

МИНИСТЕРСТВО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
КЕМЕРОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
КАФЕДРА ЮНЕСКО ПО НОВЫМ ИНФОРМАЦИОННЫМ
ТЕХНОЛОГИЯМ

А.М. Гудов, Л.Е. Шмакова

**ВВЕДЕНИЕ В ЯЗЫК СТРУКТУРИРОВАННЫХ
ЗАПРОСОВ SQL**

Учебное пособие

Кемерово 2001

ББК 3973.2-018я73

УДК 681.3

Г93

Печатается по решению редакционно-издательского совета Кемеровского государственного университета.

Рецензенты:

доктор физико-математических наук, профессор
кандидат технических наук, доцент

**В.П. Житников,
Г.Д. Буялич**

Гудов А.М., Шмакова Л.Е.

Г-93 Введение в язык структурированных запросов SQL / Учебное пособие. – Кемерово, Кемеровский госуниверситет, 2001.- 118с.

ISBN 5-8353-0065-4

В пособии рассмотрены вопросы использования языка запросов к реляционным базам данных SQL в редакции стандарта 1992 года. Особое внимание уделено формированию практических навыков по применению конструкций языка для решения различных задач. Также рассматриваются общие понятия реляционных баз данных, приводятся ER-диаграмма и описания таблиц учебной базы данных.

Пособие предназначено студентам и аспирантам математического факультета, специализирующихся по новым информационным технологиям, и всем, кто желает изучить основы работы с реляционными базами данных.

Учебное пособие выполнено в соответствии с планом работ кафедры UNESCO по новым информационным технологиям для методической поддержки общего курса «Базы данных и экспертные системы», соответствующего Государственному образовательному стандарту по специальности 01.02 «Прикладная математика».

2404010000

Г----- - 2001

ЛР №020464

ББК

3973.2-018я73

ISBN 5-8353-0065-4

© Гудов А.М., 2001

© Шмакова Л.Е., 2001

© Кемеровский госуниверситет., 2001

Оглавление

Введение.....	5
1. Основные понятия реляционной модели базы данных	9
2. Первое, что нужно знать	11
Типы данных	11
Правила присвоения имен объектам базы данных.....	12
Арифметические выражения	13
Функции.....	17
3. Среда SQL*Plus	24
Запуск SQL*Plus	24
Команды SQL*Plus	26
Определение переменных во время выполнения	29
Как использовать команды SQL в среде SQL*Plus.....	33
4. Выборка данных из базы данных	34
Команда запроса данных.....	34
Простой запрос.....	35
Сортировка строк, возвращаемых запросом.....	39
Ограничение количества выбираемых строк.....	42
Группировка строк в запросе.....	48
Выборка данных из нескольких таблиц.	54
Подзапросы.....	61
5. Определение структур данных	66
Структуры данных	66
Создание таблиц.....	67
Изменение таблиц и ограничений.....	73
Удаление таблицы.....	77
Изменение имени объекта.....	77
Усечение таблицы	78
Добавление комментариев к таблице	78
6. Манипулирование данными	79
Вставка новых строк в таблицу.....	79
Обновление строк в таблице.....	82
Удаление строк из таблицы.	84
7. Управление транзакциями	85
Состояние данных до и после завершения транзакции	86
Фиксация изменений в данных.	87
Откат результатов	87
8. Другие объекты базы данных	89

Последовательности	89
Представления	92
Индексы	96
9. Задания для практических занятий	99
Практическое занятие 1: выборка строк.....	99
Практическое занятие 2: использование в запросах однострочных функций.	100
Практическое занятие 3: ограничение количества выбираемых строк.....	100
Практическое занятие 4: использование в запросе групповых функций.	103
Практическое занятие 5: выборка данных из нескольких таблиц.	104
Практическое занятие 6: определение переменных во время выполнения.	104
Практическое занятие 7: подзапросы.	105
Практическое занятие 8: создание таблиц.	106
Практическое занятие 9: изменение таблиц и ограничений.	107
Практическое занятие 10: обработка данных.	108
Практическое занятие 11: управление транзакциями.	109
Практическое занятие 12: создание последовательностей.	110
Практическое занятие 13: создание представлений.	111
Практическое занятие 14: создание индексов.	111
Литература.	112
Приложения	114
ER-диаграмма учебной базы данных:.....	114
Структуры таблиц учебной базы данных.....	115

Введение

Основные идеи современной информационной технологии базируются на концепции, согласно которой данные должны быть организованы в базы данных с целью адекватного отображения изменяющегося реального мира и удовлетворения информационных потребностей пользователей. Эти базы данных создаются и функционируют под управлением специальных программных комплексов, называемых системами управления базами данных (СУБД) [3].

Борясь за покупателя, фирмы, производящие программное обеспечение, стали выпускать на рынок все более и более интеллектуальные и, следовательно, объемные программные комплексы. Приобретая (желая приобрести) такие комплексы, многие организации и отдельные пользователи часто не могли разместить их на собственных ЭВМ, однако не хотели и отказываться от нового сервиса. Для обмена информацией и ее обобществления были созданы сети ЭВМ, где обобществляемые программы и данные стали размещать на специальных обслуживающих устройствах - файловых серверах.

СУБД, работающие с файловыми серверами, позволяют множеству пользователей разных ЭВМ (иногда расположенных достаточно далеко друг от друга) получать доступ к одним и тем же базам данных. При этом упрощается разработка различных автоматизированных систем управления организациями, учебных комплексов, информационных и других систем, где множество сотрудников (учащихся) должны использовать общие данные и обмениваться создаваемыми в процессе работы (обучения). Однако при такой идеологии вся обработка запросов из программ или с терминалов пользователя выполняется на этих серверах. Для обеспечения эффективной работы при решении таких задач была предложена технология "Клиент-Сервер", по которой запросы пользовательских ЭВМ (Клиент) обрабатываются на специальных серверах баз данных (Сервер), а на клиента возвращаются лишь результаты обработки запроса [1].

Увеличение объема и структурной сложности хранимых данных, расширение круга пользователей информационных систем

привели к широкому распространению наиболее удобных и сравнительно простых для понимания реляционных (табличных) СУБД. Для обеспечения одновременного доступа к данным множества пользователей, нередко расположенных достаточно далеко друг от друга и от места хранения баз данных, созданы сетевые мультипользовательские версии СУБД. В них тем или иным путем решаются специфические проблемы параллельных процессов, целостности (правильности) и безопасности данных, а также санкционирования доступа.

Ясно, что совместная работа пользователей в сетях с помощью унифицированных средств общения с базами данных возможна только при наличии стандартного языка манипулирования данными, обладающего средствами для реализации перечисленных выше возможностей.

Таким языком стал SQL, разработанный в 1974 году фирмой IBM для экспериментальной реляционной СУБД System R. После появления на рынке двух пионерских СУБД этой фирмы - SQL/DS (1981 год) и DB2 (1983 год) - он приобрел статус стандарта де-факто для профессиональных реляционных СУБД. В 1987 году SQL стал международным стандартом языка баз данных, а в 1992 году вышла вторая версия этого стандарта. Поэтому все современные версии профессиональных реляционных СУБД (DB2, Oracle, Ingres, Informix, Sybase, Progress, Rdb) и даже нереляционных СУБД (например, Adabas) используют технологию "Клиент-Сервер" и язык SQL. К тому же приходят разработчики СУБД для персональных ЭВМ, многие из которых уже сегодня снабжены языком SQL [4].

Рассматриваемый же ниже непроцедурный язык SQL (Structured Query Language - структурированный язык запросов) ориентирован на операции с данными, представленными в виде логически взаимосвязанных совокупностей таблиц. Особенность предложений этого языка состоит в том, что они ориентированы в большей степени на конечный результат обработки данных, чем на процедуру этой обработки. SQL сам определяет, где находятся данные, какие индексы и даже наиболее эффективные последовательности операций следует использовать для их получения: не надо указывать эти детали в запросе к базе данных.

Реализация в SQL концепции операций, ориентированных на табличное представление данных, позволило создать компактный

язык с небольшим (менее 30) набором предложений. SQL может использоваться как интерактивный (для выполнения запросов) и как встроенный (для построения прикладных программ).

Основными конструкциями языка служат:

- предложения определения данных (определение баз данных, а также определение и уничтожение таблиц и индексов);
- запросы на выбор данных (предложение SELECT);
- предложения модификации данных (добавление, удаление и изменение данных);
- предложения управления данными (предоставление и отмена привилегий на доступ к данным, управление транзакциями и другие).

Кроме того, SQL предоставляет возможность выполнять в своих предложениях:

- арифметические вычисления (включая разнообразные функциональные преобразования), обработку текстовых строк и выполнение операций сравнения значений арифметических выражений и текстов;
- упорядочение строк и (или) столбцов при выводе содержимого таблиц на печать или экран дисплея;
- создание представлений (виртуальных таблиц), позволяющих пользователям иметь свой взгляд на данные без увеличения их объема в базе данных;
- запоминание выводимого по запросу содержимого таблицы, нескольких таблиц или представления в другой таблице (реляционная операция присваивания).
- агрегатирование данных: группирование данных и применение к этим группам таких операций, как среднее, сумма, максимум, минимум, число элементов и т.п.

Ориентированный на работу с таблицами SQL не имеет достаточных средств для создания сложных прикладных программ. Поэтому в разных СУБД он либо используется вместе с языками программирования высокого уровня (например, такими как Си или Паскаль), либо включен в состав команд специально разработанного языка СУБД (язык систем dBASE, R:BASE, Oracle и т.п.).

Здесь рассматриваются наиболее важные предложения базового варианта SQL, изложенного в стандарте ANSI 1992 года,

позволяющие познакомиться с основными средствами манипулирования данными. Недостаток места не позволил подробно рассмотреть другие его конструкции. Однако и таких знаний достаточно для получения данных из баз, находящихся под управлением большинства современных СУБД.

В разделе *«Основные понятия реляционной модели базы данных»* очень кратко даются необходимые для дальнейшего понимания материала понятия из теории реляционных баз данных.

Раздел *«Первое, что нужно знать»* посвящен описанию типов данных, арифметических и логических конструкций, а так же функций, изложенных в стандарте языка ANSI SQL-92.

Раздел *«Среда SQL*Plus»* описывает основные принципы работы со средством SQL*Plus корпорации Oracle – простого командного интерфейса пользователя для работы с сервером Oracle 7 [2]. Если читатель пользуется какой-либо другой прикладной программой, реализующей те же самые функции, он может пропустить этот раздел. Однако все примеры в тексте пособия приводятся так, как это выглядит в среде SQL*Plus.

В разделе *«Выборка данных из базы данных»* читатель найдет описание команды SELECT – единственного средства выборки данных в языке SQL. Материал содержит множество примеров, поясняющих использование этой мощной команды для решения конкретных задач.

В следующем разделе *«Определение структур данных»* описаны элементы языка определения структур данных (Database Definition Language, DDL), предназначенных для создания, модификации и удаления основной структуры для хранения данных в реляционной СУБД – таблицы.

Раздел *«Манипулирование данными»* посвящен описанию команд языка манипулирования данными (Database Manipulation Language, DML), с помощью которых данные можно заносить в таблицы, редактировать их или удалять из таблиц.

Раздел *«Управление транзакциями»* полностью описывает очень важную тему – язык управления транзакциями (Transaction Control Language, TCL). Транзакция является основной логической единицей для работы реляционных СУБД. Именно с помощью аппарата управления транзакциями СУБД содержит базу данных, с которой работают пользователи, в целостном виде.

Дополнительные объекты для хранения и обработки данных – последовательности, представления и индексы описываются в разделе «*Другие объекты базы данных*». Команды для работы с этими объектами относятся к группе команд DDL.

И наконец, в последнем разделе «*Задания для практических занятий*» приводятся контрольные упражнения, которые необходимо выполнить читателю для усвоения предлагаемого материала. Эти задания можно выполнять на практических занятиях в компьютерном классе, в домашних условиях, совместно с преподавателем или самостоятельно. В тексте каждого раздела приводятся ссылки на соответствующее практическое занятие, помеченные изображением компьютера.

В «*Приложениях*» содержится справочный материал - структуры таблиц и ER-диаграмма учебной базы данных. Эта информация постоянно требуется для выполнения практических заданий.

Текст данного пособия составлен на основании личного опыта авторов по преподаванию дисциплины «Базы данных» на математическом факультете Кемеровского госуниверситета, практических пособий слушателей авторизованных учебных курсов Oracle [2] и доступной по Интернет литературы. Авторы с удовольствием примут к сведению все замечания или пожелания при разработке следующего издания учебного пособия, которые можно присылать по адресу: good@kemsu.ru или stsle@ic.kemsu.ru.

1. Основные понятия реляционной модели базы данных

Реляционные системы берут свое начало в математической теории множеств. Они были предложены в конце 1968 года доктором Э.Ф. Коддом из фирмы IBM, который первым осознал, что можно использовать давно разработанный математический аппарат для придания надежной основы и строгости области управления базами данных.

Нечеткость многих терминов, используемых в сфере обработки данных, заставила Кодда отказаться от них и придумать новые или дать более точные определения существующим. Так, он не мог использовать широко распространенный термин "запись", который в различных ситуациях может означать экземпляр записи, либо тип

записей, запись в стиле Кобола (которая допускает повторяющиеся группы) или плоскую запись (которая их не допускает), логическую запись или физическую запись, хранимую запись или виртуальную запись и т.д. Вместо этого он использовал термин "кортеж длины n" или просто "кортеж", которому дал точное определение. В литературе [3, 4, 5] можно подробно познакомиться с терминологией реляционных баз данных, а здесь мы будем использовать неформальные их эквиваленты: таблица - для отношения, строка или запись - для кортежа, столбец или поле - для атрибута.

Появление теории реляционных баз данных и предложенного Коддом языка запросов "alpha", основанного на реляционном исчислении [6], инициировало разработку ряда языков запросов, которые можно отнести к двум классам:

1. Алгебраические языки, позволяющие выражать запросы средствами специализированных операторов, применяемых к отношениям (JOIN - соединить, INTERSECT - пересечь, SUBTRACT - вычесть и т.д.).
2. Языки исчисления предикатов, представляющие собой набор правил для записи выражения, определяющего новое отношение из заданной совокупности существующих отношений. Другими словами исчисление предикатов есть метод определения того отношения, которое нам желательно получить (как ответ на запрос) из отношений, уже имеющихся в базе данных.

Современные реляционные СУБД, как правило, поддерживают аппарат реляционной алгебры внутри своей организации, а пользователю предоставляют интерфейс для формулирования запросов к базе данных, основанный на реляционном исчислении. Операторы обоих языков замкнуты относительно понятия отношения, т.е. результатом любой операции над отношением является отношение с новыми свойствами (атрибутами или кортежами).

Само понятие "отношение" определено на множестве доменов (атрибутов с набором допустимых значений) и множестве кортежей, соответствующих множеству доменов. Множество доменов составляет схему отношения (структуру таблицы), а множество кортежей – тело отношения (набор записей в таблице). Каждое отношение должно удовлетворять следующему набору требований:

- отсутствие дубликатов кортежей (записей);
- отсутствие требования упорядоченности атрибутов (столбцов), при этом обращение к определенному атрибуту должно осуществляться по его имени;
- отсутствие требования упорядоченности кортежей (записей), обращение к конкретной записи должно происходить по значению ее ключа;
- атомарность значений атрибутов (столбцы таблицы не должны содержать более сложных значений, чем константы определенного для этого столбца типа).

Для отношения в целом можно установить ряд ограничений, которые направлены на поддержание базы данных в целостном виде. Современные СУБД отслеживают соблюдение этих правил целостности автоматически. На практике набор этих ограничений, как правило, определяется при создании структуры таблиц.

Одним из основных механизмов поддержания базы данных в целостном виде является механизм транзакции. Транзакция – это логическая единица работы СУБД по изменению данных, которая может завершиться двумя способами: либо с сохранением результатов во внешней памяти, либо с откатом на то состояние, которое база данных имела на момент начала данной транзакции. Другими словами, транзакция может начаться только в том случае, когда база данных находится в целостном виде, и закончиться так, чтобы база данных осталась в целостном виде. Современные СУБД поддерживают два уровня управления транзакции – неявный (автоматический) и явный (при котором транзакция управляется набором команд языка SQL).

2. Первое, что нужно знать

Типы данных

В SQL используются следующие основные типы данных, форматы которых могут несколько различаться для разных СУБД:

<i>Тип данных</i>	<i>Описание</i>
-------------------	-----------------

NUMBER(p,s)	Числовое значение, максимальное количество цифр в котором равно "p", а количество десятичных знаков - "s".
INTEGER	Целое число (обычно до 7 значащих цифр и знак). Аналог NUMBER(7).
VARCHAR2(s)	Символьная строка переменной длины, максимальный размер которой равен "s". В системе Oracle 7 максимально возможное значение s составляет 2000 символов.
DATE	Значение даты и времени между 1 января 4712 г. до нашей эры и 31 декабря 4712 г. нашей эры.
CHAR(s)	Символьное значение постоянной длины "s". В системе Oracle 7 максимально возможное значение s составляет 256 символов.
LONG	Символьные значения переменной длины размером до 2Гб
RAW и LONG RAW	Эквиваленты VARCHAR2 и LONG для двоичных данных.

В некоторых СУБД еще существует тип данных LOGICAL, DOUBLE и ряд других. Некоторые СУБД предоставляют пользователю возможность самостоятельного определения новых типов данных, например, плоскостные или пространственные координаты, единицы различных метрик, пяти- или шестидневные недели (рабочая неделя, где сразу после пятницы или субботы следует понедельник), дроби, графика, большие целые числа и т.п.

```
... start date DATE DEFAULT SYSDATE,...
```

Правила присвоения имен объектам базы данных

- Должны начинаться с буквы.
- Могут включать от 1 до 30 символов.
- Могут содержать только символы A-Z, a-z, 0-9, _ (подчеркивание), \$ и #.
- Не могут совпадать с именем другого объекта, принадлежащего этому же пользователю.
- Не могут совпадать с зарезервированным словом сервера базы данных.

Арифметические выражения

Иногда требуется изменить способ вывода данных, произвести вычисления или просмотреть сценарии "а что, если ...". Это можно сделать с помощью арифметических выражений. Арифметическое выражение может содержать имена столбцов, числовые константы и арифметические операторы.

Арифметические операторы

Ниже перечислены арифметические операторы, доступные в SQL. Использовать их можно в любом предложении команды SQL, кроме FROM.

<i>Оператор</i>	<i>Описание</i>
+	Сложение
-	Вычитание
*	Умножение
/	Деление
	Конкатенация

Порядок выполнения операторов

Если арифметическое выражение содержит более одного оператора, то умножение и деление выполняются в первую очередь. Если операторы в выражении имеют один и тот же приоритет, они выполняются слева направо. Изменить порядок действий при вычислении арифметического выражения можно с помощью скобок.

Пример. Вычисление годовых компенсационных. Сумма выплат за год вычисляется путем умножения заработной платы (значение переменной salary) на 12 и прибавления одноразовой премии в размере 100.

$12 * salary + 100$

Примечание: Для изменения порядка действий и упрощения чтения можно использовать скобки. Если, например, записать вышеуказанное выражение в виде $(12 * SALARY) + 100$, то

результат не изменится.

Пример. Вывод фамилии, заработной платы и суммы выплат за год для каждого служащего. Размер выплат за год вычисляется путем прибавления к заработной плате ежемесячной премии в размере 100 и умножения суммы на 12.

```
12 * (salary + 100)
```

Оператор конкатенации “||” позволяет соединять значения одних столбцов с другими столбцами, арифметическими выражениями или постоянными значениями для создания символьных выражений. Столбцы, указанные по обе стороны этого оператора, объединяются для вывода в один столбец.

Пример. Соединение двух строковых констант.

```
'Good' || 'Year' → 'GoodYear'
```

Операторы сравнения

Операторы сравнения делятся на две категории: логические и операторы SQL. Они используются для сравнения значений выражений.

Операторы сравнения проверяются следующими условиями:

Оператор	Значение
=	Равно
>	Больше
>=	Больше или равно
<	Меньше
<=	Меньше или равно

Имеется четыре оператора SQL, используемых с данными всех типов:

Оператор	Значение
BETWEEN...AND...	Между двумя значениями (включительно)
IN(список)	Совпадает с каким-то из значений в списке
LIKE	Соответствует символьному шаблону
IS NULL	Является неопределенным значением

Логические операторы:

Оператор	Значение
AND	Если обе части условия истинны, то условие истинно.
OR	Если хотя бы одна часть условия истинна, то условие истинно.
NOT	Возвращает противоположное условие.

Отрицание выражений:

Оператор	Значение
!=	Не равно (VAX. UNISX. PC)
^=	Не равно (IBM)
<>	Не равно (все операционные системы)
NOT имя столбца =	Не равно
NOT <имя столбца >	Не больше

Отрицание операторов SQ:

Оператор	Значение
NOT BETWEEN...AND...	НЕ между двумя значениями (включительно)
NOT IN(список)	НЕ входит в список значений
NOT LIKE	Не подобно заданной строке
IS NOT NULL	Не является неопределенным значением

Порядок выполнения операций:

Порядок вычисления	Оператор
1	Все операторы сравнения
2	AND
3	OR

Стандартный порядок выполнения операций отменяется скобками.

