :Ы, а также по подстроке начальных символов индексного столбца (coll LIKE 'ABCD%'). Условие вида coll LIKE '%ABCD' исключает применение сканирования по индексу.

При использовании переменных привязки оптимизатор не знает ее значения и может выбрать полное сканирование таблицы. Для исключения подобной ситуации вы можете использовать:

- литералы, однако это исключит разделение кода;
- подсказку (hint) 1Ж)ЕХ(имя_таблицы имя_индекса) для разделения курсора.

Сканирование индекса по диапазону по убыванию(1nйex Range Scans Descending)

Это то же самое, что и предыдущий метод, за исключением того, что данные возвращаются в порядке убывания. Применяется для условий типа coll < :bl или при наличии сортировки соответствующего вида. Можно применить подсказку 1Ж>EX_BE8C(имя_таблицы имя_индекса).

Поиск в индексе с пропусками (Index Skip Scans)

Метод появился в 9i и позволяет использовать составной индекс в случае, когда условие накладывается не на ведущий столбец индекса.

Полное сканирование индекса (Full Index Scans)

Данный метод может быть применен, если в условии есть ссылка на столбец индекса. Данный столбец не обязательно должен быть первым в ключе индекса. Также этот вариант рассматривается, когда выполнены следующие условия:

- все поля, необходимые запросу, есть в индексе;
- хотя бы одно поле индекса является не пустым

Full index scan применяется также для исключения операции сортировки. Данный метод использует одиночные операции чтения (событие db_file_sequential_read).

Быстрое полное сканирование по индексу (Fast Full Index Scans)

Используется как альтернатива предыдущего метода, когда индекс содержит все поля, необходимые запросу, и хотя бы одно поле индекса имеет ограничение not null. Данный метод подразумевает работу *только* с данными индекса. Этот метод не может быть использован для исключения операции сортировки, поскольку данные считываются не в отсортированном порядке, т. к. применяется механизм *многоблочного* чтения. Этот метод может быть выбран при использовании подсказки INDEXFFS. FFS не применяется к битовым индексам. Метод быстрее, чем обычное сканирование, из-за операций многоблочного чтения и возможности распараллелить их.

Объединение индексов (Index Joins)

Данный режим подразумевает хеш-объединение нескольких индексов, которые вместе содержат все поля, необходимые запросу. Таким образом исключается необходимость обращения **к** данным таблицы. Index join не может быть использован для исключения операции сортировки. Можно применить подсказку ШВЕХ_ЮЩимя_таблицы [имя_индекса]).

Объединение на основе битовых карт (Bitmap Joins)

Не следует путать с объединением битовых индексов. Данная операция означает, что для обычных b-tree индексов создается битовая карта отображения бита в ROWID, а затем проводится операция над битовыми картами (AND или OR). Метод эффективен при слиянии индексов для различных предикатов предложения WHERE.

Сканирование кластера (Cluster Scans)

Метод применяется при считывании данных из индекс-кластера. В индексном кластере все строки с одинаковым значением ключа хранятся в одном и том же блоке данных. При Cluster Scans Oracle получает ROWID первой строки соответствующего ключа. Затем обращается в нужный блок данных для считывания всех остальных значений.

Сканирование по хешу (Hash Scans)

Метод связан с таблицами, размещенными в хеш-кластере. В нем все строки с одинаковым значением хеша хранятся в одном блоке. Сервер применяет хеш-функцию к кластерному ключу и вычисляет адрес нужного блока.

Сканирование образца таблицы (Sample Table Scans)

Метод считывает случайный образец данных из таблицы. Применяется при обнаружении в предложении FROM параметра SAMPLE или SAMPLE BLOCK. В первом случае сервер определяет процент строк, которые попадают в образец, во втором случае - процент блоков БД.

Метод не применяется, если запрос является частью операции объединения или связан с удаленной таблицей. Альтернативой может быть создание таблицы командой CREATE TABLE AS SELECT и ссылкой на нее в запросе.

Вопросы

1. Назовите известные вам способы доступа к данным.
2. В чем особенности (плюсы и минусы) доступа к данным по индексу? Полного просмотра таблицы? По hash? По hash-индексу?
3. Назовите правила разбора SQL-выражений. Что такое разделяемые курсоры?
4. Что такое план выполнения? Что с его помощью можно проанализировать?
5. Продемонстрировать примеры работы подсказок с планом выполнения до и после проведения оптимизации. Привести примеры,

когда в вашей базе использование подсказок помогает и когда мешает.

6. Привести примеры с разделяемыми курсорами.
7. Назовите методы оптимизации работы.
8. Что такое подсказки оптимизатору?
9. Поясните синтаксис подсказок, расскажите подробно о каждом из видов подсказок.

Задания

1. Продемонстрировать использование и анализ плана выполнения различных запросов при помощи команд EXPLAIN PLAN и AUTOTRACE (не менее 3 различных планов). Включить AUTOTRACE только на автотрассировку. Включить сбор статистики в AUTOTRACE. Продемонстрировать различие планов выполнения при различных режимах работы оптимизатора:
   а) Choose;
   б) Rule;
   в) Alljrows;
   г) First_rows_n.
2. Привести примеры запросов, которым соответствуют планы выполнения с:
   а) простым просмотром (full table scan);
   б) просмотро'м по индексу;
   в) уникальным сканированием по индексу;
   г) сканированием по индексу по диапазону;
   д) сканированием по индексу по убыванию;
   е) быстрым сканированием по индексу;
   ж) полным сканированием по индексу;
   з) с объединением индексов;
   и) сканированием по ROWID.

При необходимости использовать подсказки оптимизатору.

Лабораторная работа №12
Курсоры

Ключевым понятием языка PL/SQL является курсор. Каждое SQL-утверждение, выполняемое сервером, имеет индивидуальный *курсор,* связанный с:

- неявным курсором, объявленным для всех DML-утверждений;
- явным курсором, объявленным и поименованным программистом.

Явный курсор - это поименованный запрос, содержащий некоторое фиксированное число строк в выборке. Чаще всего курсор содержит данные одной строки выбираемой таблицы. По существу, курсор является окном, через которое пользователь получает доступ к информации баз данных. Курсоры, в частности, могут использоваться для присваивания конкретных значений переменным программы.

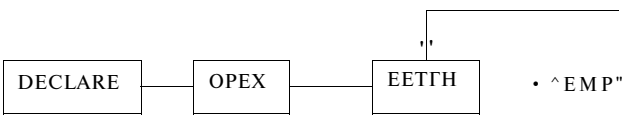

Рис. 6. Синтаксическая диаграмма работы явного курсора

Здесь:	DECLARE	создает именованную SQL область;
	OPEN	определяет активное множество;
	FETCH	передает текущий ряд в переменную;
	EMPTY	проверка на существование ряда, возврат к FETCH, если ряды существуют;
	CLOSE	реализовать активное множество.

Явный курсор

Объявление явного курсора происходит при помощи выражения

CURSOR *имя_курсора* IS *выражение_выборки;*

Здесь:	*имя_курсора*	имя, присваиваемое явному курсору;
	выражение выборки	выражение, содержащее команду SELECT.

Указания

- Нельзя включать INTO в предложение курсора.
- Если требуется использовать последовательность, то в предложении курсора необходимо применить ORDER BY.

