Глава 7

Ограничения и триггеры

В этой главе мы сосредоточим внимание на тех средствах SQL, которые позволяют создавать так называемые "активные" элементы. *Активный элемент* (active element) — это выражение или команда, которые, будучи объявленными и сохраненными в базе данных, выполняются по мере необходимости в определенные моменты времени и в связи с наступлением событий, подобных вставке кортежа в то или иное отношение или изменению состояния базы данных, при которых некоторое логическое условие приобретает значение `TRUE`.

Одна из серьезных проблем, с которыми сталкиваются разработчики приложений баз данных, связана с тем, что при обновлении элементов информации последние оказываются неверными или вступают в противоречие с другими элементами. Зачастую подобные ошибки возникают в процессе "ручного" ввода данных или преобразования их из одной формы представления в другую. Наиболее прямолинейный способ предотвращения возможности попадания в базу данных новых кортежей, содержащих ущербную информацию, или искажения существующей информации состоит в том, чтобы прикладная программа сама "заботилась" о корректности данных, предваряя каждую операцию вставки, удаления или обновления соответствующей проверкой. К сожалению, зачастую критерии, которым необходимо подчиняться, весьма сложны; помимо того, одни и те же операции проверки обычно следует повторять вновь и вновь.

Теперь о приятном: SQL предлагает множество способов выражения ограничений целостности данных и представления их в виде неотъемлемой части схемы базы данных. В этой главе мы рассмотрим основные из них. Вначале будет рассказано об ограничениях, предусматривающих создание *первичных ключей* (primary keys), содержащих в своем составе один или несколько атрибутов отношения. Затем мы познакомим вас с такими формами выражения требований ссылочной целостности, которые связаны с определением так называемых *внешних ключей* (foreign keys), предполагающих, что если некоторые компоненты атрибутов одного отношения (например, `presC#` таблицы `Studio`) содержат определенные значения, те же значения должны присутствовать в компонентах атрибутов другого отношения (соответственно `cert#` таблицы `MovieExec`).

Далее мы покажем, как определять ограничения уровня отдельного атрибута, кортежа и отношения в целом, а также ограничения *"общего вида"* (assertions), и приведем сведения о *триггерах* (triggers) — разновидности активных элементов, вызываемых системой в ответ на определенные события, такие как вставка кортежа в заданное отношение.

7.1. Ключи отношений

Возможно, наиболее важным является такое ограничение, которое позволяет сформулировать утверждение о том, что некоторое подмножество атрибутов определенного отношения формирует ключ этого отношения. Если подмножество *S* атрибутов является ключом отношения *R*, любые два кортежа *R* должны различаться в значениях компонентов хотя бы одного из атрибутов *S*. Это правило остается в силе и в применении к кортежам-дубликатам: если схема отношения *R* содержит объявление ключа, в *R* не могут быть помещены повторяющиеся кортежи.

Ограничение ключа, как и многие другие виды ограничений, формулируется в контексте SQL-команды `CREATE TABLE`. Существуют два схожих способа определения ключей: первый предусматривает использование пары служебных слов `PRIMARY KEY`, а второй — служебного слова `UNIQUE`. Для каждого отношения может быть определен только один *первичный ключ* (primary key), но любое количество *"уникальных"* (unique) ключей.

Термин "ключ" употребляется в SQL и для обозначения ограничений ссылочной целостности, называемых *ограничениями внешних ключей* (foreign-key constraints), которые предусматривают, что если некоторые значения присутствуют в определенных компонентах одного отношения, они должны наличествовать и в компонентах первичного ключа некоторого кортежа другого отношения (за подробностями обращайтесь к разделу 7.1.4 на с. 321).

7.1.1. Объявление первичного ключа

Отношение может содержать только один *первичный ключ* (primary key). Существуют два способа объявления первичного ключа в рамках выражения `CREATE TABLE`:

1) дополнить особым признаком определение атрибута в списке атрибутов реляционной схемы;
2) добавить в список элементов объявления схемы (до сих пор мы говорили о том, что такой список может содержать только определения атрибутов) специальное предложение, свидетельствующее о том, что атрибут или несколько атрибутов должны образовать первичный ключ.

При использовании первого способа следует присоединить к определению атрибута в списке элементов схемы отношения пару служебных слов `PRIMARY KEY`; второй способ предоставляет возможность вставки в этот список предложения `PRIMARY KEY` со списком ключевых элементов, заключенным в круглые скобки. Если ключ должен состоять из нескольких атрибутов, для его определения можно применять только второй способ.

Результат объявления подмножества *S* атрибутов в качестве ключа отношения *R* следует трактовать так:

1) два произвольных кортежа *R* не могут совпадать в значениях компонентов всех ключевых атрибутов одновременно; любая попытка вставки нового кортежа или обновления содержимого существующего кортежа, приводящая к нарушению этого правила, будет отвергнута системой как недопустимая операция;
2) компоненты ключевых атрибутов *S* не могут содержать значения `NULL`.

Пример 7.1. Обратимся к схеме отношения `MovieStar` ("актеры") (см. рис. 6.16 на с. 298). Ключом отношения является атрибут `name` ("имя"). Этот факт следует отразить в инструкции объявления отношения. Дополненный вариант объявления приведен на рис. 7.1.

Можно воспользоваться и таким вариантом определения первичного ключа, который предполагает включение в список элементов схемы отношения `MovieStar` отдельного предложения `PRIMARY KEY (name)`. Соответствующая версия объявления показана на рис. 7.2.

```
1)  CREATE TABLE MovieStar (
2)      name CHAR(30) PRIMARY KEY,
3)      address VARCHAR(255),
4)      gender CHAR(1),
5)      birthdate DATE
    );
```

Рис. 7.1. Придание атрибуту `name` *статуса первичного ключа*

```
1)  CREATE TABLE MovieStar (
2)      name CHAR(30),
3)      address VARCHAR(255),
4)      gender CHAR(1),
5)      birthdate DATE,
6)      PRIMARY KEY (name)
    );
```

Рис. 7.2. Объявление первичного ключа в виде отдельного предложения

❑

В примере 7.1 приемлемой можно считать любую из двух альтернативных форм определения первичного ключа, поскольку тот состоит только из одного атрибута. Если же необходимо создать ключ на основе нескольких атрибутов, следует пользоваться вторым вариантом, предусматривающим применение отдельного предложения (см. рис. 7.2). Например, при объявлении схемы отношения `Movie`, ключом которого (мы неоднократно говорили об этом прежде) является пара атрибутов `title` и `year`, после списка атрибутов нам пришлось бы включить строку вида

```
PRIMARY KEY (title, year)
```

7.1.2. Объявление ключа `UNIQUE`

Еще один способ объявления ключа отношения связан с использованием служебного слова `UNIQUE`. Слово `UNIQUE` можно употреблять в тех же "местах" конструкции `CREATE TABLE`, что и выражение `PRIMARY KEY`, т.е. после имен атрибута и его типа в списке атрибутов либо в виде отдельного предложения. Смысл объявления со словом `UNIQUE` почти аналогичен значению выражения `PRIMARY KEY`; существуют, однако, два исключения, указанные ниже.

1. Отношение может обладать несколькими ключами `UNIQUE`, но только одним первичным ключом.
2. В то время как компонентам атрибутов ключа `PRIMARY KEY` запрещено присваивать значения `NULL`, это разрешено для компонентов ключа `UNIQUE`. Кроме того, правило, запрещающее совпадение содержимого компонентов ключевых атрибутов нескольких кортежей, в ключе `UNIQUE` может быть нарушено, если этим компонентам присвоены значения `NULL` (допустима даже такая ситуация, когда все компоненты ключа `UNIQUE` нескольких кортежей содержат значения `NULL`).

Разработчику СУБД предоставлена возможность задавать дополнительные условия, касающиеся различий между ключами `PRIMARY KEY` и `UNIQUE`. Например, в конкретной реализации системы может быть предусмотрено автоматическое построение индекса для ключа,

объявленного в качестве первичного (даже в том случае, если такой ключ состоит из нескольких атрибутов), но с учетом того, что иные индексы, если таковые необходимы, должны создаваться дополнительно. В другом случае система способна сортировать отношение по значениям компонентов ключевых атрибутов и далее поддерживать его именно в таком виде.

Пример 7.2. Строку 2 рис. 7.1 допустимо изменить так:

```
2)      name CHAR(30) UNIQUE,
```

Помимо того, строка 3, например, может быть представлена в следующем виде:

```
3)      address VARCHAR(255) UNIQUE,
```

если нам кажется, что адреса актеров обязаны быть уникальными (хотя такая "догадка", вероятно, сомнительна). Аналогичным образом при необходимости можно изменить и строку 6 рис. 7.2:

```
6)      PRIMARY KEY (name)
```

❑

7.1.3. Соблюдение ограничений ключей

Давайте вновь обратимся к теме раздела 6.6.5 (см. с. 299), где, рассказывая об индексах, мы говорили о том, что хотя необходимость в их создании не регламентируется ни одним стандартом SQL, каждая реализация СУБД предусматривает те или иные средства конструирования индексов как части определения схемы базы данных. Если система предлагает создать индекс для первичного ключа отношения, это естественно и объяснимо — вполне вероятно, что запросы, предполагающие задание значений ключевых атрибутов, окажутся наиболее "популярными". Иногда целесообразно также создать индексы для других атрибутов, которые объявлены как `UNIQUE`. Если предложение `WHERE` запроса содержит условие равенства ключевого атрибута определенному значению (например, `name = 'Audrey Hepburn'` для отношения `MovieStar`, упомянутого в примере 7.1 на с. 318), при наличии подобного индекса система сможет отыскать подходящий кортеж чрезвычайно быстро, не просматривая все отношение целиком.

Во многих реализациях SQL предлагается возможность включения в команду создания индекса служебного слова `UNIQUE`, что позволяет придать атрибуту статус ключа и одновременно сконструировать индекс для этого атрибута. Например, выражение

```
CREATE UNIQUE INDEX YearIndex ON Movie(year);
```

выполняет не только функцию создания индекса, о чем говорилось в разделе 6.6.5, но и содержит объявление ограничения уникальности значений атрибута `year` отношения `Movie` (разумеется, в данном случае это не вполне правильно, поскольку уникальными должны быть *пары* значений `title` и `year`, но сейчас мы преследуем иную цель — продемонстрировать синтаксис и обсудить семантические особенности такой команды).

