Лекция 12

Создание базы данных

Большинству пользователей не приходится самостоятельно создавать базы данных. Как правило, они используют программный или интерактивный SQL для доступа к базе данных, созданной кем-то другим. Например, в типичной корпоративной базе данных ее администратор может выдать пользователю разрешение на выборку и, возможно, на изменение хранимых данных. Однако администратор не позволит пользователю создавать новые базы данных или изменять структуру существующих таблиц.

Тем не менее, по мере приобретения опыта работы с SQL, вы, вероятно, захотите создавать собственные таблицы для хранения своих данных, например результатов технических испытаний или плана-прогноза объема продаж. При работе в многопользовательской среде может возникнуть потребность в создании нескольких таблиц или даже целой базы данных для совместного использования с другими сотрудниками. Если вы работаете с базой данных, расположенной на персональном компьютере, то вам наверняка потребуется создавать собственные таблицы и базы данных для поддержки своих прикладных программ.

В настоящей главе рассматриваются средства языка SQL, позволяющие создавать таблицы и базы данных и определять их структуру.

Язык определения данных

Инструкции `SELECT`, `INSERT`, `DELETE`, `UPDATE`, `COMMIT` и `ROLLBACK`, рассмотренные в предыдущих частях книги, предназначены для обработки данных. В совокупности эти инструкции называются *языком обработки данных* или DML (Data Manipulation Language). Инструкции DML могут модифицировать информацию, хранимую в базе данных, но не могут изменять ее структуру. Например, ни одна из этих инструкций не позволяет создавать и удалять таблицы или столбцы.

Для изменения структуры базы данных предназначен другой набор инструкций SQL, так называемый *язык определения данных* или DDL (Data Definition Language). С помощью инструкций DDL можно выполнить следующее:

- определить структуру новой таблицы и создать ее;
- удалить таблицу, которая больше не нужна;
- изменить определение существующей таблицы;
- определить виртуальную таблицу (или представление) данных;
- обеспечить безопасность базы данных;
- создать индекс для ускорения доступа к таблице;
- управлять физическим размещением данных.

В большинстве случаев инструкции DDL обеспечивают высокий уровень доступа к данным и позволяют пользователю не вникать в детали хранения информации в базе данных на физическом уровне. Они оперируют абстрактными объектами базы данных, такими как таблицы и столбцы. Однако DDL не может не затрагивать вопросов, связанных с физической памятью. Естественно, что инструкции и предложения DDL, управляющие физической памятью, могут быть разными в различных СУБД.

Ядро языка определения данных образуют три команды:

- **CREATE** (создать), позволяющая определить и создать объект базы данных;
- **DROP** (удалить), служащая для удаления существующего объекта базы данных;
- **ALTER** (изменить), посредством которой можно изменить определение объекта базы данных.

Все основные реляционные СУБД позволяют использовать три указанные команды DDL во время работы. Таким образом, структура реляционной базы данных является динамической. Например, СУБД может создавать, удалять или изменять таблицы, одновременно с этим обеспечивая доступ пользователям к базе данных. Это — одно из главных преимуществ реляционных баз данных по сравнению с более ранними системами, в которых изменять структуру базы данных можно было только после прекращения работы СУБД. Это означает, что с течением времени реляционная база данных может расти и изменяться. Ее промышленная эксплуатация может продолжаться в то время, когда в базу данных добавляются все новые таблицы и приложения.

Хотя DDL и DML являются двумя отдельными частями SQL, в большинстве реляционных СУБД такое разделение существует лишь на абстрактном уровне. Обычно инструкции DDL и DML в СУБД абсолютно равноправны, и их можно произвольно чередовать как в интерактивном, так и в программном SQL. Если программе или пользователю требуется таблица для временного хранения результатов, они могут создать эту таблицу, заполнить ее, проделать с данными необходимую работу и затем удалить таблицу. Это большое преимущество по сравнению с более ранними моделями представления данных, когда структура базы данных жестко фиксировалась при ее создании.

Хотя практически все коммерческие СУБД поддерживают DDL как неотъемлемую часть языка SQL, исходный стандарт SQL1 этого не требовал. В SQL1 между инструкциями DDL и DML имеется четкое разделение и допускается реализовать DML как надстройку над нереляционным ядром базы данных. В последующих

версиях стандарта SQL все еще имеется разграничение между различными типами инструкций SQL (инструкции DDL называются инструкциями SQL-схемы, а инструкции DML — инструкциями SQL-данных и инструкциями SQL-транзакций), но в то же время сам стандарт приведен в соответствие реальным принципам реализации современных СУБД: в нем требуется, чтобы инструкции DDL можно было выполнять как в интерактивном режиме, так и в приложениях.

В стандарте SQL определены только те части DDL, которые относительно независимы от структур физического хранения, особенностей операционных систем и других специфических особенностей СУБД. На практике все СУБД включают множество дополнений к DDL, связанных со спецификой реализации каждой конкретной СУБД. Ниже в настоящей главе описываются различия между стандартом ANSI/ISO и реализацией DDL в популярных реляционных СУБД.

Создание базы данных

В СУБД, установленных на мэйнфреймах или в крупных корпоративных сетях, за создание новых баз данных отвечает только администратор. В СУБД, установленных на серверах более низкого уровня, отдельным пользователям может быть разрешено создавать собственные базы данных, но обычно в таких СУБД базы данных создаются централизованно, а пользователи затем работают с ними. Если вы работаете с базой данных на персональном компьютере, то, скорее всего, являетесь как ее администратором, так и пользователем, и вам придется создавать базу самостоятельно.

В стандарте SQL1 содержится спецификация языка SQL, используемого для описания структуры базы данных, но не указывается способ создания базы данных, поскольку в различных СУБД применялись разные подходы к этому вопросу. Эти различия остались и сегодня. Методы создания баз данных, применяемые в ведущих реляционных СУБД, отчетливо иллюстрируют эти различия.

- В СУБД DB2 структура базы данных определена по умолчанию. База данных ассоциируется с выполняемой копией серверного обеспечения DB2, и пользователь получает доступ к базе данных, подключаясь к серверу DB2. Таким образом, база данных создается в процессе инсталляции DB2 на конкретную компьютерную систему.
- В СУБД Oracle база данных создается в процессе установки программного обеспечения, как и в DB2. Однако изредка администратор Oracle самостоятельно создает базу данных при помощи команды `CREATE DATABASE` или графического приложения из поставки Oracle, устанавливая определенные параметры базы данных, наилучшим образом соответствующие ее ожидаемому использованию. Как правило, каждая копия программного обеспечения СУБД Oracle управляет единственной базой данных, указанной в конфигурационном файле; пользовательские таблицы располагаются в схемах в этой базе данных. Заметим, что конкретный сервер или мэйнфрейм могут управлять несколькими базами данных, но в этом случае каждая из них требует работы собственной копии программного обеспечения СУБД.

