Глава 6

ОСНОВЫ SQL

6.1. ВВЕДЕНИЕ В ЯЗЫК SQL

В начале 1970-х гг. прошлого столетия в одной из исследовательских лабораторий компании IBM была разработана экспериментальная реляционная система управления базами данных, для которой затем был создан специальный ***язык SQL*** (Structured Query Language — язык структурированных запросов). Стандарт на язык SQL был выпущен Американским национальным институтом стандартов (ANSI) в 1986 г., а в 1987 г. Международная организация стандартов (ISO) приняла его в качестве международного.

В настоящее время язык SQL является первым и пока единственным стандартным языком работы с базами данных. Язык SQL поддерживается многими СУБД различных типов, разработанных для самых разнообразных вычислительных платформ. Практически все крупнейшие разработчики СУБД в настоящее время создают свои продукты, нацеленные на использование языка SQL.

SQL является прежде всего информационно-логическим языком, предназначенным для описания, изменения и извлечения данных, хранимых в реляционных базах данных. Это непроцедурный язык; более того, его нельзя назвать языком программирования. Язык SQL ориентирован на операции с данными, представленными в виде логически взаимосвязанных таблиц, что позволило создать компактный язык с небольшим набором предложений. Важнейшей особенностью структуры языка SQL является ориентация на конечный результат обработки данных, а не на процедуру этой обработки. Язык SQL сам определяет, где находятся данные, индексы и даже какие наиболее эффективные последовательности операций следует использовать для получения результата, а потому указывать эти детали в запросе к базе данных не требуется. Кроме того, язык

SQL может использоваться как для выполнения запросов к данным, так и для построения прикладных программ.

Для расширения функциональных возможностей многие разработчики, придерживающиеся принятых стандартов, добавляют к стандартному языку SQL различные расширения.

Реализацией языка SQL называется программный продукт SQL соответствующего производителя.

Все конкретные реализации языка несколько отличаются друг от друга. В интересах самих же производителей гарантировать, чтобы их реализация соответствовала современным стандартам ANSI в части переносимости и удобства работы пользователей. Тем не менее каждая реализация SQL содержит усовершенствования, отвечающие требованиям того или иного сервера баз данных. Эти усовершенствования или расширения языка SQL представляют собой дополнительные команды и опции, являющиеся добавлениями к стандартному пакету и доступные в данной конкретной реализации.

Язык SQL включает только команды определения и манипулирования данными и не содержит каких-либо команд управления ходом вычислений. Подобные задачи должны решаться либо с помощью языков программирования, либо интерактивно — действиями самих пользователей. Язык SQL может использоваться двумя способами:

1) интерактивная работа, заключающаяся во вводе пользователем отдельных SQL-операторов;

2) внедрение SQL-операторов в программы на процедурных языках.

Язык SQL относительно прост в изучении. Поскольку это не процедурный язык, в нем необходимо указывать, какая информация должна быть получена, а не как ее можно получить. Иначе говоря, SQL не требует указания методов доступа к данным. Язык SQL может использоваться широким кругом специалистов, включая администраторов баз данных, прикладных программистов и множество других конечных пользователей, не имеющих навыков программирования.

По типу производимых действий различают следующие операции:

- идентификация данных и нахождение их позиции в БД;
- выборка (чтение) данных из БД;
- включение (запись) данных в БД;
- удаление данных из БД;
- модификация (изменение) данных БД.

Основные категории команд языка SQL предназначены для выполнения различных функций, включая построение объектов базы данных и манипулирование ими, начальную загрузку данных в таблицы, обновление и удаление существующей информации, выполнение запросов к базе данных, управление доступом к ней и ее общее администрирование.

Основные категории команд языка SQL:

- DDL — язык определения данных;
- DML — язык манипулирования данными;
- DQL — язык запросов;
- DCL — язык управления данными;
- команды администрирования данных;
- команды управления транзакциями.

Язык определения данных DDL (Data Definition Language) позволяет создавать и изменять структуру объектов базы данных, например, создавать и удалять таблицы. Основными командами языка DDL являются следующие:

- `CREATE TABLE` — создание таблицы;
- `ALTER TABLE` — изменение таблицы;
- `DROP TABLE` — удаление таблицы;
- `CREATE INDEX` — создание индекса;
- `ALTER INDEX` — изменение индекса;
- `DROP INDEX` — удаление индекса.

Язык манипулирования данными DML (Data Manipulation Language) используется для манипулирования информацией внутри объектов реляционной базы данных посредством трех основных команд:

- `INSERT` — вставка записей;
- `UPDATE` — обновление записей;
- `DELETE` — удаление записей.

Любая модель данных определяет множество действий, которые допустимо производить над некоторой реализацией БД для ее перевода из одного состояния в другое.

Язык запросов DQL (Data Query Language) наиболее известен пользователям реляционной базы данных, несмотря на то что он включает всего одну команду SELECT, которая возвращает строки из базы данных и позволяет делать выборку одной или нескольких строк или столбцов из одной или нескольких таблиц. Любая операция над данными включает в себя селекцию данных (select), т. е. выделение из всей совокупности именно тех данных, над которыми должна быть выполнена требуемая операция, и действие над выбранными данными, которое определяет характер операции. Усло-

вие селекции — это некоторый критерий отбора данных, в котором могут быть использованы логическая позиция элемента данных, его значение и связи между данными.

Язык управления данными DCL (Data Control Language) включает команды управления данными, которые позволяют управлять доступом к информации, находящейся внутри базы данных. Как правило, они используются для создания объектов, связанных с доступом к данным, а также служат для контроля над распределением привилегий между пользователями. Команды управления данными следующие:

- `GRANT` — установить права доступа;
- `REVOKE` — аннулировать права доступа.

Синтаксис этих команд зависит от СУБД. Для того чтобы упростить процесс управления доступом, многие СУБД предоставляют возможность объединять пользователей в группы или определять роли — совокупность привилегий, предоставляемых пользователю.

Такой подход позволяет предоставить конкретному пользователю определенную роль или соотнести его определенной группе пользователей, обладающей набором прав в соответствии с задачами, которые на нее возложены.

С помощью команд администрирования данных пользователь осуществляет контроль над выполнением действий с базой данных, анализирует операции базы данных, анализирует производительность системы и т. п. Следует отметить, что администрирование данных и администрирование базы данных не одно и то же. Администрирование базы данных представляет собой общее управление базой данных и подразумевает использование команд всех уровней.

Команды управления транзакциями включают в себя следующие команды:

- `COMMIT` — подтверждение транзакции;
- `ROLLBACK` — откат транзакции;
- `SAVEPOINT` — установка точки прерывания (неполный откат);
- `SET TRANSACTION` — начало транзакции.

Для успешного изучения языка SQL необходимо привести краткое описание структуры SQL-операторов и нотации, которые используются для определения формата различных конструкций языка. Оператор SQL состоит из зарезервированных слов, а также из слов, определяемых пользователем. Зарезервированные слова являются постоянной частью языка SQL и имеют фиксированное значение. Слова, определяемые пользователем, задаются им самим

(в соответствии с синтаксическими правилами) и представляют собой идентификаторы или имена различных объектов базы данных. Слова в операторе размещаются также в соответствии с установленными синтаксическими правилами.

Идентификаторы языка SQL предназначены для обозначения объектов в базе данных и являются именами таблиц, представлений, столбцов и других объектов базы данных. Стандарт SQL задает набор символов, который используется по умолчанию. Он включает строчные и прописные буквы латинского алфавита (A–Z, a–z), цифры (0–9) и символ подчеркивания (_). На формат идентификатора накладываются следующие ограничения:

- идентификатор может иметь длину до 128 символов;
- идентификатор должен начинаться с буквы;
- идентификатор не может содержать пробелы.

Большинство компонентов языка не чувствительны к регистру.

Язык, в терминах которого дается описание языка SQL, называется *метаязыком*. Синтаксические определения обычно задают с помощью специальной металингвистической символики, называемой Бэкуса-Наура формулами (БНФ). Для записи зарезервированных слов используются прописные буквы. Строчные буквы употребляются для записи слов, определяемых пользователем. Применяемые в нотации БНФ символы и их обозначения показаны в табл. 6.1.

Ранее мы уже определили данные как совокупную информацию, хранимую в БД в виде одного из нескольких различных типов. С помощью типов данных устанавливаются основные правила для данных, содержащихся в конкретном столбце таблицы, в том числе размер выделяемой для них памяти.

В языке SQL имеется шесть скалярных типов данных, определенных стандартом (табл. 6.2).

Таблица 6.1. Металингвистическая символика БНФ

::=	Равно по определению
|	Необходимость выбора одного из нескольких приведенных значений
<...>	Описанная с помощью метаязыка структура языка
{...}	Обязательный выбор некоторой конструкции из списка
[...]	Необязательный выбор некоторой конструкции из списка
[,...n]	Необязательная возможность повторения конструкции от нуля до нескольких раз

Таблица 6.2. Скалярные типы данных

Тип данных	Объявление
Символьный	`CHAR|VARCHAR`
Битовый	`BIT|BIT VARYING`
Точные числа	`NUMERIC|DECIMAL|INTEGER|SMALLINT`
Округленные числа	`FLOAT|REAL|DOUBLE PRECISION`
Дата/время	`DATE|TIME|TIMESTAMP`
Интервал	`INTERVAL`

Символьные данные состоят из последовательности символов, входящих в определенный создателями СУБД набор символов. Поскольку наборы символов являются специфическими для различных диалектов языка SQL, перечень символов, которые могут входить в состав значений данных символьного типа, также зависит от конкретной реализации.

При определении столбца с символьным типом данных параметр «длина» применяется для указания максимального количества символов, которые могут быть помещены в данный столбец. Символьная строка может быть определена как имеющая фиксированную (CHAR) или переменную (VARCHAR) длину. Если строка определена с фиксированной длиной значений, то при вводе в нее меньшего числа символов значение дополняется до указанной длины пробелами, добавляемыми справа. Если строка определена с переменной длиной значений, то при вводе в нее меньшего числа символов в базе данных будут сохранены только введенные символы, что позволит достичь определенной экономии внешней памяти.

Битовый тип данных используется для определения битовых строк, т. е. последовательности двоичных цифр (битов), каждая из которых может иметь значение либо 0, либо 1.

Тип точных числовых данных применяется для определения чисел, которые имеют точное представление, т. е. числа состоят из цифр, необязательной десятичной точки и необязательного символа знака. Данные точного числового типа определяются точностью и длиной дробной части. Точность задает общее количество значащих десятичных цифр числа, в которое входит длина как целой части, так и дробной, но без учета самой десятичной точки. Масштаб указывает количество дробных десятичных разрядов числа.

Типы NUMERIC и DECIMAL предназначены для хранения чисел в десятичном формате. По умолчанию длина дробной части равна нулю, а принимаемая по умолчанию точность зависит от реализации. Тип INTEGER (INT) используется для хранения больших положительных или отрицательных целых чисел. Тип SMALLINT — для хранения небольших положительных или отрицательных целых чисел; в этом случае расход внешней памяти существенно сокращается.

Тип округленных чисел применяется для описания данных, которые нельзя точно представить в компьютере, в частности действительных чисел. Округленные числа или числа с плавающей точкой представляются в научной нотации, при которой число записывается с помощью мантиссы, умноженной на определенную степень десяти (порядок).

Тип данных «дата/время» используется для определения моментов времени с некоторой установленной точностью. Стандарт SQL поддерживает следующий формат.

Тип данных DATE используется для хранения календарных дат, включающих поля YEAR (год), MONTH (месяц) и DAY (день). Тип данных TIME — для хранения отметок времени, включающих поля HOUR (часы), MINUTE (минуты) и SECOND (секунды). Тип данных TIMESTAMP — для совместного хранения даты и времени.

Данные типа INTERVAL используются для представления периодов времени.

6.2. РАБОТА С ТАБЛИЦАМИ. ОГРАНИЧЕНИЯ ЦЕЛОСТНОСТИ

6.2.1. Работа с доменами

Домен — объект реляционной базы данных, позволяющий описывать характеристики столбцов таблиц.

При создании или изменении любой таблицы при описании столбцов можно ссылаться на существующий домен для копирования всех его характеристик в столбец таблицы. Иногда в нескольких таблицах баз данных присутствуют столбцы, обладающие одними и теми же характеристиками. В таком случае можно предварительно описать тип такого столбца и его поведение с помощью домена, а затем поставить в соответствие каждому из одинаковых столбцов имя домена. При копировании в столбец отдельные характеристики, описанные в домене, могут быть изменены и дополнены. Домены

также можно использовать при описании локальных переменных и параметров в хранимых процедурах и триггерах.

Основной характеристикой домена является тип данных. Типы данных задаются при создании или изменении доменов, при создании или изменении характеристик столбцов таблиц, при описании внутренних переменных в хранимых процедурах и триггерах. Типы данных можно объединить в группы, или категории — числовые данные, строковые, данные даты и времени, а также единственный в группе двоичный тип данных BLOB.

Домен определяется оператором CREATE DOMAIN, имеющим следующий формат:

```
CREATE DOMAIN <имя домена> [AS] <тип данных>
[DEFAULT <значение по умолчанию>]
[NOT NULL] [CHECK (<условие ограничения>)]
[COLLATE <порядок сортировки>]
```

Предложение DEFAULT определяет выражение, которое по умолчанию заносится в колонку, ассоциированную с доменом, при создании записи таблицы. Это значение будет присутствовать в соответствующем столбце записи до тех пор, пока пользователь не изменит его каким-либо образом. Значения по умолчанию могут быть выражены как литерал-значение (числовое, строковое или дата).

Предложение CHECK определяет требования к значениям каждого столбца, ассоциированного с доменом. Столбцу не могут быть присвоены значения, не удовлетворяющие ограничениям, наложенным в предложении CHECK. Формат ограничения, накладываемого на значения полей, ассоциированных с доменом:

```
<условие ограничения домена>::= {
VALUE <оператор> <значение>
|VALUE [NOT] BETWEEN <значение 1> AND <значение 2>
|VALUE [NOT] LIKE <значение> [ESCAPE <значение>]
|VALUE [NOT] IN (<значение1> [, <значение2>…]
|VALUE IS [NOT] NULL
|VALUE [NOT] CONTAINING <значение>
|VALUE [NOT] STARTING [WITH] <значение>
<ограничения домена>
<ограничения домена> OR <ограничения домена>
<ограничения домена> AND <ограничения домена>
}
```

где

```
<оператор> = { = | < | > | <= | >= | !< | !> | <> | != };
```

VALUE <оператор> <значение> — значение домена находится с параметром <значение> во взаимоотношениях, определяемых параметром <оператор>. Другими словами , VALUE означает все правильные значения, которые могут быть присвоены столбцу, ассоциированному с доменом;

BETWEEN <значение1> AND <значение2> — значение домена должно находиться в промежутке между <значение1> и <значение2>, включая их;

LIKE <значение1> [ESCAPE <значение2>] — задает шаблон подобия, при этом символ «%» указывает любое значение любой длины, а символ «_» — любой одиночный символ. ESCAPE используется, если в операторе LIKE символы «%» и «_» присутствуют в шаблоне как обычные символы. В этом случае выбирается некоторый символ <значение 2>, после которого служебные символы теряют свой специальный статус и входят в поисковую строку как обычные параметры. Например, CHECK (VALUE LIKE «%!%» ESCAPE «!») — любое количество символов «!» оканчивается «%»;

IN (<значение 1> [, <значение 2>...] — значение домена должно совпадать с одним из приведенных в списке параметров;

IS [NOT] NULL — столбцы обязательно должны содержать какое-либо значение, отличное от пустого;

CONTAINING <значение> — значение домена обязательно должно содержать вхождение параметра <значение>, неважно в каком месте;

STARTING [WITH] <значение> — значение домена должно начинаться параметром «значение».

