ЯЗЫК SQL

ТИПЫ ДАННЫХ SQL (INTERBASE)

BLOB

Синтаксис: BLOB

Размер: Переменный

Диапазон/Точность: Нет

Описание: Большой двоичный объект. Сохраняет данные большого объема, такие как графика, текст и цифровой звук. Основная структура модуля: сегмент. Субтип данных BLOB описывается в их контексте.

CHAR

Синтаксис: CHAR(*n*)

Размер: *n* символов

Диапазон/Точность: от 1 до 32767 байтов. Размер символа кодировки определяет максимальное число символов, которые разместятся в 32K.

Описание: Фиксированной длины CHAR или строка текста. Альтернативное ключевое слово: CHARACTER

ТИПЫ ДАННЫХ SQL (INTERBASE)

DATE

Синтаксис: DATA
Размер: 64 бита
Диапазон/Точность: от 1 янв 100 до 11 янв 5941
Описание: Так же включает информацию о времени.

DECIMAL

Синтаксис: DECIMAL (*precision*, *scale*)
Размер: Переменный
Диапазон/Точность: *precision* = от 1 до 15. Определяет, что сохраняется, по крайней мере *precision* цифр числа. *scale* = от 1 до 15. Определяет число знаков после запятой. Должно быть меньше или равно *precision*.
Описание: Для примера, DECIMAL(10,3) сохраняет числа точно в следующем формате: ppppppp.sss

ТИПЫ ДАННЫХ SQL (INTERBASE)

DOUBLE PRECISION

Синтаксис: DOUBLE PRECISION
Размер: 64 бита
Диапазон/Точность: от 1.7E-308 до 1.7E308
Описание: Для научных вычислений: 15 цифр точности.
Обратите внимание: Текущий размер типа DOUBLE зависит от платформы. Большинство платформ поддерживает размер в 64 бита.

FLOAT

Синтаксис: FLOAT
Размер: 32 бита
Диапазон/Точность: от 3.4E-38 до 3.4E38
Описание: Одиночная точность: 7 цифр точности.

ТИПЫ ДАННЫХ SQL (INTERBASE)

INTEGER

Синтаксис: INTEGER

Размер: 32 бита

Диапазон/Точность: от -2 147 483 648 до 2 147 483 648

Описание: Длинное целое со знаком (long, longword).

NUMERIC

Синтаксис: NUMERIC (*precision*, *scale*)

Диапазон/Точность: *precision* = от 1 до 15. Определяет, что сохраняется, по крайней мере *precision* цифр числа. *scale* = от 1 до 15. Определяет число знаков после запятой. Должно быть меньше или равно *precision*.

Описание: Для примера, NUMERIC(10,3) сохраняет числа точно в следующем формате: ppppppp.sss

ТИПЫ ДАННЫХ SQL (INTERBASE)

SMALLINT

Синтаксис: SMALLINT

Размер: 16 бит

Диапазон/Точность: от -32768 до 32767

Описание: Короткое целое со знаком. (shot, word).

VARCHAR

Синтаксис: VARCHAR (*n*)

Размер: *n* символов

Диапазон/Точность: от 1 до 32767 байтов. Размер символа кодировки определяет максимальное число символов, которые разместятся в 32K.

Описание: переменной длины CHAR или строка текста. Альтернативные ключевые слова: VARYING CHAR, VARYING CHARACTER.

ТИПОВАЯ БАЗА ДАННЫХ

Рассматриваема база данных будет иметь три таблицы Продавцов (SALESPEOPLE), Клиентов (CUSTOMERS), Заказы (ORDERS)

Структура таблиц:

TABLE SALESPEOPLE

SNUM ***ID_INT NOT NULL*** /* ID_INT = INTEGER NOT NULL */, (уникальный номер назначенный каждому продавцу)

SNAME ***NAME*** /* NAME = VARCHAR(50) */, (имя продавца)

CITY ***N_CITY*** /* N_CITY = VARCHAR(50) */, (расположение продавца)

COMM ***NUM_K*** /* NUM_K = NUMERIC(5,2) */(комиссионные продовцов)

ТИПОВАЯ БАЗА ДАННЫХ

TABLE CUSTOMERS

CNUM ***ID_INT NOT NULL*** /* ID_INT = INTEGER NOT NULL */, (уникальный номер назначенный каждому клиенту)

CNAME ***NAME*** /* NAME = VARCHAR(50) */, (имя клиента)

CITY ***N_CITY*** /* N_CITY = VARCHAR(50) */, (расположение клиента)

RATING ***ID_INT*** /* ID_INT = INTEGER NOT NULL */, (число указывающее уровень предпочтения данного клиента перед другими. Большое число указывает на большее предпочтение)

SNUM ***ID_INT*** /* ID_INT = INTEGER NOT NULL */ (номер продавца, назначенного этому клиенту (из таблицы продавцов))

ТИПОВАЯ БАЗА ДАННЫХ

TABLE ORDERS

ONUM ***ID_INT NOT NULL*** /* ID_INT = INTEGER NOT NULL */, (уникальный номер данный каждому заказу)

AMT ***NUM_K*** /* NUM_K = NUMERIC(5,2) */, (сумма заказа)

ODATE ***DATE_Z*** /* DATE_Z = DATE */, (дата заказа)

CNUM ***ID_INT*** /* ID_INT = INTEGER NOT NULL */, (номер клиента, разместившего заказ (из таблицы Клиентов))

SNUM ***ID_INT*** /* ID_INT = INTEGER NOT NULL */ (номер продавца, исполняющего заказ (из таблицы продавцов))

СОДЕРЖАНИЕ ТАБЛИЦ

Таблица продавцов (SALESPEOPLE)

SNUM	SNAME	CITY	COMM
1 001	Сидоров	Благовещенск	0.12
1 002	Петров	Шимановск	0.13
1 003	Александров	Благовещенск	0.11
1 004	Иванов	Райчихинск	0.15
1 007	Питкин	Свободный	0.10

СОДЕРЖАНИЕ ТАБЛИЦ

Таблица клиентов (CUSTOMERS)

CNUM	CNAME	CITY	RATING	SNUM
2 001	Андреев	Благовещенск	100	1 001
2 002	Сергеев	Шимановск	200	1 003
2 003	Клопов	Талакан	200	1 002
2 004	Курапаткин	Бурея	300	1 002
2 006	Верещагин	Благовещенск	100	1 001
2 007	Ситкин	Талакан	100	1 004
2 008	Алексеев	Шимановск	300	1 007

СОДЕРЖАНИЕ ТАБЛИЦ

Таблица заказов (ORDERS)

ONUM	AMT	ODATE	CNUM	SNUM
3 001	18.690	10.03.2012	2 008	1 007
3 002	1 900.10	10.03.2012	2 007	1 004
3 003	767.190	10.03.2012	2 001	1 001
3 005	5 160.45	10.03.2012	2 003	1 002
3 006	1 098.16	10.03.2012	2 008	1 007
3 007	75.750	10.04.2012	2 004	1 002
3 008	4 723.00	10.05.2012	2 006	1 001
3 009	1 713.23	10.04.2012	2 002	1 003
3 010	1 309.95	10.06.2012	2 004	1 002
3 011	9 891.88	10.06.2012	2 006	1 001

ИЗВЛЕЧЕНИЕ ИНФОРМАЦИИ ИЗ ТАБЛИЦ

ЧТО ТАКОЕ ЗАПРОС?

Запрос **-** это команда, которую вы даете вашей СУБД, и которая сообщает системе какую именно информацию и из каких таблиц базы данных необходимо вывести. Эта информация обычно посылается непосредственно на экран компьютера или терминала, которым вы пользуетесь, хотя, в большинстве случаев, ее можно также послать на принтер, сохранить в файле, или представить как вводную информацию для другой команды.

ИЗВЛЕЧЕНИЕ ИНФОРМАЦИИ ИЗ ТАБЛИЦ

КОМАНДА SELECT

В самой простой форме, команда SELECT просто приказывает СУБД извлечь информацию из таблицы. Например, вы могли бы вывести таблицу Продавцов,
напечатав следующее:

SELECT snum, sname, city, comm FROM Salespeople;

Вывод для этого запроса:

SNUM	SNAME	CITY	COMM
1 001	Сидоров	Благовещенск	0.12
1 002	Петров	Шимоновск	0.13
1 004	Иванов	Благовещенск	0.11
1 007	Питкин	Свободный	0.15
1 003	Александров	Райчихинск	0.1

ИЗВЛЕЧЕНИЕ ИНФОРМАЦИИ ИЗ ТАБЛИЦ

SELECT - ключевое слово, которое сообщает СУБД, что эта команда запрос. Все запросы начинаются этим словом, сопровождаемым пробелом.

FROM - ключевое слово, которое должно быть представлено в каждом запросе. Оно сопровождается пробелом и затем именем таблицы, используемой в качестве источника информации.

; - точка с запятой используется во всех интерактивных командах SQL, чтобы сообщать СУБД что команда введена полностью и готова выполниться

ИЗВЛЕЧЕНИЕ ИНФОРМАЦИИ ИЗ ТАБЛИЦ

ВЫБИРАЙТЕ ВСЕГДА САМЫЙ ПРОСТОЙ СПОСОБ

Если вы хотите видеть каждый столбец таблицы в выводе вашего запроса, имеется необязательное сокращение, которое вы можете использовать. Звездочка * может применяться для вывода полного списка столбцов следующим образом:

SELECT * FROM Salespeople;

Это равносильно:

SELECT snum, sname, city, comm FROM Salespeople;

ИЗВЛЕЧЕНИЕ ИНФОРМАЦИИ ИЗ ТАБЛИЦ

ПРОСМОТР ТОЛЬКО ОПРЕДЕЛЕННОГО СТОЛБЦА ТАБЛИЦЫ

Команда SELECT способна извлечь строго определенную информацию из таблицы. Сначала мы продемонстрируем возможность увидеть только определенные столбцы таблицы. Это выполняется легко, простым исключением столбцов, которые вы не хотите видеть, из списка столбцов команды SELECT. Например, запрос:

SELECT sname, comm FROM Salespeople;

SNAME	COMM
Сидоров	0.12
Петров	0.13
Иванов	0.11
Питкин	0.15
Александров	0.1

ИЗВЛЕЧЕНИЕ ИНФОРМАЦИИ ИЗ ТАБЛИЦ

ПЕРЕУПОРЯДОЧЕНИЕ СТОЛБЦОВ

Даже если столбцы таблицы, по определению, упорядочены, это не означает, что вы обязаны извлекать, их в том же порядке. Звездочка * покажет все столбцы в их естественном порядке, но если вы перечислите столбцы отдельно, вы можете получить их в том порядке, в котором хотите. Например:

SELECT odate, snum, onum, amt FROM orders;

Вывод

ODATE	SNUM	ONUM	AMT
10.03.2012	1 007	3 001	18.69
10.03.2012	1 001	3 003	767.19
10.03.2012	1 004	3 002	1900.1
10.03.2012	1 002	3 005	5160.45
10.03.2012	1 007	3 006	1098.16
10.04.2012	1 003	3 009	1713.23
10.04.2012	1 002	3 007	75.75
10.05.2012	1 001	3 008	4723
10.06.2012	1 002	3 010	1309.95
10.06.2012	1 001	3 011	9891.88

ИЗВЛЕЧЕНИЕ ИНФОРМАЦИИ ИЗ ТАБЛИЦ

УДАЛЕНИЕ ИЗБЫТОЧНЫХ ДАННЫХ

Аргумент **DISTINCT** (ОТЛИЧИЕ) дает вам возможность, устранять повторяющиеся значения из вывода вашего запроса. Предположим, что вы хотите знать, какие продавцы в настоящее время имеют заказы в таблице Заказов, для этого вводим:

SELECT snum FROM Orders;

SNUM
1 007
1 001
1 004
1 002
1 007
1 003
1 002
1 001
1 002
1 001

ИЗВЛЕЧЕНИЕ ИНФОРМАЦИИ ИЗ ТАБЛИЦ

Видим повторяющиеся номера продавцов: 1001, 1002, 1007.