- В СУБД Microsoft SQL Server и Sybase имеется инструкция CREATE DATABASE, которая является частью языка определения данных. Сопутствующая ей инструкция DROP DATABASE удаляет существующие базы данных. Эти инструкции можно использовать как в интерактивном, так и в программном SQL. Имена создаваемых баз данных отслеживаются в специальной "главной" базе данных, которая связана с конкретной инсталляцией СУБД. Хотя архитектура этих СУБД отличается от DB2 и Oracle, базы данных Sybase и SQL Server концептуально аналогичны схемам Oracle или DB2. Имена баз данных в пределах инсталляции SQL Server должны быть уникальны. С помощью параметров инструкции CREATE DATABASE можно задать физическое устройство, на котором размещается база данных.
- СУБД Informix Universal Server (ныне — продукт IBM) поддерживает инструкции CREATE DATABASE и DROP DATABASE. Инструкция CREATE DATABASE позволяет создать базу данных в указанной *области*, представляющей собой именованный участок дисковой памяти, находящийся под управлением СУБД. Кроме того, можно указать тип подключения к новой базе данных, а также сделать выбор между производительностью базы данных и целостностью ее данных при системных сбоях.
- MySQL также поддерживает инструкции SQL CREATE DATABASE и DROP DATABASE. Инструкция CREATE DATABASE управляет параметрами хранения базы данных и механизма СУБД. Каждая база данных хранится в своем подкаталоге корневого каталога инсталляции MySQL.

В стандарте SQL умышленно не дано определение термина *база данных*, поскольку в различных СУБД этот термин часто трактуется по-разному. Вместо него стандарт употребляет термин *каталог*, означающий именованную коллекцию системных таблиц, описывающих структуру базы данных (которая и называется базой данных в ведущих СУБД). (Дополнительная информация о структуре базы данных в SQL приведена позже в данной главе.) В стандарте не описано, как должен создаваться и уничтожаться каталог, и специально указано, что подобные вопросы зависят от реализации. Подчеркивается также, что вопросы о количестве каталогов в системе и о том, могут ли те или иные инструкции SQL обращаться к данным из различных каталогов, тоже зависят от реализации СУБД. На практике, как было показано выше, в большинстве СУБД для создания и удаления баз данных используется пара инструкций CREATE DATABASE/DROP DATABASE.

Определения таблиц

В реляционной базе данных наиболее важным элементом ее структуры является таблица. В многопользовательской базе данных основные таблицы, как правило, создает администратор, а пользователи просто обращаются к ним в процессе работы. Тем не менее при работе с такой базой данных часто оказывается удобным создавать собственные таблицы и хранить в них собственные данные, а также данные, извлеченные из других таблиц. Эти таблицы могут быть временными и существовать только в течение одного интерактивного SQL-сеанса, но могут сохраняться и в про-

должение нескольких недель или даже месяцев. В базе данных на персональном компьютере структуры таблиц являются еще более динамичными. Поскольку вы являетесь и пользователем, и администратором базы данных, можете создавать и удалять таблицы по мере необходимости, не думая о других пользователях.

Создание таблицы (CREATE TABLE)

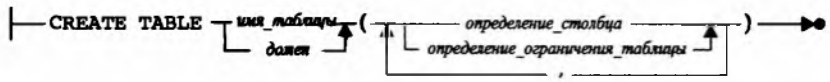
Инструкция CREATE TABLE, синтаксическая диаграмма которой изображена на рис. 13.1, определяет новую таблицу и подготавливает ее к приему данных. Различные предложения инструкции задают элементы определения таблицы. Синтаксическая диаграмма инструкции кажется довольно громоздкой, поскольку требуется указать много элементов и параметров для них. Кроме того, некоторые параметры в одних СУБД присутствуют, а в других нет. На практике же создать таблицу относительно несложно.

После выполнения инструкции CREATE TABLE вы становитесь владельцем новой таблицы, которой присваивается указанное в инструкции имя. Если вы обладаете соответствующими привилегиями, можете создавать таблицы для других пользователей, используя квалифицированные имена с указанием владельца. Имя таблицы должно быть идентификатором, допустимым в SQL, и не должно конфликтовать с именами существующих таблиц. Таблица создается пустой, но СУБД подготавливает ее к приему данных, которые записываются с помощью инструкции INSERT.

Определения столбцов

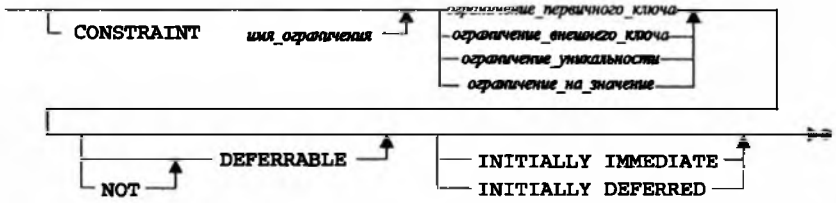
Столбцы новой таблицы задаются в инструкции CREATE TABLE. Определения столбцов представляют собой заключенный в скобки список, элементы которого отделены друг от друга запятыми. Порядок следования определений столбцов в списке определяет расположение столбцов в таблице. В инструкции CREATE TABLE, поддерживаемой основными СУБД, каждое определение столбца содержит следующую информацию.

- **Имя столбца** используется для обращения к столбцу в инструкциях SQL. Каждый столбец в таблице должен иметь уникальное имя, но в разных таблицах имена столбцов могут совпадать.
- **Тип данных столбца** указывает, данные какого вида хранятся в столбце. Типы данных были рассмотрены в главе 5, "Основы SQL". В стандарте SQL указаны и домены (описаны в главе 11, "Целостность данных"), но они поддерживаются только некоторыми из современных СУБД. Для некоторых типов, например VARCHAR и DECIMAL, требуется дополнительная информация, такая как длина или число десятичных разрядов. Эта дополнительная информация заключается в скобки за ключевым словом, определяющим тип данных.
- **Обязательность данных** определяет, допускаются ли в данном столбце значения NULL.
- **Значение по умолчанию**. Это необязательное значение *по умолчанию*, которое заносится в столбец в том случае, если в инструкции INSERT для таблицы не указано значение для данного столбца.