Может быть задана комбинация условий, которым должно соответствовать значение домена. В этом случае отдельные условия соединяются оператором AND или OR.

Например, создадим домен ОТДЕЛЫ символьного типа с условием, что данные, записываемые в этот домен, будут начинаться с комбинации символов «отд» и содержать внутри себя строку «018»:

```
CREATE DOMAIN ОТДЕЛЫ AS VARCHAR(10)
CHECK (VALUE STARTING WITH «отд» AND VALUE CONTAINING
«018»)
```

Для изменения определения домена используется оператор:

```
ALTER DOMAIN <имя>
{[SET DEFAULT {литерал |NULL |USER }]
```

```
|[DROP DEFAULT]
|[ADD [CONSTRAINT] CHECK (<ограничения домена>)]
|[DROP CONSTRAINT]}
```

Оператор позволяет изменить параметры домена, определенного ранее оператором `CREATE DOMAIN`. Однако нельзя изменить тип данных и определение `NOT NULL`, если в столбцах уже есть значения, не соответствующие устанавливаемому типу данных, либо являющимися пустыми. Все сделанные изменения будут учтены для всех столбцов, определенных с использованием данного домена. В этом операторе:

- `[SET DEFAULT]` — устанавливает значения по умолчанию подобно тому, как это делается в операторе `CREATE DOMAIN`;
- `[DROP DEFAULT]` — отмена текущих значений по умолчанию, назначенных по умолчанию;
- `[ADD [CONSTRAINT] CHECK (<ограничения домена>)]` — добавление условий, которым должны соответствовать значения столбца, ассоциированного с доменом. При этом возможно определение условий, рассмотренных выше для предложения `CHECK` оператора `CREATE DOMAIN`;
- `[DROP CONSTRAINT]` — удаление условий, определенных для домена в предложении `CHECK` оператора `CREATE DOMAIN` или предыдущих операторов `ALTER DOMAIN`.

Примеры использования доменов будут рассмотрены после того, как изучим операторы создания таблиц.

6.2.2. Управление таблицами

После создания общей структуры базы данных можно приступить к созданию таблиц, которые представляют собой отношения, входящие в состав проекта базы данных. Напомним, что таблица — основной объект для хранения информации в реляционной базе данных. Она состоит из содержащих данные строк и столбцов, занимает в базе данных физическое пространство и может быть постоянной или временной. Поле, также называемое в реляционной базе данных столбцом, является частью таблицы, за которой закреплен определенный тип данных. Каждая таблица базы данных должна содержать хотя бы один столбец. Строка данных — это запись в таблице базы данных, она включает поля, содержащие данные из одной записи таблицы.

Перед созданием таблиц базы данных необходимо продумать определение всех столбцов таблицы и характеристик каждого столб-

ца, индексов, ограничений целостности по отношению к другим таблицам. Предварительно должны быть определены домены, если они используются в таблицах. База данных, в которую добавляется создаваемая таблица, должна быть открыта, т.е. с ней должно быть установлено активное соединение.

Создание таблицы БД осуществляется оператором `CREATE TABLE`. Базовый синтаксис оператора создания таблицы имеет следующий вид:

```
CREATE TABLE <имя таблицы>
({<определение столбца>|<определение ограничения
таблицы>}
[,…,{<определение столбца>|<определение ограничения
таблицы>}]}
```

После задания имени таблицы через запятую в круглых скобках должны быть перечислены все предложения, определяющие отдельные элементы таблицы — столбцы или ограничения целостности:

- `<имя таблицы>` — идентификатор создаваемой таблицы;
- `<определение столбца>` — задание имени, типа данных и параметра отдельного столбца таблицы. Названия столбцов должны соответствовать правилам для идентификаторов и быть уникальными в пределах таблицы;
- `<определение ограничения таблицы>` — задание некоторого ограничения целостности на уровне таблицы.

С помощью предложения <определение столбца> задаются свойства столбца:

```
<определение столбца> ::=
<имя столбца> <тип данных>
[<ограничение столбца>][,…,<ограничение столбца>]
```

Рассмотрим назначение и использование параметров:

- `<имя столбца>` — идентификатор, задающий имя столбца таблицы;
- `<тип данных>` — тип данных столбца;
- `<ограничение столбца>` — с помощью этого предложения указываются ограничения, которые будут определены для столбца. Синтаксис предложения следующий:

```
<ограничение столбца>::=
[CONSTRAINT<имя ссылочной целостности>]
{[DEFAULT <выражение>]
|[NULL|NOT NULL]
```

```
|[PRIMARY KEY|UNIQUE]
|[FOREIGN KEY REFERENCES <имя главной таблицы>
[(<имя столбца> [,…,n])]
[ON DELETE {NO ACTION|CASCADE|SET DEFAULT|SET NULL}]
[ON UPDATE {NO ACTION|CASCADE|SET DEFAULT|SET NULL}]
]
|[(CHECK<условие столбца>)]
}
```

Рассмотрим назначение параметров.

CONSTRAINT — необязательное ключевое слово, после которого указывается название ограничения на значения столбца (<имя ссылочной целостности>). Имена должны быть уникальны в пределах базы данных. Имя ссылочной целостности не является обязательным. Оно присутствует в системных сообщениях относительно нарушения целостности и может использоваться при изменении структуры таблиц. В случае если это имя отсутствует, будет установлено системное имя. Для удаления непоименованной целостности придется использовать ее системное имя.

DEFAULT — задает значение по умолчанию для столбца. Это значение будет использовано при вставке строки, если для столбца не указано никакое значение.

NULL|NOT NULL — ключевые слова, разрешающие или запрещающие хранение в столбце значений NULL. Ключевое слово NULL используется для указания того, что в данном столбце могут содержаться значения NULL. Значение NULL отличается от пробела или нуля — к нему прибегают, когда необходимо указать, что данные недоступны, опущены или недопустимы. Если указано ключевое слово NOT NULL, то будут отклонены любые попытки поместить значение NULL в данный столбец. Если указан параметр NULL, помещение значений NULL в столбец разрешено. По умолчанию стандарт SQL предполагает наличие ключевого слова NULL.

PRIMARY KEY — определение первичного ключа. Если по столбцу строится первичный ключ, столбцу может быть приписан атрибут PRIMARY KEY. Если в первичный ключ входит единственный столбец, спецификатор ставится при определении столбца. Если в состав первичного ключа должны входить несколько столбцов, спецификатор ставится после определения всех столбцов. В любом случае поля, по которым строится первичный ключ, не могут быть пустыми, поэтому указывается спецификатор NOT NULL.

UNIQUE — атрибут, означающий, что в столбце не может быть два одинаковых значения. Уникальный ключ строится по столбцу (столбцам), когда столбец не входит в состав первичного ключа,

но тем не менее его значение должно всегда быть уникальным. Столбец, объявленный с этим атрибутом, как и первичный ключ, может применяться для обеспечения ссылочной целостности между родительской и дочерней таблицей. Для соединения с родительской таблицей в дочерней таблице строится внешний ключ. В таблице может быть создано несколько ограничений целостности UNIQUE.

FOREIGN KEY — внешний ключ, создается для обеспечения ссылочной целостности в подчиненной таблице.

Формат определения внешнего ключа:

```
FOREIGN KEY (<список столбцов подчиненной таблицы>)
REFERENCES <имя главной таблицы>
[<список столбцов главной таблицы>]
[ON DELETE {NO ACTION|CASCADE|SET DEFAULT|SET NULL}]
[ON UPDATE {NO ACTION|CASCADE|SET DEFAULT|SET NULL}]
```

В списке столбцов подчиненной таблицы содержатся те поля, которые входят в состав внешнего ключа. В списке столбцов главной таблицы содержатся те поля, которые являются ключевыми для связи таблиц (список можно опускать, если связь с родительской таблицей осуществляется по первичному ключу).

Необязательные параметры ON DELETE, ON UPDATE указывают, что должен делать сервер при удалении и изменении первичного ключа родительской таблицы соответственно:

- NO ACTION — удаление записей или изменение ключевого поля главной таблицы запрещено, если имеются соответствующие записи в подчиненной таблице;
- CASCADE — при удалении записи в главной таблице происходит удаление всех подчиненных записей в дочерней таблице. При изменении значений в поле первичного ключа главной таблицы изменяются значения ключевого поля во всех подчиненных ей записях дочерней таблицы;
- SET DEFAULT — при удалении записи или изменении ключевого поля в главной таблице в соответствующих записях дочерней таблицы полю внешнего ключа присваивается значение по умолчанию, указанное при определении поля; если это значение отсутствует в первичном ключе, возбуждается исключение, причем используется значение по умолчанию, имевшее место на момент определения ссылочной целостности, если впоследствии это значение будет изменено, ссылочная целостность при SET DEFAULT все равно будет использовать прежнее значение;
- SET NULL — в ключевое поле подчиненных записей дочерней таблицы заносится пустое значение NULL.

Ограничения, накладываемые на столбцы таблицы, определяются с помощью предложения CHECK (<условие столбца>), общий формат которого приводится ниже:

```
<условие столбца>::= {
VALUE <оператор> <значение>
|VALUE [NOT] BETWEEN <значение 1> AND <значение 2>
|VALUE [NOT] LIKE <значение> [ESCAPE <значение>]
|VALUE [NOT] IN (<значение1> [, <значение2>…]
|VALUE IS [NOT] NULL
|VALUE [NOT] CONTAINING <значение>
|VALUE [NOT] STARTING [WITH] <значение>
}
```

В данном условии ограничения на значение столбца описываются в том же формате, что и при определении домена:

```
<оператор> = { = | < | > | <= | >= | !< | !> | <> | != };
```

VALUE <оператор> <значение> — значение домена находится с параметром <значение> во взаимоотношениях, определяемых параметром <оператор>;

BETWEEN <значение1> AND <значение2> — значение домена должно находиться в промежутке между <значение1> и <значение2> включительно;

LIKE <значение1> [ESCAPE <значение2>] — задает шаблон подобия, при этом символ «%» указывает любое значение любой длины, а символ «_» — любой одиночный символ. ESCAPE используется, если в операторе LIKE символы «%» и «_» присутствуют в шаблоне как обычные символы. В этом случае выбирается некоторый символ <значение 2>, после которого служебные символы теряют свой специальный статус и входят в поисковую строку как обычные параметры. Например, CHECK (VALUE LIKE «%!%» ESCAPE «!») — любое количество символов «!» оканчивается «%»;

IN (<значение 1> [, <значение 2>…] — значение домена должно совпадать с одним из приведенных в списке параметров;

IS [NOT] NULL — столбцы обязательно должны содержать какое-либо значение, отличное от пустого;

CONTAINING <значение> — значение домена обязательно должно содержать вхождение параметра <значение>, неважно в каком месте;

STARTING [WITH] <значение> — значение домена должно начинаться параметром «значение».

Может быть задана комбинация условий, которым должно соответствовать значение домена. В этом случае отдельные условия соединяются оператором AND или OR.

Описание существующих в базе данных таблиц (характеристик столбцов, их порядка, наличие различных ключей или ограничения CHECK) можно изменять после создания таблиц.

Проиллюстрируем вышесказанное примерами.

Определим таблицу ОТДЕЛЫ с полями Номер и Наименование:

```
CREATE TABLE ОТДЕЛЫ
(Номер INTEGER NOT NULL,
Наименование VARCHAR(20))
```

Определим таблицу ОТДЕЛЫ с полями Номер, Наименование и первичным ключом по полю Номер:

```
CREATE TABLE ОТДЕЛЫ
(Номер INTEGER NOT NULL PRIMARY KEY,
Наименование VARCHAR(20))
```

Та же таблица, но с первичным ключом уже по двум полям (определение ограничений на уровне таблицы):

```
CREATE TABLE ОТДЕЛЫ
(Номер INTEGER NOT NULL,
Наименование VARCHAR(20) NOT NULL,
PRIMARY KEY (Номер, Наименование)
```

Теперь приведем пример использования в таблице домена. Создадим домен Домен_Номер, имеющий целочисленный тип с ограничением — больше или равно 100:

```
CREATE DOMAIN Домен_Номер AS INTEGER CHECK (VA-
LUE>=100);
```

В определении таблицы СОТРУДНИКИ поле Номер ассоциировано с доменом Домен_Номер:

```
CREATE TABLE СОТРУДНИКИ
(Номер Домен_Номер NOT NULL,
ФИО VARCHAR(20),
PRIMARY KEY (Номер))
```

Создадим две таблицы. Главная таблица ТОВАРЫ с полями Товар и цена, а также с первичным ключом по полю Товар:

```
CREATE TABLE ТОВАРЫ
```

```
(Товар VARCHAR (20) NOT NULL,
Цена INTEGER NOT NULL,
PRIMARY KEY (Товар))
```

Подчиненная таблица ПРОДАЖИ имеет первичный ключ по полю Номер и внешний по полю Товар для обеспечения ссылочной целостности с таблицей ТОВАРЫ. Поскольку не указывается поле связи в главной таблице, то для связи используется первичный ключ таблицы ТОВАРЫ:

```
CREATE TABLE ПРОДАЖИ
(Номер INTEGER NOT NULL PRIMARY KEY,
Дата DATE,
Товар VARCHAR (20) NOT NULL,
FOREIGN KEY (Товар) REFERENCES ТОВАРЫ)
```

Определение общих полей главной и подчиненной таблиц должно в точности совпадать. Если будут различия в порядке сортировки символов, то столбцы связи не будут фактически идентичными, что приведет к нарушению ссылочной целостности.

Еще один пример построения связи между таблицами ТОВАРЫ и ПРОДАЖИ.