Рассмотрим вкратце, каким образом система следит за соблюдением ограничения ключа. Вообще говоря, условие такого ограничения подлежит проверке перед каждой попыткой изменения содержимого базы данных. Впрочем, понятно, что ограничение ключа отношения R подвергается опасности нарушения только в том случае, если R каким-либо образом модифицируется. Если говорить более точно, операция удаления кортежа из R не способна привести к нарушению условия ограничения — этим чреваты команды вставки новых кортежей и обновления существующих кортежей. Поэтому обычно SQL-системы осуществляют проверку условия ограничения ключа только при обработке команд `INSERT` и `UPDATE`.

Наличие индекса для атрибутов, объявленных как ключевые, жизненно важно в том смысле, что система может эффективно следить за соблюдением ограничений ключей. Всякий раз, когда выполняется вставка кортежа или обновление значений ключевых атрибутов

существующего кортежа, система использует индекс для быстрого просмотра содержимого отношения в поисках кортежа с теми же значениями ключевых атрибутов. Если такой кортеж уже существует, система обязана предотвратить модификацию отношения.

Если индекс для ключевых атрибутов не создан, соблюдение ограничения ключа все еще необходимо и возможно. Процедуру поиска удастся упростить, отсортировав отношение по ключевым атрибутам. Однако в отсутствие какой бы то ни было отправной точки поиска при попытке отыскания кортежа с заданными значениями компонентов системе придется проверить все отношение. Подобный процесс настолько трудоемок, что задача модификации отношения с большим количеством кортежей становится попросту неразрешимой.

7.1.4. Объявление внешнего ключа

Ограничения, относящиеся ко второй важнейшей разновидности, призваны гарантировать, что значения заданных атрибутов отношения имеют строго определенный смысл. Например, атрибут `presC#` отношения `Studio` должен содержать ссылку на элемент информации с описанием конкретного руководящего лица. Рассматриваемое ограничение ссылочной целостности предполагает, что если в компоненте `presC#` некоторого кортежа отношения `Studio` содержится сертификационный номер *c*, этот номер должен принадлежать реальному лицу. Таким "реальным" лицом может быть только человек, сведения о котором представлены в отношении `MovieExec`, или, иными словами, в `MovieExec` должен существовать кортеж, компонент `cert#` которого содержит тот же номер *c*.

SQL позволяет объявить атрибут или несколько атрибутов отношения как *внешний ключ* (foreign key), *ссылающийся* (references) на некоторые атрибуты другого (или того же самого) отношения. Результат объявления внешнего ключа следует трактовать так.

1. Атрибуты, адресуемые внешним ключом, должны быть объявлены как `UNIQUE` или `PRIMARY KEY`. Если это требование не выполняется, внешний ключ не будет создан.
2. Значения атрибутов внешнего ключа должны присутствовать в компонентах адресуемых атрибутов некоторого кортежа. Если говорить более точно, положим, что внешний ключ F ссылается на множество атрибутов G определенного отношения. Положим также, что некоторый кортеж t отношения с внешним ключом обладает отличными от `NULL` значениями во всех компонентах атрибутов F; обозначим список этих значений как $t[F]$. Тогда в отношении, адресуемом внешним ключом, должен существовать некоторый кортеж s, который совпадает с $t[F]$ во всех атрибутах G, т.е. удовлетворяет условию $s[G] = t[F]$.

Как и в случае объявления первичного ключа, внешний ключ допускает две формы объявления.

1. Если внешний ключ содержит единственный атрибут, объявление последнего можно сопроводить предложением, свидетельствующим о том, что атрибут "ссылается" (`REFERENCES`) на определенный атрибут (обязанный быть ключом — первичным или "уникальным") некоторой таблицы:

   `REFERENCES` *<таблица>*`(`*<атрибут>*`)`
2. Другой способ объявления предполагает включение в список элементов команды `CREATE TABLE` отдельного предложения с утверждением о том, что некоторое подмножество атрибутов таблицы является внешним ключом (`FOREIGN KEY`), ссылающимся (`REFERENCES`) на множество ключевых атрибутов той же или другой таблицы:

   `FOREIGN KEY` *<атрибуты>* `REFERENCES` *<таблица>*`(`*<атрибуты>*`)`

Пример 7.3. Пусть необходимо объявить отношение

```
Studio(name, address, presC#),
```

обладающее первичным ключом name и внешним ключом presC#, ссылающимся на атрибут cert# отношения

```
MovieExec(name, address, cert#, netWorth).
```

Тот факт, что атрибут presC# ссылается на атрибут cert#, можно описать непосредственно в строке определения presC#:

```
CREATE TABLE Studio (
    name CHAR(30) PRIMARY KEY,
    address VARCHAR(255),
    presC# INT REFERENCES MovieExec(cert#)
);
```

Альтернативная форма объявления такова:

```
CREATE TABLE Studio (
    name CHAR(30) PRIMARY KEY,
    address VARCHAR(255),
    presC# INT,
    FOREIGN KEY (presC#) REFERENCES MovieExec(cert#)
);
```

Заметьте, что адресуемый атрибут cert# должен быть (впрочем, так оно и есть) ключом отношения MovieExec. Смысл любой из двух конструкций состоит в том, что любое значение, присваиваемое компоненту presC# кортежа Studio, должно присутствовать в компоненте cert# некоторого кортежа отношения MovieExec. Единственное исключение связано с присваиванием компоненту presC# значения NULL — требование наличия NULL в cert# не выдвигается (атрибут cert# и не может содержать NULL, поскольку является первичным ключом). ❑

7.1.5. Обеспечение ссылочной целостности

Теперь вы знаете, как объявлять внешние ключи, и осведомлены о том, что любая комбинация значений компонентов внешнего ключа, ни одно из которых не равно NULL, должна присутствовать и в компонентах соответствующих атрибутов отношения, адресуемого внешним ключом. Однако возникает закономерный вопрос, каким образом система может поддерживать подобные ограничения при осуществлении операций модификации базы данных. Разработчику базы данных предоставляется возможность выбора одной из альтернативных стратегий, рассмотренных ниже.

Стратегия по умолчанию: отмена операции

По умолчанию система всегда отвергает операцию модификации, нарушающую условие ограничения ссылочной целостности. Обратимся к примеру 7.3, в котором предполагается, что любое значение атрибута presC# отношения Studio должно присутствовать также в числе значений атрибута cert# отношения MovieExec. Каждое из перечисленных ниже действий будет отменено системой, т.е. вызовет исключение периода выполнения либо приведет к появлению сообщения об ошибке.

1. Попытка вставки (INSERT) в Studio нового кортежа, компонент presC# которого содержит значение, отличное от NULL и не совпадающее ни с одним значением компонентов cert# кортежей MovieExec. Операция вставки отвергается системой, и подобный кортеж не будет включен в состав отношения Studio.
2. Попытка присваивания (UPDATE) компоненту presC# любого из существующих кортежей Studio значения, отличного от NULL и не совпадающего ни с одним значением компонентов cert# кортежей MovieExec. Операция обновления отвергается системой, и содержимое компонента presC# кортежа отношения Studio не изменяется.

3. Попытка удаления (DELETE) из отношения MovieExec такого кортежа, значение компонента cert# которого совпадает с содержимым компонента presC# одного или нескольких кортежей отношения Studio. Операция удаления отменяется системой, и кортеж остается в составе MovieExec.
4. Попытка обновления (UPDATE) содержимого компонента cert# кортежа отношения MovieExec при условии, что прежнее значение cert# присутствует в компоненте presC# одного или нескольких кортежей отношения Studio. Операция обновления отменяется системой, и значение cert# кортежа MovieExec не изменяется.

Стратегия каскадной модификации

Другой подход к обработке операций (см. пп. 3, 4 выше) удаления и обновления кортежей отношения (такого как MovieExec), адресуемого внешним ключом, называют *стратегией каскадной модификации* (cascade strategy), в соответствии с которой аналогичные изменения вносятся и в таблицу, содержащую внешний ключ.

Если следовать стратегии каскадной модификации, при удалении (DELETE) из отношения MovieExec кортежа с информацией о каком-либо руководящем лице для поддержания ссылочной целостности необходимо изъять и кортежи отношения Studio, содержащие ссылки на компонент cert# удаляемого кортежа MovieExec. Операции обновления (UPDATE) данных обрабатываются аналогично: если значение компонента cert# некоторого кортежа MovieExec изменяется с c_1 на c_2 и существует кортеж Studio, содержащий в компоненте presC# значение c_1, система обязана изменить с c_1 на c_2 и значение компонента presC#.

Стратегия "присвоить значения NULL"

Наконец, руководствуясь еще одной возможной стратегией (ее кратко называют *set-null*) соблюдения условий ссылочной целостности, при удалении кортежа MovieExec или изменении содержимого компонента cert#, на который имеется ссылка, система должна присвоить значения NULL всем соответствующим компонентам presC# отношения Studio.

Система способна реагировать на две категории команд — DELETE и UPDATE, — нарушающих ограничения ссылочной целостности, независимо, и стратегия, которой ей надлежит придерживаться в одном и другом случае, определяется в момент объявления внешнего ключа: в конструкцию объявления вводятся предложения ON DELETE и/или ON UPDATE, сопровождаемые обозначением одного из двух вариантов действий — SET NULL или CASCADE.

Пример 7.4. Рассмотрим, каким образом следует изменить объявление отношения

```
Studio(name, address, presC#),
```

предложенное в примере 7.3 на с. 321, чтобы обеспечить обработку ситуаций, связанных с удалением и обновлением кортежей отношения

```
MovieExec(name, address, cert#, netWorth).
```

На рис. 7.3 показана первая из приведенных в примере 7.3 команд CREATE TABLE, дополненная предложениями ON DELETE и ON UPDATE. Строка 5 содержит инструкцию, в соответствии с которой при удалении кортежа MovieExec система обязана присвоить значения NULL всем компонентам presC# отношения Studio с тем же содержимым, что и компонент cert# удаляемого кортежа. Инструкция строки 6 свидетельствует, что при обновлении значения компонента cert# кортежа MovieExec системе следует выполнить аналогичные действия и для компонентов presC# отношения Studio, содержавших то же значение.