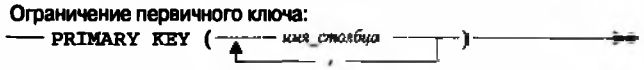


Определение столбца:

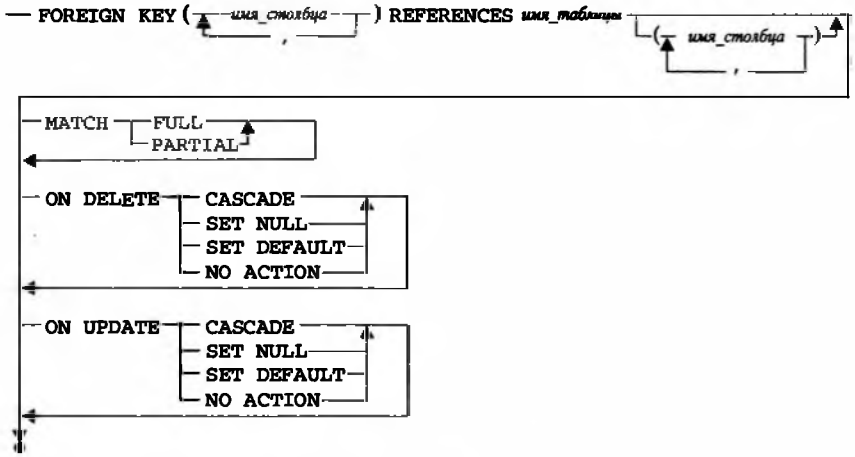
Определение ограничения таблицы:



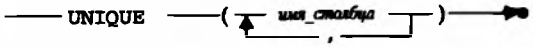
Ограничение первичного ключа:



Ограничение внешнего ключа:



Ограничение уникальности:



Ограничение на значение:

Синтаксическая диаграмма инструкции CREATE TABLE

Стандарт SQL позволяет указать в определении столбца несколько дополнительных элементов, с помощью которых можно установить, что столбец должен содержать уникальные значения либо являться первичным или внешним ключом, а также ограничить допустимые значения данных в столбце. Эти элементы, по сути, пред-

ставляют собой варианты других предложений инструкции CREATE TABLE, предназначенные для одного столбца, и описываются в последующих разделах как составляющие этой инструкции.

ставляют собой варианты других предложений инструкции CREATE TABLE, предназначенные для одного столбца, и описываются в последующих разделах как составляющие этой инструкции.

Теперь СУБД будет автоматически проверять каждую добавляемую в эту таблицу строку и выяснять, находится ли указанный в строке номер офиса в определенном диапазоне значений. Домены особенно эффективны, когда в базе данных одни и те же ограничения применимы сразу к нескольким столбцам. В нашей учебной базе данных номера офисов появляются в таблицах OFFICES и SALESREPS, поэтому домен VALID_OFFICE_ID можно применить к соответствующим столбцам обеих таблиц. В промышленных базах данных могут существовать десятки и сотни столбцов, данные которых относятся к одному и тому же домену.

Значения по умолчанию и отсутствующие значения

В определении каждого столбца указывается, могут ли в нем отсутствовать данные, т.е. допускается ли хранение в нем значений NULL. В большинстве СУБД и в стандарте SQL по умолчанию считается, что значения NULL допускаются. Если же столбец обязательно должен содержать данные в каждой строке, в его определение необходимо включить ограничение NOT NULL. В СУБД Sybase и SQL Server, наоборот, предполагается, что значения NULL недопустимы, если иное явно не объявлено в определении столбца.

Как стандарт SQL, так и многие реляционные СУБД поддерживают задание для столбцов значений по умолчанию, которые указываются в определении столбца. Вот пример инструкции CREATE TABLE для таблицы OFFICES, задающей значения по умолчанию.

Определение таблицы OFFICES со значениями по умолчанию (стандарт ANSI/ISO).

```
CREATE TABLE OFFICES
     (OFFICE INTEGER       NOT NULL,
        CITY VARCHAR(15)   NOT NULL,
      REGION VARCHAR(10)   NOT NULL DEFAULT 'Eastern',
         MGR INTEGER                DEFAULT 106,
      TARGET DECIMAL(9,2)           DEFAULT NULL,
       SALES DECIMAL(9,2)  NOT NULL DEFAULT 0.00);
```

При таком определении таблицы в процессе ввода в нее сведений о новом офисе необходимо задать только идентификатор офиса и город, в котором он расположен. По умолчанию устанавливаются: район — Eastern, менеджер офиса — Сэм Кларк (идентификатор служащего 106), текущий объем продаж — нуль, план продаж — NULL. Отметим, что в определении столбца TARGET можно не указывать описание DEFAULT NULL; по умолчанию он все равно будет принимать значения NULL.

Определения первичного и внешнего ключей

Кроме определений столбцов таблицы, в инструкции CREATE TABLE указывается информация о первичном ключе таблицы и ее связях с другими таблицами базы данных. Эта информация содержится в предложениях PRIMARY KEY и FOREIGN KEY. Вначале они поддерживались в IBM SQL, а затем были внесены в стандарт ANSI/ISO. Большинство основных реализаций SQL поддерживает эти предложения.

В предложении PRIMARY KEY задается столбец или столбцы, которые образуют первичный ключ таблицы. Как говорилось в главе 4, "Реляционные базы данных", этот столбец (или комбинация столбцов) служит в качестве уникального иденти-

фикатора строк таблицы. СУБД автоматически следит за тем, чтобы первичный ключ каждой строки таблицы имел уникальное значение. Кроме того, в определениях столбцов первичного ключа должно быть указано, что они не могут содержать значения NULL (имеют ограничение NOT NULL).