Пример

```
DECLARE
CURSOR real_curs IS
SELECT * FROM realty
/
```

Для того чтобы далее работать с курсором, его необходимо открыть командой OPEN.

Синтаксис:

OPEN *имя_курсора;*

Выборка данных из курсора может быть выполнена в набор переменных подходящих типов командой FETCH.

Синтаксис:

FETCH *имя_курсора* INTO *[переменная!, переменная!...]|*
\имя_записи
I

Здесь: *имя_курсора* имя описанного ранее курсора;

переменная выходная переменная для размещения результата;

имя записи имя ранее определенной записи.

Указания

. В предложение INTO команды FETCH следует включать столько же переменных, сколько столбцов возвращает команда SELECT. Необходимо проверить совместимость типов данных.
. Между именами переменных и столбцами устанавливается позиционное соответствие.

- Другой способ - это определить запись курсора (RECORD) и ссылаться на нее в предложении INTO.
- Команда FETCH позволяет проверить, содержит ли курсор строки. Если команда FETCH не выбирает никаких значений, значит, в активном наборе не осталось необработанных строк.

По окончании обработки команды SELECT курсор необходимо закрыть командой CLOSE.

Синтаксис:

CLOSE *имя_курсора;*

Здесь: *имя_курсора* имя ранее описанного курсора.

Для явных курсоров, как и для неявных, имеется 4 атрибута, с помощью которых можно получить информацию о курсоре.

Таблица 33. Атрибуты курсора

Атрибут	Описание
%ISOPEN	Логический атрибут, значение которого равно TRUE, если курсор открыт
%NOTFOUND	Логический атрибут, значение которого равно TRUE, если последняя команда FETCH не вернула строку
%FOUND	Логический атрибут, значение которого равно TRUE до тех пор, пока не окажется, что последняя команда FETCH не вернула строку. Это атрибут, обратный атрибуту %NOTFOUND
%ROWCOUNT	Числовой атрибут, возвращающий общее количество строк, выбранных на данный момент

Обычно для обработки нескольких строк из явного курсора создается цикл, при каждом выполнении которого выбирается одна строка. В итоге обрабатываются все строки активного набора. Прежде чем сослаться на курсор, проверяйте успех каждой выборки с помощью атрибутов данного курсора. Не забудьте указать критерий выхода из цикла!

Выборка строк возможна только при открытом курсоре. Проверить, открыт ли курсор, можно с помощью атрибута %ISOPEN.

Для выборки точного количества строк можно либо создать цикл FOR с числовым параметром, либо использовать простой цикл и определять момент выхода из цикла с помощью атрибута курсора %ROWCOUNT.

Пример

```
DECLARE
Argl Owner.Non%type;
Arg2 Owner.Ow%type;
Arg3 Owner.AdO%type;
CURSOR own curs IS SELECT * FROM owner;
BEGIN
OPEN own curs;
FOR I IN1..5 LOOP
        FETCH own curs INTO Argl, Arg2, Arg3;
        DBMS_OUTPUT.PUTJLINE ('Фамилия
владельца с номером ' ||
TO   CHAR(Argl)||':'||Arg2||
\Eгоадрес:'||Ar_E3||<.');
        END LOOP;
CLOSE owncurs;
END;
/
```

Теперь проиллюстрируем, как можно было обойтись без цикла FOR.

Пример

```
DECLARE
Argl Owner.Non%type;
Arg2 Owner.Ow%type;
Arg3 Owner.AdO%type;
CURSOR own_curs IS SELECT * FROM owner;
BEGIN
OPEN own_curs;
FETCH own_curs INTO Argl, Arg2, Arg3;
EXIT WHEN own_curs%NOTFOUND;
        DBMS_OUTPUT.PUT_LINE ('Фамилия
владельца с номером ' ||
TO_CHAR(Argl) || ': ' || Arg2 ||
'. Его адрес: ' || Arg3 || '.');
        END LOOP;
CLOSE own_curs;
END;
```

Примечание. Если вы используете атрибут %ROWCOUNT, добавьте проверку на отсутствие строк с помощью атрибута %NOTFOUND, потому что в случае, если команда FETCH не вернет строку, счетчик %ROWCOUNT не увеличивается.

Неявный курсор

Неявный курсор используется в том случае, если запрос возвращает только одну строку и вы желаете сохранить результат работы запроса в каких-либо переменных. Неявный курсор не именуется, для него не нужно выполнять открытие и закрытие курсора, не нужно указывать команду FETCH. В простейшем случае неявный курсор может быть представлен обычным запросом, результат работы которого записывается в переменные (в запросе присутствует INTO).

Пример

```
DECLARE
Argl Owner.Non%type;
Arg2 Owner.Ow%type;
Arg3 Owner.AdO%type;
BEGIN
SELECT Owner.NOn, Owner.Ow, Owner.AdO
INTO Argl, Arg2,Arg3
FROM Owner
WHERE Owner.NOn = 7;
END;
/
```

Задания

1.)Сделать выборку данных из таблицы Lease с использованием курсора и цикла с методом %NOTFOUND для получения данных о _ договорах, заключенных не позже 2005 г.

2./Сделать выборку данных из таблицы Lease с использованием курсора и цикла с методом %FOUND для получения данных об арендаторах, фамилия которых начинается с определенной буквы. Подсказка: используйте цикл WHILE.

3. Выбрать данные об арендаторах и владельцах объектов, которые заключили договор в определенную дату.

4. Поменяйте местами адреса арендаторов и владельцев недвижимости. Если арендаторов (или владельцев) меньше, чем владельцев (или арендаторов, соответственно), то «лишние» адреса должны оставаться на месте. Подсказка: используйте одновременно 2 курсора, можете еще посчитать, кого меньше.

S./Напишите неявный курсор, который будет находить самый большой номер договора и количество арендаторов, заключивших договоры.

Требования к сдаче лабораторной работы

1. Необходимо, чтобы были созданы курсоры, реализующие как метод %FOUND, так и %NOTFOUND.
2. Должен быть реализован неявный курсор.

Лабораторная работа №13
Подпрограммы. Пакеты

Подпрограмму- это поименованный блок PL/SQL, который принимает параметры и может быть вызван. PT7/SQL имеет два типа' подпрограмм, называемых ***процедурами ж функциями.*** Обычно процедуру вызывают для того, чтобы выполнить некоторое действие, а функцию - для того, чтобы вычислить некоторое значение.

Подпрограммы имеют декларативную часть, исполняемую часть и необязательную часть обработки исключений. Декларативная часть содержит объявления типов, курсоров, констант, переменных, исключений и вложенных подпрограмм. Все эти объекты локальны и перестают существовать после выхода из подпрограммы. Исполняемая часть содержит предложения, которые присваивают значения, управляют выполнением и манипулируют данными Oracle. Часть обработки исключений содержит обработчики, которые имеют дело с исключениями, возбуждаемыми при исполнении.

Подпрограммы можно определять в любом инструменте Oracle, который поддерживает PL/SQL. Их можно объявлять в блоках PL/SQL, процедурах, функциях и пакетах. Однако подпрограммы должны объявляться в конце декларативной секции, после всех других программных объектов.