Для того что их устранить вводим:

SELECT DISTINCT snum FROM Orders;

SNUM
1 001
1 002
1 003
1 004
1 007

Однако если вы не хотите потерять некоторые данные, вы не должны безоглядно использовать DISTINCT, потому что это может скрыть какую- то проблему или какие то важные данные. Например, вы могли бы предположить, что имена всех наших клиентов различны. Если кто-то помещает второго Андреева в таблицу Клиентов, а вы используете **SELECT DISTINCT cname,** вы не будете даже знать о существовании двойника. Вы можете получить не того Андреева или даже не знать об этом.

ИЗВЛЕЧЕНИЕ ИНФОРМАЦИИ ИЗ ТАБЛИЦ

Если предложение выбирает несколько полей, DISTINCT опускает строки, где все выбранные поля идентичны. Строки, в которых значения некоторых полей одинаковы, а некоторых различны, будут сохранены.

DISTINCT ВМЕСТО ALL

Вместо DISTINCT вы можете указать ALL. Это будет иметь противоположный эффект, дублирование строк вывода сохранится. Так как это тот же самое, что и не указывать ни DISTINCT ни ALL, то ALL, по существу, скорее пояснительный, а не действующий аргумент.

ИЗВЛЕЧЕНИЕ ИНФОРМАЦИИ ИЗ ТАБЛИЦ

КВАЛИФИЦИРОВАННЫЙ ВЫБОР С ИСПОЛЬЗОВАНИЕМ ПРЕДЛОЖЕНИЯ WHERE

Таблицы имеют тенденцию становиться очень большими, ведь с течением времени в них добавляются все новые строки. Поскольку обычно не все, а только определенные строки интересуют вас: в данное время, SQL дает вам возможность указать какие строки будут выбраны для вывода.

WHERE - предложение команды SELECT, позволяющее вам устанавливать предикаты (логические выражения с переменными), которые могут оказаться верными или неверными для любой строки таблицы.

ИЗВЛЕЧЕНИЕ ИНФОРМАЦИИ ИЗ ТАБЛИЦ

Команда извлекает только те строки из таблицы, для которых предикат верен.

Например, вы хотите видеть имена и комиссионные всех продавцов в Благовещенске. Вы можете ввести такую команду:

SELECT sname, city FROM Salespeople WHERE city='Благовещенск';

Когда предложение WHERE присутствует, СУБД просматривает всю таблицу по одной строке и определяет верен ли предикат для каждой строки.

SNAME	CITY
Сидоров	Благовещенск
Иванов	Благовещенск

ИЗВЛЕЧЕНИЕ ИНФОРМАЦИИ ИЗ ТАБЛИЦ

Рассмотрим пример с числовым полем в предложении WHERE . Поле rating таблицы Клиентов предназначено для того, чтобы разделять клиентов на группы на основании некоторых критериев. Возможно, это форма оценки кредита клиента, или оценка, связанная с предыдущими заказами клиента. Такие числовые оценки могут быть полезны в реляционных базах данных как способ подведения итогов обработки сложной информации. Мы можем выбрать всех клиентов с рейтингом 100, следующим образом:

SELECT * FROM Customers WHERE rating = 100;

Одиночные кавычки не используются здесь потому, что рейтинг числовое поле.

ИЗВЛЕЧЕНИЕ ИНФОРМАЦИИ ИЗ ТАБЛИЦ

Результат вывода:

CNUM	CNAME	CITY	RATING	SNUM
2 001	Андреев	Благовещенск	100	1 001
2 006	Верещагин	Благовещенск	100	1 001
2 007	Ситкин	Талакан	100	1 004

ИЗВЛЕЧЕНИЕ ИНФОРМАЦИИ ИЗ ТАБЛИЦ

********** **РАБОТА С SQL** **********

- Напишите команду SELECT, которая бы вывела номер заказа, сумму и дату для всех строк из таблицы Заказов.
- Напишите запрос, который вывел бы все строки из таблицы Клиентов, дня которых номер продавца - 1001.
- Напишите запрос, который вывел бы таблицу со столбцами в следующем порядке: city, sname, snum, comm.
- Напишите команду SELECT, которая вывела бы оценку (rating), сопровождаемую именем каждого клиента в Шимоновске.
- Напишите запрос, который вывел бы значения snum всех продавцов в текущем порядке из таблицы Заказов без каких бы то ни было повторений.

ИСПОЛЬЗОВАНИЕ РЕЛЯЦИОННЫХ И БУЛЕВЫХ ОПЕРАТОРОВ

РЕЛЯЦИОННЫЕ ОПЕРАТОРЫ

Реляционный оператор - математический символ, который указывает на определенный тип сравнения двух значений.

Предположим, что вы хотите видеть всех Продавцов, комиссионные которых превышают определенное значение. Вы можете использовать оператор "больше чем" (>). Реляционные операторы, которыми располагает SQL :

= ***Равно***

> ***Больше чем***

< ***Меньше чем***

>= ***Больше чем или равно***

<= ***Меньше чем или равно***

<> ***Не равно***

ИСПОЛЬЗОВАНИЕ РЕЛЯЦИОННЫХ И БУЛЕВЫХ ОПЕРАТОРОВ

Предположим, что вы хотите увидеть всех клиентов с оценкой (rating) выше 200. Так как 200 - это скалярное значение, как и значение в столбце rating, для их сравнения вы можете использовать реляционный оператор.

SELECT * FROM Customers WHERE rating > 200;

Вывод

CNUM	CNAME	CITY	RATING	SNUM
2 004	Курапаткин	Бурея	300	1 002
2 008	Алексеев	Шимоновск	300	1 007

ИСПОЛЬЗОВАНИЕ РЕЛЯЦИОННЫХ И БУЛЕВЫХ ОПЕРАТОРОВ

БУЛЕВЫ ОПЕРАТОРЫ

Основные логические операторы также распознаются в SQL. Выражения Буля являются или верными или неверными, подобно предикатам. Булевы операторы связывают одно или более верных или неверных значений и производят единственное верное или неверное значение.

Стандартными логическими операторами, распознаваемыми в SQL, являются **AND**, **OR** и **NOT**.

Существуют другие, более сложные, логические операторы (типа "исключающей или"), но они могут быть сформированы из этих трех простых операторов.

ИСПОЛЬЗОВАНИЕ РЕЛЯЦИОННЫХ И БУЛЕВЫХ ОПЕРАТОРОВ

Пример

SELECT * FROM Customers WHERE city = 'Шимоновск' AND rating > 200;

CNUM	CNAME	CITY	RATING	SNUM
2 008	Алексеев	Шимоновск	300	1 007

SELECT * FROM Customers WHERE city = 'Шимоновск ' OR rating >200;

CNUM	CNAME	CITY	RATING	SNUM
2 003	Клопов	Шимоновск	200	1 002
2 004	Курапаткин	Бурея	300	1 002
2 008	Алексеев	Шимоновск	300	1 007

ИСПОЛЬЗОВАНИЕ РЕЛЯЦИОННЫХ И БУЛЕВЫХ ОПЕРАТОРОВ

NOT может использоваться для инвертирования значений Буля. Пример запроса с NOT:

SELECT * FROM Customers WHERE city = 'Шимоновск' OR NOT rating > 200;

CNUM	CNAME	CITY	RATING	SNUM
2 001	Андреев	Благовещенск	100	1 001
2 002	Сергеев	Ивановка	200	1 003
2 003	Клопов	Шимоновск	200	1 002
2 006	Верещагин	Благовещенск	100	1 001
2 008	Алексеев	Шимоновск	300	1 007
2 007	Ситкин	Талакан	100	1 004

Все записи за исключением Курапаткина были выбраны. Курапаткин не был в Шимоновске, и его рейтинг больше чем 200, так что он потерпел неудачу при обеих проверках. В каждой из других строк встретился один или другой, или оба критерия.

ИСПОЛЬЗОВАНИЕ РЕЛЯЦИОННЫХ И БУЛЕВЫХ ОПЕРАТОРОВ

Обратите внимание, что оператор **NOT** должен предшествовать Булеву оператору, чье значение он изменяет, и не должен помещаться перед реляционным оператором. Например, неправильным будет:

rating NOT > 200

А как SQL оценит следующее?

SELECT * FROM Customers WHERE NOT city = 'Шимоновск' OR rating > 200;

NOT применяется здесь только к выражению : **city = 'Шимоновск',** или к выражению **rating > 200** тоже?

SQL применяет NOT к логическому выражению, которое непосредственно следует за NOT.

ИСПОЛЬЗОВАНИЕ РЕЛЯЦИОННЫХ И БУЛЕВЫХ ОПЕРАТОРОВ

Совершенно другой результат от запроса:

SELECT * FROM Customers WHERE NOT (city = 'Шимоновск' OR rating > 200);

Здесь SQL берет каждую строку и определяет, соответствует ли истине равенство **city = 'Шимоновск'** или равенство **rating >200.** Если любое условие верно, выражение Буля внутри круглых скобок верно. Однако, если выражение Буля внутри круглых скобок верно, предикат как единое целое неверен, потому что NOT преобразует верно в неверно, и наоборот.