В предложении FOREIGN KEY задается внешний ключ таблицы и определяется связь, которую он создает для нее с другой (родительской) таблицей. В этом предложении указывается следующее:

- столбец или столбцы создаваемой таблицы, которые образуют внешний ключ;
- таблица, связь с которой создает внешний ключ. Это родительская таблица; определяемая таблица в данном отношении является дочерней;
- необязательный список имен столбцов родительской таблицы, которые соответствуют столбцам внешнего ключа определяемой таблицы. Если имена столбцов опущены, в родительской таблице обязаны быть столбцы с именами, идентичными именам столбцов во внешнем ключе;
- необязательное имя для этого отношения; оно не используется в инструкциях SQL, но может появляться в сообщениях об ошибках и потребуется в дальнейшем, если будет необходимо удалить внешний ключ;
- как СУБД должна трактовать значения NULL в одном или нескольких столбцах внешнего ключа при связывании его со строками таблицы-предка;
- необязательное правило удаления для данного отношения (CASCADE, SET NULL, SET DEFAULT или NO ACTION, как описывалось в главе 11, "Целостность данных"), которое определяет действие, предпринимаемое при удалении строки родительской таблицы;
- необязательное правило обновления для данного отношения (эти правила описаны в главе 11, "Целостность данных"), которое определяет действие, предпринимаемое при обновлении первичного ключа в строке родительской таблицы;
- необязательное условие на значения, которое ограничивает данные в таблице так, чтобы они отвечали определенному критерию отбора.

Ниже приводится расширенная инструкция CREATE TABLE для таблицы ORDERS, в которую входит определение первичного ключа и трех внешних ключей таблицы.

Определение таблицы ORDERS с первичным и внешними ключами.

```
CREATE TABLE ORDERS
  (ORDER_NUM INTEGER       NOT NULL,
  ORDER_DATE DATE          NOT NULL,
        CUST INTEGER       NOT NULL,
         REP INTEGER,
         MFR CHAR(3)       NOT NULL,
     PRODUCT CHAR(5)       NOT NULL,
         QTY INTEGER       NOT NULL,
      AMOUNT DECIMAL(9,2)  NOT NULL,
```

```
PRIMARY KEY (ORDER_NUM),
 CONSTRAINT PLACEDBY
FOREIGN KEY (CUST)
 REFERENCES CUSTOMERS
  ON DELETE CASCADE,
 CONSTRAINT TAKENBY
FOREIGN KEY (REP)
 REFERENCES SALESREPS
  ON DELETE SET NULL,
 CONSTRAINT ISFOR
FOREIGN KEY (MFR, PRODUCT)
 REFERENCES PRODUCTS
  ON DELETE RESTRICT);
```

На рис. 13.2 изображены три созданные этой инструкцией связи с присвоенными им именами. В общем случае связи желательно давать имя, поскольку оно помогает лучше понять, какая именно связь создана внешним ключом. Например, каждый заказ делается клиентом, идентификатор которого находится в столбце CUST таблицы ORDERS. Связь с этим столбцом получила имя PLACEDBY (кем сделан).

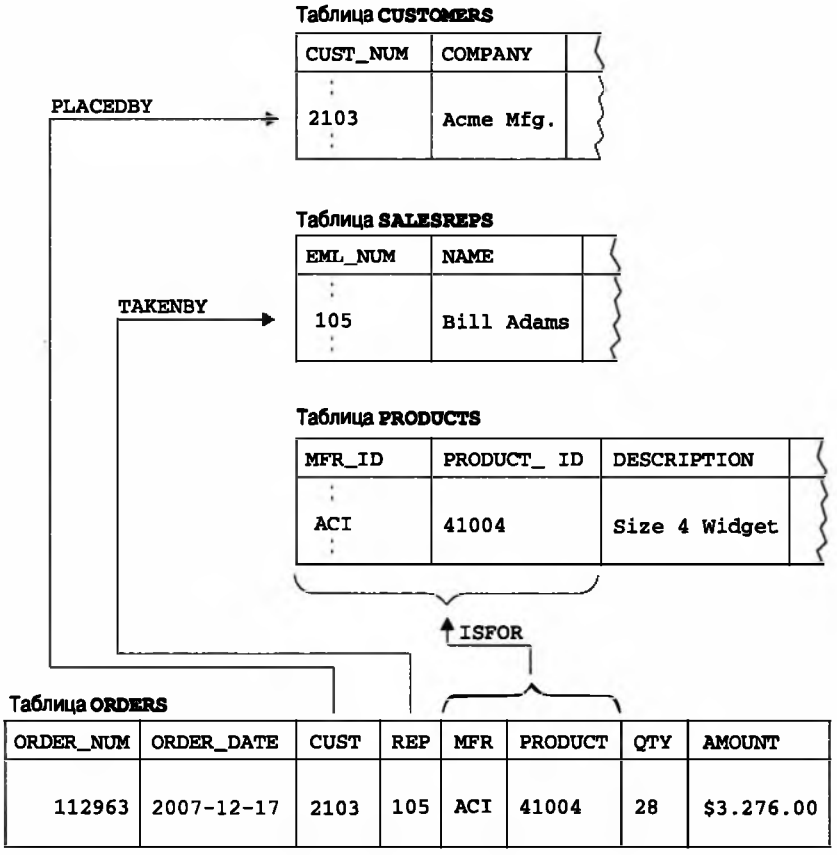


Рис. 13.2. Имена связей в инструкции CREATE TABLE

Когда СУБД выполняет инструкцию CREATE TABLE, она сравнивает определение каждого внешнего ключа с определениями связанных таблиц. СУБД проверяет, соответствуют ли друг другу внешний ключ и первичный ключ в связанных таблицах как по числу столбцов, так и по типу данных. Для того чтобы такая проверка была возможна, связанная таблица уже должна быть определена.

Заметим, что предложение FOREIGN KEY также определяет правила удаления и обновления для создаваемого им отношения "предок-потомок". Правила удаления и обновления, а также действия, которые они могут запускать, рассматриваются в главе 11, "Целостность данных". Если правила явно не указаны, СУБД использует правила по умолчанию (NO ACTION).

Если две или более таблиц образуют ссылочный цикл (как таблицы OFFICES и SALESREPS), то для первой создаваемой таблицы невозможно определить внешний ключ, так как связанная с нею таблица еще не существует. СУБД откажется выполнять инструкцию CREATE TABLE, выдав сообщение о том, что в определении таблицы присутствует ссылка на несуществующую таблицу. В этом случае необходимо создать таблицу без определения внешнего ключа и добавить данное определение позже с помощью инструкции ALTER TABLE. (В стандарте SQL и нескольких ведущих СУБД предлагается иной подход к решению данной проблемы — инструкция CREATE SCHEMA, с помощью которой одновременно создается все множество таблиц. Эта инструкция, а также объекты базы данных, входящие в состав схемы в SQL, описываются позже в данной главе.)

Условия уникальности

Стандарт SQL определяет, что условия уникальности также задаются в инструкции CREATE TABLE, с применением предложения UNIQUE, показанного на рис. 13.1. Вот как выглядит инструкция CREATE TABLE для таблицы OFFICES с включенным в нее условием уникальности для столбца CITY.