Преимущества подпрограмм

- Расширяемость — позволяют вам приспосабливать средства PL/SQL для ваших потребностей.
- Модульность - позволяют вам разбивать ваши программы на управляемые, хорошо определенные логические модули. Это поддерживает методы проектирования сверху вниз и пошагового уточнения, характерные для структурного подхода к решению проблем.
- Способствуют используемости и сопровождаемости - однажды проверенную подпрограмму можно с уверенностью использовать в любом количестве приложений.
- Способствует абстрагированию - чтобы использовать подпрограммы, вам нужно знать, что они делают, а не как они это делают.

Процедуры

Процедура - это подпрограмма, которая выполняет специфическое действие. Хранимую процедуру можно создать командой CREATE PROCEDURE.

Синтаксис:

```
CREATE [OR REPLACE] PROCEDURE
имя [{параметр [, параметр,...])] IS
 [локальные объявления]
 BEGIN
 исполняемые предложения
 [EXCEPTION
 обработчики исключений]
 END [имя];
I
```

где каждый *«параметр»* имеет следующий синтаксис:

имя_перем [IN | OUT | IN OUT] *тип_данных [{:=* | DEFAULT} *знач]*

В отличие от спецификатора типа данных в объявлении переменной, спецификатор типа данных для параметра не может иметь ограничений. Например, следующее объявление name незаконно: PROCEDURE ... (name CHAR (20)) IS - незаконно; должно быть CHAR.

Процедура имеет две части: спецификацию и тело. Спецификация процедуры начинается с ключевого слова PROCEDURE и заканчивается именем процеду_ры^ или списком параметров. Объявления параметров необязательны. Если процедура не принимает параметров, скобки также не кодируются. Тело процедуры начинается с ключевого слова IS и заканчивается ключевым словом ыми.

Тело процедуры состоит из трех частей: декларативной части, исполняемой части и необязательной части обработки исключений. Декларативная часть содержит локальные объявления, которые помещаются между ключевыми словами IS и BEGIN. Ключевое слово DECLARE, которое начинает декларативную часть в анонимном блоке PL/SQL, здесь **не** используется. Исполняемая часть содержит предложения, которые помещаются между ключевыми словами BEGIN и EXCEPTION (или END). В исполняемой части процедуры должно быть хотя бы одно предложение. Часть обработки исключений содержит обработчики

исключений, которые помещаются между ключевыми словами EXCEPTION и END.

Пример

Процедура замены адреса владельца с заданным номером.

```
CREATE OR REPLACE PROCEDURE
CHANGE_ADR (NUM IN Owner.NOn%type,
ADDRESS  Owner.AdO%type) IS
BEGIN
UPDATE Owner SET AdO=ADDRESS
WHERE NOn=NUM;
END;
/
```

Функция

Функция - это подпрограмма, которая вычисляет значение. Функции структурируются *гтг^¥,"^1я^ШЩу^Гс^гШ~Уззтш^ч!¥€Г^уякциш* -еодержат фразу RETURN.

Синтаксис:

```
FUNCTION имя [ (аргумент  [, аргумент,...]) ]
RETURN тип_данн IS
[локальные объявления]
BEGIN
исполняемые  предложения
[EXCEPTION
обработчики  исключений]
END [имя];
I
```

где каждый *«аргумент»* имеет следующий синтаксис:

имя_перем [IN | OUT | IN OUT] *тип_данных* [{:= | DEFAULT} *знач]*

Примечание. Вызовы пользовательских функций могут появляться в процедурных предложениях, но НЕ в предложениях SQL.

Предложение REXLIRN.—немедленно завершает выполнение подпрограммы и В03jр^аеТуnраВneННе вызывающей программе^Выполнение продолжается с предложенияГ следующегТ~й"вшовом подпрограммы.

Подпрограмма может содержать несколько предложений RETURN в процедурах предложение RETURN не может содержать выражение.

Фактические и формальные параметры

Подпрограммы принимают и передают информацию через параметры. Переменные или выражения, которые специфицированы в списке параметров в вызове подпрограммы, называются фактическими параметрами. Переменные, объявленные в спецификации подпрограммы и используемые в теле подпрограммы, называются формальными параметрами. Фактический параметр и соответствующий ему формальный параметр должны иметь совместимые типы данных.

Позиционная и именная нотация

При вызове подпрограммы можно записывать фактические параметры, используя позиционную или именную нотацию. Иными словами, можно указывать соответствие между фактическими и формальными параметрами через позиции этих параметров или через их имена.

Пример

```
DECLARE
nmb  Owner.NOn%type;
addr  Owner.AdO%type;
PROCEDURE CHANGE_ADR
(NUM IN Owner.NOn%type, ADDRESS
Owner AdO%type) IS ...
```

Процедуру credit теперь можно вызвать четырьмя логически эквивалентными способами:

```
BEGIN
CHANGE_ADR (nmb, addr); - позиционная нотация
CHANGE_ADR (NUM => nmb, ADDRESS -=> addr); -
именная нотация
CHANGE_ADR (ADDRESS => addr, NUM => nmb); -
именная нотация
CHANGE_ADR (nmb, ADDRESS^ addr);
смешанная нотация

 END;
/
```

Первый вызов процедуры использует позиционную нотацию. Компилятор PL/SQL ассоциирует первый фактический параметр, nmb, с первым формальным параметром, NUM, а второй фактический параметр, addr, - со вторым формальным параметром, ADDRESS.

Второй вызов процедуры использует именную нотацию. Стрелка ассоциирует формальный параметр слева от стрелки с фактическим параметром справа от стрелки.

Третий вызов процедуры также использует именную нотацию и показывает, что вы можете задавать пары параметров в любом порядке. Поэтому вы не обязаны знать порядок, в котором перечислены формальные параметры.

Четвертый вызов процедуры показывает, что вы можете смешивать позиционную и именную нотации. В данном случае первый параметр задан в позиционной, а второй - в именной нотации. Позиционная нотация в этом варианте должна предшествовать именной. Обратное не допускается. Например, следующий вызов процедуры незаконен: CHANGE_ADR (ADDRESS => addr, nmb); - незаконно.

Моды параметров

Моды параметров используются, чтобы определить поведение формальных параметров подпрограммы. Все три возможные моды: IN (по умолчанию), OUT и IN OUT - могут использоваться в любой процедуре. Что касается функций, то избегайте использования моды OUT или IN OUT в функциях.

Параметр с модой IN передает значение вызываемой подпрограмме. Внутри подпрограммы такой параметр выступает как константа, поэтому ему нельзя присвоить значение.

Параметр с модой OUT позволяет возвращать значение вызывающей программе. Внутри подпрограммы такой параметр выступает как неинициализированная переменная, поэтому его значение нельзя присваивать другим переменным или переприсвоить самому себе. Фактический параметр, соответствующий формальному параметру с модой OUT, должен быть переменной; он не может быть константой или выражением.

Параметр IN OUT позволяет вам передавать в подпрограмму начальные значения и возвращать обновленные значения вызывающей программе. Внутри подпрограммы такой параметр выступает как инициализированная переменная. Поэтому ему можно присвоить значение, а его

значение можно присваивать другим переменным. Иными словами, параметр IN OUT можно рассматривать как обычную переменную.

Перекрытие имен

PL/SQL позволяет перекрывать имена подпрограмм. Иными словами, вы можете использовать одно и то же имя для нескольких различных подпрограмм, если только их формальные параметры различаются по количеству, порядку или семействам типов данных.