CNUM	CNAME	CITY	RATING	SNUM
2 001	Андреев	Благовещенск	100	1 001
2 002	Сергеев	Ивановка	200	1 003
2 006	Верещагин	Благовещенск	100	1 001
2 007	Ситкин	Талакан	100	1 004

ИСПОЛЬЗОВАНИЕ РЕЛЯЦИОННЫХ И БУЛЕВЫХ ОПЕРАТОРОВ

Проследить логику выражения:

```
SELECT * FROM Orders
WHERE NOT ((odate = '10.03.2012' AND snum > 1002) OR
amt > 2000.00) ;
```

ИСПОЛЬЗОВАНИЕ РЕЛЯЦИОННЫХ И БУЛЕВЫХ ОПЕРАТОРОВ

```
SELECT * FROM Orders
WHERE NOT ((odate = '10.03.2012' AND snum > 1002)
OR amt > 2000.00) ;
```

ONUM	AMT	ODATE	CNUM	SNUM
3 003	767.19	10.03.2012	2 001	1 001
3 009	1713.23	10.04.2012	2 002	1 003
3 007	75.75	10.04.2012	2 004	1 002
3 010	1309.95	10.06.2012	2 004	1 002

ИСПОЛЬЗОВАНИЕ РЕЛЯЦИОННЫХ И БУЛЕВЫХ ОПЕРАТОРОВ

1. Напишите запрос, который может дать вам все заказы со значениями суммы выше чем $1,000.
2. Напишите запрос, который может выдать вам поля sname и city для всех продавцов в Благовещенске с комиссионными выше 0.10 .
3. Напишите запрос к таблице Клиентов, чей вывод может включить всех клиентов с оценкой =< 100, если они не находятся в Риме.
4. Что может быть выведено в результате следующего запроса ?

SELECT * FROM Orders
WHERE (amt < 1000 OR NOT (odate='10.03.1990' AND snum > 2003)) ;

5. Что может быть выведено в результате следующего запроса ?

SELECT * FROM Orders
WHERE NOT ((odate = '10.03.1990' OR snum > 1006) AND amt > = 1500);

6. Как можно проще переписать такой запрос ?

SELECT srmm, sname, city, com FROM Salespeople
WHERE (comm > + . 12 OR comm < .14) ;

ИСПОЛЬЗОВАНИЕ СПЕЦИАЛЬНЫХ ОПЕРАТОРОВ В ПРЕДИКАТАХ

В дополнение к реляционным и логическим операторам, SQL использует специальные операторы: **IN, BETWEEN, LIKE,** и **IS NULL**.
С помощью этих операторов можно сложные и мощные предикаты.

ИСПОЛЬЗОВАНИЕ СПЕЦИАЛЬНЫХ ОПЕРАТОРОВ ОПЕРАТОРОВ В ПРЕДИКАТАХ

ОПЕРАТОР IN

Оператор IN определяет набор значений, с: каждым из которых данное проверяемое значение сравнивается.

Если вы хотите:

найти всех продавцов, которые размещены в Благовещенске или в Шимоновске, выдолжны использовать следующий запрос:

SELECT * FROM Salespeople WHERE city = 'Благовещенск' OR city = 'Шимоновск';

Имеется и более простой способ получить ту же информацию:

SELECT * FROM Salespeople WHERE city IN ('Благовещенск', 'Шимоновск');

SNUM	SNAME	CITY	COMM
1 001	Сидоров	Благовещенск	0.12
1 002	Петров	Шимоновск	0.13
1 004	Иванов	Благовещенск	0.11

Давайте найдем всех клиентов продавцов, у которых значение snum = 2001, 2007, и 2004. Вывод для следующего запроса:

SELECT * FROM Customers WHERE cnum IN (2001, 2007, 2004) ;

CNUM	CNAME	CITY	RATING	SNUM
2 001	Андреев	Благовещенск	100	1 001
2 004	Курапаткин	Бурея	300	1 002
2 007	Ситкин	Талакан	100	1 004

ОПЕРАТОР BETWEEN

Оператор **BETWEEN** (МЕЖДУ) похож на оператор **IN**. В отличие от сравнения проверяемого значения с каждым из элементов набора, как это делает **IN**, **BETWEEN** определяет диапазон, в который должно попасть проверяемое значение, чтобы предикат стал верным.

Вы должны ввести ключевое слово **BETWEEN** с начальным значением, ключевое **AND** и конечное значение. В отличие от **IN**, **BETWEEN** чувствителен к порядку, и первое значение в предложении должно быть первым по алфавитному или числовому порядку.

ИСПОЛЬЗОВАНИЕ СПЕЦИАЛЬНЫХ ОПЕРАТОРОВ ОПЕРАТОРОВ В ПРЕДИКАТАХ

Пример извлекать из таблицы Продавцов всех продавце» комиссионными между .10 и .12:

SELECT * FROM Salespeople WHERE coram BETWEEN .10 AND .12;

Для оператора **BETWEEN,** значение, совпадающее с любым из двух значений I границ диапазона (в этом случае, .10 и .12) заставляет предикат быть верным.

SNUM	SNAME	CITY	COMM
1 001	Сидоров	Благовещенск	0.12
1 004	Иванов	Благовещенск	0.11
1 003	Александров	Райчихинск	0.1

ИСПОЛЬЗОВАНИЕ СПЕЦИАЛЬНЫХ ОПЕРАТОРОВ ОПЕРАТОРОВ В ПРЕДИКАТАХ

SQL не предусматривает непосредственной поддержки BETWEEN, не включающего граничные значения. Вы должны или определить ваши граничные значения так, чтобы включающая интерпретация была приемлема, или сделать что-нибудь типа этого:

SELECT * FROM Salespeople WHERE (comm BETWEEN .10 AND .12) AND NOT comm IN (.10, .12);

SNUM	SNAME	CITY	COMM
1 004	Иванов	Благовещенск	0.11

ИСПОЛЬЗОВАНИЕ СПЕЦИАЛЬНЫХ ОПЕРАТОРОВ ОПЕРАТОРОВ В ПРЕДИКАТАХ

Подобно реляционным операторам, BETWEEN может работать с символьными полями. Это означает что вы можете использовать BETWEEN, чтобы выбира ряд значений из упорядоченных по алфавиту значений.

Этот запрос выбирает всех клиентов, чьи имена попали в определенный алфавитиый диапозон :

SELECT * FROM Customers WHERE cname BETWEEN 'Б' AND 'Т';

Вывод для этого запроса:

CNUM	CNAME	CITY	RATING	SNUM
2 002	Сергеев	Ивановка	200	1 003
2 003	Клопов	Шимоновск	200	1 002
2 004	Курапаткин	Бурея	300	1 002
2 006	Верещагин	Благовещенск	100	1 001
2 007	Ситкин	Талакан	100	1 004

ОПЕРАТОР LIKE

LIKE применим только к полям типа CHAR или VARCHAR, с которыми он используется чтобы находить подстроки. Т.е. он ищет поле символа чтобы видеть, совпадает ли с условием часть его строки. В качестве условия он использует групповые символы (wildkards) — специальные символы которые могут соответствовать чему-нибудь. Имеются два типа групповых символов используемых с LIKE:

- символ подчеркивания (_) замещает любой одиночный символ. Например, **'b_t'** будет соответствовать словам **'bat'** или **'bit'**, но не будет соответствовать **'brat'**.

- знак процента (**%**) замещает последовательность любого числа символов (включая символы нуля). Например **'%p%t'** будет соответствовать словам **'put', 'posit',** или **'opt',** но не **'spite'**.

ИСПОЛЬЗОВАНИЕ СПЕЦИАЛЬНЫХ ОПЕРАТОРОВ ОПЕРАТОРОВ В ПРЕДИКАТАХ

SELECT * FROM Customers WHERE cname LIKE 'К%';

CNUM	CNAME	CITY	RATING	SNUM
2 003	Клопов	Талакан	200	1 002
2 004	Курапаткин	Бурея	300	1 002

ИСПОЛЬЗОВАНИЕ СПЕЦИАЛЬНЫХ ОПЕРАТОРОВ ОПЕРАТОРОВ В ПРЕДИКАТАХ

LIKE может быть удобен если вы ищете имя или другое значение, и если вы не помните как они точно пишутся. Предположим что вы неуверены как записано по буквам имя одного из ваших продавцов **Петров** или **Пестов.** Вы можете просто использовать ту часть которую вы знаете и групповые символы, чтобы найти все возможные пары

```
SELECT * FROM Salespeople
    WHERE sname LIKE 'Пе_ _ов';
```

SNUM	SNAME	CITY	COMM
1 002	Петров	Шимановск	0,13

РАБОТА С НУЛЕВЫМИ (NULL) ЗНАЧЕНИЯМИ

В таблицах могут быть записи которые не имеют никаких значений для какого либо поля, например потому что информация не завершена, или потому что это поле просто не заполнялось. SQL учитывает такой вариант, позволяя вам вводить значение **NULL** (ПУСТОЙ) в поле, вместо значения. Когда значение поля равно NULL, это означает, что программа базы данных специально промаркировала это поле как не имеющее никакого значения для этой строки (или записи). Это отличается от просто назначения полю значения нуля или пробела, которые база данных будет обрабатывать также как и любое другое значение. Точно также, как NULL не является техническим значением, оно не имеет и типа данных. Оно может помещаться в любой тип поля.

NULL ОПЕРАТОР

Так как NULL указывает на отсутствие значения, вы не можете знать, каков будет результат любого сравнения с использованием NULL. Когда NULL сравнивается с любым значением, даже с другим таким же NULL, результат будет ни верным ни неверным, он — неизвестен. Вы должны понимать различия между неверно и неизвестно. SQL предоставляет специальный оператор **IS**, который используется с ключевым словом NULL, для размещения значения NULL. Найдем все записи в нашей таблице Заказчиков с NULL значениями в city столбце:

```
SELECT * FROM Customers WHERE city IS NULL;
```

Здесь не будет никакого вывода, потому что мы не имеем никаких значений NULL в наших типовых таблицах. Значения NULL — очень важны, и мы вернемся к ним позже.

ИСПОЛЬЗОВАНИЕ NOT СО СПЕЦИАЛЬНЫМИ ОПЕРАТОРАМИ

Специальные операторы могут немедленно предшествовать Булеву NOT.