В подчиненной таблице указываются действия сервера при изменении значения в составе первичного ключа и при удалении записи в главной таблице:

```
CREATE TABLE ПРОДАЖИ
(Номер INTEGER NOT NULL PRIMARY KEY,
Дата DATE,
Товар VARCHAR (20) NOT NULL,
CONSTRAINT FK1 FOREIGN KEY (Товар) REFERENCES ТОВАРЫ
ON UPDATE NO ACTION
ON DELETE NO ACTION)
```

Теперь для таблицы ТОВАРЫ будет установлена блокировка удаления или изменения значения в столбце Товар, если в таблице ПРОДАЖИ имеются записи о продаже этого товара. При этом ссылочной целостности в данном примере было присвоено имя FK1.

Изменение структуры таблиц, уже заполненных данными, является одним из наиболее опасных действий, которое часто приводит к исключениям базы данных или к потере существующих в таблице данных.

Для изменения структуры существующих таблиц используется оператор ALTER TABLE, упрощенный синтаксис которой представлен ниже:

```
ALTER TABLE <имя таблицы>
{[ADD <имя столбца> <тип данных> [
NULL|NOT NULL ]]
|[DROP [COLUMN] <имя столбца>]}
```

В одном операторе можно выполнить произвольное количество изменений в таблице.

Различные операции по изменению существующей таблицы отделяются друг от друга запятыми. Оператор `ALTER TABLE` позволяет:

- добавить определение нового столбца;
- удалить столбец из таблицы;
- удалить атрибуты целостности таблицы или отдельного столбца;
- добавить новые атрибуты целостности.

Изменение характеристик столбца, а также удаление столбца может закончиться неудачей, если:

- столбец приобретает атрибуты `PRIMARY KEY` или `UNIQUE`, но старые значения в столбце нарушают требования уникальности данных;
- удаленный столбец входил как часть в первичный или внешний ключ, что привело к нарушению ссылочной целостности между таблицами;
- столбцу были приписаны ограничения целостности на уровне таблицы;
- столбец используется в иных компонентах базы данных в просмотрах, триггерах, выражениях для вычисляемых столбцов.

Таким образом, в случае необходимости изменения атрибутов столбца или в случае удаления столбца сначала необходимо тщательно проанализировать, какие последствия для таблицы и базы данных в целом может повлечь такое изменение или удаление.

Добавление нового столбца:

```
ALTER TABLE <имя таблицы> ADD <определения столбца>
```

Добавление нового ограничения целостности:

```
ALTER TABLE <имя таблицы> ADD [CONSTRAINT <имя ограничения>] <определения целостности>
```

Удаление столбца:

```
ALTER TABLE <имя таблицы> DROP <имя столбца>[...,...]
```

Удаление ограничения целостности:

```
ALTER TABLE <имя таблицы> DROP <имя ограничения>
```

Удаление таблицы производится оператором:

```
DROP TABLE <имя таблицы>
```

Например, добавление столбца в таблицу можно осуществить следующей командой:

```
ALTER TABLE KLIENT
ADD ADRES VARCHAR(50)
```

Добавление ограничения целостности (первичного ключа) в таблицу можно выполнить командой:

```
ALTER TABLE KLIENT
ADD CONSTRAINT PK_ KLIENT
PRIMARY KEY (ID)
```

Пример удаления таблицы:

```
DROP TABLE KLIENT
```

Удаление может быть блокировано для главных таблиц, для которых в подчиненных таблицах (на данный момент не удаленных) имеются ссылки по внешнему ключу этих таблиц. Действительно, удаление главной таблицы разрушило бы ссылочную целостность. Поэтому следует либо удалить ограничения ссылочной целостности во всех подчиненных таблицах, либо, по необходимости, сначала удалить сами подчиненные таблицы, а затем уже удалять главную таблицу.

6.3. ВЫБОРКА ДАННЫХ. ОПЕРАТОР SELECT

Оператор `SELECT` относится к подразделу языка запросов DQL DML. Это один из самых сложных и самых мощных операторов SQL. Этот оператор имеет довольно сложную и развитую структуру. Его необходимо знать любому специалисту, так или иначе связанному с базами данных. Поэтому в этом учебном пособии ему уделяется достаточно много внимания.

На рис. 6.1 приведена модель базы данных, которая в дальнейшем изложении будет использоваться в качестве примера.

Таблица `Товары` связана с таблицей `Продажи` то полю `Товар` связью «один ко многим». Аналогично таблица `Клиенты` связана с таблицей `Продажи` по полю `Клиент` связью «один ко многим». Для того чтобы более наглядно проиллюстрировать результаты выполнения запросов, в качестве примера будем использовать заполненные таблицы (рис. 6.2).

Поле	Тип поля	Описание
Товар	Строка	Наименование товара
ЕИ	Строка	Единица измерения
Цена	Число	Цена

а

Поле	Тип поля	Описание
Клиент	Строка	Клиент
ИНН	Строка	ИНН клиента
Город	Строка	Город
Телефон	Строка	Телефон

б

Поле	Тип поля	Описание
Номер	Число	Номер документа
Дата	Дата	Дата документа
Товар	Строка	Наименование товара
Количество	Число	Количество товара
Клиент	Строка	Клиент

в

Рис. 6.1. Структура таблиц, используемых для примеров:
а — таблица ТОВАРЫ; *б* — таблица КЛИЕНТЫ; *в* — таблица ПРОДАЖИ

`SELECT` (англ., «выбрать») — оператор языка SQL, возвращающий набор данных (выборку) из базы данных, удовлетворяющих заданному условию. Он позволяет производить выборки данных из одной или нескольких таблиц базы данных и преобразовывать к нужному виду полученные результаты. Этот оператор способен выполнять действия, эквивалентные операторам реляционной алгебры. При его помощи можно реализовать сложные и громоздкие условия отбора данных из различных таблиц. Если выборка осуществляется из нескольких таблиц, то говорят об операции слияния.

При формировании запроса `SELECT` пользователь описывает желаемый набор данных (набор столбцов, критерии отбора записей, группировка значений, порядок вывода записей и т. п.). Сначала извлекаются все записи из таблицы, а затем для каждой записи набора проверяется ее соответствие заданному критерию. Если осуществляется слияние из нескольких таблиц, то сначала составляется произведение таблиц, а уже затем из полученного набора отбираются требуемые записи. В самом общем виде оператор `SELECT` имеет следующий синтаксис:

```
SELECT [ALL|DISTINCT ]{*|[имя_столбца [AS новое_
имя]]} [,...n]
FROM имя_таблицы [[AS] псевдоним] [,...n]
[WHERE <условия>]
[GROUP BY имя_столбца [,...n]]
[HAVING <критерии выбора групп>]
[ORDER BY имя_столбца [,...n]]
```

Оператор `SELECT` определяет поля (столбцы), которые будут входить в результат выполнения запроса. В списке они разделяются запятыми и приводятся в такой очередности, в какой должны быть представлены в результате запроса. Если используется имя поля,

Товар	ЕИ	Цена
Халва	кг	45
Сахар	кг	60
Мука пшеничная	кг	50
Конфеты "Мишка "	кг	150
Конфеты "Коровка"	кг	90
Мармелад "Кроха"	кг	85
Сосиски молочные	кг	260
Сардельки "Рощинские"	кг	300
Шпроты "Штурвал"	шт	56

а

Клиент	ИНН	Город	Телефон
ООО "Горизонт"	025500123152	Москва	4890605
ИП Привалов И.И.	025501025666	Санкт-Петербург	2560245
ООО "Ромашка"	025501244555	Москва	3658815
ООО "Перевал"	145889898622	Тверь	221588
ИП Иванов Ф.И.	025558000554	Москва	1155488
Таран О.С.	360224005454	Москва	1215648
Федорова Д.С.	500255510055	Санкт-Петербург	4449702
Лесовая В.Н.	212585832187	Москва	3021402
БМСТ	555878970025	Санкт-Петербург	4353822
Дремина Е.Е.	025578721058	Москва	6582209
ИП Газимова К.К.	025587978766	Москва	6521588

б

Номер	Клиент	Товар	Количество	Дата
45	Федорова Д.С.	Конфеты "Коровка"	50	02.04.2013
156	ИП Газимова К.К.	Конфеты "Мишка "	30	02.04.2013
162	Федорова Д.С.	Халва	10	15.02.2013
200	Лесовая В.Н.	Мармелад "Кроха"	20	07.02.2013
25	Таран О.С.	Халва	5	04.01.2013
85	Лесовая В.Н.	Мука пшеничная	50	02.04.2013
112	ООО "Ромашка"	Халва	40	15.02.2013
254	Лесовая В.Н.	Сахар	50	01.02.2013
140	ИП Газимова К.К.	Конфеты "Коровка"	15	09.01.2013
144	Таран О.С.	Сосиски молочные	30	01.04.2013

в

Рис. 6.2. Исходное состояние таблиц:
а — таблица ТОВАРЫ; *б* — таблица КЛИЕНТЫ; *в* — таблица ПРОДАЖИ

содержащее пробелы или разделители, его следует заключить в квадратные скобки. Символом * можно выбрать все поля, а вместо имени поля применить выражение из нескольких имен. Если обрабатывается несколько таблиц, то при наличии одноименных полей в разных таблицах в списке полей используется полная спецификация поля, т. е. Имя_Таблицы.Имя_Поля.

Обработка элементов оператора SELECT выполняется в следующей последовательности:

- FROM — список таблиц базы данных, из которых будет происходить выборка данных;
- WHERE — выполняется фильтрация строк объекта в соответствии с заданными условиями;
- GROUP BY — образуются группы строк, имеющих одно и то же значение в указанном столбце;
- HAVING — фильтруются группы строк объекта в соответствии с указанным условием;
- SELECT — устанавливается, какие столбцы должны присутствовать в выходных данных;
- ORDER BY — определяется упорядоченность результатов выполнения операторов.

Порядок предложений и фраз в операторе SELECT не может быть изменен. Только два предложения SELECT и FROM являются обязательными, все остальные могут быть опущены. SELECT — закрытая операция: результат запроса к таблице представляет собой другую таблицу. Существует множество вариантов записи данного оператора, что иллюстрируется приведенными ниже примерами.

Например, создать набор данных, состоящий из всех столбцов и всех записей таблицы Клиенты, можно с помощью такого оператора:

```
SELECT * FROM Клиенты
```

Такой же набор данных можно получить, если вместо звездочки перечислить все столбцы таблицы:

```
SELECT Клиент,ИНН,Город,Телефон
FROM Клиенты
```

Результат выполнения запроса может содержать дублирующиеся значения, поскольку оператор SELECT не исключает повторяющихся значений при выполнении выборки данных. Предикат DISTINCT следует применять в тех случаях, когда требуется отбросить блоки данных, содержащие дублирующие записи в выбранных полях. Значения для каждого из приведенных в инструкции SELECT полей

должны быть уникальными, чтобы содержащая их запись смогла войти в выходной набор. Причиной ограничения в применении DISTINCT является то обстоятельство, что его использование может резко замедлить выполнение запросов.

Откорректируем предыдущий запрос следующим образом:

```
SELECT DISTINCT КЛИЕНТЫ.Город
FROM КЛИЕНТЫ
```

В результате получим список городов — местонахождение клиентов.

Оператор WHERE используется для включения в набор данных лишь нужных записей. В этом случае оператор SELECT имеет следующий формат:

```
SELECT {* | <значение1> [, <значение2> ...]}
FROM <таблица1> [, < таблица2> ...]
WHERE <условие>
```

За ключевым словом WHERE следует перечень условий поиска. В набор данных, возвращаемый оператором SELECT, будут включаться только те записи, которые удовлетворяют условию поиска, указанному после WHERE. Далее будут рассмотрены весьма сложные условия выбора данных из различных таблиц, а пока разберем простейшие.

Основные типы условий поиска (или предикатов) приведены ниже.

1. Сравнение: сравниваются результаты вычисления одного выражения с результатами вычисления другого, либо значения разных столбцов.

2. Диапазон: проверяется, попадает ли значение столбца либо результат вычисления выражения в заданный диапазон значений.

3. Принадлежность множеству: проверяется, принадлежит ли значение столбца либо результат вычислений выражения заданному множеству значений.

4. Соответствие шаблону: проверяется, отвечает ли некоторое строковое значение заданному шаблону.

5. Значение NULL: проверяется, содержит ли данный столбец определитель NULL (неизвестное значение).

Например, список клиентов из Санкт-Петербурга из таблицы КЛИЕНТЫ можно получить с помощью запроса:

```
SELECT *
FROM КЛИЕНТЫ
WHERE КЛИЕНТЫ.Город = «Санкт-Петербург»
```

В языке SQL можно использовать следующие операторы сравнения:

=	равно
<	меньше
>	больше
<=	меньше или равно
>=	больше или равно
<>	не равно

Например, показать все операции продажи товаров, где количество проданного товара больше 10:

```
SELECT *
FROM ПРОДАЖИ
WHERE Количество>10
```

Более сложные предикаты могут быть построены с помощью логических операторов AND, OR или NOT, а также скобок, используемых для определения порядка вычисления выражения. Вычисление выражения в условиях выполняется по следующим правилам:

- выражение вычисляется слева направо;
- первыми вычисляются подвыражения в скобках;
- операторы NOT выполняются до выполнения операторов AND и OR;
- операторы AND выполняются до выполнения операторов OR.

Для устранения любой возможной неоднозначности рекомендуется использовать скобки. Приведем примеры использования более сложных условий поиска. В результате следующего запроса выводится список товаров, цена которых больше или равна 50 и меньше или равна 100:

```
SELECT Товар, Цена
FROM ТОВАРЫ
WHERE Цена>=50 AND Цена<=100
```

Этот же запрос можно выполнить с помощью оператора BETWEEN. Ключевое слово BETWEEN используется для поиска значения внутри некоторого интервала, определяемого своими минимальным и максимальным значениями. При этом указанные значения включаются в условие поиска:

```
SELECT Товар, Цена
FROM ТОВАРЫ
WHERE Цена BETWEEN 50 And 100
```

Полученная выборка будет аналогична выборке из предыдущего примера.

При использовании отрицания `NOT BETWEEN` будут получены значения вне границ заданного диапазона.

Оператор `IN` используется для сравнения некоторого значения со списком заданных значений, при этом проверяется, соответствует ли результат вычисления выражения одному из значений в предоставленном списке. При помощи оператора `IN` может быть получен тот же результат, что и в случае применения оператора `OR`, однако оператор `IN` выполняется быстрее. `NOT IN` используется для отбора любых значений, кроме тех, которые указаны в представленном списке.