"Висящие" кортежи и стратегии модификации

Кортеж, значения атрибутов внешнего ключа которого отсутствуют в отношении, адресуемом этим ключом, называют *висящим* (dangling tuple). Напомним, что кортежи, не способные принять участие в соединении двух отношений, также называют "висящими". Обе ситуации весьма схожи. Если значение, на которое ссылается компонент внешнего ключа, отсутствует, кортеж с неверной ссылкой не будет соединен ни с одним кортежем адресуемого отношения.

Статус "висящих" приобретают те кортежи, которые нарушают условие ссылочной целостности для конкретного ограничения внешнего ключа.

- Стратегия, предлагаемая по умолчанию в случае удаления или обновления кортежей, на которые ссылаются компоненты внешнего ключа, предусматривает запрет на выполнение подобных операций, если и только если они приводят к возникновению висящих кортежей в отношении, содержащем внешний ключ.
- Стратегия каскадной модификации предполагает удаление или обновление висящих кортежей отношения, содержащего внешний ключ, в соответствии с теми изменениями, которые претерпевает отношение, адресуемое внешним ключом.
- Стратегия *set-null* предусматривает присваивание значений `NULL` компонентам внешнего ключа каждого из висящих кортежей.

```
1)  CREATE TABLE Studio (
2)      name CHAR(30) PRIMARY KEY,
3)      address VARCHAR(255),
4)      presC# INT REFERENCES MovieExec(cert#)
5)             ON DELETE SET NULL
6)             ON UPDATE CASCADE
    );
```

Рис. 7.3. Выбор стратегий соблюдения ограничения ссылочной целостности

В данном случае стратегия *set-null* предпочтительна для операций удаления, в то время как стратегия каскадной модификации — для операций обновления. Справедливо полагать, что если, например, президент киностудии уволен или ушел на пенсию (кортеж отношения `MovieExec` удален), студия продолжает существовать и некоторое время обходится без президента (значение компонента `presC#` кортежа отношения `Studio` равно `NULL`). Однако изменение сертификационного номера президента студии, вероятно, может быть связано только с какими-либо бюрократическими манипуляциями: студия продолжает работать, президент на месте, так что наиболее уместно подчиниться предложенному решению и внести соответствующее изменение и в содержимое компонента `presC#` кортежа отношения `Studio`. ❑

7.1.6. Отложенная проверка ограничений

Вновь обратимся к примеру 7.3 на с. 321, где речь идет о том, что атрибут `presC#` отношения `Studio` является внешним ключом, ссылающимся на атрибут `cert#` отношения `MovieExec`, и предположим, что небезызвестный Билл Клинтон, завершив карьеру президента Соединенных Штатов, решил потрудиться на киностудии "Redlight Studios" — естественно, на посту ее президента. При попытке выполнения операции вставки,

```
INSERT INTO Studio
VALUES('Redlight', 'New York', 23456);
```

возникает проблема. Причина состоит в том, что в MovieExec не существует кортежа, который содержал бы сертификационный номер 23456 (мы полагаем, что это личный номер Билла Клинтона), и поэтому ограничение внешнего ключа явно нарушается.

Одно из возможных решений состоит в том, чтобы для начала попробовать вставить кортеж с описанием студии "Redlight Studios", не вводя сертификационный номер ее президента:

```
INSERT INTO Studio(name, address)
VALUES('Redlight', 'New York');
```

Это позволит предотвратить нарушение ограничения, поскольку компоненту presC# кортежа с информацией о студии "Redlight Studios" будет присвоено значение NULL, а при наличии NULL в компонентах внешнего ключа проверка кортежей отношения, адресуемого внешним ключом, не выполняется. Однако, прежде чем можно будет выполнить команду обновления,

```
UPDATE Studio
SET presC# = 23456
WHERE name = 'Redlight';
```

нам все-таки придется вставить кортеж со сведениями о Билле Клинтоне в отношение MovieExec. Если этого не сделать, операция обновления также приведет к нарушению ограничения ссылочной целостности.

Разумеется, если упорядочить операции вставки во времени (вначале поместить кортеж с информацией о Билле Клинтоне в отношение MovieExec, а затем кортеж с описанием студии "Redlight Studios" в Studio), это определенно исключит всякую возможность нарушений. Однако существуют ситуации, когда из-за наличия так называемых *циклических ограничений* (circular constraints) проблему их возможного нарушения нельзя решить простым упорядочением операций модификации базы данных.

Пример 7.5. Предположим на время, что множество "руководителей" состоит только из президентов киностудий. В этом случае ничто не мешает объявить атрибут cert# отношения MovieExec как внешний ключ, ссылающийся на атрибут presC# отношения Studio; при этом, разумеется, придется снабдить атрибут presC# признаком UNIQUE, предусматривая, что ни одно лицо не может быть президентом двух киностудий одновременно.

Теперь окажется, что невозможно пополнить базу данных ни кортежем с описанием новой киностудии, ни кортежем с информацией о новом президенте: операция вставки в отношение Studio кортежа с уникальным значением presC# приводит к нарушению ограничения внешнего ключа, связывающего presC# с MovieExec(cert#), а операция вставки в MovieExec кортежа с уникальным значением cert# влечет нарушение условия внешнего ключа, соединяющего cert# со Studio(presC#). ❑

Проблема, освещенная в примере 7.5, поддается решению, но для этого следует прибегнуть к средствам SQL, которые до сих пор нами не рассматривались.

1. Требуется наличие способа группирования нескольких команд SQL (в данном случае двух инструкций вставки кортежей: одного — в отношение Studio, а другого — в MovieExec) в неделимую логическую единицу работы, называемую *транзакцией* (transaction) (за подробностями обращайтесь к разделу 8.6 на с. 393).
2. Необходимо иметь возможность указать системе, что условие определенного ограничения не должно проверяться до тех пор, пока транзакция не будет выполнена полностью, т.е. ее результаты не будут *зафиксированы* (committed) в базе данных.

Некие способы объявления транзакций действительно существуют — сейчас вы можете просто принять это на веру, — но относительно второго условия следует высказать два соображения.

1. Ключ, внешний ключ или любое другое ограничение, о которых мы будем говорить ниже, могут быть объявлены с использованием любого из двух предложений — DEFERABLE или NOT DEFERABLE. Последнее, подразумеваемое по умолчанию, означает, что всякий раз, когда содержимое базы данных подвергается модификации, система сразу выполняет все необходимые проверки, если таковые предусмотрены. Если же ограничение объявлено как DEFERABLE, система, прежде чем осуществить проверку, должна дождаться завершения текущей транзакции.
2. Если объявление ограничения содержит служебное слово DEFERABLE, вслед за ним можно указать также любую из двух пар служебных слов — INITIALLY DEFERRED или INITIALLY IMMEDIATE. В первом случае проверка выполнения условия ограничения должна быть отложена до момента окончания обработки текущей транзакции, если только команда возобновления проверки этого ограничения не задана принудительно. Если же ограничение объявлено с применением предложения INITIALLY IMMEDIATE, проверка ограничения изначально будет осуществляться непосредственно перед внесением любых изменений, но с условием, что команда приостановки проверки может быть введена в любой момент.

Пример 7.6. На рис. 7.4 приведено объявление отношения Studio, предусматривающее, что операции проверки ограничения внешнего ключа должны быть отложены до завершения каждой соответствующей транзакции. Помимо того, атрибут presC# помечен как UNIQUE, чтобы его можно было использовать в качестве объекта внешней ссылки со стороны других отношений (в частности, MovieExec).

```
CREATE TABLE Studio (
  name CHAR(30) PRIMARY KEY,
  address VARCHAR(255),
  presC# INT UNIQUE
         REFERENCES MovieExec(cert#)
         DEFERRABLE INITIALLY DEFERRED
);
```

Рис. 7.4. Использование инструкции отложенной проверки ограничения внешнего ключа

Объявив аналогичным образом предусмотренное примером 7.5 ограничение внешнего ключа, связывающее атрибут MovieExec(cert#) с атрибутом Studio(presC#), мы смогли бы сформировать транзакцию, охватывающую операции вставки кортежей в оба отношения, и тогда процедуры проверки ограничений внешних ключей приостанавливались бы системой до тех пор, пока обе операции вставки не были бы завершены. Таким образом, при пополнении базы данных сведениями о новых президенте и киностудии можно было бы не беспокоиться о возможных нарушениях внешних ключей. ❑

Существуют два дополнительных обстоятельства, которые следует иметь в виду при использовании механизма отложенной проверки ограничений:

- ограничению любого типа может быть присвоено наименование (за подробностями обращайтесь к разделу 7.3.1 на с. 334);
- если ограничение DEFERRABLE снабжено именем (скажем, MyConstraint), для перехода с режима *непосредственной* (immediate) проверки на режим *отложенной* (deferred) проверки необходимо выполнить SQL-команду

  ```
  SET CONSTRAINT MyConstraint DEFERRED;
  ```

  а в противном случае — команду

  ```
  SET CONSTRAINT MyConstraint IMMEDIATE;
  ```

7.1.7. Упражнения к разделу 7.1

* **Упражнение 7.1.1.** Отношения из "кинематографической" базы данных в редакции, впервые рассмотренной в разделе 5.1 на с. 204, обладают следующими ключами, отмеченными *подчеркиванием*:

```
Movie(title, year, length, inColor, studioName, producerC#)
StarsIn(movieTitle, movieYear, starName)
MovieStar(name, address, gender, birthdate)
MovieExec(name, address, cert#, netWorth)
Studio(name, address, presC#)
```

Дополните определениями первичных ключей объявления вида `CREATE TABLE`, которые были сконструированы вами при решении упражнения 6.6.1 (см. с. 304) (учтите, что ключом отношения `StarsIn` служат все три его атрибута).

Упражнение 7.1.2. Предложите объявления следующих ограничений ссылочной целостности для отношений "кинематографической" базы данных, имея в виду наличие в них ключей, созданных при решении упражнения 7.1.1.

* a) Продюсер кинофильма должен быть одним из тех лиц, кто упомянут в отношении `MovieExec`. Операции модификации `MovieExec`, нарушающие это условие, должны быть отвергнуты.

b) Повторите решение задания п. (a) с учетом того, что в результате возможного нарушения ограничения компоненту `producerC#` отношения `Movie` должно быть присвоено значение `NULL`.

c) Повторите решение задания п. (a) с учетом того, что в результате возможного нарушения ограничения соответствующий кортеж `Movie` обязан быть удален или обновлен.

d) Кинофильм, упомянутый в `StarsIn`, должен быть описан и в `Movie`. Операции, нарушающие это ограничение, подлежат отмене.

e) Информация об актере, упомянутом в `StarsIn`, должна присутствовать и в `MovieStar`. В случае нарушения этого ограничения соответствующие кортежи `StarsIn` следует удалить.