Он противоположен реляционным операторам, которые должны иметь оператор NOT вводимым выражением. Например, если мы хотим устранить NULL из нашего вывода, мы будем использовать NOT, чтобы изменить на противоположное значение предиката:

SELECT * FROM Customers WHERE city NOT NULL;

При отсутствии значений NULL (как в нашем случае), будет выведена вся таблица Заказчиков. Аналогично можно ввести следующее:

SELECT * FROM Customers WHERE NOT city IS NULL;

ИСПОЛЬЗОВАНИЕ СПЕЦИАЛЬНЫХ ОПЕРАТОРОВ ОПЕРАТОРОВ В ПРЕДИКАТАХ

Мы можем также использовать NOT с **IN**:

SELECT * FROM Salespeople WHERE city NOT IN ('Благовещенск', ‘Райчихинск');

А это — другой способ подобного же выражения:

SELECT * FROM Salespeople WHERE NOT city IN ('Благовещенск', 'Райчихинск');

Результат этих запросов один и тот же:

SNUM	SNAME	CITY	COMM
1 002	Петров	Шимановск	0,13
1 007	Питкин	Свободный	0,1

Таким же способом Вы можете использовать **NOT BETWEEN** и **NOT LIKE**.

ИСПОЛЬЗОВАНИЕ СПЕЦИАЛЬНЫХ ОПЕРАТОРОВ В ПРЕДИКАТАХ

Самостоятельная работа

1. Напишите два запроса которые могли бы вывести все заказы на 10 апреля или мая 2012 года;
2. Напишите запрос который выберет всех заказчиков обслуживаемых продавцами Петровым или Питкиным. (Подсказка: из наших типовых таблиц, поле snum связывает вторую таблицу с первой);
3. Напишите запрос, который может вывести всех заказчиков, чьи имена начинаются с буквы попадающей в диапазон от A до K.
4. Напишите запрос который выберет всех продавцов чьи имена начинаются с буквы C.
5. Напишите запрос который выберет все заказы имеющие нулевые значения или NULL в поле amt (сумма).

АГРЕГАТНЫЕ ФУНКЦИИ

АГРЕГАТНЫЕ ФУНКЦИИ

ЧТО ТАКОЕ АГРЕГАТНЫЕ ФУНКЦИИ ?

Запросы могут производить обобщенное групповое значение полей точно также как и значение одного поля. Это делает с помощью агрегатых функций. Агрегатные функции производят одиночное значение для всей группы таблицы. Имеется список этих функций:

- **COUNT** производит номера строк или не-NULL значения полей которые выбрал запрос.
- **SUM** производит арифметическую сумму всех выбранных значений данного поля.
- **AVG** производит усреднение всех выбранных значений данного поля.
- **MAX** производит наибольшее из всех выбранных значений данного поля.
- **MIN** производит наименьшее из всех выбранных значений данного поля.

КАК ИСПОЛЬЗОВАТЬ АГРЕГАТНЫЕ ФУНКЦИИ ?

Агрегатные функции используются подобно именам полей в предложении SELECT запроса, но с одним исключением, они берут имена поля как аргументы. Только числовые поля могут использоваться с SUM и AVG.

С COUNT, MAX, и MIN, могут использоваться и числовые или символьные поля. Когда они используются с символьными полями, MAX и MIN будут транслировать их в эквивалент ASCII(***American Standard Code for Information Interchange*** **— американская стандартная кодировочная таблица для печатных символов и некоторых специальных кодов**), который должен сообщать, что MIN будет означать первое, а MAX последнее значение в алфавитном порядке.

АГРЕГАТНЫЕ ФУНКЦИИ

Чтобы найти SUM всех наших покупок в таблицы Заказов, мы можем ввести следующий запрос:

SELECT SUM (amt) FROM Orders;

Результат запроса:

SUM
26514,4

Чтобы найти AVG всех наших покупок в таблицы Заказов, мы можем ввести следующий запрос:

SELECT AVG (amt) FROM Orders;

Результат запроса:

AVG
2651,44

СПЕЦИАЛЬНЫЕ АТРИБУТЫ COUNT

Функция **COUNT** несколько отличается от всех. Она считает число значений в данном столбце, или число строк в таблице. Когда она считает значения столбца, она используется с **DISTINCT**, чтобы производить счет чисел различных значений в данном поле. Мы могли бы использовать ее, например, чтобы сосчитать номера продавцов в настоящее время описаных в таблице Заказов:

SELECT COUNT (DISTINCT snum) FROM Orders;

Результат запроса:

COUNT
5

В примере, **DISTINCT**, сопровождаемый именем поля с которым он применяется, помещен в круглые скобки, но не сразу после **SELECT**, как раньше.

ИСПОЛЬЗОВАНИЕ COUNT СО СТРОКАМИ, А НЕ ЗНАЧЕНИЯМИ

Чтобы подсчитать общее число строк в таблице, используйте функцию **COUNT** со звездочкой вместо имени поля, как например в следующем примере:

SELECT COUNT (*) FROM Customers

Результат запроса: **COUNT**

7

COUNT со звездочкой включает и NULL и дубликаты, по этой причине DISTINCT не может быть использован. DISTINCT не применим с COUNT (*), потому, что он не имеет никакого действия в хорошо разработаной и поддерживаемой базе данных.

В такой базе данных, не должно быть таких строк, которые бы являлись полностью пустыми или дубликатов (первые не содержат никаких данных, а последние полностью избыточны). С другой стороны, если все таки имеются полностью пустые или избыточные строки, вы вероятно не захотите чтобы COUNT скрыл от вас эту информацию, поэтому если вы не уверены в качестве базы данных то можете применять distinct в сочетании с count(*).

АГРЕГАТЫ ПОСТРОЕННЫЕ НА СКАЛЯРНОМ ВЫРАЖЕНИИ

Можете также использовать агрегатные функции с аргументами, которые состоят из скалярных выражений, включающих одно или более полей (При этом если вы это делаете, DISTINCT не разрешается). Предположим, что таблица Заказов имеет еще один столбец который хранит предыдущий не уплаченый баланс (поле **blnc**) для каждого заказчика. Вы должны найти этот текущий баланс, добавлением суммы приобретений к предыдущему балансу. Вы можете найти наибольший неуплаченый баланс следующим образом:

SELECT MAX (blnc + amt) FROM Orders;

(в практической работе добавить данное поле заполнить и выполнить select)

ПРЕДЛОЖЕНИЕ GROUP BY

Предложение **GROUP BY** позволяет вам определять подмножество значений в особом поле с другим названием поля, и применять функцию агрегата к подмножеству. Это дает вам возможность объединять поля и агрегатные функции в едином предложении SELECT. Например, предположим что вы хотите найти наибольшую сумму приобретений полученную каждым продавцом. Вы можете сделать раздельный запрос для каждого из них, выбрав MAX (amt) из таблицы **Заказов** для каждого значения поля **snum**. **GROUP BY**, однако, позволит Вам поместить их все в одну команду:

SELECT snum, MAX (amt) FROM Orders GROUP BY snum;

АГРЕГАТНЫЕ ФУНКЦИИ

Вывод для этого запроса:

SNUM	MAX
1 001	9891,88
1 002	5016,45
1 003	1713,23
1 004	1900,1
1 007	1098,16

GROUP BY применяет агрегатные функции независимо от серий групп, которые определяются с помощью значения поля в целом. В этом случае, каждая группа состоит из всех строк с тем же самым значением поля snum, и MAX функция применяется отдельно для каждой такой группы.

АГРЕГАТНЫЕ ФУНКЦИИ

Можно также использовать **GROUP BY** с многочисленными полями. Предположим что вы хотите увидеть наибольшую сумму покупок получаемую каждым продавцом за каждый день. Чтобы сделать это, вы должны сгруппировать таблицу Заказов по датам продавцов, и применить функцию MAX к каждой такой группе, подобно этому:

SELECT snum, odate, MAX ((amt)) FROM Orders GROUP BY snum, odate;

Вывод для этого запроса

SNUM	ODATE	MAX
1 001	10.03.2012	767,19
1 001	10.05.2012	4723
1 001	10.06.2012	9891,88
1 002	10.03.2012	5016,45
1 002	10.04.2012	1309,95
1 003	10.04.2012	1713,23
1 004	10.03.2012	1900,1
1 007	10.03.2012	1098,16

пустые группы, в дни когда текущий продавец не имел заказов, не будут показаны в выводе.

ПРЕДЛОЖЕНИЕ HAVING

Предположим (предыдущий пример) вы хотели бы увидеть только максимальную сумму покупок, значение которой выше $3000.00

Чтобы увидеть максимальную стоимость покупок свыше $3000.00, вы можете использовать предложение **HAVING**. Предложение **HAVING** определяет критерии, используемые для удаления определенных групп из вывода, точно также как предложение **WHERE** делает это для индивидуальных строк.

```
SELECT snum, odate, MAX ((amt)) FROM Orders GROUP
BY snum, odate HAVING MAX ((amt)) > 3000.00;
```

Вывод для этого запроса:

SNUM	ODATE	MAX
1 001	10.05.2012	4723
1 001	10.06.2012	9891,88
1 002	10.03.2012	5016,45

АГРЕГАТНЫЕ ФУНКЦИИ

HAVING может использовать только аргументы которые имеют одно значение на группу вывода.

SELECT snum, MAX (amt) FROM Orders GROUP BY snum HAVING snum IN (1002,1007);

Результат:

SNUM	MAX
1 002	5016,45
1 007	1098,16

НЕ ДЕЛАЙТЕ ВЛОЖЕННЫХ АГРЕГАТОВ

В строгой интерпретации ANSI SQL, вы не можете использовать агрегат агрегата. Предположим что вы хотите выяснять, в какой день имелась наибольшая сумма приобретений. Если вы попробуете сделать это,

SELECT odate, MAX (SUM (amt)) FROM Orders GROUP BY odate;

то ваша команда будет вероятно отклонена.

(Некоторые реализации не предписывают этого ограничения, которое является выгодным, потому что вложенные агрегаты могут быть очень полезны, даже если они и несколько проблематичны.)