Например, для того чтобы вывести список клиентов, не проживающих ни в Москве, ни в Санкт-Петербурге, необходимо выполнить запрос:

```
SELECT Клиент, Город
FROM КЛИЕНТЫ
WHERE Город NOT IN («Москва»,»Санкт-Петербург»)
```

С помощью оператора `LIKE` можно выполнять сравнение выражения с заданным шаблоном, в котором допускается использование символов-заменителей:

- символ «%» — вместо этого символа может быть подставлено любое количество произвольных символов;
- символ «_» заменяет один символ строки.

Например, в результате выполнения следующего запроса будет выведен список клиентов, у которых в номере телефона вторая цифра — 4:

```
SELECT *
FROM КЛИЕНТЫ
WHERE КЛИЕНТЫ.Телефон LIKE '_4%'
```

В результате будет найден клиент «Федорова Д. С.» с телефоном 4449602.

Для того чтобы выбрать клиентов, у которых в фамилии или названии встречается слог «ва», нужно составить запрос с использованием двух символов «%»:

```
SELECT *
FROM КЛИЕНТЫ
WHERE КЛИЕНТЫ.Клиент LIKE '%ва%',
```

Такая форма поиска применяется в тех случаях, когда пользователь не знает или не помнит всех данных о нужном объекте.

Оператор IS NULL используется для сравнения текущего значения со значением NULL — специальным значением, указывающим на отсутствие любого значения (важно подчеркнуть, что NULL — это не знак пробела или нуль). Выражение IS NOT NULL используется для проверки присутствия значения в каком-либо поле.

Например, необходимо найти клиентов, у которых не указан телефон (поле Телефон не содержит никакого значения):

```
SELECT Клиент
FROM КЛИЕНТЫ
WHERE Телефон IS NULL
```

Выборка клиентов, у которых есть телефон (поле Телефон содержит какое-либо значение):

```
SELECT Клиент
FROM КЛИЕНТЫ
WHERE Телефон IS NOT NULL
```

В общем случае строки в результирующем наборе данных SQL-запроса никак не упорядочены. Однако их можно отсортировать в определенном порядке. Для этого в оператор SELECT помещается ключевое слово ORDER BY, которая сортирует данные выходного набора в заданной последовательности. Сортировка может выполняться по нескольким полям, в этом случае они перечисляются за ключевым словом ORDER BY через запятую.

Способ сортировки (по возрастанию или по убыванию) задается ключевым словом, указываемым в рамках параметра ORDER BY следом за названием поля, по которому выполняется сортировка. По умолчанию реализуется сортировка по возрастанию. Явно она задается ключевым словом ASC.

Для выполнения сортировки в обратной последовательности необходимо после имени поля, по которому она выполняется, указать ключевое слово DESC. Фраза ORDER BY позволяет упорядочить выбранные записи в порядке возрастания или убывания значений любого столбца или комбинации столбцов, независимо от того, присутствуют эти столбцы в результирующем наборе или нет. Фраза ORDER BY всегда должна быть последним элементом в операторе SELECT. Например, необходимо вывести список товаров в алфавитном порядке:

```
SELECT *
FROM ТОВАРЫ
ORDER BY ТОВАРЫ.Товар
```

В предложении ORDER BY может быть указано и больше одного элемента. Тогда критерием для сортировки служит результат «склеивания» всех перечисляемых полей. Например, таблицу продажи можно упорядочить по полям Товар и Дата:

```
SELECT *
FROM ПРОДАЖИ
ORDER BY ПРОДАЖИ.Товар, ПРОДАЖИ.Дата
```

В следующем запросе сортировка осуществляется сначала по имени покупателя, затем по названию товара и, наконец, по дате:

```
SELECT Клиент, Товар, Дата, Количество
FROM Продажи
ORDER BY Клиент, Товар, Дата
```

Соединение — это процесс, когда две или более таблицы объединяются в одну. Способность объединять информацию из нескольких таблиц или запросов в виде одного логического набора данных обусловливает широкие возможности SQL, которые и будут рассмотрены далее.

Ранее мы уже определили оператор WHERE как средство для наложения ограничений на результат запроса. Но если в качестве условия ограничения установить связь между таблицами по определенным столбцам, то это и будет соединением двух таблиц. В условиях объединения могут участвовать поля, относящиеся к одному и тому же типу данных и содержащие один и тот же вид данных, но они не обязательно должны иметь одинаковые имена. Блоки данных из двух таблиц объединяются, как только в указанных полях будут найдены совпадающие значения.

Чтобы выбрать все записи о продажи товаров из таблицы ПРОДАЖИ и для каждого товара указать его цену из таблицы ТОВАРЫ, можно выполнить такой запрос:

```
SELECT ПРОДАЖИ.*, ТОВАРЫ.ЦЕНА
FROM ПРОДАЖИ, ТОВАРЫ
WHERE ПРОДАЖИ.ТОВАР = ТОВАРЫ.ТОВАР
```

При выполнении этого оператора для каждой записи из таблицы ПРОДАЖИ ищется запись в таблице ТОВАРЫ, у которой значение в поле Товар совпадает со значением в поле Товар текущей записи таблицы ПРОДАЖИ (рис. 6.3).

Номер	Клиент	Товар	Количество	Дата	Цена
140	ИП Газимова К.К.	Конфеты "Коровка"	15	09.01.2013	90
45	Федорова Д.С.	Конфеты "Коровка"	50	02.04.2013	90
156	ИП Газимова К.К.	Конфеты "Мишка "	30	02.04.2013	150
200	Лесовая В.Н.	Мармелад "Кроха"	20	07.02.2013	85
85	Лесовая В.Н.	Мука пшеничная	50	02.04.2013	50
254	Лесовая В.Н.	Сахар	50	01.02.2013	60
144	Таран О.С.	Сосиски молочные	30	01.04.2013	260
25	Таран О.С.	Халва	5	04.01.2013	45
162	Федорова Д.С.	Халва	10	15.02.2013	45
112	ООО "Ромашка"	Халва	40	15.02.2013	45

Рис. 6.3. Результат внутреннего соединения таблиц

При этом безразлично, в каком порядке перечислять таблицы в условии поиска, т. е. безразлично, какая из таблиц будет упомянута слева, а какая справа. Такой способ соединения таблиц называется *внутренним соединением*.

Приведем еще один пример внутреннего соединения таблиц. Выберем все записи о расходе товара из таблицы ПРОДАЖИ и для каждого клиента укажем его Город из таблицы КЛИЕНТЫ:

```
SELECT ПРОДАЖИ.*, КЛИЕНТЫ.Город
FROM ПРОДАЖИ, КЛИЕНТЫ
WHERE КЛИЕНТЫ.Клиент = ПРОДАЖИ.Клиент
```

Как можно видеть из примера (рис. 6.4), в результирующий набор данных не включены записи из таблицы КЛИЕНТЫ, для которых нет записей в таблице ПРОДАЖИ.

При внутреннем соединении двух таблиц последовательность формирования результирующего набора данных можно представить следующим образом. Из столбцов, которые указаны после слова SELECT, составляется промежуточный набор данных путем связывания каждой записи из первой таблицы с каждой записью из второй таблицы. Из получившегося набора данных отбрасываются

Номер	Клиент	Товар	Количество	Дата	Город
140	ИП Газимова К.К.	Конфеты "Коровка"	15	09.01.2013	Москва
156	ИП Газимова К.К.	Конфеты "Мишка "	30	02.04.2013	Москва
254	Лесовая В.Н.	Сахар	50	01.02.2013	Москва
200	Лесовая В.Н.	Мармелад "Кроха"	20	07.02.2013	Москва
85	Лесовая В.Н.	Мука пшеничная	50	02.04.2013	Москва
112	ООО "Ромашка"	Халва	40	15.02.2013	Москва
25	Таран О.С.	Халва1	5	04.01.2013	Москва
144	Таран О.С.	Сосиски молочные	30	01.04.2013	Москва
162	Федорова Д.С.	Халва	10	15.02.2013	Санкт-Петербург
45	Федорова Д.С.	Конфеты "Коровка"	50	02.04.2013	Санкт-Петербург

Рис. 6.4. Соединение таблиц ПРОДАЖИ и КЛИЕНТЫ

все записи, не удовлетворяющие условию поиска в предложении WHERE.

В рассмотренных выше примерах связывания таблиц в перечнях возвращаемых столбцов после слова SELECT и в условии поиска после WHERE перед именем столбца через точку пишется название таблицы, например:

```
WHERE КЛИЕНТЫ.Клиент = ПРОДАЖИ.Клиент
```

Перед именем столбца необходимо указывать имя таблицы в тех случаях, когда в разных таблицах есть одноименные столбцы (как в данном примере). Использование имен таблиц для идентификации столбцов неудобно, так как делает его громоздким и трудночитаемым. Намного лучше присвоить каждой таблице какое-нибудь краткое имя — *псевдоним*. Псевдоним отделяется от фактического имени таблицы пробелом в списке таблиц-источников после ключевого слова FROM. Например, запрос

```
SELECT ПРОДАЖИ.*, КЛИЕНТЫ.Город
FROM ПРОДАЖИ, КЛИЕНТЫ
WHERE КЛИЕНТЫ.Клиент = ПРОДАЖИ.Клиент
```

после введения в него псевдонимов таблиц выглядит намного компактнее:

```
SELECT П.*, К.Город
FROM ПРОДАЖИ П, КЛИЕНТЫ К
WHERE К.Клиент = П.Клиент
```

Для расчета значений вычисляемых столбцов результирующего набора данных используются арифметические выражения. При этом после оператора SELECT в списке столбцов вместо имени вычисляемого столбца указывается выражение. Например, следующий запрос возвращает все записи о продаже товаров из таблицы ПРОДАЖИ и для каждого факта реализации товара рассчитывает общую стоимость отпущенного товара:

```
SELECT П.*, Т.Цена, П.Количество * Т.Цена
FROM ПРОДАЖИ П, ТОВАРЫ Т
WHERE П.Товар = Т.Товар
```

Результат выполнения данного запроса можно увидеть на рис. 6.5.

Для создания вычисляемого поля применяются арифметические операции сложения, вычитания, умножения и деления, а также встроенные функции языка SQL. Можно указать имя любого

Номер	Клиент	Товар	Количество	Дата	MULTIPLY
140	ИП Газимова К.К.	Конфеты "Коровка"	15	09.01.2013	1350
45	Федорова Д.С.	Конфеты "Коровка"	50	02.04.2013	4500
156	ИП Газимова К.К.	Конфеты "Мишка "	30	02.04.2013	4500
200	Лесовая В.Н.	Мармелад "Кроха"	20	07.02.2013	1700
85	Лесовая В.Н.	Мука пшеничная	50	02.04.2013	2500
254	Лесовая В.Н.	Сахар	50	01.02.2013	3000
144	Таран О.С.	Сосиски молочные	30	01.04.2013	7800
162	Федорова Д.С.	Халва	10	15.02.2013	450
112	ООО "Ромашка"	Халва	40	15.02.2013	1800

Рис. 6.5. Создание вычисляемого поля

столбца таблицы, но использовать имя столбца только той таблицы или запроса, которые указаны в списке предложения FROM. При построении сложных выражений используются скобки. Можно явным образом задавать имена столбцов результирующей таблицы, для чего применяется фраза AS. В следующем запросе выводится список товаров с указанием года и месяца продажи:

```
SELECT T.Товар, Year(П.Дата) AS Год, Month(П.Дата)
AS Месяц
FROM ТОВАРЫ T, ПРОДАЖИ П
WHERE T.Товар=П.Товар
```

В запросе использованы встроенные функции Year и Month для выделения года и месяца из даты.

Агрегатные (*итоговые*) *функции* предназначены для вычисления итоговых значений операций над всеми записями набора данных. К агрегатным относятся следующие функции:

- COUNT (<выражение>) — определяет количество записей в выходном наборе SQL-запроса (число вхождений значения выражения во все записи результирующего набора данных);
- SUM(<выражение>) — суммирует значения выражения;
- AVG(<выражение>) — эта функция позволяет рассчитать среднее значение множества значений, хранящихся в определенном поле отобранных запросом записей. Оно является арифметическим средним значением, т.е. суммой значений, деленной на их количество;
- MAX(<выражение>) — определяет максимальное значение;
- MIN(<выражение>) — определяет минимальное значение.

Все эти функции оперируют со значениями в единственном столбце таблицы (столбцами нескольких таблиц) или с арифметическим выражением и возвращают единственное значение. При вычислении результатов любых функций сначала исключаются все пустые значения, после чего требуемая операция применяется

только к оставшимся конкретным значениям столбца. Вариант COUNT(*) — особый случай использования функции COUNT, его назначение состоит в подсчете всех строк в результирующей таблице, независимо от того, содержатся там пустые, дублирующиеся или любые другие значения. В результате выполнения следующего запроса будет подсчитано количество строк в таблице ПРОДАЖИ:

```
SELECT Count(*) AS Количество Продаж
FROM ПРОДАЖИ
```

Если из группы одинаковых записей нужно учитывать только одну, то перед выражением в скобках включают ключевое слово DISTINCT:

```
COUNT(DISTINCT Клиент)
```

DISTINCT не имеет смысла для функций MIN и MAX, однако его использование может повлиять на результаты выполнения функций SUM и AVG, поэтому необходимо заранее обдумать, должно ли оно присутствовать в каждом конкретном случае. Кроме того, ключевое слово DISTINCT может быть указано в любом запросе не более одного раза.

Например, в следующем запросе вычисляется количество покупателей, приобретавших товары на складе:

```
SELECT COUNT(DISTINCT ПРОДАЖИ.КЛИЕНТ) AS Кол Клиентов
FROM ПРОДАЖИ
```

В другом примере вычисляется общая стоимость отпущенных товаров за один день:

```
SELECT SUM(П.Количество*Т.Цена) AS ОбщСтоимость
FROM ПРОДАЖИ П,ТОВАРЫ Т
WHERE (П.Товар=Т.Товар) AND (П.Дата='02.09.2017')
```

Функции COUNT, MIN, MAX применимы как к числовым, так и нечисловым полям. Функции SUM и AVG могут использоваться только в случае числовых полей.

Иногда в запросах требуется формировать промежуточные итоги, т.е. получить результат выполнения функции не для всего набора данных, а для определенных групп записей (например, подсчитать итог для каждого клиента, для каждого товара, за определенную дату). Для этой цели в операторе SELECT используется предложение GROUP BY. Запрос, в котором присутствует GROUP BY, называется

группирующим запросом, поскольку в нем группируются данные, полученные в результате выполнения операции SELECT. Для каждой отдельной группы создается единственная итоговая строка. После оператора SELECT перечисляются те столбцы, которые не меняют значения в группе, т. е. участвуют в группировке. Например, группируя запрос по товарам в таблице ПРОДАЖИ, нельзя в предложении SELECT указать еще и дату, так как каждому товару могут соответствовать разные даты продажи и, следовательно, подобный запрос будет неверен:

```
SELECT П.Товар, П.Дата, SUM(П.Количество)
FROM ПРОДАЖИ П
GROUP BY П.Товар
```

Для того чтобы получить общее количество проданного товара (рис. 6.6), следует использовать запрос:

```
SELECT П.Товар, SUM(П.Количество)
FROM ПРОДАЖИ П
GROUP BY П.Товар
```

А чтобы получить количество проданного товара за каждую дату, нужно откорректировать предыдущий запрос следующим образом:

```
SELECT П.Товар, П.Дата, SUM(П.Количество)
FROM ПРОДАЖИ П
GROUP BY П.Товар, П.Дата
```

Другими словами, все имена полей, приведенные в списке предложения SELECT, должны присутствовать и во фразе GROUP BY, за исключением случаев, когда имя столбца используется в агрегатной функции. В то же время в предложении GROUP BY могут быть имена столбцов, отсутствующие в списке предложения SELECT.