Строки символов (литералы)

Литерал — это любой символ, выражение или число, включенные в список SELECT и не являющиеся ни именем, ни псевдонимом столбца. Они печатаются для каждой возвращаемой строки. Литералы в виде текста произвольного формата могут быть включены в результат запроса. В списке SELECT они рассматриваются как столбцы. Символьные литералы и литералы-даты должны быть заключены в апострофы (‘’), а числовые

литералы - нет.

Пример:

```
'L ast Year'  
'01-JAN-98'  
123  
12.764
```

Выражения с датами

Для данных типа дата в SQL возможно применять некоторые арифметические операторы:

<i>Операция</i>	<i>Результат</i>	<i>Описание</i>
Дата + число	Дата	Прибавление количества дней к дате
Дата - число	Дата	Вычитание количества дней из даты
Дата – дата	Кол-во дней	Вычитание одной даты из другой
Дата + число/24	дата	Прибавление к дате часов

Обработка неопределенных значений

Неопределенным значением (NULL) называется недоступное, неприсвоенное, неизвестное или неприменимое значение. Неопределенное значение - это не ноль и не пробел. Ноль — это число, а пробел — символ. Издержки "хранения" неопределенного значения - это один байт внутренней памяти.

Неопределенные значения возможны в столбцах любых типов, если при создании таблицы они не были описаны как столбцы только с определенными значениями (NOT NULL) или столбцы, содержащие первичный ключ (PRIMARY KEY).

Если выражение содержит неопределенное значение в любом из столбцов, то и результатом вычисления выражения будет неопределенное значение. При попытке деления на ноль вы получите сообщение об ошибке, а результатом деления на неопределенное значение будет неопределенное значение.

Для преобразования неопределенного значения в фактическое используется функция NVL:

```
NVL (выражение1, выражение 2)
```

где:

выражение1 Исходное или вычисленное
 значение, которое может быть

Выражение2

неопределенным.

Значение, которое подставляется вместо неопределенного значения.

Примечание: Функцию NVL можно применять для преобразования любого типа данных, но результат всегда будет того же типа, что и *выражение1*.

Пример: при вычислении значений по следующей формуле не возникает неопределенности в интерпретации получаемого результата, поскольку если значение переменной *salary* будет неопределено, функция возвратит значение «ноль».

```
12*NVL(salary,0)/100
```

Преобразование NVL для различных типов:

<i>Тип данных</i>	<i>Пример преобразования</i>
NUMBER	NVL (<i>числовой столбец, 9</i>)
DATE	NVL (<i>столбец даты, '01-ЯНВ-95'</i>)
CHAR или VARCHAR2	NVL (<i>символы столбец, 'Недоступно'</i>)

Функции

В языке SQL существуют два класса функций – однострочные и групповые.

Однострочные функции принимают на вход одну строку (запроса или арифметического выражения) и выдают один результат. Этот результат, как и в случае понятия функции в любом языке программирования, связывается с ее именем. Однострочные функции могут быть разных типов. Мы рассмотрим следующие типы функций: символьные; числовые; для работы с датами; функции преобразования. Аргументом однострочных функций может быть: константа, заданная пользователем; значение переменной; имя столбца таблицы; выражение.

Групповая функция принимает на входе группу строк и выдает одно значение после обработки этой группы.

Различие в интерпретации входных данных обуславливает и различие в применении этих функций. Так однострочные функции

могут использоваться там, где в качестве результата запроса к базе данных подразумевается получение только одной строки данных. Для использования групповой функции необходимо сначала сформировать из «многострочного» результата запроса группы строк, а затем для каждой из них применить групповую функцию.

Синтаксис:

имя_функции (столбец | выражение, [аргумент1, аргумент2, ...])

где:

<i>имя_функции</i>	имя функции
<i>Столбец</i>	любой именованный столбец из базы данных.
<i>Выражение</i>	любая строка символов или вычисляемое выражение.
<i>аргумент1, аргумент2</i>	любой аргумент, используемый функцией.

Числовые функции

<i>Функция</i>	<i>Возвращаемое значение</i>
ABS(<i>n</i>)	Абсолютное значение величины <i>n</i>
CEIL(<i>n</i>)	Наименьшее целое, большее или равное <i>n</i>
COS(<i>n</i>)	Косинус <i>n</i> (угла, выраженного в радианах)
COSH(<i>n</i>)	Гиперболический косинус <i>n</i>
EXP(<i>n</i>)	<i>e</i> в степени <i>n</i>
FLOOR(<i>n</i>)	Наибольшее целое, меньшее или равное <i>n</i>
LN(<i>n</i>)	Натуральный логарифм <i>n</i> , где <i>n</i> >0
LOG(<i>m</i> , <i>n</i>)	Логарифм <i>n</i> по основанию <i>m</i>
MOD(<i>m</i> , <i>n</i>)	Остаток от деления <i>m</i> на <i>n</i>
POWER(<i>w</i> , <i>n</i>)	<i>w</i> в степени <i>n</i>
ROUND(<i>n</i> [, <i>m</i>])	<i>n</i> , округленное до <i>m</i> позиций после десятичной точки. По умолчанию <i>m</i> равно нулю
SIGN(<i>n</i>)	-1 (если <i>n</i> <0); 0 (если <i>n</i> =0); 1 (если <i>n</i> >0)
SIN(<i>n</i>)	Синус <i>n</i> (угла, выраженного в радианах)
SINH(<i>n</i>)	Гиперболический синус
SQRT(<i>n</i>)	Квадратный корень от <i>n</i> . Если <i>n</i> <0, возвращает значение NULL
TAN(<i>n</i>)	Тангенс <i>n</i> (угла, выраженного в радианах)
TANH(<i>n</i>)	Гиперболический тангенс <i>n</i>
TRUNC(<i>n</i> [, <i>m</i>])	<i>n</i> , усеченное до <i>m</i> позиций после десятичной

точки. По умолчанию t равно нулю

Символьные функции

Функция	Возвращаемое значение
<i>Символьные функции, возвращающие символьные значения:</i>	
CHR(n)	Символ с кодом n
CONCAT($char1, char2$)	Конкатенация символьных строк $char1$ и $char2$
INITCAP($char$)	Символьная строка $char$, первые буквы всех слов в которой преобразованы в прописные
LOWER($char$)	Символьная строка $char$, все буквы которой преобразованы в строчные
LPAD($char1, n[, char2]$)	Символьная строка $char1$, которая дополняется слева последовательностью символов из $char2$ так, чтобы общая длина строки стала равна n . Значение $char2$ по умолчанию – (один пробел). Если часть многобайтового символа не помещается в добавляемой строке, то конец строки заполняется пробелами.
LTRIM($char[, set]$)	Символьная строка $char$, в которой удалены все символы от начала вплоть до первого символа, которого нет в строке set . Значение set по умолчанию - " (один пробел).
NLS_INITCAP($char[, nls_sort]$)	Символьная строка $char$, в которой первые буквы всех слов преобразованы в прописные. Параметр nls_sort определяет последовательность сортировки
NLS_LOWER($char[, nls_sort]$)	Символьная строка $char$, все буквы которой преобразованы в строчные. Параметр nls_sort определяет последовательность сортировки
NLS_UPPER($char[, nls_sort]$)	Символьная строка $char$, все буквы которой преобразованы в прописные. Параметр nts_sort определяет последовательность сортировки
REPLACE($char, search_string [, replacement_string]$)	Символьная строка $char$, в которой все фрагменты $search_string$ заменены на $replacement_string$. Если параметр $replacement_string$ не определен, все фрагменты $search_string$ удаляются
RPAD($char1, n[, char2]$)	Символьная строка $char1$, которая дополнена справа последовательностью символов из $char2$ так, что общая длина строки равна n .

	Если часть многобайтового символа не помещается в добавляемой строке, то конец строки заполняется пробелами
RTRIM(<i>char</i> [, <i>set</i>])	Символьная строка <i>char</i> , в которой удалены все символы справа вплоть до первого символа, которого нет в строке <i>set</i> . Значение параметра <i>set</i> по умолчанию – ‘ ’ (один пробел).
SOUNDEX(<i>char</i>)	Символьная строка, содержащая фонетическое представление для <i>char</i> , на английском языке
SUBSTR(<i>char</i> , <i>m</i> [, <i>n</i>])	Фрагмент символьной строки <i>char</i> , начинающийся с символа <i>m</i> , длиной <i>n</i> символов (до конца строки, если параметр <i>n</i> не указан).
SUBSTRB(<i>char</i> , <i>m</i> [, <i>n</i>])	Фрагмент символьной строки <i>char</i> , начинающийся с символа <i>m</i> , длиной <i>n</i> байтов (до конца строки, если параметр <i>n</i> не указан).
TRANSLATE(<i>char</i> , <i>from</i> , <i>to</i>)	Символьная строка <i>char</i> , в которой все символы, встречающиеся в строке <i>from</i> , заменены на соответствующие символы из <i>to</i> .
UPPER(<i>char</i>)	Символьная строка <i>char</i> , в которой все буквы преобразованы в прописные
<i>Символьные функции, возвращающие числовые значения:</i>	
ASCII(<i>char</i>)	Возвращает десятичный код первого символа строки <i>char</i> в кодировке, принятой в базе данных. (Код ASCII в системах, использующих кодировку ASCII). Возвращает значение первого байта многобайтового символа.
INSTR(<i>char1</i> , <i>char2</i> [, <i>n</i> [, <i>m</i>]])	Позиция первого символа <i>m</i> -ого фрагмента строки <i>char1</i> , совпадающего со строкой <i>char2</i> , начиная с <i>n</i> -ого символа. По умолчанию <i>n</i> и <i>m</i> равны 1. Номер символа отсчитывается от первого символа строки <i>char1</i> , даже когда <i>n</i> > 1
INSTRB(<i>char1</i> , <i>char2</i> [, <i>n</i> [, <i>m</i>]])	Позиция первого символа <i>n</i> -ого фрагмента строки <i>char1</i> , совпадающего со строкой <i>char2</i> , начиная с <i>m</i> -ого байта. По умолчанию <i>n</i> и <i>m</i> равны 1. Номер байта отсчитывается от первого символа строки <i>char1</i> , даже когда <i>n</i> > 1.
LENGTH(<i>char</i>)	Длина строки <i>char</i> в символах

LENGTHB(<i>char</i>)	Длина строки <i>char</i> в байтах
NLSSORT(<i>char1, char2</i> [, <i>n</i>],[<i>m</i>])	Зависящее от национального языка значение, используемое при сортировке строки <i>char</i> .

Функции для работы с датами

Oracle хранит данные во внутреннем цифровом формате: век, год, месяц, число, часы, минуты, секунды. По умолчанию дата выдается в формате «DD-MON-YY».

Функция	Назначение
MONTHS_BETWEEN(<i>date1, date2</i>)	Определяет число месяцев, разделяющих две даты. Дробная часть результата представляет собой долю месяца.
ADD_MONTHS(<i>date, n</i>)	Добавление календарных месяцев к дате.
NEXT_DAY(<i>date, 'char'</i>)	Ближайшая дата, когда наступит заданный день. Аргумент ' <i>char</i> ' может задавать порядковый номер или название дня недели.
LAST_DAY(<i>date</i>)	Определение последнего дня месяца, содержащего заданную дату.
ROUND(<i>date</i> [, ' <i>fmt</i> '])	Округление до целого числа суток. Если <i>fmt</i> =YEAR, определяет первый день года.
TRUNC(<i>date</i> [, ' <i>fmt</i> '])	Возвращает первый день месяца, указанного в аргументе <i>date</i> . Если <i>fmt</i> =YEAR, возвращает дату первого дня года.
SYSDATE()	Возвращает текущую дату и время.

Функции преобразования типа

Функция	Возвращаемое значение
TO_CHAR (<i>date</i> [, ' <i>fmt</i> '])	Преобразование даты в строку символов в соответствии с форматной моделью <i>fmt</i> .
TO_CHAR (<i>number</i> [, ' <i>fmt</i> '])	Преобразование числа в строку символов в соответствии с форматной моделью <i>fmt</i> .
TO_NUMBER (<i>char</i>)	Преобразование строки символов в числовой формат.
TO_DATE (<i>char</i> [, ' <i>fmt</i> '])	Преобразование строки символов в формат даты в соответствии с форматной моделью <i>fmt</i> .

Форматные модели

Модель формата:

- должна быть заключена в апострофы;
- различает символы верхнего и нижнего регистров;
- может включать любые разрешенные элементы формата даты;
- использует элемент `fm` для удаления конечных пробелов и ведущих нулей;
- отделяется от значения даты запятой.

<i>Модель</i>	<i>Описание</i>
<i>Форматные модели для работы с датами</i>	
YY[YY]	Полный год цифрами.
YEAR	Год прописью.
MM	Двузначное цифровое обозначение месяца.
MON	Трехсимвольное сокращенное название месяца.
MONTH	Полное название месяца.
DD	День недели цифрами.
DY	Трехсимвольное сокращенное название дня недели.
DAY	Полное название дня недели.
HH	Часы цифрами в 12-ти часовом формате.
HH24	Часы цифрами в 24-х часовом формате.
MI	Минуты цифрами.
SS	Секунды цифрами.
AM	Символы 'AM' 'PM'.
<i>Числовые модели формата</i>	
9	Вывод цифры с подавлением ведущих нулей.
0	Вывод цифры, если ведущий нуль – вывод нуля.
\$	Плавающий знак доллара.
L	Плавающий символ местной валюты
.	Вывод десятичной точки.
,	Вывод разделителя троек цифр.

Групповые функции

<i>Функция</i>	<i>Возвращаемое значение</i>
----------------	------------------------------

AVG([<i>DISTINCT</i> <i>ALL</i>] <i>n</i>)	Среднее значение от <i>n</i> , нулевые значения опускаются
COUNT([<i>ALL</i>]*)	Число строк, извлекаемых в запросе или подзапросе
COUNT([<i>DISTINCT</i> <i>ALL</i>] <i>expr</i>)	Число строк, для которых <i>expr</i> принимает не пустое значение
MAX([<i>DISTINCT</i> <i>ALL</i>] <i>expr</i>)	Максимальное значение выражения <i>expr</i>
MIN([<i>DISTINCT</i> <i>ALL</i>] <i>expr</i>)	Минимальное значение выражения <i>expr</i>
STDDEV([<i>DISTINCT</i> <i>ALL</i>] <i>n</i>)	Стандартное отклонение величины <i>n</i> , нулевые значения опускаются
SUM([<i>DISTINCT</i> <i>ALL</i>] <i>n</i>)	Сумма значений <i>n</i>
VARIANCE([<i>DISTINCT</i> <i>ALL</i>] <i>n</i>)	Дисперсия величины <i>n</i> , нулевые значения опускаются

Примеры использования некоторых функций

Функция округления:

- ROUND (45.923, 2) 45.92
- ROUND (45.923, 0) 46
- ROUND (45.923, -1) 50

Функция усечения:

- TRUNC (45.923, 2) 45.92
- TRUNC (45.923) 45
- TRUNC (45.923, -1) 40

Вычисление остатка от деления двух чисел:

- MOD(1600,300) 100

Функции работы с датами:

- MONTHS_BETWEEN('01-SEP-95','11-JAN-94') 19.6774194
- ADD_MONTHS('11-JAN-94',6) '11-JUL-94'
- NEXT_DAY('01-SEP-95','FRIDAY') '08-SEP-95'
- LAST_DAY('01-SEP-95') '30-SEP-95'
- ROUND('25-MAY-95','MONTH') 01-JUN-95
- ROUND('25-MAY-95 ','YEAR') 01-JAN-95
- TRUNC('25-MAY-95 ','MONTH') 01-MAY-95
- TRUNC('25-MAY-95 ','YEAR') 01-JAN-95

3. Среда SQL*Plus

SQL*Plus - это среда, разработанная фирмой Oracle, для реализации простого командного интерфейса доступа к СУБД и выполнения команд SQL и PL/SQL с дополнительными возможностями. Использовать команды SQL*Plus можно при написании даже самых основных команд SQL. В этом разделе показано, как с помощью команд SQL*Plus делать следующее:

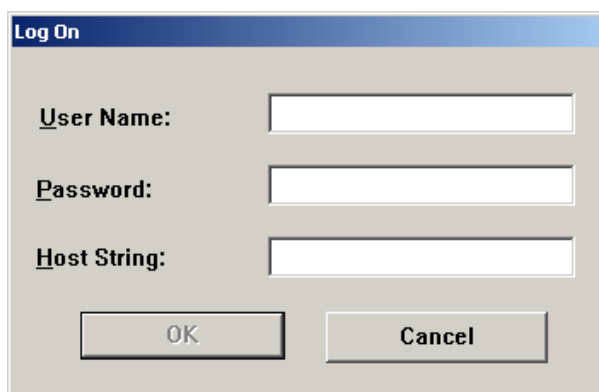
- Описывать структуру таблиц.
- Редактировать команды SQL в буфере.
- Сохранять файлы с командами SQL для редактирования.
- Выполнять сохраненные файлы.
- Загружать команды SQL из файла в буфер SQL.
- Получать оперативные справки.

Запуск SQL*Plus

Способ вызова SQL*Plus зависит от используемой операционной системы или среды Windows.

*Вход в SQL*Plus из среды Windows*

В менеджере программ дважды щелкнуть на пиктограмме SQL*Plus, после чего ввести имя пользователя и, если требуется, пароль и имя базы данных.



*Вход в SQL*Plus из командной строки*

На своей машине в ответ на приглашение операционной системы введите команду SQL*Plus.

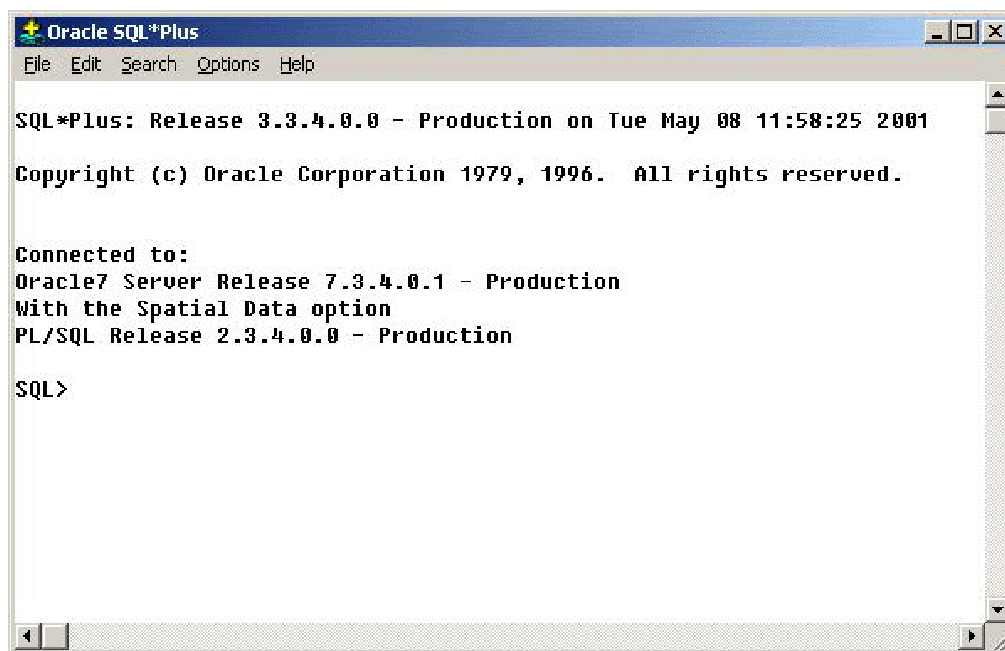
```
sqlplus [имя_пользователя [/пароль [@база_данных]]]
```


где:

<i>имя_пользователя</i>	имя пользователя базы данных
<i>пароль</i>	пароль для входа в базу данных. Если ввести его здесь, он будет виден на экране.
<i>база_данных</i>	строка подключения к базе данных.

Примечание: Для сохранения секретности своего пароля не следует вводить его в ответ на приглашение операционной системы. Лучше ввести только имя пользователя, а пароль ввести позже в ответ на приглашение "Password".

После входа в SQL*Plus на экране появляется следующее сообщение:



Вывод структуры таблицы

Структуру таблицы в SQL*Plus можно получить с помощью команды **DESCRIBE**. В результате выполнения команды на экран выводятся имена столбцов, типы данных и сведения об *обязательности* присутствия данных в столбцах.

`DESC[RIBE] имя таблицы`

где

Имя таблицы Имя существующей таблицы, представления

или синонима, доступных пользователю.

Пример Вывод информации о структуре таблицы S_DEPT.

```
SQL> DESCRIBE s_dept
```

Name	Null?	Type
ID	NOT NULL	NUMBER(7)
NAME	NOT NULL	VARCHAR2 (25)
REGION_ID		NUMBER(7)

где

Null?	Означает, что столбец должен обязательно содержать данные
Type	Показывает тип данных

Команды SQL*Plus

*Команды редактирования SQL*Plus*

При вводе команды SQL она записывается в область памяти, называемую буфером SQL, и остается там до ввода новой команды. Команды SQL*Plus вводятся по одной строке и не хранятся в буфере SQL.

Примечание: Если нажать клавишу [RETURN] до завершения команды, SQL*Plus выдает приглашение в виде номера строки. Ввод в буфер SQL прекращается вводом одного из символов окончания (точки с запятой или дробной черты) или двукратным нажатием на [RETURN]. После этого на экран выводится приглашение SQL.