СКАЛЯРНОЕ ВЫРАЖЕНИЕ С ПОМОЩЬЮ ВЫБРАННЫХ ПОЛЕЙ

Предположим что вы хотите выполнять простые числовые вычисления данных чтобы затем помещать их в форму больше соответствующую вашим потребностям. SQL позволяет вам помещать скалярные выражения и константы среди выбраных полей. Например, вы можете пожелать, представить комиссионные вашего продавца в процентном отношении а не как десятичные числа. Просто достаточно:

SELECT snum, sname, city, comm * 100 FROM Salespeople;

SNUM	SNAME	CITY	MULTIPLY
1 001	Сидоров	Благовещенск	12,00
1 002	Петров	Шимановск	13,00
1 003	Александров	Благовещенск	11,00
1 004	Иванов	Райчихинск	15,00
1 007	Питкин	Свободный	10,00

СТОЛБЦЫ ВЫВОДА

Последний столбец предшествующего примера помечен как **MULTIPLY** (т.е. автоматически), потому что это — *столбец вывода.* Столбцы вывода — это столбцы данных созданные запросом способом, иным чем просто извлечение их из таблицы. Вы создаете их всякий раз, когда вы используете агрегатные функции, константы, или выражения в предложении SELECT запроса. Так как имя столбца — один из атрибутов таблицы, столбцы которые приходят не из таблиц не имеют никаких имен или именуются произвольно. Другими словами непомеченные, столбцы вывода могут обрабатываться также как и столбцы извлеченные из таблиц, почти во всех ситуациях.

ПОМЕЩЕНИЕ ТЕКСТА В ВАШЕМ ВЫВОДЕ ЗАПРОСА

Символ 'A', когда ничего не значит сам по себе, — является константой, такой например как число 1. Вы можете вставлять константы в предложение SELECT запроса, включая и текст. Однако символьные константы, в отличие от числовых констант, не могут использоваться в выражениях. Вы можете иметь выражение 1 + 2 в вашем предложении SELECT, но вы не можете использовать выражение типа 'A' + 'B‘. Возможность вставлять текст в вывод ваших запросов очень удобная штука.

СКАЛЯРНЫЕ ВЫРАЖЕНИЯ

Вы можете усовершенствовать предыдущий пример представив комиссионные как проценты со знаком процента (%).

SELECT snum, sname, city, ' % ', comm * 100 FROM Salespeople;

SNUM	SNAME	CITY	CONSTANT	MULTIPLY
1 001	Сидоров	Благовещенск	%	12
1 002	Петров	Шимановск	%	13

. . .

или

SELECT snum, sname, city, (comm * 100) AS PROCENT FROM Salespeople;

SNUM	SNAME	CITY	PROCENT
1 001	Сидоров	Благовещенск	12,00
1 002	Петров	Шимановск	13,00

. . .

УПОРЯДОЧИВАНИЕ ВЫВОДА ПОЛЕЙ

В SQL на ряду с командой ORDER BY которая позволяет упорядочивать ваш вывод существуют команды: возрастание (**ASC**) и убывание (**DESC**) для каждого столбца. По умолчанию установлено — возрастание. Давайте рассмотрим нашу таблицу порядка приводимую в порядок с помощью номера заказчика (обратите внимание на значения в cnum столбце):

SELECT * FROM Orders ORDER BY cnum DESC;

ONUM	AMT	ODATE	CNUM	SNUM
3 001	18,69	10.03.2012	2 008	1 007
3 006	1098,16	10.03.2012	2 008	1 007
3 002	1900,1	10.03.2012	2 007	1 004
3 008	4723	10.05.2012	2 006	1 001
3 011	9891,88	10.06.2012	2 006	1 001
3 007	75,75	10.04.2012	2 004	1 002
3 010	1309,95	10.04.2012	2 004	1 002
3 005	5016,45	10.03.2012	2 003	1 002
3 009	1713,23	10.04.2012	2 002	1 003
3 003	767,19	10.03.2012	2 001	1 001

УПОРЯДОЧЕНИЕ С ПОМОЩЬЮ МНОГОЧИСЛЕНЫХ СТОЛБЦОВ

Мы можем также упорядочивать таблицу с помощью другого столбца, например с помощью поля amt, внутри упорядочения поля cnum :

SELECT * FROM Orders ORDER BY cnum DESC, amt DESC;

ONUM	AMT	ODATE	CNUM	SNUM
3 006	1098,16	10.03.2012	2 008	1 007
3 001	18,69	10.03.2012	2 008	1 007
3 002	1900,1	10.03.2012	2 007	1 004
3 011	9891,88	10.06.2012	2 006	1 001
3 008	4723	10.05.2012	2 006	1 001
3 010	1309,95	10.04.2012	2 004	1 002
3 007	75,75	10.04.2012	2 004	1 002
3 005	5016,45	10.03.2012	2 003	1 002
3 009	1713,23	10.04.2012	2 002	1 003
3 003	767,19	10.03.2012	2 001	1 001

ОБЪЕДИНЕНИЕ ТАБЛИЦ

Одна из наиболее важных особенностей запросов SQL — это их способность определять связи между многочисленными таблицами и выводить информацию из них внутри одной команды.

Этот вид операции называется — *объединением*, которое является одним из видов операций в реляционных базах данных. Главное в реляционном подходе – это связи которые можно создавать между позициями данных в таблицах.

Используя обьединения, мы непосредственно связываем информацию с любым полем таблицы, и таким образом способны создавать связи между фрагментами данных.

ИМЕНА ТАБЛИЦ И СТОЛБЦОВ

Полное имя столбца таблицы фактически состоит из имени таблицы, сопровождаемого точкой и затем именем столбца. Имеются несколько примеров имен :

Salespeople.snum
Salespeople.city
Orders.odate

До этого, опускали имена таблиц потому что запрашивали только одну таблицу. Когда делается запрос к не одной таблице, то можно опускать имена таблиц, если все ее столбцы имеют разные имена. Но это не всегда так бывает. Например, мы имеем две типовые таблицы со столбцами называемыми city.

Если мы должны связать эти столбцы, мы будем должны указать их с именами Salespeople.city или Customers.city, чтобы SQL мог их различать.

СОЗДАНИЕ ОБЬЕДИНЕНИЯ

Предположим что вы хотите поставить в соответствии вашему продавцу ваших заказчиков в городе в котором они живут, поэтому вы увидите все комбинации продавцов и заказчиков для этого города. Вы будете должны брать каждого продавца и искать в таблице Заказчиков всех заказчиков того же самого города. Вы могли бы сделать это, введя следующую команду:

SELECT Customers.cname, Salespeople.sname, Salespeople.city FROM Salespeople, Customers WHERE Salespeople.city = Customers.city;

CNAME	SNAME	CITY
Андреев	Сидоров	Благовещенск
Андреев	Александров	Благовещенск
Верещагин	Сидоров	Благовещенск
Верещагин	Александров	Благовещенск
Сергеев	Петров	Шимановск
Алексеев	Петров	Шимановск

ОБЪЕДИНЕНИЕ ТАБЛИЦ ЧЕРЕЗ СПРАВОЧНУЮ ЦЕЛОСТНОСТЬ

Эта особенность используется через встроенных связей в базу данных. В предыдущем примере, установили связь между двумя таблицами через простые поля. Но эти таблицы, уже были соединены через snum поле. Эта связь называется *состоянием справочной целостности*. Используя обьединение можно извлекать данные в терминах этой связи. Например, чтобы показать имена всех заказчиков соответствующих продавцам которые их обслуживают, мы будем использовать такой запрос:

```
SELECT Customers.cname, Salespeople.sname
FROM Customers, Salespeople
WHERE Salespeople.snum = Customers.snum;
```

ОБЪЕДИНЕНИЕ ТАБЛИЦ

Результат запроса:

CNAME	SNAME
Верещагин	Сидоров
Андреев	Сидоров
Алексеев	Питкин
Клопов	Петров
Курапаткин	Петров
Ситкин	Иванов
Сергеев	Александров

ОБЪЕДИНЕНИЯ ТАБЛИЦ ПО РАВЕНСТВУ ЗНАЧЕНИЙ В СТОЛБЦАХ И ДРУГИЕ ВИДЫ ОБЪЕДИНЕНИЙ

Обьединения которые используют предикаты основанные на равенствах называются — *объединениями по равенству*. Все наши примеры в этой главе до настоящего времени, относились именно к этой категории, потому что все условия в предложениях WHERE базировались на математических выражениях использующих знак равно (=).

Строки **'city = 'London'** и **'Salespeople.snum = Orders.snum'** — примеры таких типов равенств найденных в предикатах.

Объединения по равенству — это вероятно наиболее общий вид объединения, но имеются и другие. Вы можете, фактически, использовать любой из реляционных операторов в обьединении.

ОБЪЕДИНЕНИЕ ТАБЛИЦ

пример другого вида объединения

SELECT sname, cname FROM Salespeople, Customers WHERE sname < cname AND rating < 200;

SNAME	CNAME
Сидоров	Ситкин
Петров	Ситкин
Александров	Андреев
Александров	Верещагин
Александров	Ситкин
Иванов	Ситкин
Питкин	Ситкин

Эта команда не часто бывает полезна. Обычно, мы не создаеМ сложных связей подобно этой, и, по этой причине, мы вероятно будем строить наиболее общие ***объединения по равенству,*** но вы должны хорошо знать и другие возможности.

ОБЪЕДИНЕНИЕ БОЛЕЕ ДВУХ ТАБЛИЦ

Предположим что мы хотим найти всех заказчиков не находящихся в тех городах где находятся их продавцы. Для этого необходимо связать все три наши типовые таблицы:

SELECT onum, cname, Orders.cnum, Orders.snum FROM Salespeople, Customers, Orders WHERE Customers.city < > Salespeople.city AND Orders.cnum = Customers.cnum AND Orders.snum = Salespeople.snum;

ONUM	CNAME	CNUM	SNUM
3001	Алексеев	2008	1007
3002	Ситкин	2007	1004
3005	Клопов	2003	1002
3006	Алексеев	2008	1007
3007	Курапаткин	2004	1002
3009	Сергеев	2002	1003
3010	Курапаткин	2004	1002

ОБЪЕДИНЕНИЕ ТАБЛИЦ

РАБОТА С SQL

- Напишите запрос, который бы вывел список номеров заказов, сопровождающихся именем заказчика, который создавал эти заказы.
- Напишите запрос, который бы выдавал имена продавца и заказчика для каждого заказа после номера заказа.
- Напишите запрос, который бы выводил всех заказчиков, обслуживаемых продавцом с комиссионными выше 12%. Выведите имя заказчика, имя продавца и ставку комиссионных продавца.
- Напишите запрос, который вычислил бы сумму комиссионных продавца для каждого заказа заказчика с оценкой выше 100.