Еще один пример использования группировки в запросах — подсчет количества покупок, осуществленных каждым клиентом:

	Товар	SUM
▶	Конфеты "Коровка"	65
	Конфеты "Мишка "	30
	Мармелад "Кроха"	20
	Мука пшеничная	50
	Сахар	50
	Сосиски молочные	30
	Халва	55

Рис. 6.6. Результат выполнения группирующего запроса

```
SELECT Клиент, COUNT(*)
FROM ПРОДАЖИ
GROUP BY Клиент
```

Товар	SUM
Конфеты "Коровка"	5850
Конфеты "Мишка "	4500
Мармелад "Кроха"	1700
Мука пшеничная	2500
Сахар	3000
Сосиски молочные	7800
Халва	2475

Рис. 6.7. Результат выполнения группирующего запроса с условием

Если совместно с GROUP BY используется предложение WHERE, то оно обрабатывается первым, а группированию подвергаются только те строки, которые удовлетворяют условию поиска.

Например, для того чтобы определить, на какую сумму был продан товар каждого наименования, можно составить запрос (рис. 6.7):

```
SELECT П.Товар, SUM(П.Количество*Т.Цена)
FROM ПРОДАЖИ П, ТОВАРЫ Т
WHERE Т.Товар = П.Товар
GROUP BY П.Товар
```

Чтобы определить количество покупателей на каждую дату, нужно выполнить такой запрос:

```
SELECT Дата, COUNT(DISTINCT Клиент)
FROM ПРОДАЖИ
GROUP BY Дата
```

Если в результирующем наборе данных нужно наложить ограничения на значения итоговых строк, то после предложения GROUP BY используют предложение HAVING. Это дополнительная возможность «отфильтровать» выходной набор. Важно отметить, что итоговые функции могут использоваться только в списке предложения SELECT и в составе предложения HAVING, в то время как в предложении WHERE такие функции указывать нельзя. Например, следующий запрос позволяет вывести список товаров, проданных на сумму более 3 000 р.:

```
SELECT П.Товар, SUM(П.Количество*Т.Цена)
FROM ПРОДАЖИ П, ТОВАРЫ Т
WHERE Т.Товар = П.Товар
GROUP BY П.Товар
HAVING SUM(П.Количество*Т.Цена)>3000
```

Показать даты продажи товаров, в которых суммарное количество отпускаемого товара было не менее 10 единиц, причем в каждую дату было совершено более одной операции продажи:

```
SELECT Дата,COUNT(*)
FROM ПРОДАЖИ
WHERE Количество>=10
GROUP BY Дата
HAVING COUNT(*)>1
```

Иногда в запросах при использовании операций сравнения значение, с которым надо сравнивать результат запроса, заранее не определено и представляет собой не одно, а несколько значений, либо должно быть получено путем выполнения другого оператора SELECT. В таких случаях используются подзапросы (вложенные запросы).

Подзапрос (вложенный запрос) — это инструмент создания временной таблицы, содержимое которой извлекается и обрабатывается внешним оператором. Текст подзапроса должен быть заключен в скобки. Оператор SELECT с подзапросом имеет следующий вид:

```
SELECT ...
FROM ...
WHERE <сравниваемое значение> <оператор> (SELECT ...)
```

Существует два типа подзапросов.

1. *Скалярный подзапрос* возвращает единственное значение. В принципе, он может использоваться везде, где требуется указать единственное значение.

2. *Табличный подзапрос* возвращает множество значений, т.е. значения одного или нескольких столбцов таблицы, размещенные в более чем одной строке. Он возможен везде, где допускается наличие таблицы.

Подзапросы бывают:

- некоррелированные — не содержат ссылки на запрос верхнего уровня, вычисляются один раз для запроса верхнего уровня;
- коррелированные — содержат условия, зависящие от значений полей в основном запросе; вычисляются для каждой строки запроса верхнего уровня.

Вложенный запрос имеет тот же синтаксис, что и основной или внешний, следовательно, вложенный запрос также может содержать подзапрос. Один оператор SELECT как бы внедряется в тело другого оператора SELECT.

Внешний оператор SELECT использует результат выполнения внутреннего оператора для определения содержания окончательного результата всей операции.

Внутренние запросы могут быть помещены непосредственно после оператора сравнения (=, <, >,<=, >=, <>) в предло-

жения WHERE и HAVING внешнего оператора SELECT. Кроме того, внутренние операторы SELECT могут применяться в операторах INSERT, UPDATE и DELETE (вставка, обновление и удаление записей).

К подзапросам применяются следующие правила и ограничения:

- нельзя использовать ORDER BY, хотя эта фраза может присутствовать во внешнем запросе;
- список в предложении SELECT состоит из имен отдельных столбцов или составленных из них выражений, за исключением случая, когда в подзапросе присутствует ключевое слово EXISTS;
- по умолчанию имена столбцов в подзапросе относятся к таблице, имя которой указано в предложении FROM. Однако допускается ссылка и на столбцы таблицы, указанной во фразе FROM внешнего запроса, для чего применяются специфицированные имена столбцов (т.е. с указанием таблицы) либо их псевдонимы;
- если подзапрос является одним из двух операндов, участвующих в операции сравнения, то запрос должен указываться в правой части этой операции.

Следующий запрос возвращает даты, когда было продано максимальное количество товара:

```
SELECT Количество, Дата
FROM ПРОДАЖИ
WHERE Количество = (SELECT MAX(Количество) FROM
ПРОДАЖИ)
```

Очевидно, что вложенный оператор SELECT должен возвращать единичное значение, а не список. Во вложенном подзапросе определяется максимальное количество товара. Во внешнем подзапросе — дата, для которой количество товара оказалось равным максимальному. Необходимо отметить, что нельзя прямо использовать предложение

```
WHERE Количество = Max(Количество),
```

поскольку применять обобщающие функции в предложениях WHERE запрещено.

Усложним предыдущий пример. Определим даты, когда со склада было отгружено максимальное количество товара, и для каждого клиента, который этот товар приобрел, укажем город (рис. 6.8):

Количество	Дата	Клиент	Город
50	01.02.2013	Лесовая В.Н.	Москва
50	02.04.2013	Лесовая В.Н.	Москва
50	02.04.2013	Федорова Д.С.	Санкт-Петербург

Рис. 6.8. Результат выполнения вложенного запроса

```
SELECT П.Количество, П.Дата, К.Клиент, К.Город
FROM ПРОДАЖИ П, КЛИЕНТЫ К
WHERE (П.Клиент = К.Клиент)
AND
П.Количество =(SELECT MAX(Количество) FROM ПРОДАЖИ)
```

-

ПРОДАЖИ КЛИЕНТЫ.

'

(. 6.9):

```
SELECT Дата, Количество,
Количество-(SELECT AVG(Количество) FROM ПРОДАЖИ)
AS Отклонение
FROM ПРОДАЖИ
WHERE Количество>
(SELECT AVG(Количество)FROM ПРОДАЖИ)
```

'

-

, SELECT

'

.

, WHERE

HAVING, '

.

-

' '

:

Дата	Количество	Отклонение
02.04.2013	50	20
02.04.2013	50	20
15.02.2013	40	10
01.02.2013	50	20

Рис. 6.9. Результат выполнения вложения запроса в вычисляемом столбце

```
[NOT] IN;
{ALL | SOME | ANY};
[NOT] EXISTS;
```

IN

, ,

.

,

-

:

```
SELECT П.*
FROM ПРОДАЖИ П
WHERE П.Клиент IN
(SELECT П1.Клиент FROM ПРОДАЖИ П1
WHERE П1.Количество =
(SELECT MAX(П2.Количество) FROM ПРОДАЖИ П2))
```

. -

Количество:

```
SELECT MAX(П2.Количество) FROM ПРОДАЖИ П2
```

, -

:

```
SELECT П1.Клиент
FROM ПРОДАЖИ П1
WHERE П1.Количество =
(SELECT MAX(П2.Количество) FROM ПРОДАЖИ П2)
```

ПРОДАЖИ IN, . .

ПРОДАЖИ, КЛИЕНТ

, (. 6.10).

Номер	Клиент	Товар	Количество	Дата
45	Федорова Д.С.	Конфеты "Коровка"	50	02.04.2013
162	Федорова Д.С.	Халва	10	15.02.2013
200	Лесовая В.Н.	Мармелад "Кроха"	20	07.02.2013
85	Лесовая В.Н.	Мука пшеничная	50	02.04.2013
254	Лесовая В.Н.	Сахар	50	01.02.2013

Рис. 6.10. Список покупок клиентов, приобретавших максимальное количество товара

С помощью NOT IN можно получить выборку строк, в которых сравниваемое значение не входит в множество, полученное во вложенных запросах.

Ключевые слова ANY и ALL могут использоваться с подзапросами, возвращающими один столбец чисел.

Отношение сравниваемого значения и значений, возвращаемых подзапросом, устанавливается операторами ALL и SOME (ANY):

- ALL указывает, что условие будет истинно только тогда, когда сравниваемое значение находится в нужном отношении со всеми значениями, возвращаемыми подзапросом, иначе говоря, выполняется для всех значений в результирующем столбце подзапроса;
- SOME (или ANY) — условие поиска истинно, когда сравниваемое значение находится в нужном отношении хотя бы с одним значением, возвращаемым подзапросом, т. е. выполняется хотя бы для одного из значений в результирующем столбце подзапроса.

Если в результате выполнения подзапроса получено пустое значение, то для ключевого слова ALL условие сравнения будет считаться выполненным, а для ключевого слова ANY — невыполненным. Ключевое слово SOME является синонимом слова ANY.

Проиллюстрируем вышесказанное примерами. Определим все факты продажи товаров, в которых количество единиц продаваемого товара превышает среднее значение:

```
SELECT * FROM ПРОДАЖИ П1
WHERE П1.Количество> ALL
(SELECT AVG(П2.Количество)
FROM ПРОДАЖИ П2
GROUP BY П2.Клиент)
```

И еще один пример использования предложения HAVING и ключевого слова ALL. Нужно вывести сведения о клиенте, который приобрел наибольшее количество товаров:

```
SELECT К.*
FROM КЛИЕНТЫ К
WHERE К.Клиент =
(SELECT П.Клиент
FROM ПРОДАЖИ П
GROUP BY П.Клиент
HAVING SUM(П.Количество) >= ALL
(SELECT SUM (П1.Количество) FROM ПРОДАЖИ П1
GROUP BY П1.Клиент))
```

Перечислим все факты продажи товаров, в которых количество единиц продаваемого товара превышает среднее значение продажи хотя бы одного товара:

```
SELECT * FROM ПРОДАЖИ П1
WHERE П1.Количество> SOME
(SELECT AVG(П2.Количество)
FROM ПРОДАЖИ П2
GROUP BY П2.Клиент)
```

Ключевые слова EXISTS и NOT EXISTS предназначены для использования только совместно с подзапросами. Результат их обработки представляет собой логическое значение TRUE или FALSE.

Для ключевого слова EXISTS результат равен TRUE в том и только в том случае, если в возвращаемой подзапросом результирующей таблице присутствует хотя бы одна строка. Если результирующая таблица подзапроса пуста, результатом обработки операции EXISTS будет значение FALSE. Для ключевого слова NOT EXISTS используются правила обработки, обратные по отношению к ключевому слову EXISTS. Поскольку по ключевым словам EXISTS и NOT EXISTS проверяется лишь наличие строк в результирующей таблице подзапроса, то эта таблица может содержать произвольное количество столбцов. Используя ключевое слово EXISTS, можно составить список всех клиентов, которые хотя бы один раз приобретали товар:

```
SELECT *
FROM КЛИЕНТЫ К
WHERE EXISTS
(SELECT П.*
FROM ПРОДАЖИ П
WHERE К.Клиент = П.Клиент)
```

Для того чтобы получить список клиентов, которые не сделали ни одной покупки в этот же запрос, нужно вставить слово NOT:

```
SELECT *
FROM КЛИЕНТЫ К
WHERE NOT EXISTS
(SELECT П.*
FROM ПРОДАЖИ П
WHERE К.Клиент = П.Клиент)
```

Если в условии поиска нужно указать, что из таблицы требуется выбрать лишь те записи, для которых подзапрос возвращает

только одно значение, указывается предложение SINGULAR. Изменим немного предыдущий запрос и получим совсем другой результат:

```
SELECT *
FROM КЛИЕНТЫ К
WHERE SINGULAR
(SELECT П.ПРОДАЖИ FROM ПРОДАЖИ П WHERE К.Клиент =
П.Клиент)
```

Результатом выполнения запроса является список всех клиентов, приобретавших только один вид товара и только один раз, т.е. в таблице ПРОДАЖИ для такого клиента присутствует только одна запись.

Выше были рассмотрены внутренние соединения таблиц базы данных. Внутреннее соединение используется, когда нужно включить все строки из обеих таблиц, удовлетворяющие условию объединения. В этом случае строится декартово произведение строк первой и второй таблиц, а из полученного набора данных отбираются записи, удовлетворяющие условиям объединения.

Существует также и другой вид соединения таблиц — *внешнее соединение*. Оно определяется в предложении FROM согласно такой спецификации:

```
SELECT ...
FROM <таблица1><вид соединения> JOIN <таблица2>
ON <условие соединения>
```

Внешнее соединение отличается от внутреннего соединения тем, что в нем одна из таблиц является *ведущей*. В результирующий набор данных включаются записи *ведущей* таблицы соединения, которые могут объединяться с пустым множеством записей другой таблицы. Какая из таблиц будет ведущей, определяет вид соединения.

Левое внешнее соединение(LEFT)— это соединение, в котором ведущей является первая таблица, в предложении FROM расположенная слева от вида соединения. Соответственно, строки ведущей (первой) таблицы, не имеющие совпадающих значений в общих столбцах второй таблицы, также включаются в результирующий набор данных.