Команды редактирования SQL*Plus:

<i>Команда</i>	<i>Описание</i>
----------------	-----------------

A[PPEND] <i>текст</i>	Добавить текст в конец текущей строки.
C[HANGE] /старый/новый/	Заменить в текущей строке <i>старый</i> текст на <i>новый</i> .
QHANGE] / <i>текст</i> /	Удалить <i>текст</i> из текущей строки.
CL[EAR] BUFF[ER]	Удалить все строки из буфера SQL.
DEL	Удалить текущую строку.
DEL n	Удалить строку, заданную параметром n.
DEL m n	Удалить строки от m до n.
I[NPUT]	Вставить неопределенное количество строк.
I[NPUT] <i>текст</i>	Вставить строку, состоящую из <i>текста</i> .
L[IST]	Вывести список всех строк в буфере SQL.
L[IST] n	Вывести одну строку (с номером n).
L[IST] m n	Вывести диапазон строк от n до m.
R[UN]	Вывести и выполнить команду из буфера SQL.
N	Указать строку, которая должна стать текущей.
n <i>текст</i>	Заменить строку n <i>текстом</i> .
0 <i>текст</i>	Вставить строку перед строкой 1.

Для сервера Oracle команды SQL*Plus являются вспомогательным средством. Они используются для управления средой, форматирования результатов запросов и работы с файлами.

Команды работы с файлами

Команда	Описание
SAV[E] <i>имя_файла</i> [.ext] [REPL[ACE]]APP[END]]	Сохраняет в файле текущее содержимое буфера SQL в файле. <i>APPEND</i> используется для добавления информации в существующий файл. <i>REPLACE</i> перезаписывает существующий файл. По умолчанию файл имеет расширение <i>.sql</i> .
GET <i>имя_файла</i>	Вызывает содержимое ранее сохраненного файла в буфер SQL. По умолчанию файл имеет расширение <i>.sql</i> .
START <i>имя_файла</i>	Запускает выполнение ранее сохраненного файла команд.

@ <i>имя_файла</i>	Запускает выполнение файла команд (как и команда START).
EDIT	Вызывает редактор и сохраняет содержимое буфера в файле <i>afiedt.buf</i> .
ED[IT] [<i>имя_файла</i> [.sql]]	Вызывает редактор для редактирования сохраненного файла.
SPO[OL] [<i>имя_файла</i> [.sql] OFF OUT]	Записывает результаты запроса в файл. <i>OFF</i> закрывает буферный файл (спул-файл). <i>OUT</i> закрывает буферный файл и посылает результаты из файла на системный принтер.
EXIT	Выход из SQL*Plus.

Примечание: Для замены текстового редактора следует изменить значение переменной `_EDITOR` среды SQL*Plus с помощью команды DEFINE.

В ответ на приглашение SQL можно ввести только одну команду SQL*Plus. Чтобы продолжить команду SQL на следующей строке, поставьте в конце текущей строки знак переноса (-).

Команда COLUMN

Управление столбцом отчета осуществляется с помощью команды COLUMN. Можно, например, изменить заголовок, ширину и формат.

Синтаксис:

COL[UMN] [(<i>column\alias</i>) <i>[option ...]</i>]

Опции команды COLUMN:

<i>Опция</i>	<i>Описание</i>
CLE[AR]	Отменяет любые форматы столбцов.
FOR[MAT] <i>format</i>	Меняет отображение данных столбца.
HEA[DING] <i>text</i>	Задаёт заголовок столбца. Вертикальная линия () задаёт переход на новую строку в заголовке, если вы не используете выравнивание.
JUS[TIFY] { <i>align</i> }	Выравнивает заголовок столбца (не данные!) слева, справа или по центру.

NOPRI[NT]	Прячет столбец.
NUL[L] <i>text</i>	Задаёт <i>text</i> , который должен отображаться в случае неопределённых значений.
PRI[NT]	Показывает столбец.
TRU[NCATED]	Усекает строку в конце первой строки дисплея.
WRA[PPED]	Переходит на следующую строку в конце строки.
WOR[DWRAPPED]	То же, что и WRAPPED, но слова не разбиваются.

Длинную команду можно перенести на следующую строку. Для этого текущую строку следует закончить символом переноса (-).

Определение переменных во время выполнения

SQL*Plus позволяет создавать интерактивные отчеты, когда пользователя приглашают ввести значения, ограничивающие объем выходных данных. Для создания отчета в командный файл или в отдельные команды SQL включаются так называемые **переменные подстановки (подстановочные переменные)**. Иными словами, переменная выступает в роли контейнера, в котором временно хранятся значения.

Возможные цели использования переменных:

- Определение интервалов дат.
- Выборка данных по конкретному пользователю.
- Выборка данных по конкретному отделу.
- Обмен переменными между командами SQL.
- Динамическое изменение верхних и нижних колонтитулов страниц.

Переменные подстановки с одним амперсандом (&)

Пользователь может динамически ограничивать выбираемые строки с помощью переменной подстановки, которой предшествует один амперсанд '&'. Значение такой переменной запрашивается при каждом выполнении команды. Команда **SET VERIFY** задает вывод текста команды до и после того, как SQL*Plus замещает переменные подстановки значениями.

Пример: Создание команды, которая во время выполнения

запрашивает номер отдела у пользователя. Отчет должен содержать учетный номер, фамилию и размер заработной платы каждого служащего.

```
SQL> SELECT id, last_name, salary
2>   FROM s_emp
3>   WHERE dept_id = &department_number;
Enter value for department_number: 31
old   3: WHERE dept_id = &department_number;
new   3: WHERE dept_id = 31
```

Подстановка текстовых переменных и дат

Текстовые строки и даты должны быть заключены в апострофы. Для избежания возможных ошибок следует заранее заключить амперсанд и имя переменной в апострофы (например, '&job_title') или вводить в них значение переменной при выполнении запроса.

Пример: Вывод номера, фамилии и размера заработной платы служащих, находящихся на конкретной должности. Должность запрашивается во время выполнения.

```
SQL> SELECT id, last_name, salary
2>   FROM s_emp
3>   WHERE title = '&job_title';
Enter value for job_title: Stock Clerk
```

С помощью переменных подстановки можно задать:

- Условие WHERE.
- Предложение ORDER BY.
- Выражение для столбца.
- Имя таблицы.
- Целую команду SELECT.

Пример: Вывод номера заказа, любого другого столбца и любого условия заказа. Попробуйте разные условия и имена столбцов и сравните результаты.

```

SQL> SELECT id, &column_name
2>    FROM s_ord
3>    WHERE &condition;

Enter value for column_name: total
Enter value for condition: payment_type = 'CASH'

```

Определение переменных пользователя

Задать переменные можно до выполнения команды SELECT. Для определения переменных используются две команды SQL*Plus:

- **DEFINE** – для создания переменной типа CHAR;
- **ACCEPT** – для чтения введенного значения и сохранения его в переменной.

SQL*Plus использует команду DEFINE для подстановки переменной с двойным амперсандом (&&). Если в команде DEFINE требуется задать строку с символом пробела, эта строка должна быть заключена в апострофы.

<i>Команда</i>	<i>Описание</i>
DEFINE переменная = значение	Создание переменной типа CHAR и присвоение ей значения
DEFINE переменная	Вывод переменной, ее значения и типа данных.
DEFINE	Вывод всех переменных пользователя, их значений и типа данных.
ACCEPT (синтаксис см. ниже)	Чтение строки, введенной пользователем, и сохранение ее в переменной.

Команда ACCEPT

- Создает приглашение к вводу данных, удобное для пользователя.
- Явно задает переменные типа NUMBER и DATE.
- Предотвращает отображение данных, введенных пользователем, в целях секретности.
- Переменная сохраняет значение до ее очистки командой UNDEFINE или до окончания сеанса работы в SQL*Plus.

Сокращенный синтаксис:

ACCEPT <i>переменная</i> [<i>тип_данных</i>] [FORMAT] [PROMPT <i>текст</i>] [HIDE]
--

где

<i>переменная</i>	Имя переменной, где хранится значение. Если переменная не существует, SQL*Plus ее создает.
<i>тип_данных</i>	NUMBER, CHAR или DATE. Максимальная длина для типа CHAR – 240 байтов. DATE сверяется с моделью формата.
FOR[MAT]	Модель формата – например, A10 или 9.999.
PROMPT <i>текст</i>	<i>текст</i> , который выдается прежде, чем пользователь может ввести значение.
HIDE	Предотвращает отображение данных, введенных пользователем – например, пароля.

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

Пример.

```
ACCEPT p_dname PROMT 'Provide the department name:'
ACCEPT p_salary NUMBER PROMPT 'Начальная зарплата:'
ACCEPT pswd CHAR PROMPT 'Password:' HIDE
```

Пример. Вывод номера и названия региона для заданного отдела. Создайте командный файл 17 prompt и используйте команду ACCEPT для настройки приглашения к вводу данных.

```
SET ECHO OFF
ACCEPT p_dname PROMT 'Provide the department name:'
SELECT d.name, r.id, r.name "Region Name"
FROM s_dept d, s_region r
WHERE d.region_id = r.id AND
UPPER(d.name) LIKE UPPER ('%p_dname%')
SET ECHO ON
```



```
SQL> start 17_prompt
Provide the department name: sales
```

Команда SET ECHO

От переменной ECHO зависит, печатают ли команды START и @ каждую команду командного файла по мере ее выполнения. В случае ECHO ON команда печатается, а в случае ECHO OFF – нет.

Команда UNDEFINE

Переменная сохраняет свое значение до ее очистки командой UNDEFINE или до завершения сеанса работы в SQL*Plus. Проверить неопределенные переменные можно с помощью команды DEFINE.

Передача значений переменных в командный файл

Для того, чтобы значения переменных передать в качестве параметров в командный файл, необходимо следовать следующим правилам. В командном файле используйте ссылку типа "&номер" (0 - 9) в любой из команд языка SQL.

- При запуске файла из командной строки задайте значение его параметров после имени файла, отделяя их один от другого пробелом.

```
SQL> START my_file value1 value2
```

Как использовать команды SQL в среде SQL*Plus

Следуя этим простым правилам и указаниям, можно задавать правильные команды в среде SQL*Plus для обработки на сервере баз данных, которые легко читать и редактировать.

- Вводить команды SQL можно в виде одной или нескольких строк.
- Для удобства чтения и редактирования каждое предложение обычно вводится на отдельной строке.
- Для удобства чтения команды можно использовать табуляцию и отступы.
- Сокращение и перенос слов в командах запрещаются.
- Ключевые слова и команды обычно вводятся символами

верхнего регистра; остальные слова (например, имена таблиц и столбцов) - символами нижнего регистра.

- При отсутствии особых указаний символы верхнего и нижнего регистров в командах SQL не различаются.
- Команда SQL вводится на той же строке, что и приглашение SQL, а последующие строки нумеруются. Эти строки образуют *буфер SQL*.
- В каждый момент времени активной в буфере может быть только одна команда и выполнить ее можно несколькими способами:
 - Поставить точку с запятой (;) в конце последнего предложения.
 - Поставить точку с запятой или дробную черту в последней строке буфера.
 - Ввести дробную черту в ответ на приглашение SQL.
 - Задать команду RUN среды SQL*Plus в ответ на приглашение SQL.



Для закрепления материала можно выполнить Практическое занятие 6, но только после того, как будет изучен материал следующего раздела.

4. Выборка данных из базы данных

Команда запроса данных

Команда **SELECT** выбирает информацию из базы данных.

Основной блок запроса:

```
SELECT          [DISTINCT] {*, столбец [псевдоним],  
групповая_функция}  
FROM            таблица  
[WHERE          условие]  
[GROUP BY      выражение_группирования]  
[HAVING        условие_группы]  
[ORDER BY      {Столбец, выражение} [ASC|DESC]];
```

где

SELECT	Список, включающий, по крайней мере, один столбец.
DISTINCT	Подавляет выборку дубликатов.
*	Выбирает все столбцы.
<i>столбец</i>	Выбирает заданный столбец.
<i>псевдоним</i>	Дает выбранным столбцам другие заголовки.
<i>таблица</i>	Указывает таблицу, содержащую столбцы.
WHERE	Ограничивает запрос строками, удовлетворяющими заданному условию.
<i>Условие</i>	Состоит из имен столбцов, выражений, констант и операторов сравнения.
GROUP BY	Разбивает строки на группы.
<i>Выражение_гру п-тирования</i>	Определяет столбец, по значениям которого группируются строки.
HAVING	Вывод конкретных групп.
<i>Условие_группы</i>	Задаёт условие отбора групп для вывода.
ORDER BY	Задаёт порядок вывода строк.
ASC	Сортирует строки в порядке возрастания; используется по умолчанию.
DESC	Сортирует строки в порядке убывания.

Простой запрос

В простейшей форме команда SELECT должна включать следующее:

```
SELECT [DISTINCT] {*, столбец [псевдоним]}  
FROM таблица;
```

Предложение SELECT - задает нужные столбцы. Звездочка (*) означает выбор всех столбцов таблицы. Предложение FROM указывает, в какой таблице находятся столбцы, заданные в предложении SELECT.

Пример: Вывод содержимого всех столбцов и строк таблицы S_DEPT:

```
SQL> SELECT *  
2> FROM S_dept;
```

Результат:

ID	NAME	REGION_ID
10	Finance	1
31	Sales	1
32	Sales	2
33	Sales	3
34	Sales	4
35	Sales	5
41	Operations	1
42	Operations	2
43	Operations	3
44	Operations	4
45	Operations	5
50	Administration	1

12 rows selected

Выборка заданных столбцов

Вывод можно ограничить столбцами, имена которых указаны через запятую в предложении SELECT.

Пример: Вывод всех номеров отделов, фамилий служащих и идентификационных номеров их менеджеров из таблицы S_EMP.

```
SQL> SELECT dept_id, last_name, manager_id  
2 FROM s_emp;
```

DEPT_ID	LAST NAME	MANAG
50	Velasquez	

41	Ngao	1
31	Nagayama	1
10	Quick-To-See	1
50	Ropeburn	1
41	Urguhart	2
42	Menchu	2
43	Biri	2
44	catchpole	2
...		
25 rows	Selected.	

Столбцы в команде SELECT указываются в последовательности, в которой должен осуществляться их вывод. Запятые между именами столбцов обязательны.

Формат заголовков столбцов, используемый по умолчанию

Заголовки столбцов и данные, состоящие из символов и даты, выравниваются в столбце по левому краю, а числа — по правому. В заголовках столбцов, содержащих символы и даты, лишние символы могут быть отброшены, но числовые заголовки отображаются полностью. По умолчанию заголовки столбцов выводятся в символах верхнего регистра. На выводе можно заменить заголовок столбца псевдонимом.

Псевдонимы столбцов

При выводе результатов запроса в среде SQL*Plus в качестве заголовков столбцов обычно используются их имена. Такие заголовки часто трудны для понимания и даже бессмысленны. Изменить заголовок столбца можно с помощью его псевдонима. Псевдоним указывается в списке команды SELECT сразу за именем столбца и отделяется от него пробелом. По умолчанию такие альтернативные заголовки выводятся в символах верхнего регистра и не могут содержать пробелов, если псевдоним не заключен в кавычки (" ").

Пример. Вывод фамилии, заработной платы и суммы компенсационных выплат за год для каждого служащего. Объем выплат за год вычисляется путем прибавления к заработной плате ежемесячной премии в размере 100 долларов и умножения суммы на 12. Назвать столбец ANNUAL_SALARY.

```
SQL> SELECT last_name, salary,
2      12 * (salary + 100) AS ANNUAL_SALARY
3      FROM s_emp;
```

Замечание. Для соответствия стандарту ANSI SQL-92 перед псевдонимом может находиться ключевое слово AS.

Псевдонимы столбцов в кавычках

Псевдоним, который содержит пробелы или специальные символы (например, # или _) или в котором различаются символы верхнего и нижнего регистров, должен быть заключен в кавычки (" ").

```
SQL> SELECT last_name, salary,
2      12 * (salary + 100) "Зарплата за год"
3      FROM s_emp;
```

Пример. Вывод фамилии, заработной платы, должности и вычисленных комиссионных.

```
SQL> SELECT last_name, title,
2      Salary*commission_pct/100 COMM
3      FROM s_emp;
```

Предотвращение выборки дубликатов строк

При отсутствии указаний с вашей стороны SQL*Plus включает в результаты запросов все строки, не изымая дубликаты. Ключевое слово DISTINCT, следующее сразу за словом SELECT, исключает дублирование строк.

Пример. Вывод всех названий отделов из таблицы S_DEPT.

```
SQL> SELECT name
2      FROM s_dept;
```

NAME

Finance

Sales

Sales
Sales
Sales
Sales
Operations
...
12 rows selected

Пример. Вывод всех *неповторяющихся* названий отделов из таблицы S_DEPT.

```
SQL> SELECT DISTINCT name  
2 FROM s_dept;
```

NAME

Administration
Finance
Sales
Operations

После квалификатора DISTINCT можно указать несколько столбцов. В этом случае он будет относиться ко всем выбранным столбцам.

Пример. Вывод всех возможных комбинаций должностей и номеров отделов.

```
SQL> SELECT DISTINCT dept_id, title  
2 FROM s_dept;
```



Для закрепления материала рекомендуется выполнить Практическое занятие 1.

Сортировка строк, возвращаемых запросом

Порядок строк, возвращаемых в результате запроса, не определен. Отсортировать строки можно с помощью предложения **ORDER BY**. Для сортировки можно задать выражение или позицию столбца в списке предложения SELECT.

Пример. Вывод из таблицы S_EMP фамилии, номера отдела и даты начала работы каждого служащего. Результат сортируется по фамилиям.

```
SQL> SELECT last_name, dept_id, start_date
2     FROM s_emp
3     ORDER BY last_name;
```

LAST_NAME	DEPT_ID	START_DATE
biri	43	07-APR-90
Catchpole	44	09-FEB-92
Chang	44	30-NOV-90
Dancs	45	17-MAR-91
Dumas	35	09-OCT-91
Giljum	32	18-JAN-92
Havel	45	27-FEB-91

По умолчанию строки сортируются в порядке возрастания:

- Вывод числовых значений производится от меньших к большему – например, 1 – 999.
- Вывод дат начинается с более ранних – например, 01-ЯНВ-92 предшествует 01-ЯНВ-95.
- Вывод символьных значений производится в алфавитном порядке – например, от А до Z.
- Неопределенные значения при сортировке по возрастанию выводятся последними, а при сортировке по убыванию – первыми.

Порядок сортировки, принятый по умолчанию, меняется на противоположный с помощью слова **DESC** после имени столбца в предложении **ORDER BY**.

Пример. Вывод из таблицы S_emp фамилии, номера отдела и даты найма каждого служащего. Результат сортируется таким образом, чтобы служащие, нанятые последними, возглавляли список.

```
SQL> SELECT last_name, dept_id, start_date
2     FROM s_emp
3     ORDER BY start_date DESC;
```


LAST_NAME	DEPT_ID	START_DATE
Urquhart	41	18-JAN-91
Chang	44	30-NOV-90
Patel	34	17-OCT-90
Menchu	42	14-MAY-90

...
25 rows selected.

В предложении **ORDER BY** можно указать псевдоним столбца.

Еще один способ сортировки результатов запроса — это сортировка по позиции. Он особенно полезен при сортировке по длинному выражению. Вместо повторного ввода выражения можно указать его позицию в списке **SELECT**

```
SQL> SELECT last_name, salary * 12
2     FROM s_emp
3     ORDER BY 2;
```

Сортировать результат можно и по нескольким столбцам. Предельным количеством столбцов сортировки является количество столбцов таблицы. Столбцы указываются в предложении **ORDER BY** через запятые. Для изменения порядка сортировки по какому-либо столбцу на обратный следует задать квалификатор **DESC** после его имени или позиции. Сортировать можно и по столбцам, не входящим в список **SELECT**.

Пример. Вывод фамилии, номера отдела и заработной платы всех служащих. Результат сортируется по номерам отделов, а внутри отделов – в порядке убывания заработной платы.

```
SQL> SELECT last_name, dept_id, salary
2     FROM s_emp
3     ORDER BY dept_id, salary DESC;
```

LAST_NAME	DEPT_ID	SALARY
Quick-To-See	10	1450
Nagayama	31	1400

Magee	31	1400
Giljum	32	1490
Sedeghi	33	1515
Nguyen	34	1525
Patel	34	795
...		

Ограничение количества выбираемых строк

Ограничить набор строк, возвращаемых в результате запроса, можно с помощью предложения **WHERE**. Предложение **WHERE** следует сразу за предложением **FROM** и задает условие, которое должно быть выполнено. Условие состоит из имен столбцов, выражений, констант и операторов сравнения.

Пример. Запрос для вывода имен, фамилий и должностей, служащих с фамилией «Magee».

```
SQL> SELECT first_name, last_name, title
2     FROM s_emp
3     Where Last_name='Magee';
```

В символьных строках различаются символы верхнего и нижнего регистров. Поэтому, для совпадения, фамилия должна быть написана строчными буквами и начинаться с заглавной буквы.

Оператор **BETWEEN**

Оператор **BETWEEN** используется для проверки вхождения значения в интервал значений (включая границы интервала). Нижняя граница должна быть указана первой.

Пример. Вывод имени, фамилии и даты найма служащих, нанятых между 9 мая и 17 июня 1991 года включительно.

```
SQL> SELECT first_name, last_name, start_date
2     FROM s_dept
3     Where start_date BETWEEN '09-MAY-91'
4     AND '17-JUN-91';
```

Оператор **IN**

Для проверки принадлежности значений к заданному списку

используется оператор IN.