Определение таблицы OFFICES с условием уникальности.

```
CREATE TABLE OFFICES
     (OFFICE INTEGER       NOT NULL,
        CITY VARCHAR(15)   NOT NULL,
      REGION VARCHAR(10)   NOT NULL,
         MGR INTEGER,
      TARGET DECIMAL(9,2),
       SALES DECIMAL(9,2)  NOT NULL,
 PRIMARY KEY (OFFICE),
  CONSTRAINT HASMGR
 FOREIGN KEY (MGR)
  REFERENCES SALESREPS
   ON DELETE SET NULL,
      UNIQUE (CITY));
```

Если первичный или внешний ключ включает в себя только один столбец либо если условие уникальности или условие на значение касаются одного столбца, стандарт ANSI/ISO разрешает использовать сокращенную форму ограничения, при которой оно просто добавляется в конец определения столбца, как показано в следующем примере.

Определение таблицы OFFICES с условием уникальности (синтаксис ANSI/ISO).

```
CREATE TABLE OFFICES
     (OFFICE INTEGER      NOT NULL  PRIMARY KEY,
        CITY VARCHAR(15)  NOT NULL  UNIQUE,
      REGION VARCHAR(10)  NOT NULL,
         MGR INTEGER      REFERENCES SALESREPS,
      TARGET DECIMAL(9,2),
       SALES DECIMAL(9,2) NOT NULL);
```

Такой синтаксис поддерживается в ряде ведущих СУБД, в частности, в SQL Server, Informix, Sybase и DB2.

Ограничения на значения столбцов

Еще одна возможность обеспечения целостности данных в SQL, ограничение CHECK (которое уже рассматривалось в главе 11, "Целостность данных"), также задается в инструкции CREATE TABLE. Оно содержит условие на значение (идентичное условию отбора в запросе на выборку), проверяемое всякий раз при попытке модификации содержимого таблицы (с помощью инструкций INSERT, UPDATE или DELETE). Если после модификации условие остается истинным, такое изменение допускается; в противном случае СУБД отвергает изменения и выдает сообщение об ошибке. Ниже приведена инструкция CREATE TABLE, создающая таблицу OFFICES с простым ограничением на значение столбца TARGET, позволяющим гарантировать, что плановый объем продаж офиса всегда будет больше $0.00.

Определение таблицы OFFICES с ограничением на значение столбца TARGET.

```
CREATE TABLE OFFICES
     (OFFICE INTEGER      NOT NULL,
        CITY VARCHAR(15)  NOT NULL,
      REGION VARCHAR(10)  NOT NULL,
         MGR INTEGER,
      TARGET DECIMAL(9,2),
       SALES DECIMAL(9,2) NOT NULL,
 PRIMARY KEY (OFFICE),
  CONSTRAINT HASMGR
 FOREIGN KEY (MGR)
  REFERENCES SALESREPS
   ON DELETE SET NULL,
      CHECK (TARGET >= 0.00));
```

Дополнительно можно задать необязательное имя ограничения, которое будет отображаться в сообщениях об ошибках, выдаваемых СУБД при нарушении данного ограничения. Вот пример более сложного ограничения для таблицы SALESREPS, требующего соблюдения правила "служащим, принятым на работу после 1 января 2006 года, нельзя назначать план больше чем $300000". Этому ограничению назначено имя QUOTA_CAP.

```
CREATE TABLE SALESREPS
   (EMPL_NUM INTEGER      NOT NULL,
        NAME VARCHAR (15) NOT NULL,
          .
          .
          .
```

Удаление таблицы (DROP TABLE)

С течением времени структура базы данных изменяется. Для представления новых объектов создаются новые таблицы, а некоторые старые таблицы становятся ненужными. Эти ненужные таблицы можно удалить из базы данных инструкцией DROP TABLE, показанной на рис. 13.3.

Синтаксическая диаграмма инструкции DROP TABLE

Инструкция содержит имя удаляемой таблицы. Обычно пользователь удаляет одну из своих собственных таблиц и указывает в инструкции неквалифицированное имя таблицы. Имея соответствующее разрешение, можно удалить и таблицу другого пользователя, но в этом случае необходимо указать полное имя таблицы. Вот несколько примеров инструкции DROP TABLE.

Таблица CUSTOMERS была заменена двумя новыми таблицами — CUST_INFO и ACCOUNT_INFO — и больше не нужна.

```
DROP TABLE CUSTOMERS;
```

Сэм дает вам разрешение удалить его таблицу BIRTHDAYS.

```
DROP TABLE SAM.BIRTHDAYS;
```

Когда инструкция DROP TABLE удаляет из базы данных таблицу, ее определение и все содержимое теряются. Восстановить данные невозможно, и, чтобы повторно создать определение таблицы, следует использовать новую инструкцию CREATE TABLE. (Однако в настоящее время Oracle позволяет восстанавливать удаленные таблицы из Корзины, и другие производители могут последовать этому примеру.) Так как выполнение инструкции DROP TABLE может привести к серьезным последствиям, пользоваться ею следует крайне осторожно!

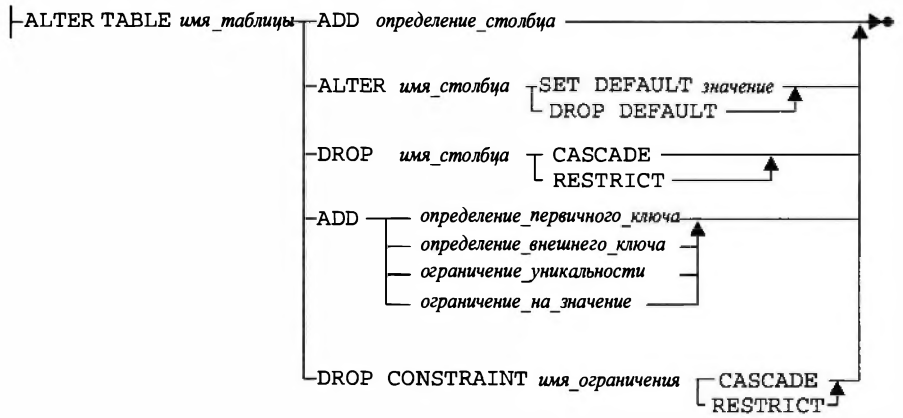
Стандарт SQL требует, чтобы инструкция DROP TABLE включала в себя либо параметр CASCADE, либо RESTRICT, которые определяют, как влияет удаление таблицы на другие объекты базы данных (например, представления, рассматриваемые в главе 14, "Представления"), зависящие от этой таблицы. Если задан параметр RESTRICT и в базе данных имеются объекты, которые содержат ссылку на удаляемую таблицу, то выполнение инструкции DROP TABLE закончится неуспешно. В большинстве коммерческих СУБД допускается применение инструкции DROP TABLE без каких-либо параметров, однако имеются и исключения.