Перекрывающиеся имена подпрограмм могут появляться только в блоке, подпрограмме или пакете. Нельзя перекрывать имена независимых подпрограмм. Нельзя перекрывать две подпрограммы, если их формальные параметры различаются лишь именами или модами параметров. Более того, нельзя перекрывать две подпрограммы, если их формальные параметры различаются лишь типами данных, причем эти типы данных относятся к одному и тому же семейству. Наконец, нельзя перекрывать две функции, если они различаются лишь типами данных результирующего значения, даже если эти типы данных относятся к разным семействам.

Вызов хранимых подпрограмм

Хранимые подпрограммы можно вызывать из триггера базы данных, другой хранимой подпрограммы, приложения прекомпилятора Oracle, приложения OCI или из инструмента Oracle, такого как SQL*Plus. Хранимая подпрограмма может вызывать другую хранимую подпрограмму. Например, в теле пакетированной подпрограммы может появиться вызов независимой процедуры. Приложение прекомпилятора или приложение OCI может вызывать хранимые подпрограммы из анонимных блоков PL/SQL.

Пример

```
EXEC SQL EXECUTE
BEGIN
CHANGE ADR(3, 'Самарская, 13-22');
END;
END-EXEC;
```

Вы можете вызывать хранимые подпрограммы интерактивно из инструментов Oracle, таких как SQL*Plus, SQL*Forms или SQL*DBA.

Пример

```
SQL> EXECUTE CHANGE_ADR
(3, 'Самарская, 13-22');
```

Этот вызов эквивалентен следующему анонимному блоку PL/SQL:

```
SQL> BEGIN CHANGE_ADR (3, 'Самарская, 13-22');
END;
```

*Примечание. Тело независимой или пакетированной хранимой подпрограммы может содержать любое предложение SQL или PL/SQL. Однако подпрограммы, участвующие в распределенной транзакции, триггерах базы данных и приложениях SQL*Forms, не могут вызывать хранимых подпрограмм, содержащих предложения. COMMIT, ROLLBACK или SAVEPOINT*

Обращения к хранимым функциям могут появляться в процедурных предложениях, но не в предложениях SQL.

Пакет

Пакет - это объект базы данных, который грущщзуех^логически связанные типы, программные объекты й~ТТодпрограмм,ьТ РТ./SOT.. Пакеты "предлагают несколько преимуществ: модульность, облегчение проектирования приложений, скрытие информации, расширенную функциональность и лучшую производительность.

Пакеты обычно сосюяииз двух частей (спецификации и тела), хотя, иногдаТтеле нет необходимости. Спецификация пакета - это интерфейс с вшшГИЕГТприложениями; она объявляет типы, переменные, константы, исключения, курсоры и подпрограммы, доступные для использования в пакете. Тело пакета полностью определяет курсоры и подпрограммы, тем самым реализуя спецификацию пакета.

В отличие от подпрограмм, пакеты нельзя вызывать, передавать им параметры илТГШлЖь1в¥ть~йх^друг"в-другаГВ остальном формат пакета аналогичен формату подпрограммы.

Синтаксис:

CREATE [OR REPLACE] PACKAGE *имя_пакета* IS
спецификация [объявления общих типов и объектов;
спецификации подпрограмм]
END *[имя_пакета]*\

CREATE [OR REPLACE] PACKAGE BODY *имя_пакета* IS
[объявления личных типов и объектов;
тела подпрограмм;]
[BEGIN
предложения инициализации]
END *[имя_пакета]*\

Спецификация содержит общие объявления, которые видимы вашему приложению. Тело - детали реализации и закрытые объявления, которые скрыты от вашего приложения. Тело пакета реализует спецификацию пакета. Оно включает определения всех курсоров и подпрограмм, объявленных в спецификации пакета. Тело пакета может также содержать закрытые объявления, которые определяют типы и объекты, необходимые для внутренней работы пакета. Сфера таких объявлений локальна в теле пакета. Поэтому объявленные здесь типы и объекты недоступны нигде, кроме тела пакета.

Для обращения к типам, объектам и подпрограммам, объявленным в спецификации пакета, используются квалифицированные ссылки:

имя_пакета.имя_типа
имя_пакета.имя_объекта
имя_пакета.имя_подпрограммы

Обращаться к содержимому пакета можно из триггеров базы данных, хранимых подпрограмм, встроенных блоков PL/SQL, а также анонимных блоков PL/SQL, посылаемых в Oracle интерактивно через SQL*Plus или SQL*DBA.

Вопросы

1. Чем отличается функция от процедуры?
2. Синтаксис функции. Синтаксис процедуры.
3. Какие существуют ограничения на вызовы пользовательских функций и процедур?
4. Фактические и формальные параметры.

5. Что такое позиционная и именная нотация? Смешанная нотация? Какие существуют ограничения при смешанной нотации?
6. Какие бывают моды параметров? В чем их особенности?
7. Значения параметров по умолчанию.
8. Ограничения на пространство имен подпрограмм.
9. В чем особенности и преимущества хранимых подпрограмм? Как синтаксически задать хранимую подпрограмму?
10. Что такое пакет? Что может находиться в спецификации пакета? В теле пакета?
11. Как вызвать пакетированную функцию?

Задания

1. Напишите процедуру, которая выводит на печать количество заключенных договоров для каждого арендатора недвижимости.
2. Напишите функцию, которая возвращает наименование самого дорогого объекта аренды. Предусмотрите обработку исключительной ситуации, когда таких объектов более одного, то функция должна вернуть наименование первого из них и выдать сообщение об ошибке.
3. Напишите процедуру, которая удаляет из базы арендаторов, не заключивших ни одного договора. Перед тем как произвести удаление записей, процедура должна напечатать фамилии этих арендаторов и сообщение об удалении.
4. Напишите процедуру, которая определяет список недвижимости и ее владельцев (с адресами), договоры на которые заключены ранее 01.01.2007 г. Вывод данных должен быть сгруппирован по типам недвижимости и оформлен в читаемом виде, например:

квартира 1 к

Марусина А.К., Самарская, 12-25
Иващенко П.Г., Кирова, 17-223

квартира 2к

Сидоренко В.К., Полевая, 158-92
Соломкина П. А., Галактионовская, 129-32
Сороконожкин В.Л., Чапаевская, 33-31

Подсказка: используйте 2 курсора, один из которых будет полностью функционировать внутри цикла второго курсора.

5. Напишите функцию, которая будет рассчитывать процент дохода фирмы. Входным параметром должен быть номер договора. Выходным - доход фирмы от заданного договора. Процент считаем по принципу: сумма договора*(0.05+коэффициент). Размер коэффициента устанавливается в зависимости от вида недвижимости, но не может быть более 0.05. Пример: дом - коэффициент 0.03, 2-комнатная квартира - коэффициент 0.01 и т. п. Все коэффициенты прописываются как локальные переменные или как локальные константы в функции.

6. Оформите пакет, в котором будут находиться вышеперечисленные подпрограммы.

Лабораторная работа Жя14
Индивидуальные задания

В индивидуальном задании вам необходимо спроектировать и создат базу (в которой будет не менее 5 таблиц), заполнить ее (требования - н менее 10 записей в каждой таблице, записи должны быть осмысленными^ Все функции и процедуры в базе должны быть оформлены в паке! триггеры должны быть рабочими. Необходимо продемонстрироват работу всех подпрограмм пакета, триггеров.