ОБЪЕДИНЕНИЕ ТАБЛИЦ

Ответы:

- `SELECT onum, cname FROM Orders, Customers WHERE Customers.cnum = Orders.cnum;`

- `SELECT onum, cname, sname FROM Orders, Customers, Salespeople WHERE Customers.cnum = Orders.cnum AND Salespeople.snum = Orders.snum;`

- `SELECT cname, sname, comm FROM Salespeople, Customers WHERE Salespeople.snum = Customers.snum AND comm * .12;`

- `SELECT onum, comm * amt FROM Salespeople, Orders, Customers WHERE rating > 100 AND Orders.cnum = Customers.cnum AND Orders.snum = Salespeople.snum;`

ОБЪЕДИНЕНИЕ ТАБЛИЦЫ С СОБОЙ

Это объединение — такое же, как и любое другое объединение между двумя таблицами, за исключением того, что в данном случае обе таблицы идентичны.

Синтаксис команды для объединения таблицы с собой, тот же что и для объединения многочисленых таблиц, в одном экземпляре.

Когда вы объединяете таблицу с собой, все повторяемые имена столбца, заполняются префиксами имени таблицы.

Вы можете сделать это с помощью определения временных имен называемых — *переменными диапазона, переменными корреляции* или просто — ***псевдонимами.***

Имеется пример, который находит все пары заказчиков имеющих один и тот же самый рейтинг:

SELECT first.cname, second.cname, first.rating FROM Customers first, Customers second WHERE first.rating = second.rating;

ОБЪЕДИНЕНИЕ ТАБЛИЦЫ С СОБОЙ

CNAME	CNAME1	RATING
Верещагин	Верещагин	100
Ситкин	Верещагин	100
Андреев	Верещагин	100
Верещагин	Ситкин	100
Ситкин	Ситкин	100
Андреев	Ситкин	100
Верещагин	Андреев	100
Ситкин	Андреев	100
Андреев	Андреев	100
Сергеев	Сергеев	200
Клопов	Сергеев	200
Сергеев	Клопов	200
Клопов	Клопов	200
Алексеев	Алексеев	300
Курапаткин	Алексеев	300
Алексеев	Курапаткин	300
Курапаткин	Курапаткин	300

УСТРАНЕНИЕ ИЗБЫТОЧНОСТИ

Обратите внимание, что наш вывод имеет два значение для каждой комбинации, причем второй раз в обратном порядке. Это потому, что каждое значение показано первый раз в каждом псевдониме, и второй раз (симметрично) в предикате.

Следовательно, значение **A** в псевдониме сначала выбирается в комбинации со значением **B** во втором псевдониме, а затем значение **A** во втором псевдониме выбирается в комбинации со значением **B** в первом псевдониме.

ОБЪЕДИНЕНИЕ ТАБЛИЦЫ С СОБОЙ

Простой способ избежать этого состоит в том, чтобы налагать порядок на два значения, так чтобы один мог быть меньше чем другой или предшествовал ему в алфавитном порядке. Это делает предикат ассиметричным, поэтому те же самые значения в обратном порядке не будут выбраны снова, например:

SELECT tirst.cname, second.cname, first.rating FROM Customers first, Customers second WHERE first.rating = second.rating AND first.cname < second.cname;

CNAME	CNAME1	RATING
Андреев	Верещагин	100
Верещагин	Ситкин	100
Андреев	Ситкин	100
Клопов	Сергеев	200
Алексеев	Курапаткин	300

ОБЪЕДИНЕНИЕ ТАБЛИЦЫ С СОБОЙ

ЕЩЕ БОЛЬШЕ КОМПЛЕКСНЫХ ОБЪЕДИНЕНИЙ

Предположим что вы еще не назначили ваших заказчиков к вашему продавцу. Компания должна назначить каждому продавцу первоначально трех заказчиков, по одному для каждого рейтингового значения. Вы лично можете решить, какого заказчика какому продавцу назначить, но следующий запрос вы используете, чтобы увидеть все возможные комбинации заказчиков, которых вы можете назначать.

```
SELECT a.cnum, b.cnum, c.cnum FROM Customers a, Customers b, Customers c WHERE a.rating = 100 AND b.rating = 200 AND c.rating = 300;
```

CNUM	CNUM 1	CNUM 2
2 001	2 002	2 004
2 001	2 002	2 008
2 001	2 003	2 004
2 001	2 003	2 008
2 006	2 002	2 004
2 006	2 002	2 008
2 006	2 003	2 004
2 006	2 003	2 008
2 007	2 002	2 004
2 007	2 002	2 008
2 007	2 003	2 004
2 007	2 003	2 008

ВСТАВКА ОДНОГО ЗАПРОСА ВНУТРЬ ДРУГОГО

С помощью SQL вы можете вкладывать запросы внутрь друг друга. Обычно, внутренний запрос генерирует значение, которое проверяется в предикате внешнего запроса, определяющего, верно оно или нет.

Например, предположим что мы знаем имя продавца: Сидоров, но не знаем значение его поля snum, и хотим извлечь все заказы из таблицы Заказов. Имеется один способ чтобы сделать это

```
SELECT * FROM Orders WHERE snum = (SELECT snum FROM
Salespeople WHERE sname = 'Сидоров');
```

ONUM	AMT	ODATE	CNUM	SNUM
3003	767,19	10.03.2012	2001	1001
3008	4723	10.05.2012	2006	1001
3011	9891,88	10.06.2012	2006	1001

Ограничения

SELECT * FROM Orders WHERE snum = (SELECT snum FROM Salespeople WHERE city = ‘Шимановск’);

Поскольку мы имеем только одного продавца в **Шимановск** — Петров, то подзапрос будет выбирать одиночное значение snum и следовательно будет принят. Но это — только в данном случае. Большинство SQL баз данных имеют многочисленных пользователей, и если другой пользователь добавит нового продавца из **Шимановска** в таблицу, подзапрос выберет два значения, и ваша команда потерпит неудачу.

DISTINCT С ПОДЗАПРОСАМИ

Вы можете, в некоторых случаях, использовать DISTINCT, чтобы вынудить подзапрос генерировать одиночное значение.

Предположим что мы хотим найти все порядки кредитований для тех продавцов которые обслуживают Алексеева (cnum = 2008). Имеется один способ, чтобы сделать это:

```
SELECT * FROM Orders WHERE snum = ( SELECT
DISTINCT snum Orders WHERE cnum = 2008 );
```

ИСПОЛЬЗОВАНИЕ ПОДЗАПРОСОВ, КОТОРЫЕ ВЫДАЮТ МНОГО СТРОК С ПОМОЩЬЮ ОПЕРАТОРА IN

Вы можете использовать подзапросы которые производят любое число строк если вы используете специальный оператор IN (операторы BETWEEN, LIKE, и IS NULL не могут использоваться с подзапросами). Мы можем использовать IN чтобы выполнить такой же подзапрос, который не будет работать с реляционным оператором, и найти все атрибуты таблицы Заказов для продавца в Благовещенске:

```
SELECT * FROM Orders WHERE snum IN ( SELECT snum FROM
Salespeople WHERE city = 'Благовещенск' );
```

ONUM	AMT	ODATE	CNUM	SNUM
3003	767,19	10.03.2012	2001	1001
3008	4723	10.05.2012	2006	1001
3009	1713,23	10.04.2012	2002	1003
3011	9891,88	10.06.2012	2006	1001

ИСПОЛЬЗОВАНИЕ АГРЕГАТНЫХ ФУНКЦИЙ В ПОДЗАПРОСАХ

Один тип функций, который автоматически может производить одиночное значение для любого числа строк, конечно же, — агрегатная функция. Любой запрос, использующий одиночную функцию агрегата без предложения GROUP BY, будет выбирать одиночное значение для использования в основном предикате. Например, вы хотите увидеть все порядки имеющие сумму приобретений выше средней на 10-е апреля:

```
SELECT * FROM Orders WHERE amt > ( SELECT AVG (amt)  FROM Orders   WHERE odate = '10.04.2012' );
```

ПОДЗАПРОСЫ

ONUM	AMT	ODATE	CNUM	SNUM
3002	1900,1	10.03.2012	2007	1004
3005	5016,45	10.03.2012	2003	1002
3006	1098,16	10.03.2012	2008	1007
3008	4723	10.05.2012	2006	1001
3009	1713,23	10.04.2012	2002	1003
3010	1309,95	10.04.2012	2004	1002
3011	9891,88	10.06.2012	2006	1001

EXISTS

EXISTS — это оператор, который производит верное или неверное значение, другими словами, выражение Буля. Это означает, что он может работать автономно в предикате или в комбинации с другими выражениями Буля, использующими Булевы операторы AND, OR, и NOT. Он берет подзапрос как аргумент и оценивает его как верный, если тот производит любой вывод или как неверный, если тот не делает этого. Например, мы можем решить, извлекать ли нам некоторые данные из таблицы Заказчиков если, и только если, один или более заказчиков в этой таблице находятся в '**Благовещенске**:

SELECT cnum, cname, city FROM Customers WHERE EXISTS (SELECT * FROM Customers WHERE city = 'Благовещенск');

ОБЪЕДИНЕНИЕ МНОГОЧИСЛЕННЫХ ЗАПРОСОВ В ОДИН

Вы можете поместить многочисленые запросы вместе и объединить их вывод, используя предложение UNION. Предложение UNION обьединяет вывод двух или более SQL запросов в единый набор строк и столбцов. Например, чтобы получить всех продавцов и заказчиков размещенных в Благовещенске и вывести их как единое целое вы могли бы ввести:

```
SELECT snum, sname FROM Salespeople WHERE city =
'Благовещенск' UNION SELECT cnum, cname FROM Customers
WHERE city = 'Благовещенск';
```

SNUM	SNAME
1001	Сидоров
1003	Александров
2001	Андреев
2006	Верещагин

Виды оператора JOIN

INNER JOIN
LEFT OUTER JOIN
RIGHT OUTER JOIN
FULL OUTER JOIN
CROSS JOIN

СОЕДИНЕНИЕ ТАБЛИЦ

SELECT O.*, S.* FROM SALESPEOPLE S INNER JOIN ORDERS O ON O.SNUM=S.SNUM where S.SNUM<>1001

ONUM	AMT	ODATE	CNUM	SNUM	SNUM1	SNAME	CITY	COMM
3 001	18,69	10.03.2012	2 008	1 007	1 007	Питкин	Свободный	0,1
3 002	1 900,100	10.03.2012	2 007	1 004	1 004	Иванов	Райчихинск	0,15
3 005	5 016,450	10.03.2012	2 003	1 002	1 002	Петров	Шимановск	0,13
3 006	1 098,160	10.03.2012	2 008	1 007	1 007	Питкин	Свободный	0,1
3 007	75,75	10.04.2012	2 004	1 002	1 002	Петров	Шимановск	0,13
3 009	1 713,230	10.04.2012	2 002	1 003	1 003	Александров	Благовещенск	0,11
3 010	1 309,950	10.04.2012	2 004	1 002	1 002	Петров	Шимановск	0,13