- MySQL допускает в целях совместимости применение RESTRICT и CASCADE, но по крайней мере до версии 5.0 они не оказывали никакого действия.
- Oracle по умолчанию использует режим RESTRICT (который не может быть указан явно) и требует применения ключевых слов CASCADE CONSTRAINTS вместо CASCADE.
- SQL Server и DB2 не поддерживают RESTRICT и CASCADE.

Изменение определения таблицы (ALTER TABLE)

В процессе работы с таблицей у пользователя часто возникает необходимость добавить в таблицу некоторую информацию. Например, в учебной базе данных может потребоваться выполнить следующее:

- добавить в каждую строку таблицы CUSTOMERS имя и номер телефона служащего компании-клиента, через которого поддерживается контакт, если необходимо использовать эту таблицу для связи с клиентами;
- добавить в таблицу PRODUCTS столбец с указанием минимального количества на складе, чтобы база данных могла автоматически предупреждать о том, что запас какого-либо товара стал меньше допустимого предела;
- сделать столбец REGION в таблице OFFICES внешним ключом для вновь созданной таблицы REGIONS, первичным ключом которой является название региона;
- удалить определение внешнего ключа для столбца CUST таблицы ORDERS, связывающего ее с таблицей CUSTOMERS, и заменить его определениями двух внешних ключей, связывающих столбец CUST с двумя вновь созданными таблицами CUST_INFO и ACCOUNT_INFO.



Синтаксическая диаграмма инструкции ALTER TABLE

Эти и другие изменения можно осуществить с помощью инструкции ALTER TABLE, синтаксическая диаграмма которой изображена на рис. 13.4. Данная инструкция, как и DROP TABLE, обычно применяется пользователем по отношению к своим собственным таблицам. При наличии соответствующих прав и используя полное имя таблицы, можно изменять таблицы других пользователей. Как видно из рисунка, с помощью инструкции ALTER TABLE можно сделать следующее:

- добавить в таблицу определение столбца;
- удалить столбец из таблицы;
- изменить значение по умолчанию для какого-либо столбца;

- добавить или удалить первичный ключ таблицы;
- добавить или удалить внешний ключ таблицы;
- добавить или удалить условие уникальности;
- добавить или удалить условие на значение.

Добавление столбца

Чаще всего инструкция ALTER TABLE применяется для добавления столбца в существующую таблицу. Предложение с определением столбца в инструкции ALTER TABLE имеет точно такой же вид, как и в инструкции CREATE TABLE, и выполняет ту же функцию. Новое определение добавляется в конец определений столбцов таблицы, и в последующих запросах новый столбец будет крайним справа. СУБД обычно предполагает, что новый столбец во всех существующих строках содержит значения NULL. Если столбец объявлен как NOT NULL и со значением по умолчанию, то СУБД считает, что все его значения — значения по умолчанию.

Обратите внимание на то, что нельзя объявлять столбец просто как NOT NULL, поскольку СУБД подставляла бы в существующие строки значения NULL, нарушая тем самым заданное условие. (В действительности, когда вы добавляете новый столбец, СУБД может заносить во все существующие строки нового столбца значения NULL или значения по умолчанию. Некоторые СУБД обнаруживают тот факт, что строка слишком коротка для нового определения таблицы, только при выборке этой строки пользователем и расширяют ее значениями NULL (или значениями по умолчанию) непосредственно перед выводом на экран или передачей в программу пользователя.)

Ниже даны примеры инструкций ALTER TABLE, добавляющих новые столбцы.

Добавить контактный телефон и имя служащего компании-клиента в таблицу CUSTOMERS.

```
ALTER TABLE CUSTOMERS
  ADD CONTACT_NAME VARCHAR(30);

ALTER TABLE CUSTOMERS
  ADD CONTACT_PHONE CHAR(10);
```

Добавить в таблицу PRODUCTS столбец минимального количества товара на складе.

```
ALTER TABLE PRODUCTS
  ADD MIN_QTY INTEGER NOT NULL DEFAULT 0;
```

В первом примере новые столбцы будут иметь значения NULL для существующих клиентов. Во втором примере столбец MIN_QTY для существующих товаров будет содержать нули (0), что вполне уместно.

Когда инструкция ALTER TABLE впервые появилась в реализациях SQL, единственными структурными элементами таблиц были определения столбцов, поэтому было понятно, что означает предложение ADD этой инструкции. Со временем таблицы стали включать определения первичных и внешних ключей и прочих ограничений, а в предложении ADD стал указываться тип добавляемого ограничения. В целях унификации стандарт SQL позволяет добавлять после ключевых слов ADD слово COLUMN для явного указания на то, что создается столбец. С учетом этого, предыдущий пример можно записать так.

Добавить в таблицу PRODUCTS столбец минимального количества товара на складе.

```
ALTER TABLE PRODUCTS
  ADD COLUMN MIN_QTY INTEGER NOT NULL DEFAULT 0;
```

Удаление столбца

С помощью инструкции ALTER TABLE можно удалить из существующей таблицы один или несколько столбцов, если в них больше нет необходимости. Вот пример удаления столбца HIRE_DATE из таблицы SALESREPS.

Удалить столбец из таблицы SALESREPS.

```
ALTER TABLE SALESREPS
 DROP HIRE_DATE;
```

Стандарт SQL требует, чтобы одна инструкция ALTER TABLE использовалась для удаления только одного столбца, но в ряде ведущих СУБД такое ограничение снято.

Следует учитывать, что операция удаления столбца вызывает те же проблемы целостности данных, которые были описаны в главе 11, "Целостность данных", на примере обновления таблиц. Например, при удалении столбца, являющегося первичным ключом в каком-либо отношении, связанные с ним внешние ключи становится недействительными. Похожая проблема возникает, когда удаляется столбец, участвующий в проверке ограничения на значение другого столбца. Те же проблемы возникают и в представлениях, основанных на удаляемом столбце.

Описанные проблемы в стандарте SQL решены так же, как и в случае инструкций DELETE и UPDATE, — с помощью *правила удаления* (которые в стандарте именуются *поведением при удалении*). Можно выбрать одно из двух правил.