***!** **Упражнение 7.1.3.** Рассмотрим ограничение, предполагающее, что каждый кинофильм, сведения о котором представлены в отношении `Movie`, должен быть упомянут по меньшей мере в одном кортеже отношения `StarsIn`. Возможно ли представить это условие в виде ограничения внешнего ключа? Почему да или почему нет?

Упражнение 7.1.4. Предложите объявления подходящих первичных ключей для отношений

```
Product(maker, model, type)
PC(model, speed, ram, hd, rd, price)
Laptop(model, speed, ram, hd, screen, price)
Printer(model, color, type, price)
```

из базы данных, рассмотренной в упражнении 5.2.1 на с. 218. Дополните определениями первичных ключей объявления вида `CREATE TABLE`, которые были сконструированы вами при решении заданий (a)–(d) упражнения 6.6.2 (см. с. 304).

Упражнение 7.1.5. Предложите объявления подходящих первичных ключей для отношений

```
Classes(class, type, country, numGuns, bore, displacement)
Ships(name, class, launched)
Battles(name, date)
Outcomes(ship, battle, result)
```

из базы данных, рассмотренной в упражнении 5.2.4 на с. 222. Дополните определениями первичных ключей объявления вида CREATE TABLE, которые были сконструированы вами при решении заданий (a)–(d) упражнения 6.6.3 (см. с. 304).

Упражнение 7.1.6. Сформулируйте следующие ограничения ссылочной целостности для отношений из базы данных о судах, упомянутой в упражнении 7.1.5. Примите к сведению ранее принятые вами решения, касающиеся выбора первичных ключей отношений, и учтите, что в результате возможного нарушения ограничений компонентам внешних ключей должны быть присвоены значения NULL.

* a) Каждый класс, упомянутый в Ships, должен быть описан в Classes.

b) Каждое сражение, упомянутое в Outcomes, должно быть представлено кортежем в Battles.

c) Сведения о каждом судне, упомянутом в Outcomes, должны присутствовать в Ships.

7.2. Ограничения уровня атрибутов и кортежей

Вы уже знакомы с ограничениями ключей, предполагающими, что определенные атрибуты всех кортежей отношений должны содержать различные значения, и внешними ключами, устанавливающими ограничения ссылочной целостности между атрибутами двух отношений. Теперь пришло время рассмотреть третью важную разновидность ограничений, в соответствии с которыми компоненты определенных атрибутов отношения могут обладать только такими значениями, которые относятся к заранее оговоренному подмножеству допустимых значений. Каждое из подобных ограничений может быть выражено одним из следующих способов:

1) как ограничение конкретного атрибута, заданное в определении этого атрибута в контексте объявления схемы отношения;

2) как ограничение уровня кортежа в целом; такое ограничение является частью объявления отношения и оформляется в виде отдельного предложения, не относящегося к каким бы то ни было атрибутам.

В разделе 7.2.1 мы расскажем о простом типе ограничений уровня атрибута, предполагающем, что компоненты атрибута не могут содержать значения NULL, а в разделе 7.2.2 — о базовой синтаксической форме подобных ограничений (об ограничениях CHECK уровня атрибута). Сведения об ограничениях CHECK уровня кортежа приведены в разделе 7.2.3 на с. 330.

Существует и иная, наиболее общая разновидность ограничений (см. раздел 7.4 на с. 336), которые способны препятствовать внесению определенных изменений как в отдельные компоненты или кортежи, так и в отношения целиком или даже в группы отношений.

7.2.1. Ограничения NOT NULL

Одно из самых простых ограничений уровня отдельного атрибута позволяет запретить присваивание компонентам атрибутов значений NULL. Ограничение объявляется с помощью пары служебных слов NOT NULL, располагаемых после имени и обозначения типа атрибута в конструкции CREATE TABLE.

Пример 7.7. Предположим, что следует запретить возможность присваивания значений NULL компонентам presC# отношения Studio. Чтобы реализовать замысел, достаточно дополнить строку 4 рис. 7.3 (см. с. 324) следующим образом:

```
4)  presC# INT REFERENCES MovieExec(cert#) NOT NULL
```

Последствия внесения подобного изменения таковы:

- вставка в отношение `Studio` нового кортежа, содержащего только значения названия (`name`) и адреса (`address`) киностудии, окажется невозможной, поскольку система предпримет попытку присвоить компоненту `presC#` кортежа значение `NULL`;
- мы не сможем применить стратегию *set-null* в тех ситуациях (взгляните на строку 5 рис. 7.3), когда во избежание нарушения ограничения внешнего ключа система обязана занести в компонент `presC#` значение `NULL`.

❑

7.2.2. Ограничения `CHECK` уровня атрибута

Более сложное ограничение уровня атрибута включается в определение последнего с помощью служебного слова `CHECK`, за которым следует условное выражение в круглых скобках, описывающее множество допустимых значений атрибута. Обычно подобные ограничения используются для задания перечислимых множеств значений или границ изменения значений. Впрочем, строго говоря, в качестве условия может быть использовано любое выражение, которое разрешается применять в контексте предложения `WHERE` SQL-запроса. Для ссылки на атрибут, значения которого должны быть ограничены, в выражении условия используется имя этого атрибута. Если, однако, условие распространяется на какое-либо другое отношение или его атрибуты, такое отношение должно быть указано в предложении `FROM` подзапроса (даже в том случае, если проверяемый атрибут принадлежит схеме того же отношения).

Ограничение `CHECK` уровня атрибута проверяется всякий раз, когда компонент кортежа, относящийся к этому атрибуту, получает новое значение — либо в процессе выполнения операции `UPDATE`, либо при вставке (`INSERT`) кортежа. Если новое значение компонента нарушает ограничение, операция отменяется. В примере 7.9 будет рассмотрена ситуация, когда условие `CHECK` уровня атрибута не проверяется, если операция модификации не затрагивает значение этого атрибута непосредственно, что способно привести к нарушению ограничения. Впрочем, для начала мы обратимся к более простому примеру.

Пример 7.8. Пусть необходимо обеспечить выполнение следующего условия: сертификационные номера президентов киностудий должны включать не менее шести цифр. Для решения задачи достаточно заменить строку 4 рис. 7.3 (см. с. 324), представляющего объявление схемы отношения

```
Studio(name, address, presC#),
```

строкой

```
4)  presC# INT REFERENCES MovieExec(cert#)
                CHECK (presC# >= 100000)
```

Еще один пример. Атрибут `gender` схемы отношения

```
MovieStar(name, address, gender, birthdate),
```

объявление которой приведено на рис. 6.16 (см. с. 298), не только относится к типу `CHAR(1)`, но и, как мы полагаем, способен принимать одно из двух значений — `'F'` (от *female* — "женщина") или `'M'` (от *male* — "мужчина"). Указанное ограничение можно выразить, дополнив строку 4 рис. 6.16 предложением `CHECK`:

```
4)  gender CHAR(1) CHECK (gender IN ('F', 'M')),
```

Список `('F', 'M')` представляет собой экземпляр отношения с двумя кортежами-компонентами, и условие `gender IN ('F', 'M')` гласит, что любой компонент атрибута `gender` может содержать только одно из двух значений — `'F'` или `'M'`. ❑

Условие, задаваемое в предложении CHECK, может содержать ссылки на другие атрибуты или кортежи того же отношения или даже адресовать другие отношения, но для этого требуется использовать конструкцию подзапроса. Как мы говорили выше, в качестве условия CHECK позволено использовать любое выражение, разрешенное для применения в контексте предложения WHERE запроса "select–from–where". Однако следует иметь в виду, что при проверке ограничения тестируются только значения того атрибута, к которому относится предложение CHECK, но не остальных атрибутов, упомянутых в предложении. Поэтому в некоторых случаях к отрицательному (FALSE) результату проверки сложного условного выражения может привести изменение не того компонента, который, собственно, и подлежит проверке, а какого-либо другого элемента выражения.

Пример 7.9. Как кажется, можно имитировать ограничение ссылочной целостности, предполагающее существование значения, адресуемого внешним ключом, с помощью ограничения CHECK уровня атрибута. Ниже рассмотрен пример *ошибочной* попытки представления посредством ограничения CHECK такого условия, которое предусматривает, что значение компонента presC# кортежа отношения

```
Studio(name, address, presC#)
```

обязано присутствовать в компоненте cert# некоторого кортежа отношения

```
MovieExec(name, address, cert#, netWorth).
```

Предположим, что строка 4 рис. 7.3 (см. с. 324) изменена следующим образом:

```
4)  presC# INT CHECK
                (presC# IN (SELECT cert# FROM MovieExec(cert#))
```

С формальной точки зрения в такой конструкции нет ничего плохого, но давайте проанализируем ее более внимательно.

- При попытке вставки (INSERT) в отношение Studio нового кортежа, компонент presC# которого содержит такое значение сертификационного номера, какого нет среди значений компонентов cert# отношения MovieExec, операция, как и ожидалось, будет отменена.
- Операция замены (UPDATE) содержимого компонента presC# некоторого кортежа Studio значением, отсутствующим в столбце cert# отношения MovieExec, также будет отвергнута.
- Если, однако, модифицировать отношение MovieExec (например, удалить кортеж, соответствующий президенту какой-либо киностудии), ограничение CHECK, объявленное для атрибута отношения Studio, никак на подобное изменение не отреагирует. Таким образом, операция удаления кортежа, нарушающая рассматриваемое ограничение CHECK, воспринимается системой как вполне допустимая.

В разделе 7.4.1 на с. 336 показано, как та же задача может быть решена с использованием более мощных разновидностей ограничений. ❑

7.2.3. Ограничения CHECK уровня кортежа

Для того чтобы определить ограничение, регламентирующее состав кортежей отношения *R*, в объявление CREATE TABLE после списка атрибутов, ключей и внешних ключей следует поместить предложение CHECK и сопроводить его условным выражением, заключенным в круглые скобки. Выражение должно следовать тем же правилам, в соответствии с которыми оформляются предложения WHERE. Выражение CHECK интерпретируется как условие уровня кортежа отношения *R*; для ссылки на атрибуты *R* употребляются их имена. Однако, как и в случае ограничений CHECK уровня атрибута, условное выражение посредством подзапросов способно адресовать другие отношения или иные кортежи того же отношения *R*.