Пример. Вывод номера, названия отдела и номера региона для отделов в регионах 1 и 3.

```
SQL> SELECT id, name, region_id
2     FROM s_dept
3     Where region_id IN(1,3);
```

Оператор LIKE

Используется для поиска строковых значений с помощью метасимволов (wildcards). Условия для поиска могут содержать символьные литералы или числа:

- ‘%’ означает отсутствие или некоторое количество символов;
- ‘_’ означает один символ.

Пример. Вывод фамилий служащих, начинающихся на букву “М”.

```
SQL> SELECT last_name
2     FROM s_emp
3     WHERE last_name LIKE 'M%';
```

Оператор LIKE может использоваться в качестве быстрого эквивалента некоторых операций BETWEEN.

Пример. Вывод фамилий и дат найма для служащих, принятых на работу в 1991 году.

```
SQL> SELECT last_name, start_date
2     FROM s_emp
3     WHERE start_date LIKE '%91';
```

В критерии поиска символы ‘%’ и ‘_’ можно сочетать с литералами в любой комбинации.

Пример. Вывод фамилий, второй буквой которых является “а”

```
SQL> SELECT last_name
2     FROM s_emp
3     WHERE last_name LIKE '_a%';
```

Параметр ESCAPE

Поиск символов ‘%’ и ‘_’ требует использования идентификатора ESCAPE (отменяющего специальное значение метасимволов).

Пример. Вывод названий фирм, содержащих сочетание “X_Y”.

```
SQL> SELECT name
2     FROM s_customer
3     WHERE name LIKE '%X\Y%' ESCAPE '\';
```

Оператор IS NULL

Неопределенные значения проверяются с помощью оператора IS NULL. Пользоваться оператором “=” для сравнения с неопределенными значениями не следует, так как неопределенное значение не может быть равно или не равно какому-то другому.

Пример. Вывод номера, наименования и кредитного рейтинга всех клиентов, не имеющих торгового представителя.

```
SQL> SELECT id, name, credit_rating
2     FROM s_customer
3     WHERE sales_rep_id IS NULL;
```



Для закрепления материала рекомендуется выполнить Практическое занятие 2.

Выборка по нескольким условиям

Использование сложных критериев для выборки возможно при сочетании условий с помощью операторов AND и OR.

Примечание. Оператор AND требует выполнения обоих условий.

Пример. Вывод фамилии, заработной платы, номера отдела и должности сотрудников, работающих в отделе с номером 41 и

имеющих должность «Stock Clerk».

```
SQL> SELECT last_name, salary, dept_id, title
2     FROM s_emp
3     WHERE dept_id = 41
4     AND title = 'Stock Clerk';
```

Примечание. Оператор OR требует выполнения хотя бы одного из условий.

Пример. Вывод фамилии, заработной платы и номера отдела для всех служащих, являющихся работниками склада или сотрудниками отдела 41.

```
SQL> SELECT last_name, salary, dept_id, title
2     FROM s_emp
3     WHERE dept_id = 41
4     OR title = 'Stock Clerk';
```

Пример. Вывод информации о служащих отдела 44 с зарплатой 1000 и более, а также о всех служащих отдела 42.

```
SQL> SELECT last_name, salary, dept_id
2     FROM s_emp
3     WHERE salary >= 1000
3     AND     dept_id = 44
4     OR     dept_id = 42;
```

Пример. Вывод фамилии, заработной платы и номера отдела для всех служащих отделов 44 и 42, зарплата которых составляет 1000 и более.

```
SQL> SELECT last_name, salary, dept_id
2     FROM s_emp
3     WHERE salary >= 1000
3     AND     (dept_id = 44
4     OR     dept_id = 42)
```

Пример. Вывод имени и фамилии каждого вице-президента строчными буквами, идентификатора пользователя с заглавной буквы и должности заглавными буквами.

```
SQL> SELECT LOWER(first_name || ' ' || last_name) VP,  
2     INITCAT(userid) USERID,  
3     UPPER(title) TITLE  
4     FROM s_emp  
5     WHERE title LIKE 'VP%';
```

Пример. Вывод имени и фамилии всех служащих с фамилией PATEL. Фамилия выводится в том виде, как она хранится в базе данных.

```
SQL> SELECT first_name, last_name  
2     FROM s_emp  
3     WHERE UPPER(last_name) = 'PATEL';
```

Пример. Вывод наименования и страны всех клиентов с хорошим кредитным рейтингом. Наименование и страна должны быть соединены.

```
SQL> SELECT CONCAT (name, country) CUSTOMER  
2     FROM s_customer  
3     WHERE LOWER(credit_rating) = 'good';
```

Пример. Вывод всех наименований товаров, три первых символа которых равны "Ace", и длины этих наименований.

```
SQL> SELECT name, LENGTH(name)  
2>     FROM s_product  
3>     WHERE SUBSTR (name, 1, 3) = 'Ace';
```

Примечание. Функция **SYSDATE** возвращает текущую дату и время. **DUAL** - это фиктивная таблица, используемая для просмотра SYSDATE.

Пример. Вывод фамилии и количества отработанных недель для служащих отдела 43.

```
SQL> SELECT last_name, ROUND((SYSDATE-start_date)/7,0)  
2>     FROM s_emp  
3>     WHERE dept_id = 43;
```

Пример. Вывод номера служащего, даты начала работы, количества отработанных месяцев и даты полугодовой аттестации для всех служащих, работающих менее 48 месяцев.

```
SQL> SELECT id, start_date,  
2>    MONTHS BETWEEN(SYSDATE, start_date) TENURE,  
3>    ADD_MONTHS(start_date,6) REVIEW  
4>    FROM s_emp  
5>    WHERE MONTHS_BETWEEN(SYSDATE, start_date)<48;
```

Пример. Вывод номера и даты каждого заказа, принятого торговым представителем номера 11. Вывод дат должен производиться в виде 08/92.

```
SQL> SELECT id, to_char(date_ordered, 'MM/YY') ORDERED  
2>    FROM s_ord  
3>    WHERE sales_rep_id=11;
```

Пример. Вывод сообщения о выполнении заказа с указанным номером для каждого заказа, сделанного 21 сентября 1992 года. Сообщение должно содержать сумму заказа.

```
SQL> SELECT 'Order' || TO_CHAR(id)  
2>    'was filled for a total of'  
3>    TO_CHAR(total, 'fm$9,999,999') NOTE  
4>    FROM s_ord  
5>    WHERE date_shipped = '21-SEP-92';
```

Пример. Вывод фамилии руководителя фирмы, у которого нет менеджера. Выходная строка должна указывать, что для этой фамилии номера менеджера нет.

```
SQL> SELECT last_name,  
2>    NVL(TO_CHAR(manager_id), 'no Manager')  
3>    FROM s_emp  
4>    WHERE manager_id IS NULL;
```

Пример. Вывод даты следующей пятницы, отстоящей на шесть месяцев от даты заказа. Выходная дата должна иметь следующий вид: Friday, March 12 th, 1993

```
SQL> SELECT TO_CHAR(NEXT_DAY(ADD_MONTHS
2>      (date_ordered,6), 'FRIDAY'),
3>      'fmDay, Month ddth, YYYY')
4>      "New 6 Month Review"
5> FROM s_ord
6> WHERE manager_id IS NULL;
```



Для закрепления материала рекомендуется выполнить Практическое занятие 3.

Группировка строк в запросе

По умолчанию все строки таблицы рассматриваются как одна группа. Для разбиения таблицы на меньшие группы строк используется предложение **GROUP BY** команды SELECT.

Синтаксис:

SELECT	{столбец, групповая_функция}
FROM	Таблица
[WHERE	условие]
[GROUP BY	<i>выражение_группирования</i>]
[ORDER BY	{Столбец, выражение} [ASC DESC]];

где

выражение_группирования определяет столбец, по значениям которого группируются строки.

Для получения сводной информации по каждой группе можно использовать групповые функции. 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.

Если в списке **SELECT** заданы столбцы одновременно с групповыми функциями, их список должен использоваться и в предложении **GROUP BY**. 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**.

Пример. Вывод номера, фамилии и номера отдела сотрудников, работающих в отделе 41:

```
SQL> SELECT id, last_name, dept_id DEPARTMENT
2> FROM s_emp
3> WHERE dept_id=41;
```

ID	LAST_NAME	DEPARTMENT
2	Ngao	41
6	Urguhart	41
16	Maduro	41
17	Smith	41

Пример. Вывод номера отдела и количества сотрудников, работающих в отделе 41:

```
SQL> SELECT dept_id, count(dept_id) NUMBER
2> FROM s_emp
3> WHERE dept_id=41
4> GROUP BY dept_id;
```

DEPT_ID	NUMBER
41	4

Пример. Вывод всех возможных кредитных рейтингов и количества клиентов в каждой категории. Столбец должен иметь заголовок “# Cust”.

```
SQL> SELECT credit_rating, COUNT(*) "# Cust"
2> FROM s_customer
3> GROUP BY credit_rating;
```

CREDIT_RATING	# CUST
EXELLENT	9
GOOD	3
POOR	3

Пример. Вывод всех должностей, кроме вице-президента, и соответствующей общей заработной платы за месяц. Список сортируется по общей заработной плате.

```
SQL> SELECT title, SUM(salary) PAYROLL
2> FROM s_emp
3> WHERE title NOT LIKE 'VP%'
4> GROUP BY title
5> ORDER BY SUM(salary);
```

TITLE	PAYROLL
President	2500
Warehouse	6157
Sales	7380
Stock	9490

Столбец, заданный в предложении GROUP BY, не обязательно должен быть задан в предложении SELECT. С другой стороны, если столбец из предложения GROUP BY входит в список SELECT, результат имеет больше смысла.

Если в одной и той же команде SELECT указаны как отдельные элементы данных, так и групповые функции, но пропущено предложение GROUP BY, описывающее эти отдельные элементы, выдается сообщение об ошибке.

Пример.

```

SQL> SELECT region_id, COUNT(name)
2>   FROM s_dept,
SELECT region_id, COUNT(name)
      *
ERROR at line 1:
ORA-00937: not a single-group group function
(Ошибка в строке 1:
групповая функция не для единственной группы).

```

Для исправления этой ошибки следует добавить предложение **GROUP BY**.

Группы внутри групп

Сводные результаты по группам и подгруппам можно получить путем указания более чем одного столбца в предложении **GROUP BY**. В следующем примере количество людей подсчитывается не только по отделам, но и по должностям. Порядок сортировки, используемый по умолчанию, определяется порядком столбцов в предложении **GROUP BY**.

Пример. Вывод количества служащих по должностям внутри отделов.

```

SQL> SELECT dept_id, title, COUNT(*)
2>   FROM s_emp
3>   GROUP BY dept_id, title;

```

DEPT_ID	TITLE	COUNT(*)
-----	-----	-----
...		
34	Sales Representative	1
34	Stock Clerk	1
35	Sales Representative	1
41	Stock Clerk	2
41	VP, Operations	1
...		

Предложение HAVING

Предложение **WHERE** для исключения групп не используется. Для исключения целиком некоторых групп следует пользоваться предложением **HAVING**.

Пример. Вывод номера отдела и средней заработной платы для отделов, где средняя заработная плата превышает 2000.

```
SQL> SELECT dept_id, AVG(salary)
2>   FROM s_emp
3>   WHERE AVG(salary) >2000
4>   GROUP BY dept_id,
WHERE AVG(salary) >2000
      *
ERROR at line 3:
ORA-00934: group function is not allowed here
(Использование здесь групповой функции невозможно)
```

Вместо этого для ограничения количества групп следует использовать предложение HAVING.

```
SQL> SELECT dept_id, AVG(salary)
2>   FROM s_emp
3>   GROUP BY dept_id
4>   HAVING AVG(salary) >2000;
```

DEPT_ID	AVG(SALARY)
----- 50	----- 2025

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

Синтаксис:

SELECT	{столбец, групповая_функция}
FROM	Таблица
[WHERE	условие]
[GROUP BY	выражение_группирования]
[HAVING	условие_группы]
[ORDER BY	{Столбец, выражение} [ASC DESC]];

где

HAVING	Вывод конкретных групп.
Условие_группы	Включает в выходной результат

только те группы, для которых заданное условие истинно (TRUE).

Если используется предложение HAVING, сервер баз данных, как правило, выполняет следующие действия.

- Группирует строки.
- Применяет групповую функцию.
- 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 происходят до того, как к группам из списка SELECT применяется ограничение, заданное в предложении HAVING.

Пример. Вывод должности и общей заработной платы для всех должностей с заработной платой более 5000 в месяц, кроме вице-президентов. Выходные строки сортируются по заработной плате.

```
SQL> SELECT title, SUM(salary) PAYROLL
2> FROM s_emp
3> WHERE title not like 'VP%'
4> GROUP BY title
5> HAVING SUM(salary) > 5000
6> ORDER BY SUM(salary) ;
```

Предложения здесь рассматриваются в следующем порядке:

- Если команда содержит предложение WHERE, то прежде всего отбираются строки, удовлетворяющие этому предложению.
- Выявляются группы, заданные предложением GROUP BY.
- Исключаются группы, не удовлетворяющие критерию, указанному в предложении HAVING.

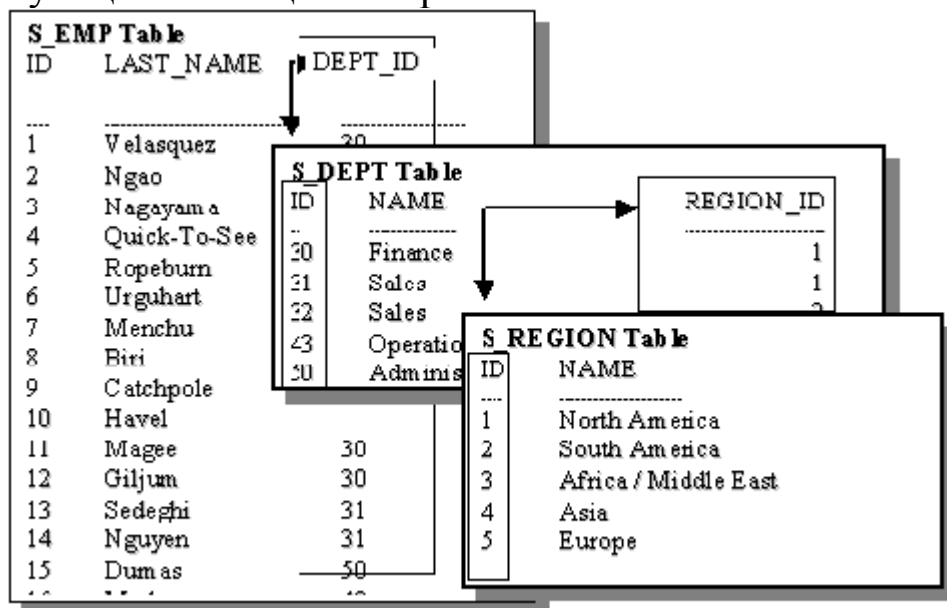
Предложение GROUP BY можно использовать без указания групповой функции в списке SELECT. Если отбор строк производится по результатам групповой функции, то использование как предложения GROUP BY, так и предложения HAVING обязательно.



Для закрепления материала рекомендуется выполнить практическое занятие 4 .

Выборка данных из нескольких таблиц

Если требуется получить данные из более, чем из одной таблицы базы данных, используется операция соединения, определенная в реляционной алгебре. Строки одной таблицы соединяются со строками другой согласно общим значениям в соответствующих столбцах – первичных и внешних ключей.



СУБД, как правило, поддерживает следующие способы соединения: *эквисоединение*; *не-эквисоединение*; *внешнее соединение*; *соединение таблицы с собой* (рекурсия).

При соединении нескольких таблиц возможен случай, когда в качестве результата запроса можно получить Декартово произведение, которое образуется, если:

- опущено условие соединения;
- условие соединения недействительно;
- все строки первой таблицы соединяются со всеми строками второй таблицы.

Во избежание получения Декартова произведения в предложение WHERE всегда необходимо включать допустимое условие соединения.

Простой запрос с соединением

Синтаксис:

```
SELECT      таблица.столбец, таблица.столбец
FROM        таблица1, таблица2;
WHERE       таблица1.столбец1           =
            таблица2.столбец2
```

где:

таблица.столбец

таблица и столбец, из которых производится выборка данных.

таблица1.столбец1=таблица2.столбец2

условие, соединяющее таблицы (или задающее их связь).

В предложении **WHERE** указывается допустимое для этого вида соединения логическое условие. Указывать имя таблицы в предложении **SELECT** необязательно, однако читать предложение легче, если каждому имени столбца предшествует имя таблицы. Если столбцы с одинаковыми именами имеются более, чем в одной таблице, имя таблицы перед именем столбца обязательно.

Эквисоединение

Этот вид соединения возникает, когда в качестве условия для соединения указывается точное равенство значений одного столбца значениям другого. Часто эти столбцы являются компонентами первичного и внешнего ключа.

Пример. Вывести фамилию служащего, номер и название отдела.

```
SQL> SELECT s_emp.last_name, s_emp.dept_id, s_dept.name
2> FROM s_emp, s_dept
3> WHERE s_emp.dept_id = s_dept.id;
```

LAST_NAME	DEPT_ID	NAME
-----	-----	-----
Velasquez	50	Administration
Ngao	41	Operations
Nagayama	31	Sales
Quick-To-See	10	Finance

Ropeburn	50	Administration
Urquhart	41	Operations
Menchu	42	Operations
Biri	43	Operations
Catchpole	44	Operations
Havel	45	Operations
Magee	31	Sales
Giljum	32	Sales
Sebeghi	33	Sales

Строки двух таблиц комбинируются и в результат включаются лишь те, у которых значения столбцов S_EMP.DEPT_ID и S_DEPT.ID равны.

Псевдонимы таблиц

Для различения одноименных столбцов из разных таблиц используются префиксы в виде имен таблиц. Использование префиксов в некоторых случаях увеличивает производительность запроса. Хотя одноименные столбцы из разных таблиц можно различать по их псевдонимам, однако использование имен таблиц становится обязательным, если имена столбцов в них совпадают. В случае, если имена таблиц очень громоздки, рекомендуется использовать вместо них псевдонимы (алиасы) таблиц. При этом необходимо следовать следующим правилам:

- перед именами столбцов рекомендуется указывать псевдонимы таблиц;
- псевдонимы таблиц действительны только для данной команды **SELECT**;
- если псевдоним таблицы создан, перед ссылкой на столбец следует указывать его, а не имя таблицы.

Пример. Вывод наименования клиента, номера региона и названия региона. Вместе с псевдонимами столбцов используются и псевдонимы таблиц для упрощения ссылок на них.

```
SQL> SELECT c.name "Customer Name", c.region_id "Region_ID",
2>    r.name "Region Name"
3>    FROM s_customer c, s_region r
4>    WHERE c.region_id = r.id;
```


Псевдонимы таблиц могут содержать до тридцати символов, но чем они короче, тем лучше. Действие псевдонима таблицы распространяется лишь на текущую команду SELECT. Использование псевдонимов таблиц позволяет уменьшить объем команды SQL.

Дополнительные условия поиска

Помимо условий соединения в предложении **WHERE** можно задавать и другие дополнительные критерии для ограничения получаемых в запросе строк. Поскольку условия соединения необходимо для исключения Декартова произведения, дополнительное условие добавляется с помощью логического оператора **AND**.

Пример. Вывод фамилии, номера отдела и названия отдела для сотрудника по фамилии “Menchu”.

```
SQL> SELECT e.last_name, e.dept_id, d.name
2> FROM s_emp e, s_dept d
3> WHERE e.dept_id = d.id
4> AND INITCAP(e.last_name) = 'Menchu';
```

LAST_NAME	DEPT_ID	NAME
-----	-----	-----
Menchu	42	Operations

Пример. Вывод фамилии, названия региона и процента комиссионных всех служащих, получающих комиссионные.

```
SQL> SELECT e.last_name, r.name, e.commission_pct,
2> FROM s_emp e, s_dept d, s_region r
3> WHERE e.dept_id = d.id
4> AND d.region_id = r.id
5> AND e.commission_pct <> 0;
```

LAST_NAME	NAME	COMMISSION_PCT
-----	-----	-----
Magee	North America	10
Giljum	Couth America	12.5

Sedeghi	Africa / Middle East	10
Nguyen	Asia	15
Dumas	Europe	17.5

Не-эквисоединения

Не-эквисоединение возникает в случае, если ни один столбец одной таблицы не соответствует точно столбцу другой таблицы. Условие соединения содержит оператор, не являющийся оператором равенства (=).

Пример. Вычисления категории служащего по заработной плате. Заработная плата должна быть между нижним и верхним значениями диапазона зарплат.

```
SQL> SELECT e.last_name name, e.title job, e.salary sal, s.grade
2> FROM s_emp e, s_salgrade s
3> WHERE e.salary BETWEEN s.losal AND s.hisal;
```

NAME	JOB	SAL	S.GRADE
-----	-----	-----	-----
SMITH	CLERK	800.00	1
ADAMS	CLERK	1,100.00	1
JAMES	CLERK	950.00	1
WARD	SALESMAN	1,250.00	2
MARTIN	SALESMAN	1,250.00	2
MILLER	CLERK	1,300.00	2
ALLEN	SALESMAN	1,600.00	3
TURNER	SALESMAN	1,500.00	3
JONES	MANAGER	2,975.00	4
BLAKE	MANAGER	2,850.00	4
CLARK	MANAGER	2,450.00	4
SCOTT	ANALYST	3,000.00	4
FORD	ANALYST	3,000.00	4
KING	PRESIDENT	5,000.00	5

14 rows selected

Здесь могут быть использованы и другие операторы, (например, <= и >=), но самый простой способ – оператор BETWEEN.