Правое внешнее соединение (RIGHT) — это соединение, когда ведущей является вторая таблица, в предложении FROM расположенная справа от вида соединения. В результирующем отношении содержатся все строки правой таблицы.

Полное внешнее соединение (FULL) — когда ведущими таблицами являются все таблицы. В результирующий набор данных включаются все записи обеих таблиц. Если для записи первой таблицы имеются записи второй таблицы, удовлетворяющие условию соединения, в результирующий набор данных будут включены все комбинации соединения таких записей обеих таблиц. Иначе в результирующий набор данных будет включена запись первой таблицы, соединенная с пустой записью.

С другой стороны, если для записи второй таблицы имеются записи первой таблицы, удовлетворяющие условию соединения, в результирующий запрос будут включены все комбинации соединения таких записей обеих таблиц. Иначе в результирующий набор данных будет включена запись второй таблицы, соединенная с пустой записью.

Проиллюстрируем это на примере. Выполнение оператора реализующего внешнее левое соединение таблиц КЛИЕНТЫ и ПРОДАЖИ приведет к созданию такого результирующего набора данных (рис. 6.11):

```
SELECT ПРОДАЖИ.*, КЛИЕНТЫ.*
FROM КЛИЕНТЫ LEFT JOIN ПРОДАЖИ
ON ПРОДАЖИ.Клиент = КЛИЕНТЫ.Клиент
```

Как видно из результата, некоторые строки таблицы КЛИЕНТЫ не имеют парных записей в таблице ПРОДАЖИ, удовлетворяющих условию соединения. Поэтому эти данные таблицы КЛИЕНТЫ показаны в соединении с пустыми записями.

Номер	Клиент	Товар	Количество	Дата	Клиент1	ИНН	Город	Телефон
<null>	<null>	<null>	<null>	<null>	ООО "Горизонт"	025500123152	Москва	4890605
<null>	<null>	<null>	<null>	<null>	ИП Привалов И.И.	025501025666	Санкт-Петербург	2560245
112	ООО "Ромашка"	Халва	40	15.02.2013	ООО "Ромашка"	025501244555	Москва	3658815
<null>	<null>	<null>	<null>	<null>	ООО "Перевал"	145889898622	Тверь	221588
<null>	<null>	<null>	<null>	<null>	ИП Иванов Ф.И.	025558000554	Москва	1155488
25	Таран О.С.	Халва	5	04.01.2013	Таран О.С.	360224005454	Москва	1215648
144	Таран О.С.	Сосиски молочные	30	01.04.2013	Таран О.С.	360224005454	Москва	1215648
45	Федорова Д.С.	Конфеты "Коровка"	50	02.04.2013	Федорова Д.С.	500255510055	Санкт-Петербург	4449702
162	Федорова Д.С.	Халва	10	15.02.2013	Федорова Д.С.	500255510055	Санкт-Петербург	4449702
200	Лесовая В.Н.	Мармелад "Кроха"	20	07.02.2013	Лесовая В.Н.	212585832187	Москва	3021402
85	Лесовая В.Н.	Мука пшеничная	50	02.04.2013	Лесовая В.Н.	212585832187	Москва	3021402
254	Лесовая В.Н.	Сахар	50	01.02.2013	Лесовая В.Н.	212585832187	Москва	3021402
<null>	<null>	<null>	<null>	<null>	БМСТ	555878970025	Санкт-Петербург	4353822
<null>	<null>	<null>	<null>	<null>	Дремина Е.Е.	025578721058	Москва	6582209
156	ИП Газимова К.К.	Конфеты "Мишка "	30	02.04.2013	ИП Газимова К.К.	025587978766	Москва	6521588
140	ИП Газимова К.К.	Конфеты "Коровка"	15	09.01.2013	ИП Газимова К.К.	025587978766	Москва	6521588

Рис. 6.11. Результат внешнего левого соединения таблиц КЛИЕНТЫ и ПРОДАЖИ

Аналогичным образом осуществляется правое внешнее соединение:

```
SELECT ПРОДАЖИ.*, КЛИЕНТЫ.*
FROM КЛИЕНТЫ RIGHT JOIN ПРОДАЖИ
ON ПРОДАЖИ.Клиент = КЛИЕНТЫ.Клиент,
```

и полное внешнее соединение:

```
SELECT ПРОДАЖИ.*, КЛИЕНТЫ.*
FROM КЛИЕНТЫ FULL JOIN ПРОДАЖИ
ON ПРОДАЖИ.Клиент = КЛИЕНТЫ.Клиент
```

Иногда нужно объединить два или более результирующих набора данных, возвращаемых после выполнения операторов SELECT. Такое объединение производится при помощи оператора UNION. Объединяемые наборы данных должны иметь одинаковую структуру, т.е. одинаковый состав полей (тип, размер). Если в результирующих наборах данных имеются одинаковые записи, то в объединенный набор они будут записаны одной строкой:

```
<оператор SELECT>
UNION
<оператор SELECT>
```

С помощью оператора SELECT можно реализовать весьма сложные условия выбора данных из различных таблиц.

6.4. ИЗМЕНЕНИЕ ДАННЫХ. ОПЕРАТОРЫ INSERT, UPDATE, DELETE

Для заполнения базы данных пользовательскими данными, изменения и удаления существующих данных используются операторы SQL подраздела DML. Операторы задают, что должно быть сделано с данными базы данных, не указывая, как именно это должно быть сделано. Для добавления новых строк в таблицы или в представления базы данных используется оператор INSERT. Для изменения данных в таблицах базы данных применяется оператор UPDATE. Для удаления строк таблиц используется оператор DELETE.

Все действия по изменению данных выполняются в контексте (под управлением) какой-либо транзакции. Это может быть предварительно запущенная в клиенте оператором SET TRANSACTION транзакция с необходимыми характеристиками или транзакция по умолчанию, запускаемая системой автоматически при выполнении любых операций с данными и метаданными базы данных.

Язык SQL ориентирован на выполнение операций над группами записей, хотя в некоторых случаях операция может проводиться и над отдельной записью. Поэтому операторы добавления, изменения и удаления записей в общем случае вызывают соответствующие операции над группами записей.

Добавление новых строк в обычную таблицу или таблицу, лежащую в основе представления, осуществляется с помощью оператора INSERT. Его синтаксис представлен следующим образом:

```
INSERT INTO <объект>
[(столбец1[, столбец2 …])]
{VALUES (<значение>[,< значение 2>…]) | <оператор SE-
LECT>}
```

<объект> — это таблица базы данных. Список столбцов указывает столбцы, которым будут присвоены значения в добавляемых записях. Список столбцов может быть опущен. В этом случае подразумеваются все столбцы объекта, причем в том порядке, в котором они определены в данном объекте.

Поставить в соответствие столбцам списки значений можно двумя способами: явное указание значения после слова VALUES и формирование списка значений при помощи оператора SELECT.

Явное указание списка значений применяется для добавления одной записи и имеет формат:

```
INSERT INTO <объект>
[(столбец1[, столбец2 …])]
VALUES (<значение>[,< значение 2>…])
```

Значения присваиваются столбцам по порядку следования тех и других в операторе: первому по порядку столбцу присваивается первое значение, второму столбцу второе значение и т. д. Например, в таблицу ТОВАРЫ новую запись можно добавить следующим оператором:

```
INSERT INTO ТОВАРЫ
(Товар, ЕИ, Цена)
VALUES («Сахар», «кг.», 100)
```

Поскольку столбцы таблицы ТОВАРЫ указаны в полном составе и именно в том порядке, в котором они перечислены при создании таблицы ТОВАРЫ, оператор можно упростить:

```
INSERT INTO ТОВАРЫ
VALUES («Сахар», «кг.», 100)
```

Второй формой оператора INSERT является следующий формат:

```
INSERT INTO <объект>
[(столбец1[, столбец2 …])]
<оператор SELECT>
```

При этом значениями, которые присваиваются столбцам, являются значения, возвращаемые оператором SELECT. Поскольку оператор SELECT в общем случае возвращает множество записей, то и оператор INSERT в данной форме приведет к добавлению в объект аналогичного количества новых записей.

Предположим, в какой либо базе данных определена таблица ПРОДАЖИ_ГЛ, по составу и порядку следования полей аналогичная таблице ПРОДАЖИ:

```
CREATE TABLE ПРОДАЖИ
(Номер INTEGER NOT NULL,
Клиент VARCHAR(20) NOT NULL COLLATE PXW_CYRL,
Товар VARCHAR(20) NOT NULL COLLATE PXW_CYRL,
Количество INTEGER NOT NULL,
Дата DATE NOT NULL,
PRIMARY KEY (Номер)
```

Предположим, что данные из этой таблицы необходимо ежедневно добавлять в какую-либо территориально удаленную базу данных, например ПРОДАЖИ_ГЛ. Тогда ежедневная выгрузка записей из таблицы ПРОДАЖИ в таблицу ПРОДАЖИ_ГЛ будет реализована оператором:

```
INSERT INTO ПРОДАЖИ_ГЛ
SELECT *
FROM ПРОДАЖИ
WHERE Дата = <дата>,
```

где <дата> — значение текущей даты.

Для изменения данных в столбцах существующих строк в обычной таблице или в таблице, являющейся базовой для изменяемого представления, используется оператор UPDATE. Оператор за одно обращение позволяет изменить данные во всех строках или только в части строк в таблице или в представлении. Его синтаксис представлен следующим форматом:

```
UPDATE <объект>
SET столбец1=<значение1> [,столбец2=<значение2>…]
[WHERE <условие>]
```

При изменении каждому из перечисленных столбцов присваивается соответствующее значение. Изменения выполняются для всех записей, удовлетворяющих условию, которое задается так же, как в операторе SELECT. Предложение SET задает список выполняемых изменений: указывается имя изменяемого столбца и после знака равенства новое значение столбца. Значением, как и в случае оператора INSERT, может быть сколь угодно сложное выражение. Через запятую в следующих предложениях SET можно перечислять значения других столбцов таблицы. Предложение WHERE в операторе определяет множество строк, к которым будет применяться операция изменения существующих данных. Если это предложение не указано, то будут изменены все строки таблицы:

```
UPDATE ПРОДАЖИ
SET Цена=Цена*2
WHERE Товар = «Мука»
```

В результате выполнения оператора в таблице ТОВАРЫ значение в поле Цена будет увеличено вдвое во всех записях, в которых значение поля Товар равно «Мука».

Оператор DELETE предназначен для удаления группы записей из объекта. В частном случае может быть удалена только одна запись.

Формат оператора DELETE:

```
DELETE FROM <объект>
[WHERE <условие>]
```

Удаляются все записи, удовлетворяющие условию поиска. Если убрать предложение WHERE, то в объекте будут удалены все записи.

Далее приводится пример, в котором из таблицы ПРОДАЖИ будут удалены все записи о фактах продажи сахара:

```
DELETE ПРОДАЖИ
WHERE Товар = «Сахар»
```

6.5. ХРАНИМЫЕ ПРОЦЕДУРЫ И ТРИГГЕРЫ

6.5.1. Язык хранимых процедур и триггеров

Хранимые процедуры и триггеры являются программами, которые хранятся в области метаданных базы данных. Они выполняются на стороне сервера, что во многих случаях позволяет сэкономить

ресурсы системы и уменьшить сетевой трафик. К хранимым процедурам возможно обращение из хранимых процедур, триггеров и клиентских приложений. К триггерам напрямую обращение невозможно — они вызываются автоматически при наступлении конкретного события для таблицы (изменения данных) или события базы данных. Для каждого события таблицы существует две фазы — до наступления соответствующего события (`BEFORE`) и после наступления этого события (`AFTER`).

Для описания алгоритмов обработки данных в хранимых процедурах и триггерах используется расширение языка SQL. Это расширение называется процедурным SQL (PSQL) или языком хранимых процедур и триггеров. Язык содержит обычные операторы присваивания, операторы ветвления и операторы циклов. В триггерах могут применяться специфические контекстные переменные.

Язык хранимых процедур и триггеров содержит все основные конструкции классических языков программирования. Кроме того, в нем присутствуют несколько модифицированные операторы добавления, изменения, удаления и выборки существующих данных из таблиц базы данных (`INSERT`, `UPDATE`, `DELETE` и `SELECT`). В языке хранимых процедур нельзя выполнять какие-либо изменения метаданных.

В хранимых процедурах и триггерах можно использовать средства оповещения клиентов о некоторых событиях (events), возможности выдавать пользовательские исключения (exceptions). Недопустимо выполнять операции соединения с базой данных, нельзя манипулировать транзакциями, включая старт транзакции, создание точек сохранения, возврата на точки сохранения, подтверждения или отмены транзакций. Хранимая процедура и триггер выполняются в контексте той транзакции, при которой была явно вызвана хранимая процедура или выполнялась операция манипулирования данными в базе данных, в результате чего был автоматически запущен триггер. Если триггер вызывается при соединении с базой данных и отсоединении от нее, то для него запускается транзакция по умолчанию.

В синтаксисе создания хранимых процедур и триггеров можно выделить заголовок и тело. Заголовок содержит имя программного объекта, описание локальных переменных. Для триггеров в заголовке указывается событие базы данных и фаза, при которой автоматически вызывается триггер. В заголовке хранимой процедуры можно указать входные и выходные параметры. Тело хранимой процедуры или триггера представляет собой блок

операторов, содержащий описание выполняемых программой действий. Блок операторов заключается в операторные скобки BEGIN и END. В самих программах возможно присутствие произвольного количества блоков, как последовательных, так и вложенных друг в друга.

Все действия по выборке, обработке и изменению данных в хранимых процедурах и триггерах выполняются в блоке операторов, который заключается в операторные скобки BEGIN и END.

При написании триггеров и хранимых процедур в текстах скриптов, создающих требуемые программные объекты базы данных, во избежание двусмысленности относительно использования символа завершения операторов (по нормам SQL, точка с запятой) применяется оператор SET TERM, который, строго говоря, не является оператором SQL. С помощью этого псевдооператора перед началом создания триггера или хранимой процедуры задается символ, являющийся завершающим в конце текста триггера или хранимой процедуры. После описания текста соответствующего программного объекта при помощи того же оператора SET TERM значение терминатора возвращается к обычному варианту — точка с запятой.

Например, при создании некоторого триггера следует выполнить следующие операторы:

```
SET TERM ^;
/* Формирование значения первичного ключа таблицы
ТОВАРЫ */
CREATE TRIGGER TBI_STAFF FOR ТОВАРЫ BEFORE INSERT
... [Текст триггера}
END ^
```

Операторы в блоке выполняются последовательно. В расширениях SQL существуют операторы ветвления (IF-THEN-ELSE), операторы цикла (WHILE-DO, FOR-SELECT-DO, FOR EXECUTE STATEMENT), операторы обращения к данным в базе данных (операторы выборки данных, добавления, изменения, удаления).