Как создавать правильные ограничения

Во многих случаях ограничения должны предотвратить появление кортежей, удовлетворяющих нескольким условиям (как в примере 7.10, где необходимо исключить возможность существования кортежей, отвечающих условию "актер является мужчиной и его имя начинается с заданного префикса"; условие запрета состоит их двух операндов, "актер является мужчиной" и "имя актера начинается с заданного префикса", соединенных оператором (союзом) AND ("и")). Следующее за CHECK условие-"разрешение", получаемое на основании исходного условия запрета, можно представить в виде оператора OR, примененного к операндам-"отрицаниям": согласно одному из законов Моргана, отрицание (NOT) выражения AND равнозначно оператору OR с теми же операндами, что и в выражении AND, к каждому из которых применен оператор отрицания. Так, первым операндом выражения запрета в примере 7.10 является условие "актер является мужчиной". Поэтому в выражение CHECK мы вправе включить его отрицание — `gender = 'F'` (возможно, выражение `gender <> 'M'` оказалось бы более понятным). Второй операнд выражения запрета, "строка name начинается с символов `'Ms.'`", в форме отрицания выглядит так: `name NOT LIKE 'Ms.%'`.

Условие, лежащее в основе ограничения CHECK уровня кортежа, проверяется перед каждой операцией вставки в *R* нового кортежа или обновления одного из существующих кортежей. Если содержимое компонентов кортежа таково, что условие обращается в FALSE, это значит, что ограничение нарушено и операция вставки или обновления, спровоцировавшая нарушение, подлежит отмене. Если, однако, в условии посредством подзапроса адресуется некоторое отношение (даже то же *R*) и модификация этого отношения приводит к тому, что для некоторого кортежа *R* условие обращается в FALSE, это не препятствует успешному завершению операции. Иными словами, как и в случае ограничений CHECK уровня атрибута, ограничения CHECK уровня кортежа не реагируют на изменения, вносимые в другие отношения.

Хотя с помощью ограничений CHECK уровня кортежа может быть представлена весьма сложная логика поведения системы, особенно изощренные условия зачастую лучше выражать посредством ограничений *"общего вида"* (assertions), о которых рассказано в разделе 7.4.1 на с. 336. Причина состоит в том, что, как мы упоминали, при некоторых обстоятельствах система не в состоянии верно отреагировать на нарушения ограничений CHECK уровня кортежа. Впрочем, если в условии ограничения упоминаются только компоненты проверяемого кортежа и оно не содержит подзапросов, такое ограничение гарантированно поддерживается системой во всех случаях. Ниже приведен пример простого ограничения CHECK, которое охватывает несколько компонентов одного кортежа.

Пример 7.10. Обратимся к примеру 6.39 (см. с. 298), где содержится объявление схемы отношения MovieStar. Объявление воспроизведено на рис. 7.5 и дополнено определениями ключа и одного из возможных ограничений "целостности", которые, как мы полагаем, целесообразно ввести. Ограничение предусматривает, что в строке имени актера не должен присутствовать префикс "Ms.", если актер является мужчиной.

```
1)  CREATE TABLE MovieStar (
2)      name CHAR(30) PRIMARY KEY,
3)      address VARCHAR(255),
4)      gender CHAR(1),
5)      birthdate DATE,
6)      CHECK (gender = 'F' OR name NOT LIKE 'Ms.%')
    );
```

Рис. 7.5. Простой пример ограничения CHECK *уровня кортежа*

Неполная проверка ограничений CHECK: недостаток или благо?

Может показаться странным, почему ограничения CHECK уровня атрибута и кортежа допускают возможность нарушения в тех случаях, когда они содержат ссылки на другие отношения или другие кортежи того же отношения. Причина состоит в том, что ограничения такого рода могут (и должны) быть реализованы более эффективно, нежели мощные ограничения *"общего вида"* (assertions) (см. раздел 7.4.1 на с. 336). Имея дело с ограничениями уровня атрибута или кортежа, системе достаточно проверить выполнение их условий только для тех кортежей, которые подлежат вставке или обновлению. Если же речь идет об ограничении "общего вида", система обязана выполнять проверку всякий раз, когда модификации подвергается любое из отношений, упомянутых в условии этого ограничения. Грамотный проектировщик базы данных должен обращаться к ограничениям уровня атрибута или кортежа только в тех случаях, когда заведомо известно, что они не могут быть нарушены ни при каких обстоятельствах, и использовать иные механизмы, такие как ограничения "общего вида" или триггеры (см. раздел 7.4.2 на с. 339), если задачу нельзя решить более простыми средствами.

Строка 2 содержит объявление, гласящее, что атрибут name является первичным ключом отношения. В строке 6 сформулировано рассматриваемое ограничение. Условие ограничения обращается в TRUE для каждого кортежа, содержащего информацию о любой актрисе либо о любом актере, чье имя не начинается с префикса "Ms.". Напротив, кортеж об актере-мужчине, имя которого содержит префикс "Ms.", считается недопустимым. ❑

7.2.4. Упражнения к разделу 7.2

Упражнение 7.2.1. Сформулируйте перечисленные ниже ограничения, касающиеся содержимого атрибутов отношения

```
Movie(title, year, length, inColor, studioName, producerC#).
```

* a) Значение атрибута year не должно быть меньше 1985.

b) Значение атрибута length обязано принадлежать интервалу 60–250.

* c) Допустимыми значениями атрибута name являются такие, как "Disney", "Fox", "MGM" или "Paramount".

Упражнение 7.2.2. Сформулируйте перечисленные ниже ограничения, касающиеся содержимого атрибутов отношений

```
Product(maker, model, type)
PC(model, speed, ram, hd, rd, price)
Laptop(model, speed, ram, hd, screen, price)
Printer(model, color, type, price)
```

из базы данных, рассмотренной в упражнении 5.2.1 на с. 218.

a) Значение быстродействия процессора (speed) переносного компьютера (Laptop) не должно быть менее 800 Мгц.

b) В качестве устройства (rd), работающего со съемными дисками, в персональных компьютерах (PC) могут использоваться 32- или 40-скоростные приводы компакт-дисков ('32xCD' или '40xCD') либо 12- или 16-скоростные приводы DVD ('12xDVD' или '16xDVD').

c) Допустимыми значениями типа (type) принтера (Printer) являются 'laser' ("лазерный"), 'ink-jet' ("струйный") и 'bubble' ("пузырьковый").

d) Допустимыми значениями типа (type) продукта (Product) являются 'pc' ("персональный компьютер"), 'laptop' ("переносной компьютер") и 'printer' ("принтер").

! e) Номер модели (model) продукта (Product) должен быть номером модели (model) персонального компьютера (PC), переносного компьютера (Laptop) либо принтера (Printer).

Упражнение 7.2.3. В примере 7.13 на с. 337 утверждается, что ограничение CHECK уровня кортежа, приведенное на рис. 7.7 (см. с. 338), выполняет только "половину" функций ограничения *"общего вида"*, текст которого показан на рис. 7.6. Сформулируйте ограничение CHECK, касающееся отношения MovieExec и гарантирующее осуществление всех остальных функций.

Упражнение 7.2.4. Сформулируйте перечисленные ниже условия в виде ограничений CHECK уровня кортежа, каждое из которых определяется в контексте объявления одного из отношений

```
Movie(title, year, length, inColor, studioName, producerC#)
StarsIn(movieTitle, movieYear, starName)
MovieStar(name, address, gender, birthdate)
MovieExec(name, address, cert#, netWorth)
Studio(name, address, presC#)
```

"кинематографической" базы данных. Если ограничение в действительности должно охватывать два отношения, введите по одному ограничению в объявление каждой из таблиц таким образом, чтобы они (ограничения) проверялись для каждой операции вставки или обновления кортежей отношений. Операции удаления кортежей рассматривать не следует, поскольку в общем случае ограничения уровня кортежа не способны гарантировать корректность и целостность одних порций данных при удалении других.

* a) Кинофильм (Movie) не мог быть снят на цветной пленке (значение TRUE атрибута color), если он выпущен до (year) 1939 года.

b) Актер (MovieStar) не мог сняться в фильме (StarsIn), вышедшем на экран (movieYear) до момента рождения (birthdate) самого актера.

! c) Никакие две студии (Studio) не могут быть расположены по одному адресу (address).

*! d) Значение атрибута name отношения MovieStar не может присутствовать в одноименных компонентах отношения MovieExec.

! e) Каждое название (name) студии (Studio) должно присутствовать (studioName) по меньшей мере в одном кортеже отношения Movie.

!! f) Если продюсер кинофильма является еще и президентом киностудии, он должен быть президентом той студии, которая выпустила этот кинофильм.

Упражнение 7.2.5. Сформулируйте перечисленные ниже условия в виде ограничений CHECK уровня кортежа для отношений PC и Laptop, схемы которых воспроизведены в упражнении 7.2.2.

a) Персональный компьютер (PC), оснащенный процессором с быстродействием (speed) менее 1200 МГц, не должен продаваться по цене (price), превосходящей 1500 долларов.

b) Переносной компьютер (Laptop) с экраном размером менее 15 дюймов обязан обладать диском (hd) с емкостью не менее 20 Гбайт либо должен продаваться по цене (price) менее 2000 долларов.

Упражнение 7.2.6. Сформулируйте перечисленные ниже ограничения CHECK, касающиеся содержимого кортежей отношений

```
Classes(class, type, country, numGuns, bore, displacement)
Ships(name, class, launched)
Battles(name, date)
Outcomes(ship, battle, result)
```

из базы данных, рассмотренной в упражнении 5.2.4 на с. 222.

a) Суда ни одного из классов (Classes) не могут быть оснащены главными орудиями калибра (bore), превышающего 16 дюймов.

b) Если суда класса (Classes) оснащены более чем 9 орудиями (numGuns), их калибр (bore) не должен превосходить 14 дюймов.

!c) Ни одно из судов (Ships) не могло участвовать в сражении (Battles), происходившем прежде (date), чем судно было спущено на воду (launched).