1. База данных хроники восхождений в альпинистском клубе

В базе данных должны записываться даты начала и завершения каждог восхождения, имена и адреса участвовавших в нем альпинистов, название высота горы, страна и район, где эта гора расположена. Дайте выразительны имена таблицам и полям, в которые могла бы заноситься указанна информация. Написать пакет, состоящий из процедур и функций, которы позволили бы выполнить следующие действия с базой данных:

- для каждой горы показать список групп, осуществлявши восхождение, в хронологическом порядке;
- предоставить возможность добавления новой вершины с указание названия вершины, высоты и страны местоположения;
- предоставить возможность изменения данных о вершине, если н нее не было восхождения;
- показать список альпинистов, осуществлявших восхождени в указанный интервал дат;
- предоставить возможность добавления нового альпиниста в соста указанной группы;
- показать информацию о количестве восхождений каждого альш ниста на каждую гору;
- показать список восхождений (групп), которые осуществлялис в указанный пользователем период времени;
- предоставить возможность добавления новой группы, указав < название, вершину, время начала восхождения;
- предоставить информацию о том, сколько альпинистов побывали к каждой горе.

Предусмотреть разработку триггеров, обеспечивающих каскадны изменения в связанных таблицах. Создать также триггеры w автоматического добавления значения первичного ключа в таблицы.

2. База данных медицинского кооператива

Базу данных использует для работы коллектив врачей. В таблицы должны быть занесены имя, пол, дата рождения и домашний адрес каждого их пациента. Всякий раз, когда врач осматривает больного, явившегося к нему на прием, или сам приходит к нему на дом, он записывает дату и место, где проводится осмотр, симптомы, диагноз и предписания больному, проставляет имя пациента, а также свое имя. Если врач прописывает больному какое-либо лекарство, в таблицу заносится название лекарства, способ его приема, словесное описание предполагаемого действия и возможных побочных эффектов. Создать пакет, состоящий из функций и процедур, позволяющих:

- по заданной дате определить количество вызовов в этот день;
- определить количество больных, заболевших данной болезнью и вывести их фамилии и адреса;
- по заданному лекарству определить его побочный эффект;
- по диагнозу вывести список назначенных лекарств;
- по заданной дате и фамилии врача вывести все адреса, которые он посетил в этот день;
- предоставить возможность добавления нового лекарства с описанием его свойств в БД.

Предусмотреть разработку триггеров, обеспечивающих каскадные изменения в связанных таблицах. Вставку числовых значений первичного ключа сделать при помощи соответствующих триггеров.

3. База данных Городской Думы

В базе хранятся имена, адреса, домашние и служебные телефоны всех членов Думы. В Думе работает порядка сорока комиссий, все участники которых являются членами Думы. Каждая комиссия имеет свой профиль, например, вопросы образования, проблемы, связанные с жильем, и так далее. Данные по каждой из комиссий включают: ее нынешний состав и председателя; прежних председателей и членов этой комиссии, участвовавших в ее работе за прошедшие 10 лет; даты включения и выхода из состава комиссии, избрания ее председателей. Члены Думы могут заседать в нескольких комиссиях. В базу заносятся время и место проведения каждого заседания комиссии с указанием депутатов **и** служащих Думы, которые участвуют в его организации. Создать пакет с процедурами и функциями, которые позволяют выполнять следующие действия:

- показать список комиссий, для каждой ее состав и ФИО председателя;
- предоставить возможность добавления нового члена комиссии;
- показать список членов муниципалитета, для каждого из них - список комиссий, в которых он участвовал и/или был председателем;
- предоставить возможность добавления новой комиссии с указанием ФИО председателя;
- для указанного интервала дат и конкретной комиссии выдать список ее членов с указанием количества пропущенных заседаний;
- предоставить возможность добавления нового заседания с указанием присутствующих;
- по каждой комиссии показать количество проведенных заседаний в указанный период времени.

Предусмотреть разработку триггеров, обеспечивающих каскадные изменения в связанных таблицах. Вставку числовых значений первичного ключа произвести при помощи соответствующих триггеров.

4. База данных рыболовной фирмы

Фирме принадлежит небольшая флотилия рыболовных катеров. Каждый катер имеет «паспорт», куда занесены его название, тип, водоизмещение и дата постройки. Фирма регистрирует каждый выход на лов, записывая название катера, имена и адреса членов команды с указанием их должностей (капитан, боцман и т. д.), даты выхода и возвращения, а также вес пойманной рыбы отдельно по сортам (например, трески). За время одного рейса катер может посетить несколько банок. Фиксируется дата прихода на каждую банку и дата отплытия, качество выловленной рыбы (отличное, хорошее, плохое). На борту улов не взвешивается. Написать запросы, осуществляющие следующие операции:

- для каждого катера вывести даты выхода в море с указанием улова;
- предоставить возможность добавления выхода катера в море с указанием команды;
- для указанного интервала дат вывести для каждого сорта рыбы список катеров с наибольшим уловом;
- для указанного интервала дат вывести список банок с указанием среднего улова за этот период;
- предоставить возможность добавления новой банки с указанием данных о ней;
- для заданной банки вывести список катеров, которые получили улов выше среднего;

- вывести список сортов рыбы и для каждого сорта список рейсов с указанием даты выхода и возвращения, количества улова;
- для выбранного пользователем рейса и банки добавить данные о сорте и количестве пойманной рыбы;
- предоставить возможность пользователю изменять характеристики выбранного катера;
- предоставить возможность добавления нового катера;
- для указанного сорта рыбы и банки вывести список рейсов с указанием количества пойманной рыбы.

Предусмотреть разработку триггеров, обеспечивающих каскадные изменения в связанных таблицах. Вставку числовых значений первичного ключа реализовать при помощи соответствующих триггеров.

5. База данных фирмы, проводящей аукционы

Фирма занимается продажей с аукциона антикварных изделий и произведений искусства. Владельцы вещей, выставляемых на проводимых фирмой аукционах, юридически являются продавцами. Лица, приобретающие эти вещи, именуются покупателями. Получив от продавцов партию предметов, фирма решает, на котором из аукционов выгоднее представить конкретный предмет. Перед проведением очередного аукциона каждой из выставляемых на нем вещей присваивается отдельный номер лота, играющий ту же роль, что и введенный ранее шифр товара. Две вещи, продаваемые на различных аукционах, могут иметь одинаковые номера лотов.

В книгах фирмы делается запись о каждом аукционе. Там отмечаются дата, место и время его проведения, а также специфика (например, выставляются картины, написанные маслом и не ранее 1900 г.). Заносятся также сведения о каждом продаваемом предмете: аукцион, на который он заявлен, номер лота, продавец, отправная цена и краткое словесное описание. Продавцу разрешается выставлять любое количество вещей, а покупатель имеет право приобретать любое количество вещей. Одно и то же лицо или фирма может выступать и как продавец, и как покупатель. После аукциона служащие фирмы, проводящей аукционы, записывают фактическую цену, уплаченную за проданный предмет, и фиксируют данные покупателя.