В любом месте текста, где допустим пробел, могут быть помещены комментарии, которые располагаются между символами /* и */. Один такой комментарий может занимать любое количество строк. Существует и другая форма комментариев: подряд идущие два символа минус (--). Текст комментария в этом случае продолжается лишь до конца текущей строки.

Для описания одной локальной переменной используется оператор DECLARE VARIABLE. Его упрощенный синтаксис может быть представлен следующим образом:

```
DECLARE [VARIABLE] <имя локальной переменной>
{ <тип данных>
| <имя домена>
| TYPE OF <имя домена>
}
```

В одном операторе можно объявить только одну локальную переменную. В триггере и хранимой процедуре можно объявлять произвольное количество локальных переменных, используя для каждой переменной отдельный оператор DECLARE VARIABLE. Имя локальной переменной должно быть уникальным среди имен локальных переменных, входных и выходных параметров хранимой процедуры в пределах данного программного объекта. Типом данных может быть любой тип данных, используемый в SQL. Вместо типа данных можно указать имя ранее созданного домена.

6.5.2. Работа с триггерами

Триггер является программой, которая хранится в области метаданных базы данных и выполняется на стороне сервера. Напрямую обращение к триггеру невозможно. Он вызывается автоматически при наступлении одного или нескольких событий, относящихся к одной конкретной таблице (к представлению), или при наступлении одного из событий базы данных. Триггер, вызываемый при наступлении события таблицы, связан с одной таблицей или представлением, одним или более событиями для этой таблицы или представления (добавление, изменение или удаление данных) и ровно с одной фазой такого события (до наступления события или после этого). Триггер выполняется в контексте той транзакции, в контексте которой выполнялась программа, вызвавшая соответствующее событие. Исключением являются триггеры, реагирующие на события базы данных. Для некоторых из них запускается транзакция по умолчанию.

Существует шесть вариантов соотношения «событие — фаза» для таблицы (представления):

1) до добавления новой строки (BEFORE INSERT);
2) после добавления новой строки (AFTER INSERT);
3) до изменения строки (BEFORE UPDATE);
4) после изменения строки (AFTER UPDATE);
5) до удаления строки (BEFORE DELETE);
6) после удаления строки (AFTER DELETE).

Триггер, связанный с событиями базы данных, может вызываться при следующих событиях:

1) при соединении с базой данных (CONNECT), перед выполнением триггера автоматически запускается транзакция по умолчанию;
2) при отсоединении от базы данных (DISCONNECT), перед выполнением триггера запускается транзакция по умолчанию;
3) при старте транзакции (TRANSACTION START), триггер выполняется в контексте этой транзакции;
4) при подтверждении транзакции (TRANSACTION COMMIT), триггер выполняется в контексте этой транзакции;
5) при отмене транзакции (TRANSACTION ROLLBACK), триггер выполняется в контексте транзакции.

Существует возможность создавать триггеры, вызываемые автоматически для одной таблицы (представления), для одной фазы и одного события, а также для одной фазы и нескольких событий.

Если для одной таблицы (представления), одного события и одной фазы существует несколько триггеров, то можно задать порядок их выполнения, указав позицию триггера в этой цепочке.

Для создания триггера используется оператор CREATE TRIGGER, синтаксис которого представлен ниже:

```
CREATE TRIGGER <имя триггера>
[ACTIVE | INACTIVE}
{ ON <событие базы данных>
| FOR {<имя таблицы> | <имя представления>}
{BEFORE | AFTER}
<событие таблицы (представления)>
[OR <событие таблицы (представления)>] ...
| {BEFORE | AFTER}
<событие таблицы (представления)>
[OR <событие таблицы (представления)>] ...
ON {<имя таблицы> | <имя представления>}
}
[POSITION <целое>]
AS <тело триггера>;
<событие таблицы (представления)> ::=
{INSERT | UPDATE | DELETE}
<событие базы данных> ::=
{ CONNECT
| DISCONNECT
| TRANSACTION START
```

```
| TRANSACTION COMMIT
| TRANSACTION ROLLBACK
}
```

Триггер может быть активным (ACTIVE) или неактивным (INACTIVE). Если триггер активен (значение по умолчанию), то он автоматически вызывается при наступлении соответствующего события (событий) таблицы или базы данных. Если триггер неактивен, то вызов триггера не происходит.

Структура тела триггера:

```
[<объявление локальных переменных>]
BEGIN
 <оператор>
 [<оператор> … ]
END
```

После описания локальных переменных в теле триггера следует блок операторов, заключенных в операторные скобки BEGIN и END.

Для триггеров существуют специфические контекстные переменные OLD и NEW. Более правильное название этих ключевых слов — префиксы имен столбцов. В триггерах можно обращаться к значению любого столбца таблицы (представления) до его изменения в клиентской программе (для этого перед именем столбца помещается ключевое слово OLD и точка) и после изменения (перед именем столбца помещается NEW и точка).

Контекстная переменная OLD (префикс имени столбца) для всех видов триггеров является переменной только для чтения. Она недоступна в триггерах, вызываемых при добавлении данных, независимо от фазы события.

Контекстная переменная NEW в триггерах для фазы события после (AFTER) также является переменной только для чтения. Она недоступна в триггерах для события удаления данных.

Значение OLD.ИмяСтолбца — позволяет обратиться к состоянию столбца, имевшему место до внесения возможных изменений.

Значение NEW.ИмяСтолбца — позволяет обратиться к состоянию столбца после внесения возможных изменений.

Например:

```
CREATE TRIGGER BU_TOVARY FOR TOVARY
ACTIVE
BEFORE UPDATE
AS
```

```
BEGIN
IF (OLD.TOVAR<>NEW.TOVAR) THEN
UPDATE PRODAJA
SET TOVAR=NEW.TOVAR
WHERE TOVAR=OLD.TOVAR;
END
```

В этом примере триггер вносит соответствующие изменения в таблицу PRODAJA, если в записи таблицы TOVARY изменилось значение столбца TOVAR.

Нельзя создавать один триггер для нескольких событий таблицы (представления) или нескольких событий базы данных.

Для изменения заголовка и/или тела существующего триггера используется оператор ALTER TRIGGER.

В операторе изменения триггера можно изменить его состояние активности (ACTIVE / INACTIVE), событие (события) таблицы (представления) и фазу события, позицию триггера и выполняемые триггером действия.

Для удаления существующего триггера используется оператор

```
DROP TRIGGER <имя триггера>
```

Триггеры являются полезным инструментом при выполнении различных действий в случае изменения данных базы данных. Основным назначением триггеров является автоматическое выполнение функций по формированию значений искусственного первичного ключа, выдачи информационных сообщений базы данных, связанных с изменениями данных, специфические способы поддержания декларативной целостности данных и выполнение определенных действий при добавлении (изменении) некоторых данных базы данных. Триггеры также могут быть использованы при наступлении событий, связанных с базой данных в целом, — соединение с базой данных, отсоединение от базы данных, запуск, подтверждение или отмена транзакций.

6.5.3. Работа с хранимыми процедурами

Хранимая процедура, так же как и триггер, является программой, хранящейся в области метаданных базы данных и выполняющейся на стороне сервера. В отличие от триггера к хранимой процедуре могут обращаться хранимые процедуры, триггеры и клиентские программы. Допустима рекурсия — хранимая про-

цедура может обращаться сама к себе. Хранимые процедуры выполняются в контексте той же транзакции, что и вызывающие их программы.

Существует два вида хранимых процедур — выполняемые хранимые процедуры (executed stored procedures) и хранимые процедуры выбора (selected stored procedures).

Выполняемые хранимые процедуры осуществляют обработку данных, находящихся в базе данных или не связанных с базой данных. Эти процедуры могут получать входные параметры и возвращать выходные параметры. Обращение к выполняемым хранимым процедурам осуществляется при выполнении оператора SQL `EXECUTE PROCEDURE`.

Хранимые процедуры выбора, как правило, осуществляют выборку данных из базы данных, возвращая произвольное количество полученных строк. Процедуры выбора также могут получать входные параметры. Значение каждой очередной прочитанной строки возвращается вызвавшей программе в выходных параметрах. Для временной приостановки выполнения такой процедуры и передачи выбранных данных вызвавшей программе в хранимой процедуре используется оператор `SUSPEND`. Обращение к хранимой процедуре выбора осуществляется при помощи оператора `SELECT`.

Синтаксически создание выполняемой хранимой процедуры от хранимой процедуры выбора никак не отличается. Для создания хранимой процедуры используется оператор `CREATE PROCEDURE`, синтаксис которого представлен ниже:

```
CREATE PROCEDURE <имя_хранимой_процедуры>
[(<список входных параметров>)]
[RETURNS (<список выходных параметров>)]
AS
[<список объявления переменных>]
BEGIN <блок операторов> END
```

Хранимой процедуре от вызвавшей программы могут передаваться входные параметры. Параметры передаются по значению, т. е. любые изменения значений входных параметров никак не влияют на значения этих параметров в вызвавшей программе. Входным параметрам может присваиваться значение по умолчанию. Параметры, для которых заданы значения по умолчанию, должны располагаться в самом конце списка. Если входной параметр основан на домене, которому также задано значение по умолчанию в предложении `DEFAULT`, то новое значение по умолчанию перекрывает указанное при описании домена.

Хранимая процедура может возвращать вызвавшей программе произвольное количество выходных параметров. Если при описании параметра, локальной переменной процедуры указано имя домена, то для него копируются все характеристики этого домена. В теле хранимой процедуры может быть описано произвольное количество локальных переменных.

Список входных параметров хранимой процедуры записывается после имени хранимой процедуры и заключается в круглые скобки:

```
<список входных параметров> ::= (<описание параметра>
[, <описание параметра>]...)
```

Выходные параметры описываются в предложении RETURNS:

```
RETURNS (<список выходных параметров>)
```

Описание выходного параметра соответствует описанию входного параметра.

Локальные переменные триггеров и процедур, входные и выходные параметры, используемые только в хранимых процедурах, являются *внутренними переменными*. Имена внутренних переменных могут совпадать с именами столбцов используемых таблиц базы данных. Это не вызовет проблемы двусмысленности имен. При использовании внутренних переменных в операторах SELECT, INSERT, UPDATE или DELETE именам внутренних переменных всегда должно предшествовать двоеточие, чтобы не спутать их с именами столбцов таблицы. Во всех остальных случаях в любых других операторах имена внутренних переменных записываются обычным образом без двоеточия.

Синтаксис операции присваивания значения внутренней переменной имеет следующий вид:

```
<имя переменной> = <выражение>;
```

Выражением может быть любое правильное выражение SQL.

Оператор EXIT позволяет из любой точки триггера или хранимой процедуры перейти на конечный оператор END, т. е. завершить выполнение программы:

```
EXIT [<метка>]
```

Оператор SUSPEND временно приостанавливает выполнение хранимой процедуры выбора (процедуры, которая чаще всего содержит оператор SELECT, выбирающий множество строк таблицы, обращение к такой процедуре также выполняется при помощи

оператора SELECT) и передает вызвавшей программе значения выходных параметров.

Триггеры и хранимые процедуры могут вызывать хранимые процедуры при использовании оператора EXECUTE PROCEDURE:

```
EXECUTE PROCEDURE <имя процедуры>
[(<параметр> [, <параметр>] ...)]
[RETURNING_VALUES (<параметр>[,<параметр>]...)
| <параметр>[,<параметр >...]]
```

Оператор FOR SELECT DO имеет следующий формат:

```
FOR
<оператор SELECT>
DO
<оператор>
```

Алгоритм работы оператора FOR SELECT DO заключается в следующем. Выполняется оператор SELECT, и для каждой строки полученного результирующего набора данных выполняется оператор, следующий за словом DO. Этим оператором часто бывает SUSPEND, который приводит к возврату выходных параметров в вызывающее приложение.

Для изменения существующей хранимой процедуры используется оператор ALTER PROCEDURE. Для удаления существующей хранимой процедуры используется оператор DROP PROCEDURE.

Приведем примеры создания хранимых процедур.

Следующая процедура выдает все факты продажи конкретного товара, определяемого содержимым входного параметра Товар_вход.

```
CREATE PROCEDURE PRODAJA_TOVARA
(Товар_вход VARCHAR(20))
RETURNS (Дата_вых DATE, Клиент_вых VARCHAR(20),
Количество_вых INTEGER) AS
BEGIN
FOR SELECT Дата, Клиент, Количество
FROM Продажи
WHERE Товар =:Товар_вход
INTO :Дата_вых, :Клиент_вых, :Количество_вых
DO
SUSPEND;
END
```

Сначала выполняется оператор SELECT, который возвращает дату продажи, наименование покупателя и количество расхода

товара для каждой записи, у которой поле Товар содержит значение, идентичное значению во входном параметре Товар_вход. Указанные значения записываются в выходные параметры (соответственно Дата_вых, Клиент_вых, Количество_вых). После выдачи каждой записи результирующего набора данных выполняется оператор SUSPEND. Он возвращает значения выходных параметров вызвавшему приложению и приостанавливает выполнение процедуры до запроса следующей порции выходных параметров от вызывающего приложения. Такая процедура является процедурой выбора, поскольку она может возвращать множественные значения входных параметров в вызывающее приложение.

Другая процедура Клиенты_список возвращает имена всех клиентов, которые сделали покупки товара Товар_вход в количестве, превосходящем средний размер покупки по этому товару. В случае если наименование клиента — пустое значение, вместо имени клиента выводится «Данные отсутствуют».

```
CREATE PROCEDURE Клиенты_список (Товар_вход VAR-
CHAR(20))
RETURNS (Клиент_вых VARCHAR(20)) AS
DECLARE VARIABLE Сред_количество INTEGER;
BEGIN
SELECT AVG(Количество)
FROM Продажи
WHERE Товар =:Товар_вход
INTO :Сред_количество;
FOR
SELECT Клиенты
FROM Продажи
WHERE Количество >: Сред_количество
INTO :Клиент_вых
DO
BEGIN
IF (:Клиент_вых IS NULL) THEN
Клиент_вых = «Данные отсутствуют»;
SUSPEND;
END
END
```

Достоинствами хранимых процедур является то, что они, во-первых, выполняются на стороне сервера, что во многих случаях может резко сократить сетевой трафик; во-вторых, один раз написанная и отлаженная хранимая процедура может использоваться

многими программами — хранимыми процедурами, триггерами, клиентскими программами.

Хранимая процедура выбора используется, как правило, для выборки достаточно большого количества данных из базы данных. Алгоритм выборки данных в таких случаях достаточно сложный. Подобного вида процедуры обычно используются в том случае, когда декларативных средств оператора SELECT недостаточно для выполнения всех действий по выборке релевантных данных из таблиц или представлений. Такая процедура также может получать входные параметры и возвращать выходные параметры.