Внешние соединения

Внешнее соединение используется для выборки строк, не удовлетворяющих обычным условиям соединения. Оператором внешнего соединения является знак плюс, заключенный в скобки “(+)”. Этот оператор указывается с той стороны, где нет значения, по которому можно было бы произвести соединение.

Синтаксис:

SELECT	<i>таблица.столбец, таблица.столбец</i>
FROM	<i>таблица1, таблица2;</i>
WHERE	<i>таблица1.столбец1 = таблица2.столбец2(+)</i>

где:

таблица1.столбец1 = условие, соединяющее таблицы (или задающее их отношение).
таблица2.столбец2
(+) Символ внешнего соединения; может использоваться на любой стороне условия в предложении WHERE. (+) указывается после имени таблицы, в которой нет соответствующих строк.

Оператор внешнего соединения может использоваться только на одной стороне выражения (там, где недостаточно информации). Он возвращает строки таблицы, для которых в другой таблице нет соответствующей строки.

Условие, предполагающее внешнее соединение, не может использовать оператор **IN** и быть связанным с другими условиями с помощью оператора **OR**.

Пример. Вывод для каждого клиента его наименования, а также фамилии и идентификационного номера торгового представителя. В список включаются наименования даже тех клиентов, которые не имеют торгового представителя.

```
SQL> SELECT e.last_name, e.id, c.name  
2> FROM s_emp e, s_customer c  
3> WHERE e.id(+) = c.sales_rep_id  
4> ORDER BY e.id;
```

E.LAST_NAME	E.ID	C.NAME
Magee	11	Womansport
Magee	11	Beisbol Si!
Magee	11	Ojibway Retail
Magee	11	Big John's Sports Emporium
Giljum	12	Unisports
Giljum	12	Futbol Sonora
Sedeghi	13	Hanada Sport
Nguyen	14	Smms Athletics
Nguyen	14	Delhi Sports
Dumas	15	Kam's Sporting Goods
Dumas	15	Sportique
Dumas	15	Muench Sports
Dumas	15	Sporta Russia
Dumas	15	Kuhn's Sports
		Sweet Rock Sports

15 rows selected

Соединение таблицы с собой

Рекурсивное соединение возникает в том случае, когда строки таблицы соединяются со строками этой же самой таблицы. При этом в предложении **FROM** наличие двух таблиц имитируется путем использования двух различных псевдонимов одной и той же таблицы.

Пример. Вывод имен сотрудников и их менеджеров.

```
SQL> SELECT worker.last_name || ' works for ' ||
2>      manager.last_name
3>      FROM s_emp worker, s_emp manager
4>      WHERE worker.manager_id = manager.id;
```

WORKER.LAST_NAME || ' WORKS FOR ' || MANAGER.LAST_NAME

Ngao works for Velasquez
Nagayama works for Velasquez
Quick-To-See works for Velasquez
Ropeburn works for Velasquez
Urguhart works for Ngao
Menchu works for Ngao
Biri works for Ngao

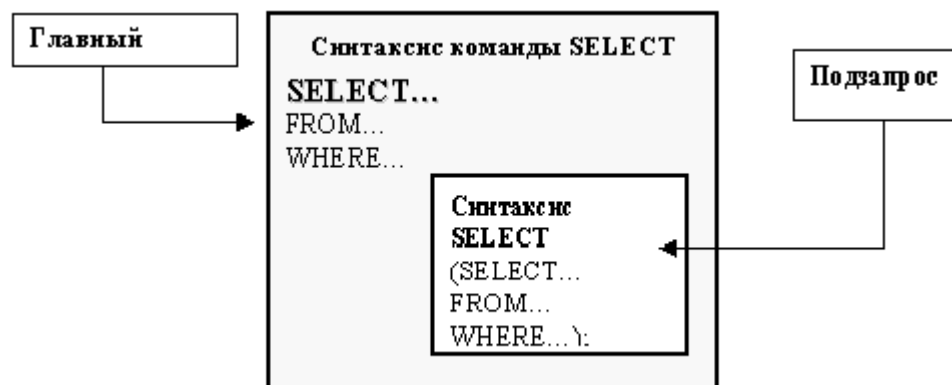
Catchpole works for Ngao
Havel works for Ngao
Magee works for Nagayama
Sedeghi works for Nagayama
...
24 rows selected.



Для закрепления материала рекомендуется выполнить практическое занятие 5.

Подзапросы

Подзапрос - это команда SELECT, вложенная в другую команду SQL (например, SELECT, CREATE, INSERT и т.д.). Механизм подзапросов является мощным и гибким инструментом языка SQL, позволяющим создавать сложные команды при работе с данными. Это может быть удобно для выборки строк таблицы по условию, зависящему от данных в самой таблице. Структура подзапроса представлена на рисунке.



Подзапросы очень полезны при написании команд SELECT для выборки значений по некоторому условию, значения операндов которого заранее неизвестны. Подзапросы можно использовать в разных предложениях команд SQL:

- предложение WHERE;
- предложение HAVING;
- предложение FROM команды SELECT или DELETE.

Синтаксис запроса с подзапросом:

```

SELECT      Список_выбора
FROM        Таблица
WHERE       выражение оператор
      (SELECT Список_выбора
        FROM Таблица
         .....);

```

где

оператор Оператор сравнения (например, >, = и т.д.) или оператор IN.

По количеству возвращаемых строк, удовлетворяющих условию запроса, подзапросы разделяют на **однотрочные** и **многострочные**. В однотрочных подзапросах в качестве операторов сравнения могут быть использованы операторы: “>”, “=”, “>=”, “<”, “<>”, “<=”. Для многострочных запросов правильным будет использование только оператора IN (NOT IN). Правила оформления подзапроса следующие:

- подзапрос должен быть заключен в круглые скобки;
- подзапрос должен находиться справа от оператора сравнения в логическом выражении;
- в подзапросе нельзя использовать предложение **ORDER BY**.

Как обрабатываются вложенные подзапросы?

Вложенная команда SELECT выполняется первой. Результат передается в условие главного запроса.

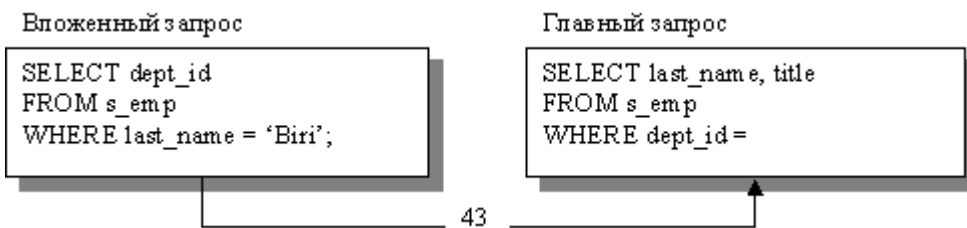
Пример. Выборка фамилий и должностей сотрудников, работающих в том же отделе, где работает сотрудник по фамилии “Biri”.

```

SQL> SELECT last_name, title
2>   FROM s_emp
3>   WHERE dept_id = (SELECT dept_id
4>                     FROM s_emp
5>                     WHERE last_name = 'Biri');

```

Диаграмма выполнения этого запроса следующая:



1. Первой выполняется команда SELECT вложенного блока запроса. Результат - 43.
2. Далее обрабатывается главный блок запроса. Результат подзапроса используется при этом для вычисления условия поиска.

Однострочные подзапросы

Однострочный подзапрос возвращает из вложенной команды SELECT только одну строку. В подзапросах этого типа используется **однострочный оператор сравнения**.

Пример. Вывод фамилий сотрудников, имеющих такую же должность, как Smith.

```
SQL>SELECT last_name
2>   FROM s_emp
3>   WHERE title =
4>   (SELECT title
5>   FROM s_emp
6>   WHERE last_name = 'Smith');
```

LAST_NAME	TITLE
-----	-----
Maduro	Stock Clerk
Smith	Stock Clerk
Nozaki	Stock Clerk
Patel	Stock Clerk
Newman	Stock Clerk
Markarian	Stock Clerk
Chang	Stock Clerk
Patel	Stock Clerk
Danes	Stock Clerk
Schwartz	Stock Clerk
10 rows selected.	

Групповая функция в подзапросе

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 фамилии, должности и зарплаты всех служащих с зарплатой ниже средней.

```
SQL> SELECT last_name, title, salary
2>   FROM s_emp
3>   WHERE salary <
4>   (SELECT AVG(salary)
5>   FROM s_emp);
```

LAST NAME	TITLE	SALARY
Urguhart	Warehouse Manager	1200
Menchu	Warehouse Manager	1250
Biri	Warehouse Manager	1100
Smith	Stock Clerk	940
Nozaki	Stock Clerk	1200
Patel	Stock Clerk	795
Newman	Stock Clerk	750
Markarian	Stock Clerk	850
Chang	Stock Clerk	800
Patel	Stock Clerk	795
Dancs	Stock Clerk	860
Schwartz	Stock Clerk	1100

12 rows selected.

Групповая функция AVG возвращает единственное значение.

Многострочные подзапросы

Подзапросы, возвращающие более одной строки, называются многострочными. В них следует использовать **многострочный оператор сравнения IN**, т.к. этот оператор предоставляет возможность выбора из списка значений.

Пример. Вывод списка служащих, приписанных к отделу “Finance” или региону 2.


```
SQL> SELECT last_name, first_name, title FROM s_emp
2>   WHERE dept_id IN (SELECT id
3>                     FROM s_dept
4>                     WHERE name = 'Finance' OR region_id = 2);
```

LAST_NAME	FIRST_NAME	TITLE
Quick-To-See	Mark	VP, Finance
Menchu	Roberta	Warehouse Manager
Giljum	Henry	Sales Representative
Nozaki	Akira	Stock Clerk
Patel	Vikram	Stock Clerk

Предложение HAVING с подзапросами

Подзапросы можно использовать не только в предложении **WHERE**, но и в предложении **HAVING**. При этом подзапросы выполняются первыми и результаты возвращаются в предложение **HAVING** главного запроса.

Пример. Вывод всех отделов, где средняя зарплата выше, чем в отделе 32.

```
SQL> SELECT dept_id, AVG(salary)
2>   FROM s_emp
3>   GROUP BY dept_id
4>   HAVING AVG(salary) > (SELECT AVG(salary)
5>                         FROM s_emp
6>                         WHERE dept_id = 32);
```

DEPT_ID	AVG(SALARY)
33	1515
50	2025

Пример. Поиск должности, чья средняя заработная плата больше средней по всем сотрудникам.

```

SQL> SELECT title, AVG(salary)
2>    FROM s_emp
3>    GROUP BY title
4>    HAVING AVG(salary) > (SELECT AVG(salary)
5>                          FROM s_emp);

```

TITLE	AVG(salary)
-----	-----
President	2550
Sales Representative	1480
VP, Administration	1550
VP, Finance	1450
VP, Operations	1450
VP, Sales	1400
6 rows selected.	



Для закрепления материала рекомендуется выполнить практическое занятие 7.

5. Определение структур данных

Структуры данных

База данных может содержать различные структуры данных, например: таблица (Table) – основная структура для хранения данных; представление (View) - подмножество данных из одной или нескольких таблиц; последовательность (Sequence) – структура для генерации значения первичного ключа; индекс (Index) – структура специализированных данных для повышения производительности некоторых запросов и т.д. Определять структуры объектов следует на этапе проектирования базы данных. Структуры данных могут быть созданы в любой момент и изменены в оперативном режиме. SQL содержит ряд команд (группы DDL), которые позволяют создавать, изменять, удалять и переименовывать структуры объектов, предназначенные для хранения данных:

<i>Команда</i>	<i>Описание</i>
----------------	-----------------

CREATE	Создает структуры для хранения данных (таблицы, представления, индексы, последовательности).
ALTER	Изменяет структуры данных или ограничения.
DROP	Удаляет структуры данных.
TRUNCATE	Усекает объекты, удаляя все записи и оставляя при этом саму структуру без изменений.
COMMENT	Добавляет комментарии к объекту.
RENAME	Переименовывает объект.

После выполнения этих команд DDL происходит автоматическая фиксация транзакций.

Создание таблиц

Синтаксис:

```
CREATE TABLE [схема.]таблица
( столбец тип_данных [DEFAULT выраж]
[ограничение_столбца],
...
[ограничение таблицы] );
```

где

<i>схема</i>	То же, что имя владельца.
<i>таблица</i>	Имя таблицы.
DEFAULT <i>выраж</i>	Задает значение по умолчанию, используемое при отсутствии значения в команде INSERT.
<i>столбец</i>	Имя столбца.
<i>тип_данных</i>	Тип данных и длина столбца.
<i>ограничение_столбца</i>	Ограничение целостности для столбца.
<i>ограничение_таблицы</i>	Ограничение целостности для таблицы в целом.

Параметр **DEFAULT**

Задает значение столбца по умолчанию при вставке строк. Допускаются строковые константы, выражения и такие функции

SQL, как SYSDATE и USER. Недопустимыми значениями являются имена других столбцов и псевдостолбцы. Тип данных значения, используемого по умолчанию, должен совпадать с типом данных, допустимым для этого столбца.

Ограничения

Ограничения реализуют правила по обеспечению целостности данных на уровне столбца или таблицы, предотвращают удаление таблицы при наличии ссылок.

Существуют следующие виды ограничений:

- **NOT NULL** – обязательность значений для столбца;
- **UNIQUE** – уникальность значений в столбце;
- **PRIMARY KEY** – первичный ключ;
- **FOREIGN KEY** – внешний ключ;
- **CHECK** – дополнительное ограничение на вид значений столбца.

По умолчанию сервер Oracle 7 задает имя любому ограничению в формате SYS_Cn. Для того, чтобы в дальнейшем можно было без особых неудобств изменять ограничения, необходимо присвоить им имена. Рекомендуется задавать имена ограничениям в следующем формате: [схема_[таблица_[столбец_]]]тип_ограничения (например, s_emp_id_pk). Видно, что ограничение с этим именем было создано для таблицы «s_emp» на уровне столбца «id», причем тип ограничения первичный ключ «pk» (сокращение от “Primary Key”).

Ограничения можно создавать либо в момент создания таблицы, либо после создания таблицы. Использовать ограничения следует на уровне столбца или на уровне таблицы.

Ограничение на уровне столбца

```
column [CONSTRAINT constraint_name] constraint_type,
```

Ограничение на уровне таблицы

```
column,...  
[CONSTRAINT constraint_name] constraint_type
```

```
(column,...),
```

Ограничение NOT NULL

Запрещает наличие неопределенных значений в столбце. Задается только на уровне столбца.

Пример. В этом примере ограничение NOT NULL задано для столбца PHONE, поскольку имя ограничению не присвоено, Oracle7 создает имя сам.

```
CREATE TABLE friend...  
  Phone VARCHAR2(15) NOT NULL,...  
  Last_name VARCHAR2(25)  
  CONSTRAINT friend_last_name_nn NOT NULL,...
```

Ограничение UNIQUE

Задаёт один или несколько столбцов, значение или комбинация значений в которых не могут повторяться в двух строках таблицы. Может быть задано как для таблицы, так и для столбца. Ограничение допускает наличие неопределенных значений, если задано на уровне столбца. Автоматически создает уникальный индекс.

Пример.

```
... phone VARCHAR2 (10)  
  CONSTRAINT s_emp_phone_uk UNIQUE,...
```

Ограничение PRIMARY KEY

Ограничение создает первичный ключ таблицы, значения которого должны быть уникальны (UNIQUE) и обязательно определены (NOT NULL). Каждая таблица может иметь только один первичный ключ. Может быть задано как на уровне таблицы, так и на уровне столбца. Автоматически создает уникальный индекс.

Пример.

```
... ID NUMBER (7)  
  CONSTRAINT s_emp_id_pk PRIMARY KEY,...
```

Ограничение FOREIGN KEY

Определяет столбец или набор столбцов в качестве внешнего ключа таблицы. Устанавливает связь с первичным или уникальным ключом в той же самой таблице или между таблицами. Значение внешнего ключа должно совпадать с существующим значением первичного ключа в родительской таблице или быть неопределенным (NULL). Может быть задано как на уровне таблицы, так и на уровне столбца.

При создании ссылок на таблицы других пользователей следует помнить, что

- ограничения должны ссылаться на таблицы в пределах одной базы данных;
- при ссылке на таблицу другого пользователя имя владельца следует использовать в качестве префикса к имени таблицы.

Для ограничения FOREIGN KEY могут быть использованы ключевые слова:

- **FOREIGN KEY** - определяет столбец в дочерней таблице как внешний ключ.
- **REFERENCES** - указывает таблицу и столбец в родительской таблице.
- **ON DELETE CASCADE** – при удалении строки в родительской таблице разрешает удаление зависимых строк в дочерней таблице.

Пример.

```
...Dept_id NUMBER (7)
      CONSTRAINT s_emp_dept_id_fk
      REFERENCES s_dept(id),...
```

Ограничение CHECK

Задаёт условие, которому должны удовлетворять значения столбца или группы столбцов в каждой строке таблицы. Может быть задано как на уровне таблицы, так и на уровне столбца. В выражениях, которые входят в состав ограничения, запрещены:

- ссылки на псевдостолбцы CURRVAL, NEXTVAL, LEVEL и ROWNUM;
- вызовы функций SYSDATE, UID, USER и USERENV;
- Запросы со ссылками на другие значения в других строках.

Пример.

```
CHECK (dept_id BETWEEN 31 AND 55),
```

Создание таблицы на основе бланка экземпляра

При создании таблицы многие производители СУБД рекомендуют использовать специальный «Бланк экземпляра таблицы», в который записываются все параметры создаваемой таблицы. На основе этого бланка затем легко составить правильную команду CREATE TABLE. Ниже приводится рекомендуемая последовательность шагов, которые необходимо предпринять при создании таблицы с помощью бланка экземпляра:

1. Создать командный файл. Включить команду CREATE TABLE *имя_таблицы*.
2. Отобразить имена столбцов, типы данных и их длину.
3. Ограничение NOT NULL задавать на уровне столбцов во всех случаях, кроме главного ключа (PRIMARY KEY).
4. Задать ограничение PRIMARY KEY.
5. Задать ограничения UNIQUE, CHECK и FOREIGN KEY.
6. Сохранить и выполнить командный файл.

Пример. На основании бланка экземпляра таблицы s_dept создается таблица с этим же именем.

Бланк экземпляра таблицы: S_DEPT

Имя столбца	ID	NAME	REGION_ID
Тип ключа	PK		FK
NN/UK	NN, U	NN, U2	U2
Таблица FK			REGION
Столбец FK			ID
Тип данных	NUMBER	CHAR	NUMBER
Длина	7	25	7
Пример данных	10	Finance	1
	31	Sales	1
	32	Sales	2

Соответствующая команда для создания таблицы

```

CREATE TABLE s_dept
(id          NUMBER(7)
 CONSTRAINT s_dept_id_pk PRIMARY KEY,
Name        VARCHAR2(25)
 CONSTRAINT s_dept_name_nn NOT NULL,
Region_id   NUMBER(7)
 CONSTRAINT s_dept_region_id_fk REFERENCES
s_region(id),
 CONSTRAINT s_dept_name_region_id_uk UNIQUE
(name, region_id);

```

Пример. Создание таблицы с именем S_EMP. Обратите внимание, что в таблице уже есть один первичный ключ, поэтому для столбца userid определяются два ограничения – NOT NULL и UNIQUE.

```

SQL> CREATE TABLE s_emp
2  (id NUMBER(7) CONSTRAINT s_emp_id_pk PRIMARY KEY,
3  last_name VARCHAR2(25) CONSTRAINT s_emp_last_name_nn NOT NULL,
4  first_name  VARCHAR2(25),
5  userid VARCHAR2(8) CONSTRAINT s_emp_userid_nn NOT NULL
6  CONSTRAINT s_emp_userid uk UNIQUE,
7  start_date  DATE DEFAULT SYSDATE,
8  comments   VARCHAR2(25),  manager_id NUMBER(7),
9  title      VARCHAR2(25),
10 dept_id    NUMBER(7)
11 CONSTRAINT s_emp_dept_id_fk REFERENCES s_dept (id),
12 salary     NUMBER(11,2),
13 commission_pct  NUMBER(4,2)
14 CONSTRAINT s_emp_commission_pct_ck CHECK (commission_pct
IN(10,12.5,15,17.5,20));

```

Создание таблицы посредством подзапроса

Создавать таблицы можно на основе уже существующих таблиц и при этом сразу же вставлять необходимые строки в новую таблицу. Для этого следует использовать команду CREATE TABLE с подзапросом.

Синтаксис:

```

CREATE TABLE таблица [(столбец,
столбец...)]
AS подзапрос;

```

где

<i>таблица</i>	имя таблицы,
<i>столбец</i>	имя столбца, значение по умолчанию и ограничение целостности,

подзапрос команда SELECT, определяющая строки для вставки в новую таблицу.

Применяя этот синтаксис для создания таблицы следует учитывать следующие особенности:

- количество заданных столбцов должно совпадать с количеством столбцов в подзапросе;
- для столбцов можно указать только имя, значение по умолчанию и правила целостности;
- в новую таблицу копируется только ограничение NOT NULL.