Создать пакет, состоящий из процедур и функций,' позволяющий осуществить следующие операции:

- для указанного интервала дат вывести список аукционов с указанием наименования, даты и места проведения;
- добавить на указанный пользователем аукцион на продажу предмет искусства с указанием начальной цены;
- вывести список аукционов с указанием суммарного дохода от продажи, отсортированных по доходу;
- для указанного интервала дат вывести список предметов, которые были проданы на аукционах в этот период времени;
- предоставить возможность добавления факта продажи на указанном аукционе заданного предмета;
- для указанного интервала дат вывести список продавцов с указанием общей суммы, полученной от продажи предметов в этот промежуток времени;
- вывести список покупателей, которые сделали приобретения в указанный интервал дат;
- предоставить возможность добавления записи о проводимом аукционе (место, время);
- для указанного места вывести список аукционов;
- для указанного интервала дат вывести список продавцов, которые принимали участие в аукционах, проводимых в этот период времени;
- предоставить возможность добавления и изменения информации о продавцах и покупателях;
- вывести список покупателей с указанием количества приобретенных предметов в указанный период времени.

Предусмотреть разработку триггеров, обеспечивающих каскадные изменения в связанных таблицах. Вставку числовых значений первичного ключа осуществить при помощи соответствующих триггеров.

6. База данных библиотеки

Разработать информационную систему обслуживания библиотеки, которая содержит следующую информацию: названия книг, ФИО авторов, наименования издательств, год издания, количество страниц, количество иллюстраций, стоимость, название филиала библиотеки или книгохранилища, в которых находится книга, количество имеющихся в библиотеке экземпляров конкретной книги, количество студентов, которым выдавалась конкретная книга, названия факультетов, в учебном процессе которых используется указанная книга. Необходимо составить пакет из процедур и функций, который позволяет:

- для указанного филиала посчитать количество экземпляров указанной книги, находящихся в нем;
- для указанной книги посчитать количество факультетов, на которых она используется в данном филиале, и вывести названия этих факультетов;
- предоставить возможность добавления и изменения информации о книгах в библиотеке;
- предоставить возможность добавления и изменения информации о филиалах;
- предусмотреть разработку триггеров/процедур, срабатывающих на пользовательские исключительные ситуации.

Предусмотреть разработку триггеров, обеспечивающих каскадные изменения в связанных таблицах. Вставку числовых значений первичного ключа реализовать при помощи соответствующих триггеров.

7. База данных по учету успеваемости студентов

База данных должна содержать данные о контингенте студентов (фамилия, имя, отчество, год поступления, форма обучения (дневная/вечерняя/заочная), номер или название группы); об учебном плане (название специальности, дисциплина, семестр, количество отводимых на дисциплину часов, форма отчетности (экзамен/зачет)); о журнале успеваемости студентов (год/семестр, студент, дисциплина, оценка). Разработать пакет, состоящий из процедур и функций, позволяющий:

- для указанной формы обучения посчитать количество студентов этой формы обучения;
- для указанной дисциплины получить количество часов и формы отчетности по этой дисциплине;
- предоставить возможность добавления и изменения информации о студентах, об учебных планах, о журнале успеваемости; при этом предусмотреть курсоры/триггеры, срабатывающие на некоторые пользовательские исключительные ситуации;
- предоставить возможность добавления и изменения информации о журнале успеваемости;
- для конкретного студента получить список всех оценок;
- для указанной группы и указанной дисциплины получить список всех неуспевающих и их количество.

Предусмотреть разработку триггеров, обеспечивающих каскадные изменения в связанных таблицах. Вставку числовых значений первичного ключа осуществить при помощи соответствующих триггеров.

8. База данных для учета аудиторного фонда университета

База данных должна содержать следующую информацию об аудиторном фонде университета. Наименование корпуса, в котором расположено помещение, номер комнаты, расположение комнаты в корпусе, ширина и длина комнаты в метрах, назначение и вид помещения, подразделение университета, за которым закреплено помещение. В базе данных также должна быть информация о высоте потолков в помещениях в зависимости от места расположения помещений в корпусе. Следует также учитывать, что структура подразделений университета имеет иерархический вид, когда одни подразделения входят в состав других (факультет, кафедра, лаборатория). Учесть такие характеристики комнаты, как количество мест, на которые она рассчитана, наличие доски, наличие розетки, наличие экрана и затемнения. Для машзалов учесть количество компьютеров и число стульев в классе, а также наличие доски.

Помимо SQL-запросов для создания таблиц базы данных, разработать пакет, состоящий из процедур и функций, позволяющий:

- рассчитать данные о площадях и объемах каждого помещения;
- для указанного корпуса получить количество факультетов, их названия и структуру;
- предоставить возможность добавления и изменения информации о корпусах в университете, при этом предусмотреть курсоры/триггеры, срабатывающие на некоторые пользовательские исключительные ситуации;
- предоставить возможность добавления и изменения информации о комнатах в корпусах университета, при этом предусмотреть курсоры/триггеры, срабатывающие на некоторые пользовательские исключительные ситуации;
- рассчитать на каждом факультете количество аудиторий, пригодных для проведения лекций, практик у доски, занятий с презентационным оборудованием и занятий с аудиооборудованием, лабораторий; вывести номера аудиторий, сгруппировав их по вышеперечисленным признакам и разбив их по корпусам.

Предусмотреть разработку триггеров, обеспечивающих каскадные изменения в связанных таблицах. Вставку числовых значений первичного ключа реализовать при помощи соответствующих триггеров.

9. База данных для регистрации происшествий

Необходимо создать базу данных для регистрации происшествий. База данных должна содержать данные для регистрации сообщений о происшествиях (регистрационный номер сообщения, дата регистрации, краткая фабула (тип происшествия); информацию о принятом по происшествию решении (отказано в возбуждении дел, удовлетворено ходатайство о возбуждении уголовного дела с указанием регистрационного номера заведенного дела, отправлено по территориальному признаку); о районе (месте) происшествия; информацию о лицах, виновных или подозреваемых в совершении происшествия (регистрационный номер лица, фамилия, имя, отчество, адрес, количество судимостей...), отношение конкретных лиц к конкретным происшествиям (виновник, потерпевший, подозреваемый, свидетель...). Помимо SQL-запросов для создания таблиц базы данных, разработать пакет, состоящий из процедур и функций, позволяющий:

- рассчитать данные о количестве происшествий в указанный промежуток времени;
- для указанного района вывести список всех происшествий;
- вывести список дат происшествий по конкретному району и виду происшествия;
- определить самый «криминальный» и самый «спокойный» районы;
- для указанного лица получить количество происшествий, в которых он зарегистрирован;
- предоставить возможность добавления и изменения информации о происшествиях, при этом ^предусмотреть курсоры/триггеры, срабатывающие на некоторые пользовательские исключительные ситуации;
- предоставить возможность добавления и изменения информации о лицах, участвующих в происшествиях, при этом предусмотреть курсоры/триггеры, срабатывающие на некоторые пользовательские исключительные ситуации.

Предусмотреть разработку триггеров, обеспечивающих каскадные изменения в связанных таблицах. Вставку числовых значений первичного ключа осуществить при помощи соответствующих триггеров.