SELECT O.*, S.* FROM SALESPEOPLE S LEFT OUTER JOIN ORDERS O ON O.SNUM=S.SNUM where S.SNUM<>1001

ONUM	AMT	ODATE	CNUM	SNUM	SNUM1	SNAME	CITY	COMM
3 005	5 016,450	10.03.2012	2 003	1 002	1 002	Петров	Шимановск	0,13
3 007	75,75	10.04.2012	2 004	1 002	1 002	Петров	Шимановск	0,13
3 010	1 309,950	10.04.2012	2 004	1 002	1 002	Петров	Шимановск	0,13
3 009	1 713,230	10.04.2012	2 002	1 003	1 003	Александров	Благовещенск	0,11
3 002	1 900,100	10.03.2012	2 007	1 004	1 004	Иванов	Райчихинск	0,15
3 001	18,69	10.03.2012	2 008	1 007	1 007	Питкин	Свободный	0,1
3 006	1 098,160	10.03.2012	2 008	1 007	1 007	Питкин	Свободный	0,1
<null>	<null>	<null>	<null>	<null>	1 008	<null>	<null>	<null>

СОЕДИНЕНИЕ ТАБЛИЦ

SELECT O.*, S.* FROM SALESPEOPLE S RIGHT OUTER JOIN ORDERS O ON O.SNUM=S.SNUM where S.SNUM<>1001

ONUM	AMT	ODATE	CNUM	SNUM	SNUM1	SNAME	CITY	COMM
3 001	18,69	10.03.2012	2 008	1 007	1 007	Питкин	Свободный	0,1
3 002	1 900,100	10.03.2012	2 007	1 004	1 004	Иванов	Райчихинск	0,15
3 005	5 016,450	10.03.2012	2 003	1 002	1 002	Петров	Шимановск	0,13
3 006	1 098,160	10.03.2012	2 008	1 007	1 007	Питкин	Свободный	0,1
3 007	75,75	10.04.2012	2 004	1 002	1 002	Петров	Шимановск	0,13
3 009	1 713,230	10.04.2012	2 002	1 003	1 003	Александров	Благовещенск	0,11
3 010	1 309,950	10.04.2012	2 004	1 002	1 002	Петров	Шимановск	0,13

SELECT O.*, S.* FROM SALESPEOPLE S FULL OUTER JOIN ORDERS O ON O.SNUM=S.SNUM where S.SNUM<>1001

ONUM	AMT	ODATE	CNUM	SNUM	SNUM1	SNAME	CITY	COMM
3 001	18,69	10.03.2012	2 008	1 007	1 007	Питкин	Свободный	0,1
3 002	1 900,100	10.03.2012	2 007	1 004	1 004	Иванов	Райчихинск	0,15
3 005	5 016,450	10.03.2012	2 003	1 002	1 002	Петров	Шимановск	0,13
3 006	1 098,160	10.03.2012	2 008	1 007	1 007	Питкин	Свободный	0,1
3 007	75,75	10.04.2012	2 004	1 002	1 002	Петров	Шимановск	0,13
3 009	1 713,230	10.04.2012	2 002	1 003	1 003	Александров	Благовещенск	0,11
3 010	1 309,950	10.04.2012	2 004	1 002	1 002	Петров	Шимановск	0,13
<null>	<null>	<null>	<null>	<null>	1 008	<null>	<null>	<null>

СОЕДИНЕНИЕ ТАБЛИЦ

SELECT O.*, S.* FROM SALESPEOPLE S CROSS JOIN ORDERS O where S.SNUM<>1001

ONUM	AMT	ODATE	CNUM	SNUM	SNUM1	SNAME	CITY	COMM
3 001	18,69	10.03.2012	2 008	1 007	1 002	Петров	Шимановск	0,13
3 002	1 900,100	10.03.2012	2 007	1 004	1 002	Петров	Шимановск	0,13
3 003	767,19	10.03.2012	2 001	1 001	1 002	Петров	Шимановск	0,13
3 005	5 016,450	10.03.2012	2 003	1 002	1 002	Петров	Шимановск	0,13
3 006	1 098,160	10.03.2012	2 008	1 007	1 002	Петров	Шимановск	0,13
3 007	75,75	10.04.2012	2 004	1 002	1 002	Петров	Шимановск	0,13
3 008	4 723,000	10.05.2012	2 006	1 001	1 002	Петров	Шимановск	0,13
3 009	1 713,230	10.04.2012	2 002	1 003	1 002	Петров	Шимановск	0,13
3 010	1 309,950	10.04.2012	2 004	1 002	1 002	Петров	Шимановск	0,13
3 011	9 891,880	10.06.2012	2 006	1 001	1 002	Петров	Шимановск	0,13
3 001	18,69	10.03.2012	2 008	1 007	1 003	Александров	Благовещенск	0,11
3 002	1 900,100	10.03.2012	2 007	1 004	1 003	Александров	Благовещенск	0,11
3 003	767,19	10.03.2012	2 001	1 001	1 003	Александров	Благовещенск	0,11
3 005	5 016,450	10.03.2012	2 003	1 002	1 003	Александров	Благовещенск	0,11
3 006	1 098,160	10.03.2012	2 008	1 007	1 003	Александров	Благовещенск	0,11
3 007	75,75	10.04.2012	2 004	1 002	1 003	Александров	Благовещенск	0,11
3 008	4 723,000	10.05.2012	2 006	1 001	1 003	Александров	Благовещенск	0,11
3 009	1 713,230	10.04.2012	2 002	1 003	1 003	Александров	Благовещенск	0,11
3 010	1 309,950	10.04.2012	2 004	1 002	1 003	Александров	Благовещенск	0,11
3 011	9 891,880	10.06.2012	2 006	1 001	1 003	Александров	Благовещенск	0,11
3 001	18,69	10.03.2012	2 008	1 007	1 004	Иванов	Райчихинск	0,15
3 002	1 900,100	10.03.2012	2 007	1 004	1 004	Иванов	Райчихинск	0,15
3 003	767,19	10.03.2012	2 001	1 001	1 004	Иванов	Райчихинск	0,15
3 005	5 016,450	10.03.2012	2 003	1 002	1 004	Иванов	Райчихинск	0,15
3 006	1 098,160	10.03.2012	2 008	1 007	1 004	Иванов	Райчихинск	0,15
3 007	75,75	10.04.2012	2 004	1 002	1 004	Иванов	Райчихинск	0,15
3 008	4 723,000	10.05.2012	2 006	1 001	1 004	Иванов	Райчихинск	0,15
3 009	1 713,230	10.04.2012	2 002	1 003	1 004	Иванов	Райчихинск	0,15
3 010	1 309,950	10.04.2012	2 004	1 002	1 004	Иванов	Райчихинск	0,15
3 011	9 891,880	10.06.2012	2 006	1 001	1 004	Иванов	Райчихинск	0,15
3 001	18,69	10.03.2012	2 008	1 007	1 007	Питкин	Свободный	0,1
3 002	1 900,100	10.03.2012	2 007	1 004	1 007	Питкин	Свободный	0,1
3 003	767,19	10.03.2012	2 001	1 001	1 007	Питкин	Свободный	0,1
3 005	5 016,450	10.03.2012	2 003	1 002	1 007	Питкин	Свободный	0,1
3 006	1 098,160	10.03.2012	2 008	1 007	1 007	Питкин	Свободный	0,1
3 007	75,75	10.04.2012	2 004	1 002	1 007	Питкин	Свободный	0,1
3 008	4 723,000	10.05.2012	2 006	1 001	1 007	Питкин	Свободный	0,1
3 009	1 713,230	10.04.2012	2 002	1 003	1 007	Питкин	Свободный	0,1
3 010	1 309,950	10.04.2012	2 004	1 002	1 007	Питкин	Свободный	0,1
3 011	9 891,880	10.06.2012	2 006	1 001	1 007	Питкин	Свободный	0,1
3 001	18,69	10.03.2012	2 008	1 007	1 008	<null>	<null>	<null>
3 002	1 900,100	10.03.2012	2 007	1 004	1 008	<null>	<null>	<null>
3 003	767,19	10.03.2012	2 001	1 001	1 008	<null>	<null>	<null>
3 005	5 016,450	10.03.2012	2 003	1 002	1 008	<null>	<null>	<null>
3 006	1 098,160	10.03.2012	2 008	1 007	1 008	<null>	<null>	<null>
3 007	75,75	10.04.2012	2 004	1 002	1 008	<null>	<null>	<null>
3 008	4 723,000	10.05.2012	2 006	1 001	1 008	<null>	<null>	<null>
3 009	1 713,230	10.04.2012	2 002	1 003	1 008	<null>	<null>	<null>
3 010	1 309,950	10.04.2012	2 004	1 002	1 008	<null>	<null>	<null>
3 011	9 891,880	10.06.2012	2 006	1 001	1 008	<null>	<null>	<null>

КОМАНДЫ МОДИФИКАЦИИ ЯЗЫКА DML

КОМАНДЫ МОДИФИКАЦИИ ЯЗЫКА DML

Значения могут быть помещены и удалены из полей, тремя командами языка DML (Язык Манипулирования Данными):

INSERT (ВСТАВИТЬ),
UPDATE (МОДИФИЦИРОВАТЬ),
DELETE (УДАЛИТЬ).

КОМАНДЫ МОДИФИКАЦИИ ЯЗЫКА DML

Все строки в SQL вводятся с использованием команды модификации INSERT. В самой простой форме, INSERT использует следующий синтаксис:

INSERT INTO <table name>
VALUES (<value>, <value> . . .);

Так, например, чтобы ввести строку в таблицу Продавцов, вы можете использовать следующее условие:

INSERT INTO Salespeople
VALUES (1009, 'Лукашев', 'Белогорск', .16);

Команды DML не производят никакого вывода, но ваша программа должна дать вам некоторое подтверждение того, что данные были использованы.

ВСТАВКА ПУСТЫХ УКАЗАТЕЛЕЙ (NULL)

Если вам нужно ввести *пустое* значение (NULL), вы вводите его точно также как и обычное значение. Предположим, что еще не имелось поля city для мистера Лукашев. Вы можете вставить его строку со значением=NULL в это поле, следующим образом:

```
INSERT INTO Salespeople VALUES (1001, 'Лукашев', NULL, .16);
```

Так как значение NULL — это специальный маркер, а не просто символьное значение, он не включяется в одиночные кавычки.