- **RESTRICT**. Если с удаляемым столбцом связан какой-либо объект в базе данных (внешний ключ, ограничение и т.п.), инструкция ALTER TABLE завершится выдачей сообщения об ошибке и столбец не будет удален.
- **CASCADE**. Любой объект базы данных (внешний ключ, ограничение и т.п.), связанный с удаляемым столбцом, *также* будет удален.

Правило CASCADE может вызвать лавину изменений, поэтому применять его следует с осторожностью. Лучше указывать правило RESTRICT, а связанные внешние ключи или ограничения обрабатывать с помощью дополнительных инструкций типа ALTER или DROP.

Индексы (CREATE/DROP INDEX)

Одним из структурных элементов физической памяти, присутствующих в большинстве реляционных СУБД, является *индекс*. Индекс — это средство, обеспечивающее быстрый доступ к строкам таблицы на основе значений одного или нескольких столбцов. На рис. 13.6 изображена таблица PRODUCTS и два созданных для нее индекса. Один из индексов обеспечивает доступ к таблице на основе столбца DESCRIPTION. Другой обеспечивает доступ на основе первичного ключа таблицы, представляющего собой комбинацию столбцов MFR_ID и PRODUCT_ID.

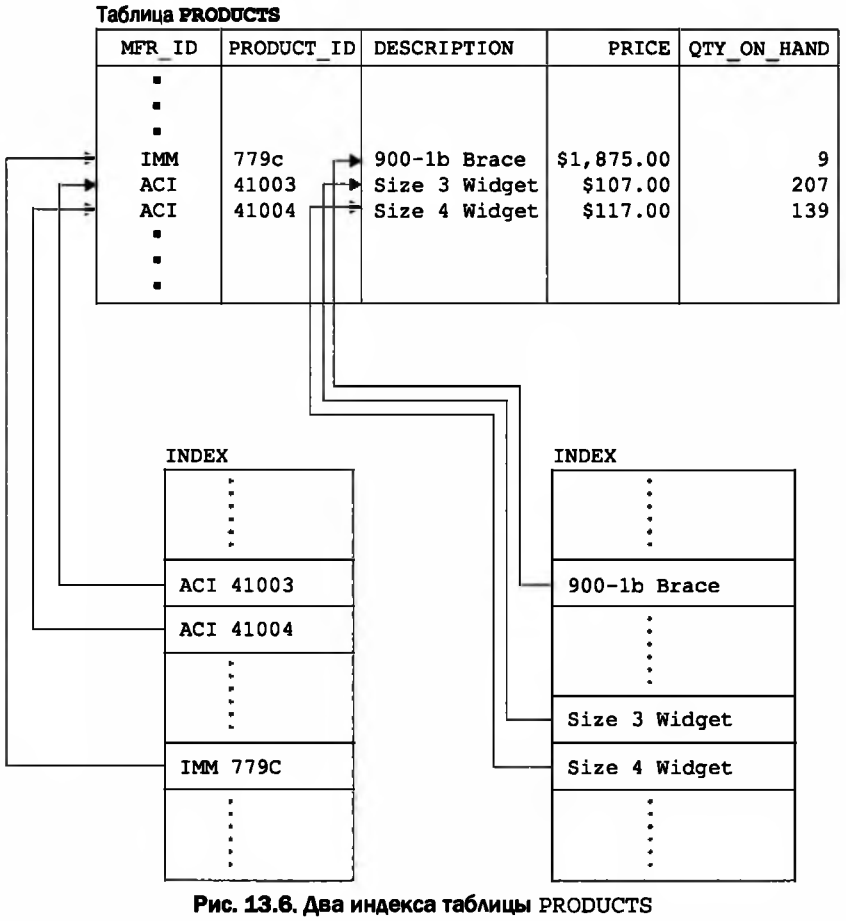


Рис. 13.6. Два индекса таблицы PRODUCTS

СУБД пользуется индексом так же, как вы пользуетесь предметным указателем книги. В индексе хранятся значения данных и указатели на строки, где эти данные встречаются. Данные в индексе располагаются в убывающем или возрастающем порядке, чтобы СУБД могла быстро найти требуемое значение. Затем по указателю СУБД может быстро найти строку, содержащую искомое значение.

Наличие или отсутствие индекса совершенно незаметно для пользователя, обращающегося к таблице. Рассмотрим, например, такую инструкцию SELECT.

Найти количество на складе и цену изделия "Size 4 Widget".

```
SELECT QTY_ON_HAND, PRICE
  FROM PRODUCTS
 WHERE DESCRIPTION = 'Size 4 Widget';
```

В инструкции ничего не говорится о том, имеется ли индекс для столбца DESCRIPTION или нет, и СУБД выполнит запрос в любом случае.

Если бы индекса для столбца DESCRIPTION не существовало, то СУБД была бы вынуждена выполнять запрос путем последовательного сканирования таблицы PRODUCTS, строка за строкой, просматривая в каждой строке столбец DESCRIPTION. Для получения гарантии того, что она нашла все строки, удовлетворяющие условию отбора, СУБД должна просмотреть *каждую* строку таблицы. Просмотр больших таблиц, содержащих миллионы строк, может занять минуты и даже часы.

Если для столбца DESCRIPTION имеется индекс, СУБД находит требуемые данные с гораздо меньшими усилиями. Она просматривает индекс, чтобы найти требуемое значение (изделие "Size 4 Widget"), а затем с помощью указателя находит требуемую строку (строки) таблицы. Поиск в индексе осуществляется достаточно быстро, так как индекс отсортирован и его строки очень короткие. Переход от индекса к строке (строкам) также происходит довольно быстро, поскольку в индексе содержится информация о том, где именно на диске располагается эта строка (строки).

Как видно из этого примера, преимущество индекса в том, что он в огромной степени ускоряет выполнение инструкций SQL с условиями отбора, имеющими ссылки на индексный столбец (столбцы). К недостаткам индекса относится то, что, во-первых, он занимает на диске дополнительное место и, во-вторых, индекс необходимо обновлять каждый раз, когда в таблицу добавляется строка или обновляется индексный столбец таблицы. Это требует дополнительных затрат на выполнение инструкций INSERT и UPDATE, которые обращаются к данной таблице.