7.3. Модификация ограничений

Вполне возможно добавить, изменить или удалить то или иное ограничение в любой момент. Способ модификации зависит от того, к какому "объекту" относится ограничение — атрибуту, таблице или (как в разделе 7.4.1 на с. 336) к схеме базы данных в целом.

7.3.1. Именование ограничений

Чтобы получить возможность изменения или удаления существующего ограничения, последнему необходимо присвоить имя. Для этого достаточно предпослать объявлению ограничения служебное слово CONSTRAINT и соответствующую строку имени.

Пример 7.11. Чтобы обозначить именем ограничение первичного ключа, приведенное в строке 2 рис. 7.1 (см. с. 319), следует изменить строку, скажем, так:

```
2)  name CHAR(30) CONSTRAINT NameIsKey PRIMARY KEY,
```

Аналогичным образом можно присвоить подходящее имя и ограничению CHECK уровня атрибута, рассмотренному в примере 7.8 на с. 329:

```
4)  gender CHAR(1) CONSTRAINT NoAndro
                CHECK (gender IN ('F', 'M')),
```

Наконец, конструкция

```
6)  CONSTRAINT RightTitle
                CHECK (gender = 'F' OR name NOT LIKE 'Ms.%')
```

является уточненной редакцией ограничения CHECK уровня кортежа (см. рис. 7.5 на с. 331). ❑

7.3.2. Модификация ограничений и команда ALTER TABLE

В разделе 7.1.6 на с. 324 мы упоминали, каким образом с помощью команды SET CONSTRAINT изменить режим обработки ограничения с "непосредственного" (immediate) на "отложенный" (deferred) и наоборот. Команда ALTER TABLE позволяет выполнять другие действия по модификации ограничений. В разделе 6.6.3 на с. 298 уже приводились примеры использования этой команды с целью добавления и удаления атрибутов.

С помощью команды ALTER TABLE можно воздействовать как на ограничения CHECK уровня атрибута, так и на ограничения уровня кортежа. Чтобы удалить ограничение, следует ввести предложение DROP, содержащее имя ограничения. Для добавления ограничения используется служебное слово ADD, после которого дается определение ограничения. Заметьте,

что ограничение добавить нельзя, если в текущем экземпляре базы данных существует одноименное ограничение.

Пример 7.12. Покажем, каким образом с помощью команды `ALTER TABLE` можно удалить и добавить ограничения, касающиеся отношения `MovieStar` (см. пример 7.11). Для удаления ограничений следует использовать команды

```
ALTER TABLE MovieStar DROP CONSTRAINT NameIsKey;
ALTER TABLE MovieStar DROP CONSTRAINT NoAndro;
ALTER TABLE MovieStar DROP CONSTRAINT RightTitle;
```

Если ограничения требуется восстановить, достаточно изменить схему отношения `MovieStar`, добавив одноименные ограничения:

```
ALTER TABLE MovieStar ADD CONSTRAINT NameIsKey
    PRIMARY KEY (name);
ALTER TABLE MovieStar ADD CONSTRAINT NoAndro
    CHECK (gender IN ('F', 'M'));
ALTER TABLE MovieStar ADD CONSTRAINT RightTitle
    CHECK (gender = 'F' OR name NOT LIKE 'Ms.%');
```

Вновь созданные ограничения относятся к категории ограничений уровня кортежа — придать им статус ограничений уровня атрибута уже нельзя.

При воссоздании ограничений давать им имена не обязательно (но весьма желательно). Однако мы не вправе полагаться на то, что система каким-либо образом "помнит" о ранее удаленных ограничениях. Поэтому при необходимости добавления в схему ранее существовавшего ограничения следует объявить его заново и целиком. ❑

7.3.3. Упражнения к разделу 7.3

Упражнение 7.3.1. Покажите, каким образом следует модифицировать схемы отношений

```
Movie(title, year, length, inColor, studioName, producerC#)
StarsIn(movieTitle, movieYear, starName)
MovieStar(name, address, gender, birthdate)
MovieExec(name, address, cert#, netWorth)
Studio(name, address, presC#)
```

из "кинематографической" базы данных, чтобы реализовать перечисленные ниже ограничения.

* a) Атрибуты `title` и `year` образуют первичный ключ отношения `Movie`.

b) Продюсер (`producerC#`) каждого кинофильма (`Movie`) должен быть упомянут в отношении `MovieExec`.

c) Значение атрибута `length` отношения `Movie` обязано принадлежать интервалу 60–250.

*! d) Значение атрибута `name` отношения `MovieStar` не может присутствовать в одноименных компонентах отношения `MovieExec` (операции удаления кортежей рассматривать не следует).

! e) Никакие две студии (`Studio`) не могут быть расположены по одному адресу (`address`).

Упражнение 7.3.2. Покажите, каким образом следует модифицировать схемы отношений

```
Classes(class, type, country, numGuns, bore, displacement)
Ships(name, class, launched)
Battles(name, date)
Outcomes(ship, battle, result)
```

Почему ограничениям желательно давать имена

Помните о том, что каждое из определяемых ограничений целесообразно снабдить подходящим именем — даже в том случае, если, как вы полагаете, на него не придется ссылаться в дальнейшем. Если ограничение объявлено без имени и требуется внести в ограничение какие-либо изменения, дать ему имя вы уже не сможете. Впрочем, если подобная ситуация все-таки произошла, не забывайте, что используемая вами СУБД, вероятно, предоставляет возможность запросить список всех ограничений, объявленных в базе данных: ограничениям, определенным без указания имени, система обычно присваивает собственные внутренние имена, на каждое из которых при необходимости можно сослаться.

из базы данных, рассмотренной в упражнении 5.2.4 на с. 222, чтобы реализовать перечисленные ниже условия в виде ограничений уровня кортежа.

a) Атрибуты `class` и `country` образуют первичный ключ отношения `Classes`.

b) Каждое судно (`ship`), упомянутое в отношении `Outcomes`, должно быть описано кортежем в `Ships`.[40]

c) Ни одно из судов не должно быть оснащено более чем 14 орудиями (`numGuns`).

! d) Ни одно из судов (`Ships`) не могло участвовать в сражении (`Battles`), происходившем прежде (`date`), чем судно было спущено на воду (`launched`).

7.4. Ограничения уровня схемы и триггеры

Возможности наиболее "мощных" активных элементов в SQL не ограничены уровнем кортежа или отдельного атрибута кортежа отношения. Подобные элементы (ограничения "общего вида" и триггеры) являются частью схемы базы данных — наравне с хранимыми отношениями и виртуальными таблицами.

- Ограничение *"общего вида"* (assertion) — это булево SQL-выражение, значением которого всегда должно быть `TRUE`.
- *Триггер* (trigger) — последовательность команд, поставленная в соответствие определенному событию (такому как вставка кортежа в заданное отношение) и выполняемая в ответ на это событие.

Ограничения "общего вида", предполагающие задание некоторого условия, выполнение которого система обязана гарантировать, более просты для программирования. Триггеры, которые принято относить к категории активных элементов общего назначения, однако, находят более широкое применение. Причина состоит в том, что ограничения "общего вида" с трудом поддаются эффективной реализации: всякий раз, когда выполняется модификация содержимого базы данных, система обязана самостоятельно проследить, не повлияет ли результат операции на истинность условия ограничения. Текст триггера, с другой стороны, совершенно точно сообщает системе, что именно необходимо делать в том или ином случае.

7.4.1. Ограничения "общего вида"

В стандарте SQL предусмотрена возможность задания ограничений *"общего вида"* (assertions), позволяющих оговорить любое условие (в формате, принятом для описания предложений `WHERE`), подлежащее выполнению. Подобно другим элементам схемы базы данных, ограничения "общего вида" объявляются с помощью конструкции, содержащей служебное слово `CREATE`. Объявление ограничения "общего вида" состоит из следующих частей:

[40] Задание (b) в оригинале содержит ошибочное условие и поэтому здесь опущено. Номера остальных заданий соответствующим образом "сдвинуты". — *Прим. перев.*

1) пары служебных слов CREATE ASSERTION;
2) названия *A* ограничения;
3) служебного слова CHECK;
4) условия *C*, заключенного в круглые скобки.

Иными словами, выражение создания ограничения "общего вида" можно представить так:

```
CREATE ASSERTION A CHECK (C);
```

Условие *C* ограничения должно равняться значению TRUE в момент выполнения команды и обязано оставаться истинным и в дальнейшем; любая операция модификации базы данных, чреватая нарушением условия, подлежит отмене. (Напомним, что некоторые из рассмотренных выше ограничений CHECK допускают возможность нарушения, если они содержат подзапросы.)

Способы оформления ограничений CHECK уровня кортежа и ограничений "общего вида" различаются. Ограничение уровня кортежа способно свободно ссылаться на атрибуты того отношения, к схеме которого относится само ограничение. Например, в строке 6 рис. 7.5 (см. с. 331) атрибуты gender и name адресуются напрямую, без указания на то, схеме какого отношения они принадлежат, и ссылаются на компоненты кортежа, подлежащего вставке или обновлению в MovieStar (это единственное отношение, упомянутое в рассматриваемой конструкции CREATE TABLE).

При создании ограничения "общего вида" мы лишены подобных привилегий. Любые атрибуты, на которые необходимо сослаться, должны быть однозначно описаны — как правило, с упоминанием соответствующих отношений в выражении вида "select-from-where". Поскольку выражение в целом должно относиться к булеву типу, результат вычисления выражения некоторым образом агрегируется и приводится к единственной величине, способной принимать значения TRUE или FALSE. Например, условие может выглядеть как выражение, возвращающее отношение, к которому применяется оператор NOT EXISTS; иными словами, ограничение состоит в том, что отношение всегда обязано быть пустым. В другом случае к столбцу отношения можно применить оператор агрегирования (подобный SUM) и сопоставить его результат с некоторой константой, т.е. предусмотреть, что сумма определенных значений, скажем, не должна превышать заданной границы.