Хранимые процедуры улучшают целостность приложений и БД, гарантируют актуальность коллективных операций и вычислений. Улучшается сопровождение таких процедур, а также безопасность (нет прямого доступа к данным).

Однако следует помнить, что перегрузка хранимых процедур прикладной логикой может перегрузить сервер, что приведет к потере производительности. Эта проблема особенно актуальна при разработке крупных информационных систем, когда к серверу может одновременно обращаться большое количество клиентов. Поэтому в большинстве случаев следует принимать компромиссные решения: часть логики приложения размещать на стороне сервера, часть — на стороне клиента (системы с разделенной логикой).

6.6. РАБОТА С ИНДЕКСАМИ

Индекс — это объект базы данных, содержащий значения указанных столбцов конкретной таблицы и ссылки на строки этой таблицы, содержащие данные значения.

Индекс создается пользователем или системой для конкретной таблицы, что позволяет во многих случаях ускорить процесс поиска данных в этой таблице, а иногда и ускорить упорядочение данных, полученных по запросу пользователя на основании предложения ORDER BY в операторе SELECT. Каждая строка индекса содержит значение столбцов, входящих в состав индекса и указатель на строку в таблице, которая имеет те же самые значения столбцов.

При наличии индексов во многих случаях поиск данных может выполняться гораздо быстрее, чем при отсутствии индекса, потому что значения в индексе упорядочены, а сам индекс относительно мал. Не следует создавать индексов для столбцов, которые имеют небольшое количество вариантов значений, например для столбцов, имеющих два значения, в частности для столбцов, моделирующих

логический тип данных, где столбец может иметь только значения TRUE и FALSE, или в случае задания пола человека — мужской или женский. Такие индексы только занимают место во внешней памяти и не дают никакого выигрыша в производительности при выполнении операций выборки и упорядочения данных.

Для ограничений первичного ключа, уникального ключа и внешнего ключа система автоматически строит индексы.

Важное правило: нельзя создавать индекс по структуре и по упорядоченности соответствующий индексу, который автоматически создается системой для первичного, уникального или внешнего ключа, при попытке выборки данных это может привести к аварийному завершению работы сервера базы данных.

Индекс может быть создан как уникальный (ключевое слово UNIQUE). В этом случае в таблице не допускается присутствие двух различных строк, имеющих одинаковое значение столбцов, входящих в состав уникального индекса.

Индекс может быть упорядочен по возрастанию значений столбцов, входящих в его состав (ASCENDING — значение по умолчанию) или по убыванию этих значений (DESCENDING). В любой момент времени работы с базой данных индекс может быть сделан активным (ACTIVE), т. е. все изменения столбцов таблицы, входящих в состав этого индекса, тут же отражаются в самом индексе, или неактивным (INACTIVE), когда никакие изменения в строках соответствующей таблицы базы данных не затрагивают содержание индекса.

Для создания индекса для существующей таблицы базы данных используется оператор:

```
CREATE [UNIQUE] [ASC[ENDING] | DESC[ENDING]]
INDEX <имя индекса> ON <таблица>
(<столбец> [, <столбец>] ...)
```

Имя индекса должно быть уникальным среди имен всех индексов базы данных, а также среди имен ограничений на уровне столбцов таблицы и ограничений на уровне таблиц. Когда при задании ограничений первичного, уникального или внешнего ключа вы указываете и имя ограничения в предложении CONSTRAINT, система строит индекс с тем же самым именем.

Ключевое слово UNIQUE задает создание уникального индекса, оно указывает, что в индексе не может быть двух строк с одинаковыми значениями всех столбцов индекса. Столбцы, входящие в состав уникального индекса, не могут также иметь пустого значения NULL.

Ключевое слово ASCENDING (сокращенный вариант ASC) означает, что записи индекса упорядочиваются по возрастанию значений столбцов, входящих в состав индекса. Этот вариант принимается по умолчанию.

Ключевое слово DESCENDING (сокращение DESC) указывает, что записи индекса упорядочиваются по уменьшению значений столбцов индекса.

В состав индекса не могут входить столбцы, имеющие тип данных BLOB, а также столбцы любого типа данных, являющиеся массивами.

Например. Если для таблицы ПРОДАЖИ требуются и возрастающий и убывающий индексы по столбцу, хранящему наименование товаров Товары, то нужно создать два индекса, выполнив следующие операторы:

```
CREATE ASCENDING INDEX I_ASC ON ПРОДАЖИ (Товары)
CREATE DESCENDING INDEX I_DESC ON ПРОДАЖИ (Товары)
```

При первоначальном создании индекс становится по умолчанию активным — все вновь добавленные строки в таблицу или выполненные изменения в индексированных столбцах базовой таблицы тут же отражаются на состоянии индекса.

В некоторых случаях бывает полезным на некоторое время «отключить» индекс, сделать его неактивным. Это может сэкономить время при выполнении так называемых пакетных операций с таблицей, когда в таблицу, для которой создан индекс, из какого-либо файла записывается достаточно большое количество строк или в таблице изменяется или из таблицы удаляется большое количество строк. Перед началом такой операции индекс переводится в неактивное (INACTIVE) состояние, а после завершения операции — снова в активное (ACTIVE). При этом при активизации индекса осуществляется полное пересоздание индекса. Все вновь введенные, измененные или удаленные строки будут учтены в новом состоянии индекса.

Изменение состояния индекса осуществляется при помощи оператора:

```
ALTER INDEX <имя индекса> {ACTIVE | INACTIVE}
```

Ключевое слово ACTIVE задает перевод неактивного индекса в активное состояние. Ключевое слово INACTIVE указывает, что индекс переводится в неактивное состояние. После перевода индекса из неактивного в активное состояние система заново полностью создает весь индекс.

Этот оператор не может быть использован для изменения структуры индекса или его упорядоченности. Если есть необходимость внести изменения в структуру индекса или изменить порядок, то следует удалить существующий индекс (см. следующий раздел), а затем создать индекс с тем же именем и с требуемыми характеристиками.

Для удаления индекса, созданного пользователем, используется оператор

```
DROP INDEX <имя индекса>
```

Нельзя таким образом удалить индекс, созданный автоматически системой для первичного, уникального или внешнего ключа. Можно только удалить индекс, который был создан пользователем.

6.7. ГЕНЕРАТОРЫ

***Генератор* (*generator*)** — это самый простой объект базы данных. Генератором называется хранимый на сервере БД механизм, возвращающий уникальные значения, никогда не совпадающие со значениями, выданными тем же самым генератором в прошлом. Он позволяет хранить целые числа в очень широком диапазоне значений. Под него отводится 8 байтов памяти. Это подходящее средство для формирования значений искусственных первичных ключей. Для каждого искусственного первичного ключа любой таблицы базы данных пользователем создается свой собственный генератор, с которым выполняются все действия по формированию значений этого первичного ключа.

В принципе, генераторы могут использоваться и для получения последовательностей неповторяющихся целых чисел для любых других целей.

Для создания генератора используется следующий оператор:

```
CREATE GENERATOR <имя генератора>
```

Имя генератора должно быть уникальным среди имен всех генераторов базы данных.

В момент создания генератора ему присваивается значение 0. Последующие обращения к функции `GEN_ID`, в параметрах которой используется имя этого генератора, изменяют это значение на указанную величину. Обычно приращением является единица, но можно использовать и любые другие целые числа, отличные от нуля (нуль по правилам синтаксиса тоже можно использовать,

однако это не позволит получить уникальную последовательность чисел).

Установка значения генератора осуществляется оператором:

```
SET GENERATOR <имя генератора> TO <стартовое значение>
```

Для получения уникального значения используется функция

```
GEN_ID (ИмяГенератора, шаг)
```

Не рекомендуется переустанавливать стартовое значение генератора или менять шаг при разных обращениях к функции GEN_ID. В этих случаях генератор может выдать не уникальное значение и как следствие будет возбуждено исключение при попытке запоминания новой записи в таблице базы данных.

Генератор можно удалить, используя оператор SQL

```
DROP GENERATOR <имя генератора>
```

Удалять генератор следует только после того, как будут удалены из базы данных все триггеры и хранимые процедуры, ссылающиеся на этот генератор.

Следующий пример иллюстрирует использование генератора при добавлении новой строки. Пусть в базе данных определен генератор для столбца Номер в таблице ПРОДАЖИ:

```
CREATE GENERATOR Ген_номер;
SET GENERATOR Ген_номер TO 0;
```

Обращение к генератору непосредственно из оператора INSERT можно записать следующим образом:

```
INSERT INTO PRODAJA
(Номер, Дата, Количество, Товар, Клиент)
VALUES (
GEN_ID (Ген_номер, 1),
‘01.05.2013’,
100, «Сахар»,
«ООО Ромашка»)
```

С помощью триггера может быть реализовано присваивание ключевому столбцу уникального значения, вызываемого перед записью новой строки в базе данных:

```
CREATE TRIGGER BI_PRODAJA FOR ПРОДАЖА
ACTIVE
BEFORE INSERT
```

```
AS
   BEGIN
     NEW. Номер = GEN_ID (Ген_номер, 1)
   END
```

Для генерирования уникальных значений можно использовать хранимую процедуру:

```
CREATE PROCEDURE GEN_PROC
RETURNS (N INTEGER)
AS
BEGIN
   N= GEN_ID (Ген_номер, 1)
   SUSPEND
END
```

Важной особенностью генераторов является то, что работа с ними выполняется вне контекста какой-либо транзакции. Это означает, что при одновременном обращении к одному и тому же генератору разных конкурирующих транзакций никогда не возникнет конфликта блокировки, и каждый параллельный процесс получит уникальное новое числовое значение. Значение зависит от времени обращения к генератору.

6.8. ДОСТОИНСТВА ЯЗЫКА SQL

Язык SQL в настоящее время получил очень широкое распространение и фактически превратился в стандартный язык реляционных баз данных. Все крупнейшие разработчики систем управления базами данных в настоящее время создают свои продукты с использованием языка SQL. В него делаются огромные инвестиции со стороны и разработчиков, и пользователей. Он стал частью архитектуры приложений, является стратегическим выбором многих крупных и влиятельных организаций. Таким образом, SQL можно представить как мощнейший инструмент, который обеспечивает пользователям, программам и вычислительным системам доступ к информации.

Язык SQL предоставляет пользователю возможности:

- создавать различные объекты базы данных, в том числе таблицы с полным описанием их структуры;
- выполнять основные операции манипулирования данными: добавление, модификацию и удаление данных из таблиц;
- выполнять различные запросы, в том числе и очень сложные и громоздкие, осуществляющие преобразование данных.

Следует отметить, что язык SQL решает все указанные выше задачи при минимальных усилиях со стороны пользователя, а структура и синтаксис его команд достаточно просты и доступны для изучения.

Основные достоинства языка SQL:

- *стандартность* — использование языка SQL в программах стандартизировано международными организациями;
- *независимость от конкретных СУБД* — все распространенные СУБД используют SQL, так как реляционную базу данных можно перенести с одной СУБД на другую с минимальными доработками;
- *возможность переноса с одной вычислительной системы на другую*;
- *реляционная основа языка* — SQL является языком реляционных баз данных, поэтому он стал популярным тогда, когда получила широкое распространение реляционная модель представления данных. Табличная структура реляционной БД хорошо понятна, а потому язык SQL прост для изучения;
- *возможность создания интерактивных запросов* — в интерактивном режиме можно получить результат запроса за очень короткое время без написания сложной программы;
- *язык SQL легко использовать в приложениях, которым необходимо обращаться к базам данных*. Одни и те же операторы SQL употребляются как для интерактивного, так и программного доступа, поэтому части программ, содержащие обращение к БД, можно вначале проверить в интерактивном режиме, а затем встраивать в программу;
- *обеспечение различного представления данных* — с помощью SQL можно представить такую структуру данных, что тот или иной пользователь будет видеть различные их представления. Кроме того, данные из разных частей БД могут быть скомбинированы и представлены в виде одной простой таблицы, а значит, представления пригодны для усиления защиты БД и ее настройки под конкретные требования отдельных пользователей;
- *возможность динамического изменения и расширения структуры базы данных* — язык SQL позволяет манипулировать структурой базы данных, тем самым обеспечивая гибкость. Создавать и удалять объекты базы данных, изменять их структуру можно во время выполнения приложения. Это очень важно с точки зрения приспособленности базы данных к изменяющимся требованиям предметной области;

- *возможность работы в архитектуре «клиент-сервер»* — SQL является одним из лучших средств для реализации взаимодействия клиента и сервера, более того, служит связующим звеном между взаимодействующей с пользователем клиентской системой и серверной системой, управляющей БД, позволяя каждой из них сосредоточиться на выполнении своих функций.

Создание языка SQL способствовало не только выработке необходимых теоретических основ, но и подготовке успешно реализованных технических решений. Начали появляться специализированные реализации языка, предназначенные для новых рынков — систем управления обработкой транзакций (OnLine Transaction Processing, OLTP) и систем оперативной аналитической обработки или системы поддержки принятия решений (OnLine Analytical Processing, OLAP).

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Какие требования предъявляются к языку работы с базами данных?
2. Что называется реализацией языка SQL?
3. Почему язык SQL стал популярен среди разработчиков?
4. Какие категории имеются в SQL?
5. Какие команды относятся к каждой категории языка?
6. Какие типы данных используются в SQL?
7. Каким образом используются домены при создании таблиц?
8. Какие ограничения могут быть описаны в домене?
9. Как задается первичный ключ при создании таблицы в языке SQL?
10. Что называется ограничением ссылочной целостности и как оно создается в языке SQL?
11. Как создается связь при определении таблиц в языке SQL?
12. Опишите формат оператора SELECT.
13. Какие функции можно использовать в операторе SELECT?
14. Как сгруппировать результат запроса?
15. Как наложить ограничение на результат запроса?
16. Как отсортировать результат запроса?
17. Что называется внутренним соединением таблиц?
18. В каких случаях в запросах используется предложение HAVING?
19. Что называется подзапросом?
20. Какой подзапрос называется скалярным, а какой — табличным?

21. Для чего в подзапросах используются предложения SINGULAR, EXISTS, ALL, SOME?
22. Что называется внешним соединением таблиц и чем внешнее соединение отличается от внутреннего?
23. Что называется генератором?
24. Для чего используются триггеры?
25. При каких событиях происходит вызов триггера?
26. Каковы преимущества использования хранимых процедур и триггеров?
27. В чем основное отличие триггера от хранимой процедуры?
28. В каких случаях следует создавать индексы?
29. Как осуществляется добавление новых строк в таблицу с помощью SQL?
30. Как осуществляется удаление строк в таблице с помощью SQL?
31. Как осуществляется изменение данных в таблице с помощью SQL?
32. Каковы достоинства использования языка SQL?