В общем-то, полезно создавать индекс лишь для тех столбцов, которые часто используются в условиях отбора. Индексы удобны также в тех случаях, когда инструкции SELECT обращаются к таблице гораздо чаще, чем инструкции INSERT и UPDATE. Большинство СУБД *всегда* создает индекс для первичного ключа таблицы, так как ожидает, что доступ к таблице чаще всего будет осуществляться через первичный ключ. Кроме того, индекс первичного ключа помогает СУБД быстро найти дублирующиеся строки при вставке новых строк в таблицу.

Большинство СУБД также автоматически создает индекс для всех столбцов (или комбинаций столбцов), указанных в ограничении уникальности. Как и в случае первичного ключа, СУБД должна при каждой вставке новой строки или обновлении существующей проверять значение такого столбца — нет ли уже такого значения в таблице. Без индекса для такого столбца (столбцов) при проверке СУБД пришлось бы выполнять последовательное сканирование всех строк таблицы. При наличии индекса СУБД может просто воспользоваться индексом для поиска строки (если таковая существует) с интересующим нас значением, что гораздо быстрее последовательного поиска.

В учебной базе данных было бы уместно создать дополнительные индексы на основе следующих столбцов.

- COMPANY таблицы CUSTOMERS, если данные из таблицы часто извлекаются по названию компании.
- NAME таблицы SALESREPS, если данные о служащих часто извлекаются по имени служащего.
- REP таблицы ORDERS, если данные о заказах часто извлекаются с использованием имени служащего, принявшего их.
- CUST таблицы ORDERS, если данные о заказах часто извлекаются с использованием имени клиента, сделавшего их.
- MFR и PRODUCT таблицы ORDERS, если данные о заказах часто извлекаются по названию заказанного товара.

В стандарте SQL ничего не говорится об индексах и о том, как их создавать. Они относятся к "деталям реализации", выходящим за рамки ядра языка SQL. Тем не менее индексы весьма важны для обеспечения требуемой производительности любой серьезной базы данных уровня предприятия.

На практике в большинстве популярных СУБД (включая Oracle, Microsoft SQL Server, MySQL, Informix, Sybase и DB2) для создания индекса используется та или иная форма инструкции CREATE INDEX (рис. 13.7). В инструкции индексу назначается имя и указывается таблица, для которой он создается. Задается также индексируемый столбец (столбцы) и порядок его сортировки (по возрастанию или убыванию). Представленная на рис. 13.7 версия инструкции CREATE INDEX для СУБД DB2 — одна из наиболее простых. В ней можно использовать ключевое слово UNIQUE для указания того, что индексный столбец (столбцы) должен содержать уникальные значения в каждой строке таблицы.

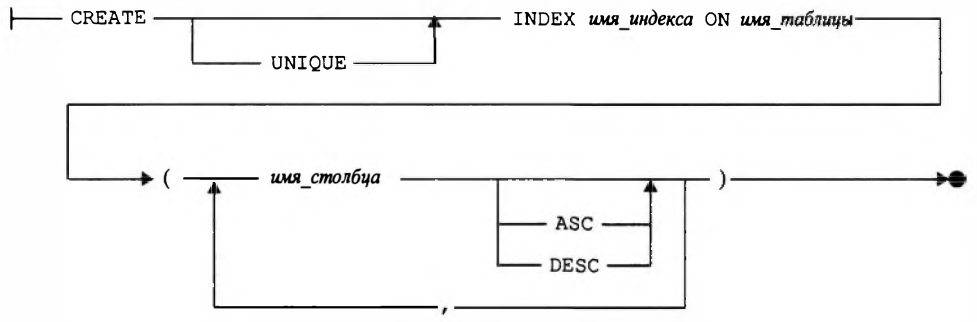


Рис. 13.7. Синтаксическая диаграмма базовой инструкции CREATE INDEX

Ниже дан пример инструкции CREATE INDEX, которая создает индекс для таблицы ORDERS на основе столбцов MFR и PRODUCT и содержит требование уникальности для комбинаций этих столбцов.

Создать индекс для таблицы OFFICES.

```
CREATE UNIQUE INDEX OFC_MGR_IDX
    ON OFFICES (MGR);
```

Создать индекс для таблицы ORDERS.

```
CREATE UNIQUE INDEX ORD_PROD_IDX
    ON ORDERS (MFR, PRODUCT);
```

В большинстве СУБД инструкция CREATE INDEX содержит дополнительные, специфические для каждой СУБД предложения, в которых задаются местоположение индекса на диске и рабочие параметры, такие как размер страниц индекса; сколько свободного пространства (в процентах) должен резервировать индекс для дополнительных строк; тип создаваемого индекса; должна ли выполняться кластеризация (т.е. должны ли записи на диске физически располагаться в том же порядке, что и в самом индексе) и т.д.

Некоторые СУБД поддерживают два или более различных типов индексов, оптимизированных для разных видов обращений к базе данных. Например, индекс *B-tree* использует древовидную структуру записей и блоков (групп записей) индекса для организации значений содержащихся в нем данных в возрастающем или убывающем порядке. Этот тип индекса (используемый по умолчанию почти во всех СУБД) обеспечивает эффективный поиск конкретного значения (или диапазона значений), требующийся при выполнении сравнения или проверке на принадлежность диапазону (BETWEEN).

Другой тип индекса, *хеш*-индекс, использует метод рандомизации для размещения всех возможных значений данных в небольшом количестве блоков индекса. Например, при наличии 10 миллионов возможных значений данных может оказаться разумным индекс с 500 хеш-блоками. Поскольку заданное значение всегда попадает в один и тот же блок, СУБД может выполнять поиск, находя соответствующий значению блок и выполняя поиск в нем. При наличии 500 блоков количество сканируемых при поиске элементов уменьшается в среднем в 500 раз. Это делает хеш-индексы очень быстрым средством при поиске точного значения. Однако распределение значений по блокам не сохраняет порядок данных, так что хеш-индекс нельзя использовать для работы с неравенствами или диапазонами.

Есть и иные типы индексов, подходящие для решения тех или иных задач базы данных, в том числе следующие.

- **T-tree** представляет собой вариацию индекса B-tree, оптимизированную для баз данных, работающих в оперативной памяти.
- **Битовая карта** полезна при относительно малом количестве возможных значений данных.
- **Индексная таблица** представляет собой относительно новую методику, когда в индексе хранится вся таблица целиком. Эта технология хорошо работает с таблицами с малым количеством столбцов, не входящих в первичный ключ, такими как таблицы поиска кодов, в которых обычно имеется только код (например, номер отдела) и описание (например, название этого отдела).

Если СУБД поддерживает различные типы индексов, инструкция CREATE INDEX не только определяет и создает индекс, но и указывает его тип.