Пример 7.13. Предположим, что требуется реализовать ограничение "общего вида", согласно которому лицо не может стать президентом киностудии, если оно не обладает совокупным годовым доходом в размере не менее 10 миллионов долларов. Ограничение по сути равнозначно утверждению о том, что множество киностудий, президенты которых являются владельцами состояний с размером менее 10 миллионов долларов, пусто. В ограничение вовлечены два отношения:

```
MovieExec(name, address, cert#, netWorth)
Studio(name, address, presC#).
```

Текст ограничения приведен на рис. 7.6.

```
CREATE ASSERTION RichPres CHECK
  (NOT EXISTS
         (SELECT *
          FROM Studio, MovieExec
          WHERE presC# = cert# AND netWorth < 10000000
         )
  );
```

Рис. 7.6. Пример ограничения "общего вида"

Между прочим, рассмотренное ограничение "общего вида", затрагивающее два отношения, можно заменить парой ограничений CHECK уровня кортежа, объявленных в составе тех же отношений. Например, нетрудно ввести в объявление отношения Studio (см. пример 7.3 на с. 321) конструкцию CHECK, показанную на рис. 7.7.

```
CREATE TABLE Studio (
  name CHAR(30) PRIMARY KEY,
  address VARCHAR(255),
  presC# INT REFERENCES MovieExec(cert#),
  CHECK (presC# NOT IN
              (SELECT cert# FROM MovieExec
               WHERE netWorth < 10000000)
  )
);
```

Рис. 7.7. Попытка замены ограничения "общего вида" ограничением CHECK *уровня атрибута*

Необходимо заметить, что ограничение, показанное на рис. 7.7, будет проверяться только при изменении содержимого "родного" отношения Studio. Система не может выявить, например, такую ситуацию, когда значение компонента netWorth кортежа MovieExec с информацией о президенте какой-либо студии снижается и становится меньше заветных 10 миллионов долларов. Чтобы получить решение, которое можно было бы считать полноценной альтернативой ограничению "общего вида", приведенному на рис. 7.6, в составе отношения MovieExec надлежит объявить дополнительное ограничение CHECK, гласящее, что значение компонента netWorth не должно быть меньше 10000000, если руководящее лицо является президентом какой-либо киностудии. ❑

Пример 7.14. Рассмотрим еще один образец ограничения "общего вида". Оно касается отношения

```
Movie(title, year, length, inColor, studioName, producerC#)
```

и гласит, что значение общей продолжительности воспроизведения кинофильмов, выпущенных одной студией, не должно превышать 10000 минут:

```
CREATE ASSERTION SumLength CHECK (10000 >= ALL
    (SELECT SUM(length) FROM Movie GROUP BY studioName)
);
```

Поскольку указанное ограничение "общего вида" относится только к Movie, его можно выразить в виде ограничения CHECK уровня кортежа и объявить в контексте схемы Movie. Другими словами, следует пополнить объявление отношения Movie конструкцией

```
CHECK (10000 >= ALL
    (SELECT SUM(length) FROM Movie GROUP BY studioName));
```

Указанное условие применяется к каждому кортежу отношения Movie. Однако оно не содержит явных ссылок на атрибуты кортежа, и основная доля "обязанностей" возлагается на подзапрос.

Обратите внимание, что если условие представлено в форме ограничения CHECK уровня кортежа, оно не будет проверяться системой при удалении кортежей из Movie. В данном случае такое поведение не опасно, поскольку если ограничение выполнялось до операции удаления кортежа, оно определенно останется в силе и после удаления. Однако, если заменить условие на обратное, т.е. установить не верхнюю границу суммарной продолжительности воспроизведения кинофильмов, а нижнюю, использование ограничения CHECK уровня кортежа вместо аналогичного ограничения "общего вида" грозит неприятностями, поскольку при удалении кортежей в какой-то момент времени условие может быть нарушено, а система никак на это не отреагирует. ❑

О классификации ограничений по функциональным возможностям

В следующей таблице представлены основные различия между ограничениями CHECK уровня атрибута, уровня кортежа и ограничениями "общего вида".

Вид ограничения	Контекст объявления	Когда проверяется	Гарантии выполнения
Ограничение CHECK уровня атрибута	Объявление атрибута	При вставке кортежа или обновлении атрибута	Да, если не используются подзапросы
Ограничение CHECK уровня кортежа	Объявление схемы отношения	При вставке или обновлении кортежа	Да, если не используются подзапросы
Ограничение "общего вида"	Схема базы данных	При любой модификации одного из упомянутых отношений	Да

Наконец, стандарт SQL предусматривает возможность удаления ограничений "общего вида". Соответствующая команда сродни другим командам удаления элементов схемы базы данных:

```
DROP ASSERTION A;
```

где *A* — наименование ограничения.

7.4.2. Правила "событие-условие-действие"

Триггеры (triggers), или, как их еще называют, *правила "событие-условие-действие"* (event-condition-action rules — ECA rules), отличаются от других ограничений, рассмотренных выше, в трех аспектах.

1. Триггер "срабатывает" только при наступлении определенного *события* (event), описание которого содержится в тексте триггера. К числу допустимых событий обычно относят операции вставки, удаления и обновления кортежа указанного отношения. Во многих СУБД "событием", на которое способен отреагировать триггер, разрешено считать и факт завершения транзакции. В разделе 7.1.6 на с. 324 мы уже вкратце рассказывали о транзакциях; за подробностями обращайтесь к разделу 8.6 на с. 393.
2. Вместо того чтобы сразу отвергнуть операцию, вызвавшую его срабатывание, триггер выполняет проверку *условия* (condition). Если результат проверки равен значению FALSE, работа триггера завершается.
3. Если же условие, сформулированное в тексте триггера, справедливо, СУБД выполняет *действие* (action), поставленное в соответствие событию, — например, препятствует продолжению операции либо наряду с ней выполняет другие операции. Под действием понимается последовательность операций с базой данных, отвечающая логике определенного события.

7.4.3. Триггеры в SQL

Синтаксис и семантика выражения триггера в SQL предоставляют программисту множество разнообразных возможностей, касающихся реализации каждой из частей конструкции — "событие", "условие" и "действие". Основные особенности конструкции объявления триггера перечислены ниже.

1. Действие может выполняться либо до, либо после события триггера.

2. При выполнении действия триггер способен ссылаться как на *прежние* (old), так и на *новые* (new) значения компонентов кортежа, операция модификации которого рассматривается как событие, повлекшее срабатывание триггера.
3. События обновления данных могут быть ограничены операциями над отдельным компонентом или множеством компонентов.
4. Условие может задаваться с помощью предложения `WHEN`; действие выполняется при наступлении события и только в том случае, если в этот момент условие оставалось справедливым.
5. Действие может выполняться в одном из двух возможных режимов:
   a) один раз для каждого модифицируемого кортежа;
   b) единожды для всех кортежей, которые подверглись изменению при выполнении одной операции.

Прежде чем приступить к изложению строгих синтаксических правил, рассмотрим пример, который поможет прояснить наиболее существенные аспекты программирования триггеров. Подразумевается, что триггер выполняется один раз для каждого кортежа, подлежащего модификации.

Пример 7.15. Пусть необходимо создать триггер, реагирующий на событие обновления (`UPDATE`) компонента `netWorth` кортежа отношения

```
MovieExec(name, address, cert#, netWorth).
```

Триггер обязан воспрепятствовать любой попытке снижения текущего значения годового дохода руководящего лица.[41] Объявление триггера приведено на рис. 7.8.

```
 1)  CREATE TRIGGER NetWorthTrigger
 2)  AFTER UPDATE OF netWorth ON MovieExec
 3)  REFERENCING
 4)       OLD ROW AS OldTuple,
 5)       NEW ROW AS NewTuple
 6)  FOR EACH ROW
 7)  WHEN (OldTuple.netWorth > NewTuple.netWorth)
 8)       UPDATE MovieExec
 9)       SET netWorth = OldTuple.netWorth
10)       WHERE cert# = NewTuple.cert#;
```

Рис. 7.8. Пример триггера уровня кортежа

Строка 1 содержит заголовок объявления (`CREATE TRIGGER`) триггера с указанием его наименования. Строка 2 описывает событие триггера — а именно факт обновления (`UPDATE`) компонента `netWorth` кортежа отношения `MovieExec`. В строках 3–5 регламентируется способ, посредством которого фрагменты кода триггера, отвечающие за реализацию "условия" и "действия", способны обращаться к кортежу и в его прежнем (`OLD`) состоянии (т.е. до выполнения операции обновления), и в новом (`NEW`) (после операции). Эти кортежи названы для ясности как `OldTuple` и `NewTuple` соответственно. Во фрагментах кода "условия" и "действия" триггера эти имена могут использоваться совершенно так же, как и переменные кортежа, объявляемые в предложении `FROM` обычного SQL-запроса.

[41] Ах, если бы проблема похудания кошелька исчерпывалась определением триггера в базе данных! — *Прим. перев.*

Строка 6, содержащая предложение FOR EACH ROW, выражает требование о том, что код триггера должен выполняться по одному разу для каждого кортежа, подвергшегося обновлению. Если бы эта фраза была опущена или заменена альтернативной, FOR EACH STATEMENT, предусмотренной по умолчанию, триггер срабатывал бы единожды для SQL-команды целиком, независимо от того, какое количество связанных с ней операций обновления должно быть выполнено. В подобном случае вместо псевдонимов кортежей, OLD ROW и NEW ROW, можно было бы определить псевдонимы таблиц, OLD TABLE и NEW TABLE (о последних — ниже).

Строка 7 представляет "условие" триггера, которое гласит, что последующее "действие" должно выполняться только в том случае, если новое значение компонента netWorth меньше, нежели прежнее.

Строки 8–10 содержат порцию кода триггера, описывающую "действие", которое подлежит выполнению, если "условие" справедливо. В данном случае "действие" представляет собой заурядную команду UPDATE, которая призвана восстановить прежнее содержимое компонента netWorth, т.е. то значение, которое хранилось в компоненте до выполнения операции обновления, вызвавшей срабатывание триггера. Как может показаться на первый взгляд, "восстановлению" способен поддаться любой кортеж MovieExec, но на самом деле операция затронет только тот, в котором, как предусмотрено предложением WHERE, содержится соответствующее значение cert#. ❑

Разумеется, в рамках одного простого примера нам удалось проиллюстрировать только некоторые из множества возможностей, предусмотренных механизмом SQL-триггеров. Ниже кратко перечислены иные важные аспекты практического использования триггеров.