Пример. Создание таблицы с данными о всех сотрудниках отдела номер 41 из таблицы S_EMP:

```
CREATE TABLE emp_41
AS   SELECT id, last_name, userid, start_date
      FROM s_emp
      WHERE dept_id = 41;
```

С помощью команды DESCRIBE среды SQL*Plus можно убедиться в существовании любой таблицы базы данных и проверить определение ее столбцов.



Для закрепления материала рекомендуется выполнить практическое занятие 8.

Изменение таблиц и ограничений

После создания таблицы может потребоваться изменение ее структуры. Например, вы хотите включить новый столбец, переопределить существующий, отменить или разрешить ограничение. Эти действия можно произвести с использованием команды **ALTER TABLE**.

Добавление столбца

Синтаксис:

```
ALTER TABLE таблица
ADD (столбец тип_данных [DEFAULT выраж] [NOT NULL]
[, столбец тип_данных]...);
```

где

<i>таблица</i>	Имя таблицы.
<i>столбец</i>	Имя столбца.
<i>тип_данных</i>	Тип данных и длина столбца.
DEFAULT <i>выраж</i>	Определение значения нового столбца по умолчанию
NOT NULL	Ограничение NOT NULL для нового столбца

Новый столбец становится в таблице последним.

Пример. Добавление столбца COMMENTS в таблицу S_REGION.

```
SQL> ALTER TABLE s_region
2 ADD (comments VARCHAR2 (255));
Table altered.
```

Изменение столбца

Можно изменить (при соблюдении определенных условий) такие параметры столбца, как тип данных, размер, значение по умолчанию, ограничение NOT NULL.

Синтаксис:

```
ALTER TABLE таблица
MODIFY (столбец тип_данных [DEFAULT выраж] [NOT NULL]
[,столбец тип_данных]...);
```

Примечание. Можно увеличить ширину столбца или точность числовых значений; уменьшить ширину столбца, если столбец содержит неопределенные значения или в таблице нет строк; изменить значения по умолчанию для последующих добавлений. Использовать ограничение NOT NULL можно только в случае, если столбец содержит неопределенные значения. Изменение значения по умолчанию доступно только для тех строк, которые впоследствии будут вставляться в таблицу.

Пример. Увеличение максимальной ширины столбца TITLE таблицы S_EMP до 50 символов.

```
SQL> ALTER TABLE s_emp
2      MODIFY      (title VARCHAR2 (50));
Table altered.
```

Добавление ограничения

С помощью команды ALTER TABLE можно добавить или удалить (но не изменить) ограничения; разрешить или запретить действие ограничения.

Синтаксис:

```
ALTER TABLE таблица
ADD|MODIFY ([CONSTRAINT ограничение] тип (столбец));
```

Пример. Добавление ограничения FOREIGN KEY для таблицы S_EMP.

```
SQL> ALTER TABLE s_emp
2      ADD CONSTRAINT s_emp_manager_id_fk
3      FOREIGN KEY (manager_id)
4      REFERENCES s_emp(id).
Table altered.
```

Удаление ограничения

Синтаксис:

```
ALTER TABLE таблица
DROP [PRIMARY KEY | UNIQUE (столбец)]
      CONSTRAINT ограничение [CASCADE];
```

Пример. Удаление ограничения для менеджеров из таблицы S_EMP.

```
SQL> ALTER TABLE s_emp
2 DROP CONSTRAINT s_emp_manager_id_fk;
Table altered.
```

Пример. Удаление ограничения PRIMARY KEY для таблицы S_DEPT и связанного с ним ограничения FOREIGN KEY для столбца S_EMP.DEPT_ID.

```
SQL> ALTER TABLE s_dept
2 DROP PRIMARY KEY CASCADE;
Table altered.
```

Запрет и разрешение ограничений

Для запрета ограничения используется предложение DISABLE команды ALTER TABLE. Для одновременного запрета всех зависимых ограничений используется параметр CASCADE.

Синтаксис:

```
ALTER TABLE таблица
DISABLE | ENABLE CONSTRAINT ограничение [CASCADE];
```

Разрешение ранее запрещенного ограничения достигается с помощью предложения ENABLE. Разрешение ограничений UNIQUE и PRIMARY KEY вызывает автоматическое создание индексов UNIQUE и PRIMARY KEY.

Примеры.

```
SQL> ALTER TABLE s_emp
2 ENABLE CONSTRAINT s_emp_id_pk;
Table altered.
```

```
SQL> ALTER TABLE s_emp
2 DISABLE CONSTRAINT s_emp_id_pk;
Table altered.
```

Удаление таблицы

Синтаксис:

```
DROP TABLE таблица [CASCADE CONSTRAINTS];
```

где

<i>таблица</i>	Имя таблицы
CASCADE CONSTRAINTS	При указании этого параметра будут удалены все зависимые ограничения.

Команда удаляет все данные из таблицы, все незафиксированные транзакции фиксируются, все индексы удаляются. Откат этой команды **невозможен**.

Изменение имени объекта

Для изменения имени таблицы, представления, последовательности или синонима используется команда RENAME.

Синтаксис:

```
RENAME имя_1 TO имя_2
```

где

<i>имя_1</i>	Старое имя объекта.
<i>имя_2</i>	Новое имя.

Пример. Таблица S_ORD переименовывается в S_ORDER. При выполнении этой команды Вы должны быть владельцем объекта.

```
SQL> RENAME s_ord TO s_order;  
Table renamed.
```

Усечение таблицы

Команда TRUNCATE удаляет все строки таблицы и освобождает память, занятую под таблицу.

Синтаксис:

```
TRUNCATE TABLE таблица;
```

Пример. Удаляются все строки из таблицы S_ITEM, структура таблицы сохраняется.

```
SQL>TRUNCATE TABLE s_item;  
Table truncated.
```

Примечание. Вернуть удаленные строки после выполнения команды TRUNCATE невозможно. Удалять строки можно и командой DELETE.

Добавление комментариев к таблице

Синтаксис:

```
COMMENT ON таблица IS 'текст';
```

где

<i>таблица</i>	Имя таблицы, для которой добавляется комментарий.
<i>текст</i>	Текст комментария.

Пример. Добавляется комментарий к таблице S_EMP.

```
SQL> COMMENT ON TABLE s_emp  
2 IS 'Employee Information';  
Comment created.
```

Для отмены комментария используется пустая строка символов. Увидеть комментарии можно с помощью следующих представлений словаря данных: ALL_COL_COMMENTS, USER_COL_COMMENTS, ALL_TAB_COMMENTS и USER_TAB_COMMENTS.



Для закрепления материала рекомендуется выполнить практическое занятие 9.

6. Манипулирование данными

После создания собственных таблиц вам потребуется добавлять, изменять и удалять их строки с помощью команд обработки данных. Язык манипулирования данными (DML) является сердцем SQL. Для каждого добавления, изменения или удаления данных из базы данных выполняется команда DML. Совокупность команд DML, результаты действия которых еще не стали постоянными, организуют транзакцию.

Вставка новых строк в таблицу.

Команда INSERT позволяет вставлять только по одной строке в таблицу.

Синтаксис:

```
INSERT INTO таблица [ (столбец [, столбец... ] ) ]  
VALUES (значение [, значение... ] );
```

где

<i>таблица</i>	имя таблицы.
<i>столбец</i>	имя столбцов таблицы, в которые вносятся изменения.
<i>значение</i>	соответствующие значения столбцов.

Команда INSERT вставляет новую строку целиком, содержащую значения для каждого из столбцов (даже неопределенные). Для того, чтобы вставить значения для нескольких столбцов, необходимо их перечислить в предложении INSERT после названия таблицы (при этом значения для столбцов, не указанных в списке, считаются неопределенными). Значения указываются в порядке следования столбцов в структуре таблицы

(в последовательности, используемой по умолчанию) или в порядке их перечисления в списке предложения INSERT. Символьные значения и даты заключаются в апострофы.

Пример. В таблицу S_DEPT вставляется строка, содержащая значения для всех ее столбцов.

```
SQL> INSERT INTO s_dept
2     VALUES    (11, 'Finance', 2);
1 row created.
```

Вставка строк с неопределенными значениями

Неопределенные значения в таблицу можно вставить двумя способами: неявно, опуская значение для этого столбца в списке столбцов, или явно, указав в качестве неопределенного значения ключевое слово NULL (или пустую символьную константу '') в списке предложения VALUES. При неявном способе для столбцов, которым заданы значения по умолчанию (элемент DEFAULT выражение команды CREATE TABLE), будут присваиваться эти значения. Если такие значения не заданы – неопределенные значения.

Пример. Неопределенное значение вставляется неявно.

```
SQL> INSERT INTO s_dept (id, name)
2     VALUES    (12, 'MIS');
1 row created.
```

Пример. Явно указывается неопределенное значение (сравните с предыдущим примером).

```
SQL> INSERT INTO s_dept
2     VALUES    (13, 'Administration', NULL);
1 row created.
```

Пример. Имя пользователя и текущее значение даты вставляются с

использованием функций USER и SYSDATE.

```
SQL> INSERT INTO s_emp (id, first_name,  
2 last_name, userid, salary, start_date)  
3 VALUES (26, 'Dorna',  
4 'Smith', USER, NULL, SYSDATE);  
1 row created.
```

Пример. Вставка строки в таблицу S_EMP с указанием конкретной даты, используя функцию TO_DATE.

```
SQL> INSERT INTO s_emp (id, first_name,  
2 last_name, userid, salary, start_date)  
3 VALUES (26, 'Dorna',  
4 'Smith', USER, NULL,  
5 TO_DATE ('01-JAN-96 08:00',  
6 'DD-MON-YY HH:MI'));  
1 row created.
```

Пример. Запись информации об отделе в таблицу S_EMP. У пользователя запрашивается номер отдела, название отдела и номер региона.

```
SQL> INSERT INTO s_dept (id, name, region_id)  
2 VALUES (&department_id, '&department_name', &region_id);  
  
Enter value for department_id: 67  
Enter value for department_name: Accounting  
Enter value for region_id: 2  
1 row created.
```

Копирование строк из другой таблицы

Синтаксис:

```
INSERT INTO таблица (столбец[, столбец])  
подзапрос;
```

где

<i>таблица</i>	имя таблицы.
<i>столбец</i>	имя столбца в таблице, значение которого необходимо ввести.
<i>подзапрос</i>	Подзапрос, возвращающий строки в <i>таблицу</i> .

Пример. Запись информации о слушателе курса в таблицу S_EMP.

```
SQL> INSERT INTO history(id, last_name, salary,
2 title, start_date)
3 SELECT id, last_name, salary, title, start_date
4 FROM s_emp
5 WHERE start_date < '01-JAN-94';
10 rows created.
```

При таком способе вставки строк в таблицу предложение VALUES не используется. Количество столбцов, указанных в предложении INSERT, должно совпадать с количеством столбцов в подзапросе.

Обновление строк в таблице.

Синтаксис:

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

где

<i>таблица</i>	имя таблицы.
<i>столбец</i>	имя обновляемого столбца таблицы.
<i>значение</i>	соответствующие значения или подзапрос для этого столбца.
<i>условие</i>	Задаёт строки, которые необходимо изменить, и состоит из имен столбцов, выражений, констант, подзапросов и операторов сравнения.

Пример. Перевод служащего номер 2 в отдел 10.

```
UPDATE    s_emp
SET       dept_id = 10
WHERE     id = 2;

1 row updated.
```

Пример. Перевод служащего номер 1 в отдел 32 и повышение его заработной платы до 2550.

```
UPDATE    s_emp
SET       dept_id = 32, salary = 2550
WHERE     id = 1;

1 row updated.
```

Примечание. Нужно помнить о том, что если в команде предложение WHERE отсутствует, изменяются все строки таблицы.

Пример. Ввод данных о назначении комиссионных в размере 10 процентов каждому служащему компании.

```
UPDATE    s_emp
SET       commission_pct = 10

25 rows updated.
```

Примечание. Если вы пытаетесь обновить запись и новое значение столбца противоречит ограничению, выдается сообщение об ошибке.

Пример. Указанное в команде значение внешнего ключа не существует в родительской таблице для первичного ключа. Поэтому выдается сообщение о том, что “родительский ключ” не обнаружен, что является нарушением ограничения.

```
SQL> UPDATE s_emp
2      SET dept_id = 60
3      WHERE dept_id = 10;
Update s_emp
      *

ERROR at line 1:
ORA-02291: integrity constraint (USR.S_EMP_DEPT_ID_FK)
violated - parent key not found
```

Удаление строк из таблицы

Синтаксис:

```
DELETE FROM таблица  
[WHERE условие];
```

где

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

Примечание. Если предложение WHERE отсутствует, то удаляются все строки таблицы.

Пример. Удаление всех строк из таблицы TEST.

```
SQL> DELETE FROM test;  
25,000 rows deleted.
```

Пример. Удаление всех служащих, которые были приняты на работу после 1 января 1996 года.

```
SQL> DELETE FROM s_emp  
2 WHERE start_date>  
3 TO_DATE ('01.01.1996', 'DD.MM.YYY');  
1 row deleted.
```

Примечание. Если вы пытаетесь удалить запись, на значение которой имеется ограничение, выдается сообщение об ошибке.

Пример. Попытка удалить все отделы, расположенные в регионе номер 1.

```
SQL> DELETE FROM s_dept
2     WHERE     region_id;
delete from s_region
      *
ERROR at line 1:
ORA-02292: integrity constraint
(USR.S_EMP_DEPT_ID_FK) violated – child record found
```



Для закрепления материала рекомендуется выполнить практическое занятие 10.

7. Управление транзакциями

Совокупность команд SQL, результаты действия которых для базы данных представляет собой единое целое, называется *транзакцией* или логической единицей работы. В упрощенном представлении, транзакции содержат либо команды DML, выполняющие единое согласованное изменение данных, либо одну команду DDL или DCL.

Транзакции начинаются с выполнения первой исполняемой команды SQL и заканчиваются либо фиксацией изменений в базе данных, либо отказом от фиксации (откатом). Окончанием транзакции может служить одно из следующих событий:

- выполнение команды COMMIT или ROLLBACK;
- выполнение команды DDL или DCL (автоматическая фиксация);
- ошибка, завершение сеанса работы или аварийный останов системы (автоматический откат).

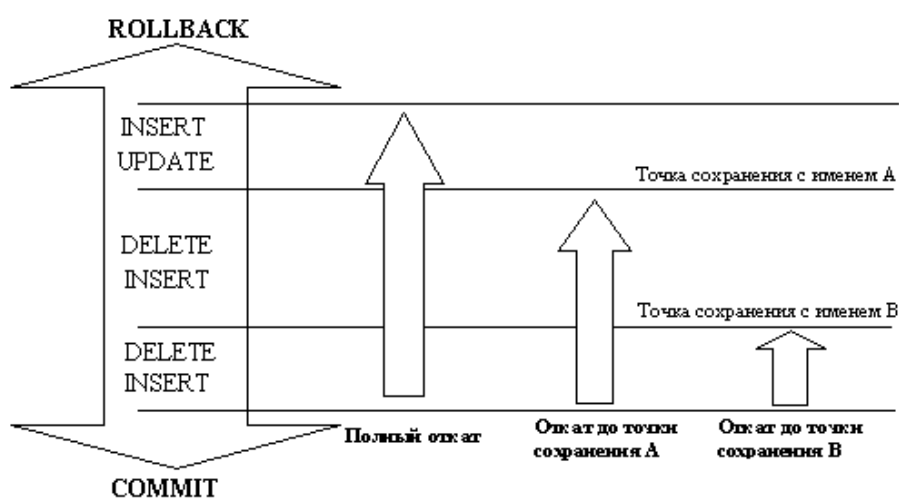
При выполнении команд SQL COMMIT или ROLLBACK происходит явная обработка транзакции. При этом обеспечивается согласованность данных; появляется возможность проверить изменения в данных прежде, чем сделать их постоянными; взаимосвязанные операции логически группируются.

Неявная обработка транзакций приводит к автоматической фиксации изменений или автоматическому откату.

Автоматическая фиксация изменений (COMMIT) происходит в следующих случаях:

- выполнение команды DDL (например, CREATE);
- выполнение команды DCL (например, GRANT);
- нормальный выход из SQL*Plus без явной отправки команды COMMIT или ROLLBACK.

Автоматический откат (ROLLBACK) выполняется в случае аварийного прекращения сеанса работы в SQL*Plus или отказа системы.



Состояние данных до и после завершения транзакции

Состояние данных перед выполнением команд COMMIT или ROLLBACK:

- Предыдущее состояние данных может быть восстановлено, т.к. изменения производятся в буфере базы данных.
- Текущий пользователь может просмотреть результаты своих операций DML с помощью команды SELECT.
- Другие пользователи *не могут* видеть результаты команд DML, выполняемых текущим пользователем.
- Изменяемые строки *блокируются*, и другие пользователи не могут обновлять их содержимое.

Состояние данных после выполнения команды COMMIT:

- Измененные данные записываются в базу данных.

- Предшествующее состояние данных теряется.
- Все пользователи могут видеть результаты.
- Измененные строки разблокируются, и другие пользователи получают доступ к ним для обработки данных.
- Все точки сохранения стираются.

Состояние данных после выполнения команды ROLLBACK:

- Все незавершенные изменения отменяются.
- Данные возвращаются в прежнее состояние.
- Блокировка строк, над которыми выполнялись операции, отменяется.

Фиксация изменений в данных

Синтаксис:

```
COMMIT;
```

Примеры. Создание нового отдела обучения и добавление данных, по крайней мере, об одном служащем. Фиксация изменений.

```
SQL> INSERT INTO s_dept (id, name, region_id)
2   VALUES (54, 'Education', 1);
1 row created.

SQL> UPDATE s_emp
2   SET dept_id = 54
3   WHERE id = 2;
1 row updated.

SQL> COMMIT;
Commit complete.
```

Откат результатов

Синтаксис:

```
ROLLBACK [TO метка];
```

где

метка Имя маркера, определяющего точку сохранения

Пример. Во время удаления записи из таблицы TEST случайно стёрты все данные этой таблицы. Ошибка исправляется, посылается правильная команда, и изменения фиксируются.

```
SQL> DELETE FROM test;
25,000 rows deleted.

SQL> ROLLBACK;
Rollback complete.

SQL> DELETE FROM test
2   WHERE      id = 100;
1 row deleted.

SQL> SELECT *
2   FROM test
3   WHERE id = 100;
no rows selected.

SQL> COMMIT;
Commit complete.
```

Откат до маркера

С помощью команды SAVEPOINT можно создать в текущей транзакции маркеры для отката. Откат до такого маркера выполняется с помощью команды ROLLBACK TO.

```
SQL> UPDATE...

SQL> SAVEPOINT update_done;
Sevepoint created

SQL> INSERT...

SQL> ROLLBACK TO update_done;
Rollback complete.
```

Откат на уровне команды

Если ошибка возникла при выполнении одной конкретной команды DML, отменяются только результаты этой команды. Для этого Oracle 7 использует неявную точку сохранения.

При откате на уровне команды все прочие изменения сохраняются и пользователь должен завершить транзакцию явно командой COMMIT или ROLLBACK.



Для закрепления материала рекомендуется выполнить практическое занятие 11.

8. Другие объекты базы данных

Последовательности

Для автоматической генерации номеров строк в таблице можно использовать такой объект базы данных, как последовательность. Последовательность – это объект базы данных, который создается одним пользователем, но может совместно использоваться несколькими пользователями. Основное назначение последовательности - автоматическая генерация уникальных чисел, которые обычно применяются для получения значений первичного ключа.

Создание последовательности

Синтаксис:

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

где

<i>Последовательность</i>	Имя последовательности.
INCREMENT BY <i>n</i>	Интервал между двумя последовательными номерами; <i>n</i> является целым числом. Если это предложение опущено, приращение при генерации чисел равно 1.
START WITH <i>n</i>	Первое генерируемое число в последовательности. Если это предложение опущено, последовательность начинается с 1.
MAXVALUE <i>n</i>	Максимальное значение, которое может генерировать последовательность.
NOMAXVALUE	Максимальное значение по умолчанию, 10^{27}

		равное 10^{27} .
MINVALUE <i>n</i>		Минимальное значение последовательности.
NOMINVALUE		Задает минимальное значение, равное 1.
CYCLE		Продолжается ли циклическая генерация чисел после достижения максимального или минимального значения.
NOCYCLE		
CACHE <i>n</i>		Количество чисел, которые Oracle7 распределяет предварительно и хранит в памяти. По умолчанию сервер хранит в кэш-памяти 20 значений.
NOCACHE		

Подчеркнутые параметры используются по умолчанию.

Пример. Создать последовательность S_DEPT_ID для первичного ключа таблицы S_DEPT. Параметр CYCLE использоваться не должен.

```
SQL> CREATE SEQUENCE s_dept_id
2 INCREMENT BY 1
3 START WITH 51
4 MAXVALUE 9999999
5 NOCACHE
6 NOCYCLE;
Sequence created.
```

Проверка параметров последовательности

Проверить значения параметров последовательности можно в таблице USER_SEQUENCES словаря данных. Столбец LAST_NUMBER содержит следующее свободное число.

```
SQL> SELECT sequence_name, min_value,
2 max_value, increment_by,
3 last_number
4 FROM user_sequences;
```

Изменение последовательности

Изменение шага приращения, максимального и минимального значений, режима циклической генерации значений и кэширования определяется командой ALTER SEQUENCE.

Синтаксис:

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

Примечание. Для изменения параметров необходимо быть владельцем последовательности или иметь для нее привилегию ALTER. Команда влияет только на числа, генерируемые после изменения. Чтобы начать генерацию с другого числа, необходимо удалить последовательность и создать заново.