10. База данных для обслуживания работы конференции

База данных должна содержать справочник персоналий участников конференции (фамилия, имя, отчество, ученая степень, ученое звание, научное направление, место работы, кафедра (отдел), должность, страна, город, почтовый индекс, адрес, рабочий телефон, домашний телефон, e-mail) и информацию, связанную с участием в конференции (докладчик или участник, дата рассылки 1-го приглашения, дата поступления заявки, тема доклада, отметка о поступлении тезисов, дата рассылки 2-го приглашения, дата поступления оргвзноса, размер поступившего оргвзноса, дата приезда, дата отъезда, потребность в гостинице). Помимо SQL-запросов для создания таблиц базы данных, разработать пакет, состоящий из процедур и функций, позволяющий:

- для указанной даты 1-й рассылки вывести список приглашенных и посчитать их количество;
- предоставить возможность добавления приглашенных на конференцию с указанием оргвзноса и даты его уплаты;
- вывести список приглашенных с указанием даты об уплате оргвзноса;
- для указанного интервала дат вывести список участников, уплативших оргвзнос в этом диапазоне;
- для указанного города вывести список участников с названиями докладов, поступивших из этого города;
- для указанного города вывести названия всех вузов, из которых приедут участники;
- провести статистический анализ - сколько студентов, аспирантов, профессоров приезжают на конференцию (сгруппировать по каждому городу) в процентном соотношении;
- вывести сумму полученного оргвзноса и сумму ожидаемого оргвзноса;
- для указанного города вывести список нуждающихся в гостинице.

Предусмотреть разработку триггеров, обеспечивающих каскадные изменения в связанных таблицах. Вставку числовых значений первичного ключа реализовать при помощи соответствующих триггеров.

11. База данных для обслуживания склада

База данных должна обеспечить автоматизацию складского учета. В ней должны содержаться следующие данные:

- информация о «единицах хранения» - номер ордера, дата, код поставщика, балансный счет, код сопроводительного документа по справочнику документов, номер сопроводительного документа, код материала по справочнику материалов, счет материала, код единицы измерения, количество пришедшего материала, цена единицы измерения;
- информация о хранящихся на складе материалах (справочник материалов - код класса материала, код группы материала, наименование материала);
- информация о единицах измерения конкретных видов материалов - код материала, единица измерения (метры, килограммы, литры и т. д.);
- информация о поставщиках материалов - код поставщика, его наименование, ИНН, юридический адрес (индекс, город, улица, дом), адрес банка (индекс, город, улица, дом), номер банковского счета.

Помимо SQL-запросов для создания таблиц базы данных, разработать пакет, состоящий из процедур и функций, позволяющий:

- посчитать количество поставщиков данного материала;
- предоставить возможность добавления единицы хранения с указанием всех реквизитов;
- вывести список поставщиков с указанием всех реквизитов данного материала на склад;
- вывести сведения о самых крупных поставщиках по данному виду материала;
- подсчитать общее количество данного материала по всем поставщикам;
- вывести информацию о материалах, количество которых минимально (меньше заданного), и о поставщиках, у которых можно его заказать;
- для указанного адреса банка посчитать количество поставщиков склада, пользующихся услугами этого банка;
- по заданному поставщику определить его банковские реквизиты.

Предусмотреть разработку триггеров, обеспечивающих каскадные изменения в связанных таблицах. Вставку числовых значений первичного ключа осуществить при помощи соответствующих триггеров.

12. База **данных фирмы**

Фирма отказалась от приобретения у своих поставщиков некоторых товаров, решив самостоятельно наладить их производство. С этой целью она организовала сеть специализированных цехов, каждый из которых принимает определенное участие в технологическом процессе. Каждому виду выпускаемой продукции присваивается, как обычно, свой шифр товара, под которым он значится в файле товарных запасов. Этот же номер служит и шифром продукта. В записи с этим шифром указывается, когда была изготовлена последняя партия этого продукта, какова ее стоимость, сколько операций потребовалось.

Операцией считается законченная часть процесса производства, которая целиком выполняется силами одного цеха в соответствии с техническими требованиями, перечисленными на отдельном чертеже. Для каждого продукта и для каждой операции в базе данных фирмы заведена запись, содержащая описание операции, ее среднюю продолжительность и номер чертежа, по которому можно отыскать требуемый чертеж. Кроме того, указывается номер цеха, обычно производящего данную операцию.

В запись, связанную с конкретной операцией, заносится требуемое количество расходуемых материалов, а также присвоенные им шифры товара. Расходуемыми называют такие материалы, как, например, электрический кабель, который нельзя использовать повторно. Когда, готовясь к выполнению операции, расходуемый материал забирают со склада, регистрируется фактически выданное количество соответствующий шифр товара; номер служащего, ответственного за выдачу; дата и время выдачи; номер операции и номер наряда на проведение работ, который будет обсуждаться ниже. Реально затраченное количество материала может не совпадать с расчетным, из-за того, например, что часть изготовленной продукции бракуется.

Каждый из цехов располагает многочисленными инструментами и приспособлениями. При выполнении некоторых операций их все же не хватает, и цех вынужден обращаться в центральную инструментальную за недостающими. Каждый тип инструмента снабжен отдельным номером, и на него заведена запись со словесным описанием. Кроме того, там отмечено, какое количество инструментов этого типа выделено цехам и какое осталось в инструментальной. Экземпляры инструмента конкретного типа, например, гаечные ключи одного размера, различаются по своим индивидуальным номерам. На фирме для каждого типа инструмента имеется запись, содержащая перечень всех индивидуальных номеров. Кроме того, указаны даты их поступления на склад. По каждой операции в

фирме отмечают типы и количество инструментов этих типов, которые должны использоваться при ее выполнении. Когда инструменты действительно берутся со склада, фиксируется индивидуальный номер каждого экземпляра, указываются номер заказавшего их цеха и номер наряда на проведение работ. И в этом случае затребованное количество не всегда совпадает с заказанным.

Наряд на проведение работ по форме напоминает заказ на приобретение товаров, но, в отличие от последнего, он направляется не поставщику, а в один из цехов. Оформляется этот наряд после того, как руководство фирмы сочтет необходимым выпустить партию некоторого продукта. В наряд заносятся шифр продукта, дата оформления наряда, срок, к которому должен быть выполнен заказ, а также требуемое количество продукта. Разработайте структуру таблиц базы данных, подберите имена таблиц и полей, в которых могла бы разместиться вся эта информация.

Помимо SQL-запросов для создания таблиц базы данных, разработать пакет, состоящий из процедур и функций, позволяющий:

- для выбранного цеха выдать список операций, выполняемых им; для каждой операции - список расходуемых материалов с указанием количества;
- показать список инструментов и предоставить возможность добавления нового;
- выдать список используемых инструментов;
- для указанного интервала дат вывести список нарядов;
- показать список операций и предоставить возможность добавления новой операции;
- выдать список расходуемых материалов, используемых в различных нарядах;
- выдать список товаров с указанием используемых инструментов;
- показать список нарядов и предоставить возможность добавления нового;
- выдать отчет о производстве товаров различными цехами, указав наименование цеха, название товара и его количество.

Предусмотреть разработку триггеров, обеспечивающих каскадные изменения в связанных таблицах. Вставку числовых значений первичного ключа сделать при помощи соответствующих триггеров.

13. База данных музыкального магазина

Таблицы базы данных содержат информацию о музыкантах, музыкальных произведениях и обстоятельствах их исполнения. Нескольких музыкантов, образующих единый коллектив, называют ансамблем. Это может быть классический оркестр, джазовая группа, квартет, квинтет и т.д. К музыкантам причисляют исполнителей (играющих на одном или нескольких инструментах), композиторов, дирижеров и руководителей ансамблей.