Если у нас есть триггер на вставку первичного ключа, то записываем:

```
INSERT INTO Salespeople VALUES (NULL, 'Лукашев',
'Белогорск', .16);
```

ИМЕНОВАНИЕ СТОЛБЦА ДЛЯ ВСТАВКИ (INSERT)

Вы можете также указывать столбцы, куда вы хотите вставить значение имени. Это позволяет вам вставлять имена в любом порядке.

INSERT INTO Customers (city, cname, cnum) VALUES ('Шимановск', 'Пупкин', 2001);

Обратите внимание, что столбцы rating и snum — отсутствуют. Это значит, что эти строки автоматически установлены в значение по умолчанию. По умолчанию может быть введено или значение NULL или другое значение, определяемое как значение по умолчанию. Если ограничение запрещает использование значения NULL в данном столбце, и этот столбец не установлен как по умолчанию, этот столбец должен быть обеспечен значением для любой команды INSERT, которая относится к таблице.

ВСТАВКА РЕЗУЛЬТАТОВ ЗАПРОСА

Вы можете также использовать команду INSERT чтобы получать или выбирать значения из одной таблицы и помещать их в другую, чтобы использовать их вместе с запросом. Чтобы сделать это, вы просто заменяете предложение VALUES (из предыдущего примера) на соответствующий запрос:

```
INSERT INTO Blagastaff SELECT * FROM Salespeople
WHERE city = 'Благовещенск';
```

Здесь выбираются все значения произведенные запросом — то-есть все строки из таблицы Продавцов со значениями city = "**Благовещенск**" — и помещаются в таблицу называемую **Blagastaff**.

Чтобы это работало, таблица **Blagastaff** должна отвечать следующим условиям:

- **Она должна уже быть создана командой CREATE TABLE.**
- **Она должна иметь четыре столбца которые совпадают с таблицей Продавцов в терминах типа данных;**

то-есть первый, второй, и так далее, столбцы каждой таблицы, должны иметь одинаковый тип данных (причем они не должны иметь одинаковых имен).

УДАЛЕНИЕ СТРОК ИЗ ТАБЛИЦ

Вы можете удалять строки из таблицы командой модификации — DELETE. Она может удалять только введеные строки, а не индивидуальные значения полей, так что параметр поля является необязательным или недоступным. Чтобы удалить все содержание таблицы Продавцов, вы можете ввести следующее условие:

```
DELETE FROM Salespeople;
```

Теперь, когда таблица пуста, ее можно окончательно удалить командой DROP TABLE (об этом позже).

Обычно, вам нужно удалить только некоторые определенные строки из таблицы. Чтобы определить какие строки будут удалены, вы используете предикат, так же как вы это делали для запросов. Например, чтобы удалить продавца Сидоров из таблицы, вы можете ввести

DELETE FROM Salespeople WHERE snum = 1003;

Вы можете также использовать DELETE с предикатом, который бы выбирал группу строк, как показано в этом примере:

```
DELETE FROM Salespeople WHERE city = 'Благовещенск';
```

ИЗМЕНЕНИЕ ЗНАЧЕНИЙ ПОЛЯ

Эта команда содержит предложение UPDATE, в котором указано имя используемой таблицы и предложение SET, которое указывает на изменение, которое нужно сделать для определенного столбца. Например, чтобы изменить оценки всех заказчиков на 200, вы можете ввести

```
UPDATE Customers SET rating = 200;
```

МОДИФИЦИРОВАНИЕ ТОЛЬКО ОПРЕДЕЛЕННЫХ СТРОК

Конечно, вы не всегда захотите указывать все строки таблицы для изменения единственного значения, так что UPDATE, наподобии DELETE, может брать предикаты. Вот как например можно выполнить изменение одинаковое для всех заказчиков продавца Андреев (имеющего snum=2001):

```
UPDATE Customers SET rating = 200 WHERE snum = 1001;
```

КОМАНДЫ МОДИФИКАЦИИ ЯЗЫКА DML

С командами **insert**, **update**, **delete** можно использовать подзапросы. Например:

```
INSERT INTO Sjpeople SELECT * FROM Salespeople WHERE city = 'Шимановск';

DELETE FROM Salespeople WHERE 100 IN ( SELECT rating FROM Customers WHERE Salespeople.snum = Customers.snum);

UPDATE Salespeople SET comm = comm + .01 WHERE 2 <= ( SELECT COUNT (cnum) FROM Customers WHERE Customers.snum = Salespeople.snum );
```

КОМАНДЫ МОДИФИКАЦИИ ЯЗЫКА DML

1. Напишите команду, которая бы удаляла всех заказчиков, не имеющих текущих заказов.

2. Напишите команду которая бы увеличила на двадцать процентов комиссионные всех продавцов, имеющих общие текущие заказы выше чем $3000.

СОЗДАНИЕ И ИЗМЕНЕНИЕ ТАБЛИЦ

Таблицы создаются командой **CREATE TABLE**. Эта команда создает пустую таблицу — таблицу без строк. Значения вводятся с помощью DML команды **INSERT**. Команда CREATE TABLE в основном определяет имя таблицы, в виде описания набора имен столбцов указанных в определенном порядке. Она также определяет типы данных и размеры столбцов. Каждая таблица должна иметь по крайней мере один столбец.

Синтаксис команды CREATE TABLE:

CREATE TABLE <table-name > (<column name > <data type>[(<size>)], <column name > <data type> [(<size>)] ...);

СОЗДАНИЕ И ИЗМЕНЕНИЕ ТАБЛИЦ

Например, ниже команда создаст таблицу Продавцов:

```
CREATE TABLE Salespeople
( snum  integer,
  sname char (10),
  city  char (10),
  comm  declmal );
```

СОЗДАНИЕ И ИЗМЕНЕНИЕ ТАБЛИЦ

Индекс — это упорядоченный (буквенный или числовой) список столбцов или групп столбцов в таблице. Таблицы могут иметь большое колличество строк, а, так как строки не находятся в каком-нибудь определенном порядке, на их поиск по указанному значению может потребовать время.

Синтаксис для создания индекса — обычно следующий (помните, что это не ANSI стандарт):

CREATE INDEX <index name> ON <table name> (<column name> [,<column name>]...);

Если, например, таблица Заказчиков будет наиболее часто упоминаемой в запросах продавцов к их собственной клиентуре, было бы правильно создать такой индекс в поле snum таблицы Заказчиков.

CREATE INDEX Clientgroup ON Customers (snum);

Теперь, тот продавец, который имеет отношение к этой таблице, сможет найти собственную клиентуру очень быстро.

Но этот индекс не уникальный!!!

СОЗДАНИЕ И ИЗМЕНЕНИЕ ТАБЛИЦ

Уникальность индекса будет создана если мы используем ключевое слово **UNIQUE** перед ключевым словом **INDEX**. Поле cnum, в качестве первичного ключа, станет первым кандидатом для уникального индекса:

CREATE UNIQUE INDEX Custid ON Customers (cnum)

ПРИМЕЧАНИЕ: эта команда будет отклонена, если уже имеются идентичные значения в поле cnum.

Удаление индексов

Синтаксис используемый для удаления индекса:

DROP INDEX <Index name>;

Удаление индекса не изменяет содержание полей.

Изменение таблицы

Типичный синтаксис чтобы добавить столбец к таблице:

ALTER TABLE <table name> ADD <column name> <data type> <size>;

Столбец будет добавлен со значением NULL для всех строк таблицы. Новый столбец станет последним по порядку столбцом таблицы.

УДАЛЕНИЕ ТАБЛИЦ

Вы должны быть собственником (т.е. быть создателем) таблицы, чтобы иметь возможность удалить ее. Случайное разрушение данных не возможно, SQL сначала потребует чтобы вы очистили таблицу прежде, чем удалит ее из базы данных.

Таблица с находящимися в ней строками, не может быть удалена.

СОЗДАНИЕ И ИЗМЕНЕНИЕ ТАБЛИЦ

Синтаксис для удаления вашей таблицы, если конечно она является пустой, следующая:

DROP TABLE <table name>;

При подаче этой команды, имя таблицы больше не распознается и нет такой команды, которая могла быть дана этому объекту.

Вы должны убедиться, что эта таблица не ссылается внешним ключом к другой таблице и что она не используется в определении Представления.

СОЗДАНИЕ И ИЗМЕНЕНИЕ ТАБЛИЦ

Типы таблиц, с которыми вы имели дело до сих пор, назывались — *базовыми таблицами*. Имеется другой вид таблиц — представления.

Представления — это таблицы, чье содержание выбирается или получается из других таблиц. Они работают в запросах и операторах DML точно также как и основные таблицы, но не содержат никаких собственных данных.

Представления подобны окнам, через которые вы просматриваете информацию (как она есть, или в другой форме, как вы потом увидите), которая фактически хранится в базовой таблице.

СОЗДАНИЕ И ИЗМЕНЕНИЕ ТАБЛИЦ

Представление — это фактически запрос, который выполняется всякий раз, когда представление становится темой команды. Вывод запроса при этом в каждый момент становится содержанием представления.

КОМАНДА CREATE VIEW

Вы создаете представление командой **CREATE VIEW.** Она состоит из слов **CREATE VIEW** (*СОЗДАТЬ ПРЕДСТАВЛЕНИЕ*), *имени представления*, которое нужно создать, слова **AS** (*КАК*), и далее *запроса*, как в следующем примере:

```
CREATE VIEW Blagstaff AS SELECT * FROM Salespeople
WHERE city = ‘Благовещенск';
```

Теперь Вы имеете представление, называемое **Blagstaff**.

Вы можете использовать это представление точно так же, как и любую другую таблицу. Она может быть запрошена, модифицирована, вставлена в, удалена из, и соединена с, другими таблицами и представлениями. Если сделать запрос такого представления:

SELECT *FROM Blagstaff;

Выводом этого представления будут две строки с данными из таблицы **Salespeople** где содержание поля **city** будет: Благовещенск

СОЗДАНИЕ И ИЗМЕНЕНИЕ ТАБЛИЦ

ИСПОЛЬЗОВАНИЕ СПЕЦИАЛЬНЫХ ОПЕРАТОРОВ В ПРЕДИКАТАХ

Самостоятельная работа

1. Оператор LIKE
   - Символ подчеркивания _
   - Символ %
   - Символ / (экранирования)
2. АГРЕГАТНЫЕ ФУНКЦИИ
   - COUNT
   - AVG
   - MAX
   - MIN
3. УПОРЯДОЧЕНИЕ ВЫВОДА
   - Команда ORDER BY
   - Команда DESC
4. ПРЕДЛОЖЕНИЕ GROUP BY