Генерация значений последовательности

Псевдостолбец **NEXTVAL** генерирует следующее свободное число в последовательности. При каждой ссылке на этот столбец он возвращает уникальное значение - даже для разных пользователей. Псевдостолбец **CURRVAL** выдает текущее число в последовательности. Чтобы **CURRVAL** содержал значение, необходимо прежде сгенерировать значение последовательности используя **NEXTVAL**.

Пример. Включение нового отдела под названием "Finance" в регионе 2.

```
SQL> INSERT INTO s_dept(id, name, region_id)
2     VALUES (s_dept_id.NEXTVAL,
3     'Finance', 2);
1 row created.
```

Пример. Просмотр текущего значения последовательности **S_DEPT_ID**.

```
SQL> SELECT s_dept_id.CURRVAL
2     FROM SYS.dual;
```

Примечания. Запись значений последовательности в свехоперативную память (кэш) ускоряет доступ к ним. При использовании последовательностей возможные пропуски значений при

генерации чисел по причинам: откат транзакции, отказ системы, использование последовательности другим пользователем или для другой таблицы.

Удаление последовательности

Удаление последовательности из словаря данных производится с помощью команды **DROP SEQUENCE**. После удаления последовательности ссылки на нее невозможны.

Пример. Удаление последовательности S_DEPT_ID.

```
SQL> DROP SEQUENCE s_dept_id;
Sequence dropped.
```



Для закрепления материала рекомендуется выполнить практическое занятие 12.

Представления

Представление – это логический образ таблицы, созданный 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**.

Создание представлений

Представление создается командой **CREATE VIEW** с использованием подзапроса.

Синтаксис:

```

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

```

где

FORCE	Создание представления независимо от того, существуют ли базовые таблицы.
NOFORCE	Создание представления только при условии существования базовых таблиц (умолчанию).
<i>представление</i>	Имя представления.
<i>псевдоним</i>	Имена выражений, выбранных в запросе для представления.
<i>подзапрос</i>	Полная команда SELECT.
WITH CHECK OPTION	Режим, при котором добавлять или обновлять можно только строки, доступные в представлении.
<i>ограничение</i>	Имя, присвоенное ограничению CHECK OPTION.
WITH READ ONLY	Запрет применения к данному представлению операций DML.

Для подзапроса, участвующего при создании представления, действуют все правила, определенные для подзапроса (см. команду SELECT с подзапросом). Получить описание представления можно точно так же, как и описание таблицы (команда DESCRIBE среды SQL*Plus или команда SELECT * FROM *имя_представления*). Выборка данных из представления производится посредством команды SELECT со ссылкой на это представление.

Рекомендуется использовать представление для:

- ограничения доступа к базе данных;
- упрощения запросов;
- достижения независимости данных от приложений пользователя;
- организации различных способов показа одних и тех же данных.

Представления делятся на *простые* и *сложные*. В следующей таблице приведены ограничения, связанные с простыми или сложными представлениями.

	<i>Простые представления</i>	<i>Сложные представления</i>
<i>Количество таблиц</i>	Одна	Одна или больше
<i>Содержит функции</i>	Нет	Да
<i>Содержит группы данных</i>	Нет	Да
<i>Операции DML над представлением</i>	Да	Нет

Пример. Создается представление EMPVU45, включающее учетный номер, фамилию и должность каждого служащего отдела номер 45.

```
SQL> CREATE VIEW empvu45
2   AS SELECT id, last_name, title
3   FROM s_emp
4   WHERE dept_id = 45;
View created
```

Изменение представления

Изменить параметры представления можно командой **CREATE OR REPLACE VIEW**. При этом будет создано новое представление с тем же именем.

Пример. Изменение представления EMPVU45 с добавлением псевдонимов для каждого столбца. Псевдонимы столбцов в команде указаны в том же порядке, что и столбцы в подзапросе.

```
SQL> CREATE OR REPLACE VIEW empvu45
2   (id_number, employee, job)
3   AS SELECT id, last_name, title
4   FROM s_emp
5   WHERE dept_id = 45;
View created
```

Пример. Создание сложного представления с групповыми функциями для выборки данных из двух таблиц.

```

SQL> CREATE VIEW dept_sum_vu
2   (name e, minsal, maxsal, avgsal)
3   AS SELECT d.name e, MIN(e.salary),
4   MAX(e.salary), AVG(e.salary)
5   FROM s_emp e,s_dept d
5   WHERE e.dept_id= d.id
6   GROUP BY d.name;
View created.

```

Правила выполнения операций DML над представлением.

Операции DML могут быть выполнены только с простым представлением! При этом даже для простого представления существуют некоторые ограничения.

Удаление строк невозможно, если представление содержит следующее:

- ключевое слово DISTINCT.

Нельзя изменять данные в представлении, если оно содержит:

- одно из вышеуказанных условий;
- столбцы, описанные как выражения;
- псевдостолбец ROWNUM.

Нельзя добавлять данные в представление, если оно содержит:

- одно из вышеуказанных условий;
- какие-либо столбцы NOT NULL, не выбранные представлением.

Параметр WITH READ ONLY запрещает операции DML над этим представлением. Этот параметр актуален только для простого представления, сложное представление всегда доступно пользователю только для выборки данных.

Предложение WITH CHECK OPTION позволяет задать ограничения, накладываемые на представление. Условие для ограничения значений, изменяемых или добавляемых через представление, содержится в предложении WHERE подзапроса в определении представления. При использовании такого представления необходимо следить за тем, чтобы результаты операций DML оставались в пределах домена представления.

Пример. Попытка изменить номер отдела для какой-либо строки в представлении закончится неудачей, т.к. при этом нарушится ограничение CHECK OPTION.

```
SQL> CREATE OR REPLACE VIEW empvu41
2   AS SELECT *
3   FROM s_emp
4   WHERE dept_id = 41
5   WITH CHECK OPTION CONSTRAINT empvu41_ck;
View created.
```

Пример. Создание представления с параметром READ ONLY. При попытке выполнить операцию DML над какой-либо строкой представления сервер Oracle7 выдает сообщение об ошибке ORA-01732.

```
SQL> CREATE OR REPLACE VIEW empvu45
2   (id_number, employee, job)
3   AS SELECT id, last_name, title
4   FROM s_emp
5   WHERE dept_id = 45
6   WITH READ ONLY;
View created.
```

Пример. Получение имени представления и его параметров из таблицы словаря данных USER_VIEWS.

```
SQL> SELECT view_name, text
2   FROM user_views;
```

Удаление представления

Удалить представление можно с помощью команды **DROP VIEW**. Удаление представления не вызывает потери данных.

Пример.

```
SQL> DROP VIEW empvu45;
View dropped.
```



Для закрепления материала рекомендуется выполнить практическое занятие 13.

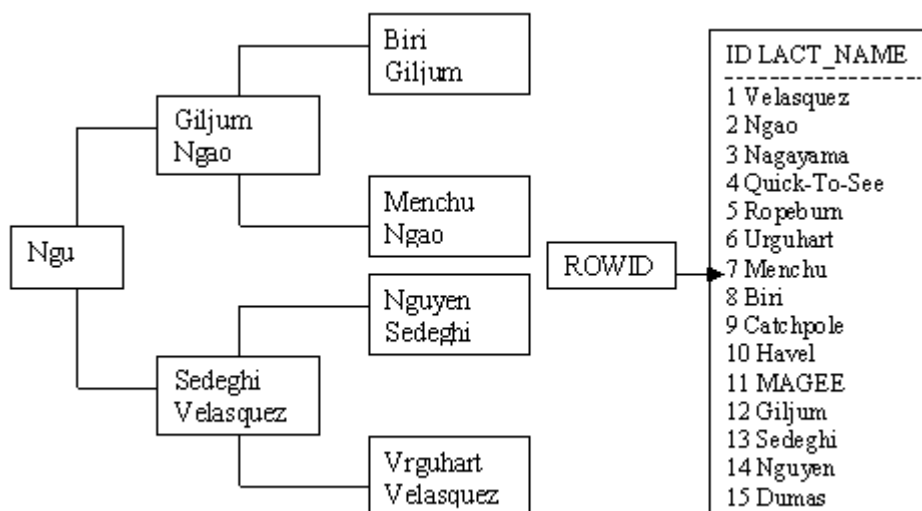
Индексы

Объект базы данных, обеспечивающий прямой и быстрый доступ к строкам в таблице, называется индексом. Индексы используются сервером базы данных для ускорения выборки строк с помощью указателя. При использовании индексов уменьшается количество операций обмена с внешней памятью за счет использования быстрого метода поиска данных. Индексы не зависят от таблицы, для которой они были созданы.

Индекс можно создать автоматически (при описании ограничений, типа PRIMARY KEY или UNIQUE) или вручную (используя команду CREATE INDEX).

Как правило, для представления индексов используется структура B-дерева. Каждый индекс состоит из значений столбцов и указателей (ROWID), организованных в виде страниц (ветвей). При выполнении запроса осуществляется поиск по ветвям дерева до обнаружения листа со значением, содержащим указатель.

Структура индекса в виде B-дерева:



Типы индексов:

- уникальный - обеспечивает уникальность значений в столбце;
- неуникальный - ускоряет запросы;
- простой - в индексе использован только один столбец;
- составной или сложный - в индексе использовано несколько столбцов.

Создание индекса

Синтаксис:

```
CREATE INDEX index
ON table (column [, column]...);
```

где

<i>index</i>	Имя создаваемого индекса.
<i>table</i>	Имя таблицы, на основе которой он создается.
<i>column</i>	Имя столбца (столбцов) для создания индекса.

Пример. Создание индекса для таблицы S_EMP на столбце LAST_NAME.

```
SQL> CREATE INDEX s_emp_last_name_idx
2     ON S_EMP(LAST_NAME);
Index created.
```

В каких случаях оправдано создание индекса:

- Столбец часто используется в предложении WHERE или условии 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 большинство запросов будут выбирать менее 10-15% строк.

Когда не следует создавать индекс:

- Таблица небольшого размера.
- Столбцы не очень часто используются как параметры в условиях при запросе.
- Большая часть запросов будет выбирать более, чем 10-15% строк.
- Таблица часто обновляется.

Определение индекса содержится в представлении словаря данных USER_INDEXES. Представление USER_IND_COLUMNS содержит имя индекса, имя таблицы и имя столбца.

```
SQL> SELECT ic.index_name, ic.column_name,  
2         ic.column_position col_pos, ix.uniqueness  
3       From user_indexes ix, user_ind_columns ic  
4       WHERE ic.index_name = ix.index_name  
5       AND ic.table_name = 'S_EMP';
```

Удаление индекса

Удаление индекса из производится командой **DROP INDEX**.

Пример. Удаление индекса с именем s_emp_last_name_idx.

```
SQL> DROP INDEX s_emp_last_name_idx;  
Index dropped.
```



Для закрепления материала рекомендуется выполнить практическое занятие 14.

9. Задания для практических занятий

Практическое занятие 1: выборка строк

- 1 Дайте ответы на следующие вопросы:
 - a. Команды SQL всегда хранятся в буфере? (*Да/Нет*).
 - b. Команды SQL *Plus помогают запрашивать данные? (*Да/Нет*).
- 2 Покажите структуру таблицы S_DEPT. Выберите всю информацию из таблицы S_DEPT.
- 3 Покажите структуру таблицы S_CUSTOMER. Выполните с этой таблицей следующие действия:
 - a. Получите всю информацию из таблицы S_CUSTOMER.
 - b. Получите список названий и номеров телефона всех фирм-клиентов.
 - c. Получите список названий и номеров телефона всех фирм-клиентов; номер телефона должен быть в строке первым.

Практическое занятие 2: использование в запросах однострочных функций

- 1 Дайте ответы на следующие вопросы:
 - a. Однострочные функции обрабатывают множество строк для получения единственного результата? (*Да/Нет*).
 - b. К значениям дат можно применять любые арифметические операторы? (*Да/Нет*).
 - c. Как называется функция, возвращаемая текущую дату?
- 2 Выведите номер служащего, его фамилию и заработную плату, повышенную на 15 % и округленную до целого.
- 3 Выведите фамилию каждого служащего и должность в скобках.
- 4 Для каждого служащего выведите фамилию, дату найма и дату пересмотра зарплаты, которая приходится на первый понедельник после шести месяцев работы. Формат даты на выводе: «день-месяц-год».
- 5 Выведите все наименования товаров, включающие слово “ski”.
- 6 Для каждого служащего вычислите количество месяцев со дня начала работы до настоящего времени. Результаты отсортируйте по количеству отработанных месяцев. Количество месяцев округлите до ближайшего целого.
- 7 (*) Выведите фамилию каждого служащего и день недели, когда он был нанят на работу. Результаты отсортируйте по дням недели, начиная с понедельника.
- 8 (*) Составьте запрос для получения следующей информации по каждому служащему: *<имя служащего>* зарабатывает *<зарплата>* в месяц, но желает *<утроенная зарплата>*.
Например: *ALLEN зарабатывает 1100 в месяц, но желает 3300.*

Практическое занятие 3: ограничение количества выбираемых строк

- 1 Дайте ответы на следующие вопросы:
 - a. Сортировка по столбцу, который не был выбран, невозможна? (*Да/Нет*).
 - b. Следующая команда SELECT будет успешно выполнена? (*Да/Нет*).

```
SQL> Select Last_name, title, salary Ann_sal
2 From S_emp
3 Where Last_name = 'Dancs';
```

- c. Следующая команда SELECT будет успешно выполнена? (Да/Нет).

```
SQL> Select *
2 From S_emp
3 Where Salary*12 ==9600;
```

- d. Сколько ошибок содержит следующая команда SELECT? Укажите их.

```
SQL> Select Id, last_name,
2 Salary x 12 Annual SALARY
3 From S_emp
4 Where Sal > 3000
5 And Start_date LIKE %84;
```

- 2 Выполните следующие действия, пользуясь таблицей S_CUSTOMER.

- Создайте запрос для вывода названия, номера и кредитного рейтинга всех фирм-клиентов, имеющих торгового представителя под номером 11. Сохраните команду SQL в файле.
- Выполните запрос из файла.
- Измените команду присвоив столбцам заголовки Company, Company ID, Rating. Выполните запрос еще раз. Формат вывода показан ниже. Сохраните отредактированный запрос в файле.

Company	Company ID	Rating
-----	-----	-----
Womansport	204	GOOD
Beisbol Si!	209	GOOD
Big John's Sports	213	GOOD
Emporium		
Ojibway Retail	214	GOOD

- d. Загрузите файл в буфер SQL. Отсортируйте результат запроса в порядке убывания номеров клиентов. Выполните запрос.
- 3 Выполните следующие упражнения с таблицей S_EMP.
- a. Покажите структуру таблицы.
 - b. Получите имя пользователя для сотрудника с номером 23.
 - c. Получите список имен, фамилий и номеров отделов для служащих отделов 10 и 50. Отсортируйте список по фамилиям в алфавитном порядке. Объедините имя с фамилией и назовите столбец “Employees”.
 - d. Получите информацию по всем служащим, в фамилии которых имеется буква “s”.
 - e. Выведите имя пользователя и дату начала работы всех служащих, нанятых между 14 мая 1990 года и 26 мая 1991 года. Результаты запроса отсортируйте по убыванию дат начала работы.
 - f. (*) Напишите запрос для вывода фамилий и заработной платы всех служащих, месячный заработок которых не лежит в интервале от 1000 до 2500.
 - g. (*) Получите список фамилий и заработной платы всех служащих отделов 31, 42, и 50, зарабатывающих более 1350. Назовите столбец “Employee Name”, а столбец заработной платы – “MONTHLY SALARY”.
 - h. Получите список фамилий и дат найма всех служащих, пришедших в 1991г.
 - i. (*) Получите список имен и фамилий всех служащих, не имеющих менеджера.
- 4 (*) Выполните упражнения, используя таблицу S_PRODUCT.
- a. Покажите структуру таблицы.
 - b. Перечислите в алфавитном порядке все товары, названия которых начинаются с “Pro”.
 - c. Выведите названия и краткие описания всех продуктов, в описании которых содержится слово “bicycle”.
 - d. Выведите все краткие описания. Сравните с результатом предыдущего упражнения. Содержал ли ответ, все описания со словом “bicycle”?

Практическое занятие 4: использование в запросе групповых функций

1. Определите истинность следующих утверждений.
 - a. Групповые функции обрабатывают большое количество строк для получения одного результата? *(Да/Нет)*.
 - b. Во время вычисления групповых функций учитываются неопределенные значения? *(Да/Нет)*.
 - c. Предложение HAVING используется для исключения строк из расчета для группы? *(Да/Нет)*.
 - d. Предложение HAVING используется для исключения групп из выходных результатов? *(Да/Нет)*.
2. Выведите наибольшую и наименьшую общую сумму заказа из таблицы S_ORD.
3. Составьте запрос для вывода минимальной и максимальной заработной платы по всем должностям в алфавитном порядке.
4. Определите количество менеджеров без вывода информации о них.
5. Выведите номер каждого заказа и количество позиций в нем. Столбец с количеством позиций озаглавьте “Number of Items”.
6. Выведите номер каждого менеджера и заработную плату самого низкооплачиваемого из его подчиненных. Исключите группы с минимальной заработной платой менее 1000. Отсортируйте результаты по размеру заработной платы.
7. Какова разница между самой высокой и самой низкой заработной платой?
8. (*) Для каждого вида товара, заказанного, по крайней мере, три раза, выведите номер этого товара и количество заказов на него. Столбец с количеством заказов на товар озаглавьте “Times Ordered”. Отсортируйте данные по номерам заказанных товаров.
9. (*) Получите список номеров и названий всех регионов с указанием количества отделов в каждом регионе.
10. (*) Для каждого заказа с общим количеством заказанных товаров 100 или более выведите номер заказа и общее количество заказанных товаров в нем. (Если, например, заказ номер 99 содержит заказ на один товар в количестве 30, а на другой – в количестве 75, то общее количество заказанных товаров равно 105).
11. (*) Выведите наименование каждого клиента и количество сделанных им заказов.

Практическое занятие 5: выборка данных из нескольких таблиц

Примечание: Для выполнения следующих упражнений используются таблицы S_EMP, S_DEPT, S_CUSTOMER, S_REGION, S_ORD, S_ITEM, S_PRODUCT.

1. Напишите отчет, содержащий фамилию, номер отдела и название отдела для каждого служащего.
2. Составьте запрос для вывода фамилии, названия отдела и названия региона для всех служащих, получающих комиссионные.
3. Покажите фамилию и название отдела для сотрудника по фамилии "Smith".
4. Выведите наименование товара, номера товара и заказанное количество по всем позициям заказа номер 106.
5. Выведите номер каждого клиента и фамилию его торгового представителя. Отсортируйте список по фамилиям.
6. (*) Для всех заказчиков и всех их заказов выведите номер заказчика, его наименование и номер заказа. Даже если клиент не делал заказ, его номер и наименование должны быть включены в список. Примерный вид отчета:

Customer ID	Customer Name	Order ID
201	Unisports	97
202	Simms Atheletics	98
...
207	Sweet Rock Sports	

7. (*) Выведите фамилии и номера всех служащих вместе с фамилиями и номерами их менеджеров.
8. (*) Для каждого заказчика, общая сумма заказа которого превышает 100 000, выведите наименование заказчика, заказанные им товары, их количество.

Практическое занятие 6: определение переменных во время выполнения

1. Дайте ответы на следующие вопросы:

- a. Значение подставляемой переменной с одним амперсандом запрашивается только один раз? (*Да/Нет*).
- b. Команда ACCEPT является командой SQL? (*Да/Нет*).

Примечание. Следующие упражнения предполагают использование таблиц S_EMP, S_CUSTOMER, S_PRODUCTS.

2. Создайте командный файл для выборки информации о каждом служащем, дата начала работы которого находится в пределах определенного диапазона. Выходные данные должны включать идентификатор пользователя, а также имя и фамилию, связанные вместе. Даты, определяющие временной диапазон, должны запрашиваться у пользователя с помощью команды ACCEPT. Используйте формат “MM/DD/YY”. Сохраните этот командный файл под именем “test62.sql”.
3. Создайте командный файл для получения списка имен и номеров клиентов. Условие поиска должно позволять производить поиск имен независимо от регистра символов. Сохраните файл под именем “test63.sql”.
4. Создайте отчет, содержащий имя торгового представителя, имя клиента и общую сумму продаж по каждому клиенту. Подсчитайте общую сумму по каждому торговому представителю. Пользователю должно быть выдано приглашение ввести номер региона. Сохраните файл под именем “test64.sql”.

Практическое занятие 7: подзапросы

1. Ответьте на следующие вопросы:
 - a. Если используется подзапрос, то какой запрос выполняется первым?
 - b. Сколько раз выполняется первый запрос?
 - c. Если подзапрос возвращает более одного значения, то нельзя использовать оператор (=). (*Да/Нет*). Если “*Да*”, то почему и какой оператор следует использовать? Если “*Нет*”, то почему?

Примечание. В следующих упражнениях используются таблицы S_EMP, S_DEPT, S_ORD, S_ITEM, S_PRODUCT.