Кроме того, в базе данных хранится информация о пластинках, которыми магазин торгует. Каждая пластинка, а точнее, ее наклейка, идентифицируется отдельным номером, так что всем копиям, отпечатанным с матрицы в разное время, присвоены одинаковые номера. На пластинке может быть записано несколько исполнений одного и того же произведения - для каждого из них в базе заведена отдельная запись. Когда выходит новая пластинка, регистрируется название выпустившей ее компании (например, EMI), а также адрес оптовой фирмы, у которой магазин может приобрести эту пластинку. Не исключено, что компания-производитель занимается и оптовой продажей своих пластинок. Магазин фиксирует текущие оптовые и розничные цены на каждую пластинку, дату ее выпуска, количество экземпляров, проданных за прошлый год и в нынешнем году, а также число еще не распроданных пластинок.

Помимо SQL-запросов для создания таблиц базы данных, разработать пакет, состоящий из процедур и функций, позволяющий:

- определить количество музыкальных произведений заданного ансамбля;
- вывести название всех пластинок заданного ансамбля;
- вывести всех исполнителей заданной композиции;
- вывести все новинки (за последние 2 недели);
- вывести аутсайдеров продаж;
- показать лидеров продаж текущего года, то есть названия пластинок, которые чаще всего покупали в текущем году;
- предусмотреть изменения данных о пластинках и ввод новых данных;
- предусмотреть ввод новых данных об ансамблях.

Предусмотреть разработку триггеров, обеспечивающих каскадные изменения в связанных таблицах. Вставку числовых значений первичного ключа осуществить при помощи соответствующих триггеров.

14. База данных соревнований по биатлону

В базе данных должны записываться даты начала и завершения этапов, место проведения этапа, виды гонок в данном этапе, страны, выставившие команды на этап, фамилии и имена спортсменов, время каждой гонки для каждого спортсмена и количество промахов на каждой гонке для каждого спортсмена, а также количество очков, набранных спортсменом в гонке (по принципу: 1-е место = 50 очков, 2-е = 45, 3-е = 42, 4-е = 40 … 30-е = 1 очко). На одном этапе не может быть более 3 гонок, Гонки идут с интервалом в 1-2 дня. Написать пакет, состоящий из процедур и функций, выполняющих следующее:

- для каждого этапа соревнований показать призеров (это первые б мест в биатлоне);
- предоставить возможность добавления нового этапа (с указанием места проведения, сроков проведения и типов гонок. На каждом этапе не более 3 гонок, сроки проведения - 5 дней);
- показать список спортсменов, участвующих в соревнованиях в заданный период времени и в заданной гонке;
- предоставить возможность добавления новой страны-участницы;
- предоставить возможность замены спортсмена и снятия спортсмена с соревнования;
- показать информацию о количестве набранных очков каждым спортсменом в рейтинговом порядке;
- показать список этапов (с видами гонок), в которых участвовал данный биатлонист, и его результаты по каждой гонке. Вывести также его положение в общем зачете;
- вывести информацию о первой тройке стран, чьи команды лидируют в общем зачете (считается по количеству золотых медалей, затем серебряных и бронзовых).

Предусмотреть разработку триггеров, обеспечивающих каскадные изменения в связанных таблицах. Вставку числовых значений первичного ключа реализовать при помощи соответствующих триггеров.

15. База данных чемпионата по футболу

В базе должны храниться названия команд, их составы, фамилии тренеров команд, даты проведения и место проведения игр, счет, количество забитых и пропущенных мячей в каждой игре для каждой команды, очки, полученные командой в каждой игре, нарушения и карточки для игроков.

Написать пакет, состоящий из процедур и функций, выполняющих следующее:

- показать даты соревнований и участвующие в них команды, счет игры;
- показать список игроков команды и их тренера;
- показать общее количество пропущенных и забитых мячей для каждой команды;
- выявить команду-лидера и команду-аутсайдера;
- составить сводную таблицу по матчам каждого круга;
- предусмотреть возможность замены игрока;
- для заданного игрока определить количество матчей, проведенных на поле;
- для заданного игрока определить количество нарушений;
- выявить лучшего форварда и вратаря;
- определить победителей.

Предусмотреть разработку триггеров, обеспечивающих каскадные изменения в связанных таблицах. Вставку числовых значений первичного ключа реализовать при помощи соответствующих триггеров.

16. База данных системного администратора

В базе данных должны храниться сведения о группах пользователей: годе их поступления и годе их окончания; фамилии, имена и отчества пользователей; их логины и пароли; дата последней смены пароля; количество и вид нарушений правил работы в сети; сведения об отключении пользователя от сети; список ресурсов, к которым открыт доступ определенной группы пользователей.

Написать пакет, состоящий из процедур и функций, выполняющих следующее:

- позволить добавлять и удалять группы пользователей;
- позволить добавлять и удалять пользователей;
- позволить восстановление пароля для пользователя;
- разрешить смену пароля пользователю;
- добавлять и изменять ресурсы, к которым есть доступ у групп пользователей;
- вывести всех нарушителей. Если количество нарушений равно 3, то отключить пользователя от сети;
- вывести дату последней смены пароля. Если пароль меняли ранее, чем полгода назад, отключить пользователя от сети;

- предусмотреть возможность изменения фамилии пользователя при сохранении логина.

Предусмотреть разработку триггеров, обеспечивающих каскадные изменения в связанных таблицах. Вставку числовых значений первичного ключа осуществить при помощи соответствующих триггеров.

17. База данных оператора сотовой связи

База должна хранить данные о датах операций, фамилии, имена и отчества подключившихся в этот день, их адреса и паспортные данные, номера SM-карт, тарифный план, баланс, пополнение счетов ранее подключившихся клиентов (дату и сумму).

Написать пакет, состоящий из процедур и функций, которые должны:

- определить сумму выручки в заданный период времени;
- определить количество подключившихся на каждый тарифный план;
- определить самый популярный и самый непопулярный тарифные планы;
- определить баланс для каждого клиента. Предусмотреть возможность изменения баланса;
- блокировать номера SIM-карт для клиентов с нулевым и отрицательным балансом;
- предусмотреть' возможность добавления нового тарифного плана и перевод на него некоторых клиентов.

Предусмотреть разработку триггеров, обеспечивающих каскадные изменения в связанных таблицах. Вставку числовых значений первичного ключа реализовать при помощи соответствующих триггеров

Список литературы

1. Чекалов А. Базы данных : от проектирования до разработки приложений. СПб.: БХВ-Петербург, 2003. 384 с.
2. Дейт К. Дж. Введение в системы баз данных ; пер. с англ. 7-е изд. М. : Издательский дом «Вильяме», 2001. 1072 с.: ил.
3. Кайт Т. Oracle для профессионалов ; пер. с англ. СПб. : ООО «ДиаСофтЮП», 2003. 672 с.
4. Генник Дж. Oracle SQL*Plus : карманный справочник. 2-е изд. СПб. : Питер, 2004. 189 с. : ил.
5. Oracle PL/SQL для профессионалов / С. Фейерштейн, Б. Прибыл. 3-е изд. СПб.: Питер, 2004. 941 с.: ил.
6. Миллсап К., Хольт Д. Oracle. Оптимизация производительности ; пер. с англ. СПб. : Символ-Плюс, 2006. 464 с.: ил.

Содержание