2. Выведите имя, фамилию и дату начала работы всех служащих, работающих в одном отделе с “Magee”.
3. Выведите номер служащего, имя, фамилию и имя пользователя для всех служащих, заработная плата которых выше средней.
4. Выведите фамилию, номер отдела и должность всех служащих, относящихся к регионам 1 или 2.
5. Выведите фамилию и заработную плату всех подчиненных “LaDoris Ngao”.
6. Выведите номер, имя и фамилию каждого служащего, который получает заработную плату выше средней и работает в одном отделе с любым сотрудником, фамилия которого содержит букву “t”.
7. Выведите номер и наименование клиента, а также кредитный рейтинг и фамилию торгового представителя для всех клиентов, которые расположены в Северной Америке или чьим торговым представителем является “Nguyen”.
8. Выведите наименование и краткое описание каждого товара, который ни разу не был заказан в сентябре 1992г.
9. (*) Выведите наименование и кредитный рейтинг всех клиентов, чьим торговым представителем является “Andre Dumas”. Что получилось и почему?
10. (*) Выведите фамилию каждого торгового представителя в регионах 1 и 2, наименования их клиентов и итоговые суммы заказов каждого клиента.

Практическое занятие 8: создание таблиц

1. Правильны ли синтаксис в следующих примерах? Если нет, то почему?

a.

```
SQL> CREATE TABLE T_3000
2      (id NUMBER (7),
3       Name VARCHAR2 (@%)
4       CONSTRAINT table_id_pk PRIMARY KEY (id));
```

b.

```

SQL> CREATE TABLE 1995_ORDERS
2      (id NUMBER (7),
3       customer_id NUMBER(7)
4       CONSTANT ord_cust_id_nn NOT NULL
5       total NUMBER (11,2),
6       filled CHAR (1)
7       CONSTANT ord_filled_ck CHECK
8       (filled IN('Y', 'N')),
9       CONSTANT ord_id_pk PRIMARY KEY);

```

- Создайте две таблицы на основе следующих бланков экземпляра таблицы. Введите команды в командный файл "Test82.sql". Затем выполните этот файл, чтобы создать таблицы. Убедитесь в том, что таблицы созданы.

Имя таблицы: DEPARTMENT

Имя столбца	ID	NAME
Тип ключа	PK	
Nulls/Unique	NN,U	
Таблица FK		
Столбец FK		
Тип данных	NUMBER	CHAR
Длина	7	25

Имя таблицы:EMPLOYEE

Имя столбца	ID	LAST NAME	FIRST NAME	DEPT ID
Тип ключа	PK			FK
Nulls/Unique	NN,U			NN
Таблица FK				DEPARTMENT
Столбец FK				ID
Тип данных	NUMBER	CHAR	CHAR	NUMBER
Длина	7	25	25	7

Практическое занятие 9: изменение таблиц и ограничений

- Создайте таблицу WORCER как копию таблицы EMPLOYEE. Просмотрите описание таблицы для проверки ее структуры.
- Просмотрите ограничения для этой таблицы. Сохраните эту команду в файле "test92.sql". Запишите типы и имена ограничений.
- Сравните эти ограничения с ограничениями для таблицы EMPLOYEE. Запишите типы и имена ограничений.
- Добавьте ограничение PRIMARY KEY из таблицы WORCER, используя столбец ID. Ограничение должно вступить в силу немедленно.

5. Добавьте ограничение типа FOREIGN KEY из таблицы DEPARTMENT на столбец DEPT_ID таблицы WORCER. Убедитесь в том, что ограничения добавлены, выполнив еще раз файл “test92.sql”.
6. Просмотрите имена и типы объектов с помощью представления USER_OBJECTS словаря данных. Для удобства просмотра данных может потребоваться изменение формата столбцов. Обратите внимание на то, что создана новая таблица и новые индексы.

Практическое занятие 10: обработка данных

1. Вставка данных в таблицы DEPARTMENT и EMPLOYEE. Вы можете использовать файлы предыдущего занятия для создания таблиц DEPARTMENT и EMPLOYEE.
 - a. Просмотрите описания таблиц DEPARTMENT и EMPLOYEE для выяснения имен столбцов.
 - b. Просмотрите ограничения, относящиеся к каждой таблице (первичный ключ и т.д.). Создайте командный файл “test101.sql”, который содержит типичный запрос для проверки ограничений.
 - c. Добавьте строку данных в таблицу DEPARTMENT с номером отдела 10 и названием отдела “Finance”. В команде INSERT столбцы не указывайте.
 - d. Добавьте две строки данных в таблицу EMPLOYEE. Создайте командный файл “test102.sql”, запрашивающий у вас значение для каждого столбца. Пусть первым служащим будет “Donna Smith”, номер отдела – 10, персональный номер – 200. Второй служащий – “Albert Jones”, номер отдела – 54, персональный номер – 201. Какой результат и почему?
 - e. Вставьте в таблицы DEPARTMENT и EMPLOYEE сведения об отделе маркетинга: номер отдела – 10, название “Marketing”. Что получилось и почему?
 - f. Убедитесь в том, что внесенные вами данные записаны в таблицы.
 - g. Создайте командный файл “test103.sql”, добавляющий в таблицу DEPARTMENT следующие строки: отдел “Marketing” (номер 37), отдел “Sales” (номер 54) и отдел “Personnel” (номер 75)).

- h. Выполните файл “test103.sql”, чтобы добавить в таблицу EMPLOYEE следующие строки “Albert Jones” из отдела номер 54, личный номер – 201; “Harry Chin” из отдела номер 75, личный номер – 202; “Rey Guiliani” из отдела номер 37, личный номер – 203.
 - i. Убедитесь в том, что данные действительно добавлены в таблицы.
 - j. Сделайте эти изменения данных постоянными.
2. Изменение и удаление данных в таблицах DEPARTMENT и EMPLOYEE.
- a. Замените название отдела кадров с “Personnel” на “Human Resourcer”.
 - b. Измените фамилию служащего номер 202 на “Korsgaard”.
 - c. Проверьте, правильно ли сделаны изменения в таблицах.
 - d. Попытайтесь удалить сведения об отделе номер 54. Каков результат и почему?
 - e. Удалите информацию о служащем “Albert Jones” из таблицы EMPLOYEE.
 - f. Снова попытайтесь удалить сведения об отделе номер 54 из таблицы DEPARTMENT. Каков результат и почему?
 - g. Проверьте, внесены ли изменения в таблицы.

Практическое занятие 11: управление транзакциями

1. Управление транзакциями с данными на примере таблиц DEPARTMENT и EMPLOYEE.
- a. Выполните файл “test103.sql” для восстановления отдела Sales как отдела номер 54.
 - b. Проверьте правильность внесенных изменений.
 - c. Создайте точку сохранения в транзакции.
 - d. Удалите все данные из таблицы EMPLOYEE.
 - e. Убедитесь в том, что таблица EMPLOYEE действительно пуста.
 - f. Отмените результаты последней операции DELETE, сохранив результаты последней операции INSERT.
 - g. Убедитесь в том, что новая строка в таблице DEPARTMENT цела. Проверьте, содержит ли таблица EMPLOYEE все три строки.
 - h. Сделайте эти добавления данных постоянными.

Практическое занятие 12: создание последовательностей

1. Создайте последовательность DEPT_ID_SEQ для генерации первичного ключа таблицы DEPARTMENT. Первое число последовательности—76, максимальное значение 80. Приращение должно быть равным единице.
2. Создайте еще одну последовательность WORKER_ID_SEQ. Она будет использоваться для столбца первичного ключа таблицы WORKER. Начните последовательность со значения 204: , максимальное значение 9999999. Проверьте, что числа увеличиваются на единицу. Задайте кэширование пяти чисел.
3. Напишите командный файл для вывода следующей информации о ваших последовательностях: размер кеша, максимальное значение, шаг приращения и последнее сгенерированное число. Назовите файл “test121.sql”.
4. Напишите интерактивный командный файл для вставки строки в таблицу DEPARTMENT. Назовите его “test122.sql”. Воспользуйтесь последовательностью, созданной вами для столбца ID. Создайте собственное приглашение на ввод названия отдела. Выполните свой файл. Добавьте два отдела – “Education” и “Administration”. Проверьте внесенные изменения.
5. Получите на экране информацию о своих последовательностях с помощью командного файла “test121.sql”. Обратите внимание на то, что последнее число последовательности WORKER_ID_SEQ отличается от самого большого значения первичного ключа в упражнении 2. Почему?
6. Напишите командный файл для вставки двух строк в таблицу WORKER. Назовите его “test123.sql”. Используйте последовательность, созданную вами для столбца ID. Выполните файл. Добавьте служащего “Tomas Lira” в качестве президента в отдел, который вы только что внесли в таблицу. Второй новый служащий – “Anna Seigher”, вице-президент в отделе “Finance”.
7. Проверьте данные, добавленные в таблицы DEPARTMENT и WORKER. Запишите самые большие значения первичного ключа для каждой из таблиц.

Практическое занятие 13: создание представлений

1. На основе таблицы WORKER создайте представление EMP_VU, включающее номер служащего, фамилию и номер отдела. Присвойте столбцу с фамилией заголовок EMPLOYEE.
2. Выведите на экран содержимое представления EMP_VU.
3. Напишите скрипт-файл для вывода на экран определения представления. Передайте скрипт-файлу имя представления. Сохраните файл под именем "test131.sql". Выполните его для вывода определения EMP_VU.
4. В представлении EMP_VU для служащего с фамилией "Smith" смените номер отдела на 37.
5. Проверьте, что "Smith" теперь приписан к отделу 37.
6. На основе таблиц DEPARTMENT и WORKER создайте представление MNS_VU для вывода данных о всех служащих отделов маркетинга и продаж. Выходные данные должны включать номер служащего, полное имя и номер отдела. Сохраните команду в скрипт-файле "test132.sql".
7. Выведете структуру и содержимое представления MNS_VU.
8. Выведите на экран определение представления MNS_VU, выполнив скрипт-файле "test131.sql".
9. Выведите на экран название каждого отдела и количество служащих в нем.
10. Измените представление EMP_VU так, чтобы оно содержало данные только о служащих отдела 37. Добавьте ограничение, запрещающее изменять номер отдела.
11. Выведите содержимое представления EMP_VU.
12. В представлении EMP_VU верните служащего с фамилией Smith номер отдела 54. Получилось или нет? Почему?

Практическое занятие 14: создание индексов

1. Могут ли какие-либо из перечисленных индексов использоваться с указанными запросами и почему?
 - a. Неуникальный индекс по столбцу LAST_NAME. (Да/Нет).

```
SQL> SELECT *
2     FROM S_EMP
3     WHERE LAST_NAME='Biri';
```

- b. Уникальный индекс по столбцу ID и неуникальный индекс по столбцу CUSTOMER_ID. (Да/Нет).

```
SQL> SELECT ID, Customer_ID, Total
2     FROM S_ORD
3     WHERE DATE_ORDERED='31-AUG-92';
```

- c. Уникальный индекс по столбцу S_DEPT.ID и неуникальный индекс по столбцу S_EMP.DEPT_ID. (Да/Нет).

```
SQL> SELECT E.LAST_NAME, D.NAME*
2     FROM S_EMP E, S_DEPT D
3     WHERE E.DEPT_ID=D.ID;
```

2. Создайте неуникальный индекс по столбцу внешнего ключа в таблице WORKER.
3. Так как пользователи часто запрашивают данные по фамилии служащих, создайте неуникальный индекс для этого столбца таблицы WORKER.
4. Выведите из словаря данных индексы и информацию об уникальности для таблиц WORKER и DEPARTMENT.
5. Удалите ограничения PRIMARY KEY для таблицы WORKER.
6. Еще раз выведите из словаря данных индексы и информацию об уникальности для таблиц WORKER и DEPARTMENT. Что изменилось и почему?
7. Вновь создайте ограничение PRIMARY KEY для таблицы WORKER. Убедитесь в том, что ограничение присутствует в словаре данных. Убедитесь в наличии уникального индекса по словарю данных.
8. Удалите индекс по столбцу фамилий служащих из таблицы WORKER.

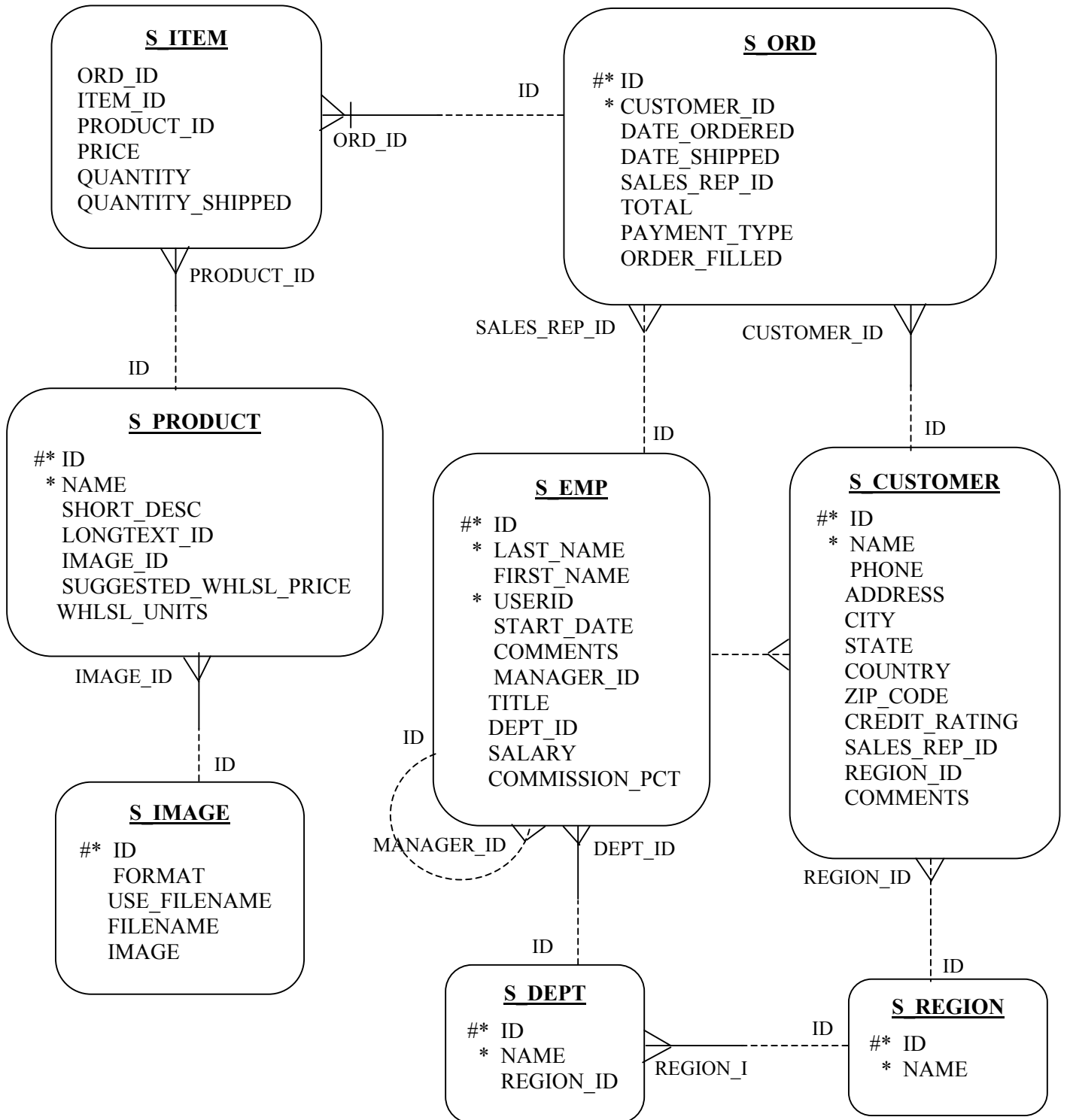
Литература

1. Бобровски С. Oracle7 и вычисления клиент/сервер. Пер. с англ. – М.: изд-во «ЛОРИ», 1996.

2. Neena Kochhar, Debby Kramer Introduction in Oracle: SQL and PL/SQL using Procedure Builder./ Part 1-2., Oracle Corporation, 1996.
3. К. Дейт Введение в системы баз данных. –М.: Наука, 1980.
4. К. Дейт Руководство по реляционной СУБД DB2. –М.: Финансы и статистика, 1988.
5. Дж. Ульман Основы систем баз данных. –М.: Финансы и статистика, 1983.
6. Кузнецов С.Д. Основы современных баз данных. / Информационно-аналитические материалы / Центр Информационных Технологий МГУ, <<http://www.citforum.ru/database/osbd/contents.shtml>>.

Приложения

ER-диаграмма учебной базы данных:



Структуры таблиц учебной базы данных

Структура таблицы *S_EMP*:

SQL> desc S_EMP		
<i>Name</i>	<i>Null?</i>	<i>Type</i>
ID		NUMBER(7)
LAST_NAME	NOT NULL	VARCHAR2(25)
FIRST_NAME		VARCHAR2(25)
USERID	NOT NULL	VARCHAR2(8)
START_DATE		DATE
COMMENTS		VARCHAR2(25)
MANAGER_ID		NUMBER(7)
TITLE		VARCHAR2(25)
DEPT_ID		NUMBER(7)
SALARY		NUMBER(11,2)
COMMISSION_PCT		NUMBER(4,2)

Структура таблицы *S_CUSTOMER*:

SQL> desc S_CUSTOMER		
<i>Name</i>	<i>Null?</i>	<i>Type</i>
ID		NUMBER(7)
NAME	NOT NULL	VARCHAR2(50)
PHONE		VARCHAR2(25)
ADDRESS		VARCHAR2(400)
CITY		VARCHAR2(30)
STATE		VARCHAR2(20)
COUNTRY		VARCHAR2(30)
ZIP_CODE		VARCHAR2(75)
CRÉDIT_RATING		VARCHAR2(9)
SALES_REP_ID		NUMBER(7)
REGION_ID		NUMBER(7)
COMMENTS		VARCHAR2(255)

Структура таблицы *S_DEPT*:

SQL> desc S_DEPT		
<i>Name</i>	<i>Null?</i>	<i>Type</i>
ID		NUMBER(7)
NAME	NOT NULL	VARCHAR2(25)
REGION_ID		NUMBER(7)

Структура таблицы *S_ITEM*:

SQL> desc S_ITEM		
<i>Name</i>	<i>Null?</i>	<i>Type</i>
ORD_ID		NUMBER(7)
ITEM_ID		NUMBER(7)
PRODUCT_ID		NUMBER(7)

PRICE	NUMBER(11,2)
QUANTITY	NUMBER(9)
QUANTITY_SHIPPED	NUMBER(9)

Структура таблицы S_ORD:

SQL> desc S_ORD		
<i>Name</i>	<i>Null?</i>	<i>Type</i>
-----	-----	-----
ID		NUMBER(7)
CUSTOMER_ID	NOT NULL	NUMBER(7)
DATE_ORDERED		DATE
DATE_SHIPPED		DATE
SALES_REP_ID		NUMBER(7)
TOTAL		NUMBER(11,2)
PAYMENT_TYPE		VARCHAR2(6)
ORDER_FILLED		VARCHAR2(1)

Структура таблицы S_PRODUCT:

SQL> desc S_PRODUCT		
<i>Name</i>	<i>Null?</i>	<i>Type</i>
-----	-----	-----
ID		NUMBER(7)
NAME	NOT NULL	VARCHAR2(50)
SHORT_DESC		VARCHAR2(255)
LONGTEXT_ID		NUMBER(7)
IMAGE_ID		NUMBER(7)
SUGGESTED_WHLSL_PRICE		NUMBER(11,2)
WHLSL_UNITS		VARCHAR2(25)

Структура таблицы S_REGION:

SQL> desc S_REGION		
<i>Name</i>	<i>Null?</i>	<i>Type</i>
-----	-----	-----
ID		NUMBER(7)
NAME	NOT NULL	VARCHAR2(50)

Структура таблицы S_IMAGE:

SQL> desc S_IMAGE		
<i>Name</i>	<i>Null?</i>	<i>Type</i>
-----	-----	-----
ID	NOT NULL	NUMBER(7)
FORMAT		VARCHAR2(25)
USE_FILENAME		VARCHAR2(25)
FILENAME		VARCHAR2(25)
IMAGE		LONG RAW

Структура таблицы S_TITLE:

SQL> desc S_TITLE		
<i>Name</i>	<i>Null?</i>	<i>Type</i>
-----	-----	-----
TITLE	NOT NULL	VARCHAR2(25)

Структура таблицы S_INVENTORY:

SQL> desc S_INVENTORY		
Name	Null?	Type
PRODUCT_ID		NUMBER(7)
WAREHOUSE_ID	NOT NULL	NUMBER(7)
AMOUNT_IN_STOCK		NUMBER(9)
REORDER_POINT		NUMBER(9)
MAX_IN_STOCK		NUMBER(9)
OUT_OF_STOCK_EXPLANATION		VARCHAR2(255)
RESTOCK_DATE		DATE

Александр Михайлович Гудов
Людмила Евгеньевна Шмакова

ВВЕДЕНИЕ В ЯЗЫК СТРУКТУРИРОВАННЫХ ЗАПРОСОВ SQL

Учебное пособие

Редактор З.А. Кунашева

Лицензия на издательскую деятельность ЛР №020464 от 9.06.1997 г.
Лицензия на полиграфическую деятельность ПЛД №44-12 от 10.08.2000 г.

Подписано к печати 4.06.2001 г. Формат 60x84 1/16.
Печать офсетная. Печ. л 7,375. Уч-изд. л 6. Тираж 250 экз.
Заказ № _____.

Кемеровский госуниверситет. 650043, Кемерово, ул. Красная, 6.
Отпечатано в издательстве «Кузбассвуиздат». 650043, Кемерово, ул. Ермака, 7.