- Строка 2 рис. 7.8 гласит, что "действие" триггера должно выполняться *после* (AFTER) возникновения его "события". Служебное слово AFTER может быть заменено словом BEFORE (*до*) — это будет означать, что условие предложения WHEN должно проверяться до внесения изменения, спровоцировавшего срабатывание триггера. Если условие верно, вначале выполняется код "действия" триггера, а затем операция, которая активизировала триггер, — независимо от того, справедливо условие или нет.
- Помимо UPDATE, к категории допустимых (т.е. способных привести к срабатыванию триггера) относятся события-операции INSERT и DELETE. Предложение OF (см. строку 2 рис. 7.8) при описании события UPDATE задавать не обязательно. Если служебное слово OF присутствует, после него перечисляются наименования атрибутов, операции обновления которых расцениваются как событие триггера. Предложение OF нельзя использовать для определения событий INSERT и DELETE, поскольку последние связаны с модификацией кортежа в целом.
- Предложение WHEN не является обязательным. В его отсутствие "действие" выполняется всякий раз после срабатывания триггера без учета каких бы то ни было условий.
- Код "действия" триггера может состоять из произвольной[42] последовательности SQL-команд, завершаемых символом точки с запятой; последовательность размещается между служебными словами BEGIN и END.
- Если событие триггера связано с операцией обновления (UPDATE), существует возможность адресовать варианты обновляемого кортежа, соответствующие состоянию

[42] В секции "действия" текста триггера разрешено использовать далеко не все команды SQL: так, например, возбраняется применять команды управления транзакциями (подобные COMMIT и ROLLBACK), организации соединения между клиентом и сервером базы данных (скажем, CONNECT и DISCONNECT), создания (*CREATE*), изменения (*ALTER*) и удаления (*DROP*) элементов схемы, определения параметров сеанса (SET) и пр. — *Прим. перев.*

базы данных до и после выполнения операции: с помощью предложений OLD ROW AS и NEW ROW AS кортежам могут быть присвоены имена, как это сделано в строках 4, 5 рис. 7.8. Если событие триггера обусловлено операцией вставки (INSERT), на кортеж, подлежащий вставке, разрешено ссылаться посредством предложения NEW ROW AS, а предложение OLD ROW AS использовать запрещено. В случае события-удаления (DELETE) допустимо ссылаться на удаляемый кортеж с помощью предложения OLD ROW AS, а предложение NEW ROW AS, напротив, применять возбраняется.

- Если опустить предложение FOR EACH ROW, *триггер уровня кортежа* (row-level trigger) (см. строку 6 рис. 7.8) "превратится" в *триггер уровня команды* (statement-level trigger). Триггер уровня команды выполняется один раз для команды модификации соответствующего типа, независимо от того, какое именно количество кортежей подлежит обновлению, вставке или удалению. Например, при задании инструкции обновления, затрагивающей множество кортежей таблицы, триггер уровня команды срабатывает один раз, в то время как триггер уровня кортежа должен выполняться для каждого кортежа, подлежащего обновлению. В тексте триггера уровня команды ссылаться на прежнюю и новую версии кортежа, как показано в строках 4 и 5, запрещено. Однако любому триггеру — и уровня кортежа, и уровня команды — позволено адресовать *множества* "старых" (old) кортежей (удаляемых кортежей либо прежних версий обновляемых кортежей) и/или "новых" (new) кортежей (вставляемых кортежей либо новых версий обновляемых кортежей): с этой целью применяются предложения, подобные таким, как OLD TABLE AS OldStuff и NEW TABLE AS NewStuff.[43]

Пример 7.16. Предположим, что необходимо предотвратить возможность уменьшения среднего значения годового дохода руководящих лиц ниже отметки в 500 тыс. долларов. К подобному результату способна привести команда вставки в отношение

```
MovieExec(name, address, cert#, netWorth)
```

нового кортежа, а также удаления любого существующего кортежа либо обновления его компонента netWorth. Один из тонких моментов связан с тем, что единственная команда INSERT или UPDATE может предусматривать вставку или обновление нескольких кортежей MovieExec, и в процессе ее выполнения среднее значение дохода будет то падать ниже заданной границы, то возрастать, пока процедура обработки команды не завершится полностью. Поэтому триггер обязан отреагировать на результат команды в целом (если, разумеется, условие триггера обратится в TRUE).

Необходимо создать отдельный триггер для каждого из трех возможных событий — вставка (INSERT), удаление (DELETE) и обновление (UPDATE). На рис. 7.9 приведен текст триггера, запускаемого системой в ответ на событие обновления кортежа отношения MovieExec. Код двух других триггеров аналогичен и несколько более прост.

В строках 3–5 содержатся объявления псевдонимов OldStuff и NewStuff, позволяющих ссылаться на отношения со "старыми" и "новыми" кортежами, которые затрагиваются в процессе выполнения операции обновления, вызывающей срабатывание триггера. Единственная команда модификации способна повлиять на состояние многих кортежей, и поэтому

[43] В дополнительных пунктах списка стоило бы упомянуть синтаксическую форму команды удаления триггера (DROP TRIGGER *T*, где *T* — наименование триггера), а также вкратце осветить еще один важный аспект, который касается правил, регламентирующих последовательность запуска нескольких триггеров, если таковые определены для одних и тех же события и таблицы. За подробностями обращайтесь к стандарту SQL-99 (см. раздел 6.9 на с. 315) и к документации из комплекта поставки конкретной СУБД. — *Прим. перев.*

столько же кортежей будет помещено системой в каждое из отношений OldStuff и NewStuff.

Если речь идет об операции обновления (как в нашем примере), в отношения OldStuff и NewStuff заносятся прежние и новые версии обновляемых кортежей. Если рассмотреть аналогичный триггер, реагирующий на событие удаления, удаляемые кортежи будут помещены в отношение OldStuff (предложение вида NEW TABLE AS NewStuff, однако, в этом случае использовать нельзя). При написании триггера для события вставки, напротив, можно предусмотреть предложение NEW TABLE AS NewStuff, которое позволит ссылаться на отношение с вновь вставленными кортежами, но отношение вида OldStuff, по понятным причинам, в этом случае системой поддерживаться не будет.

```
 1)  CREATE TRIGGER AvgNetWorthTrigger
 2)  AFTER UPDATE OF netWorth ON MovieExec
 3)  REFERENCING
 4)       OLD TABLE AS OldStuff,
 5)       TABLE AS NewStuff
 6)  FOR EACH STATEMENT
 7)  WHEN (500000 > (SELECT AVG(netWorth) FROM MovieExec))
 8)  BEGIN
 9)       DELETE FROM MovieExec
10)       WHERE (name, address, cert#, netWorth) IN NewStuff;
11)       INSERT INTO MovieExec
12)              (SELECT * FROM OldStuff);
13)  END;
```

Рис. 7.9. Пример триггера уровня команды

Строка 6 свидетельствует о том, что триггер должен выполняться один раз для каждой команды обновления в целом, независимо от того, какое именно количество кортежей подлежит обновлению. Строка 7 представляет условие, которое удовлетворяется, если среднее значение, вычисленное на основе содержимого компонентов netWorth *после* их обновления, меньше 500000.

Порция кода "действия" триггера, охватывающая строки 8–13, выполняется, если условие предложения WHEN обращается в TRUE, и состоит из двух команд: первая (см. строки 9, 10) предполагает удаление из MovieExec всех "новых" (т.е. уже обновленных) кортежей, а вторая (см. строки 11, 12) — восстановление кортежей в том состоянии, в котором они пребывали до операции UPDATE, вызвавшей срабатывание триггера. ❑

7.4.4. Триггеры INSTEAD OF

Существует весьма многообещающая разновидность триггеров, которая не описана в стандарте SQL-99, но активно обсуждалась в период его разработки и была реализована во многих коммерческих СУБД. В подобных триггерах разрешается заменять предложения BEFORE (*до*) и AFTER (*после*) предложением INSTEAD OF (*вместо*), означающим, что всякий раз при срабатывании триггера код "действия" последнего должен выполняться вместо той операции, которая спровоцировала запуск триггера.

Это средство мало что дает, если речь идет о хранимых отношениях[44], но в полной мере проявляет все свои качества в применении к *виртуальным таблицам* (views), поскольку последние редко допускают возможность непосредственной модификации (см. раздел 6.7.4 на с. 308). Триггер `INSTEAD OF` способен "распознать" попытку изменения содержимого виртуальной таблицы и вместо этого выполнить другую операцию, уместную в конкретном случае. Рассмотрим пример.

Пример 7.17. Вновь обратимся к определению виртуальной таблицы

```
CREATE VIEW ParamountMovie AS
    SELECT title, year
    FROM Movie
    WHERE studioName = 'Paramount';
```

из примера 6.45 (см. с. 305), представляющей информацию обо всех кинофильмах, выпущенных студией "Paramount". Как было показано в примере 6.49 на с. 308, содержимое таблицы `ParamountMovie`, вообще говоря, поддается обновлению, но при этом возникает одна неприятность: при вставке кортежа система не в состоянии определить, что в компонент `studioName` всегда должна заноситься строка `'Paramount'` (хотя для нас с вами это очевидно), и поэтому присваивает компоненту значение `NULL`, предусмотренное по умолчанию.

Более изящное решение проблемы связано с использованием триггера `INSTEAD OF`, текст которого показан на рис. 7.10. Большая часть кода триггера не таит никаких сюрпризов. Однако полезно обратить внимание на строку 2, содержащую предложение `INSTEAD OF`, которое свидетельствует, что любая операция вставки кортежа в таблицу `ParamountMovie` должна быть отменена.

Строки 5, 6 содержат описание "действия" триггера: вставить соответствующий кортеж в хранимое отношение `Movie`. Содержимое всех компонентов кортежа, подлежащего вставке в `Movie`, известно: значения `title` и `year` взяты из кортежа, команда вставки которого в таблицу `ParamountMovie` вызвала срабатывание триггера (мы ссылаемся на них с помощью псевдонима `NewRow`, определенного в строке 3), а значением компонента `studioName` служит знакомая константа `'Paramount'` (в составе исходного кортежа она не задавалась).

```
1)  CREATE TRIGGER ParamountInsert
2)  INSTEAD OF INSERT ON ParamountMovie
3)  REFERENCING NEW ROW AS NewRow
4)  FOR EACH ROW
5)  INSERT INTO Movie(title, year, studioName)
6)  VALUES(NewRow.title, NewRow.year, 'Paramount');
```

Рис. 7.10. Пример триггера `INSTEAD OF`

❑

7.4.5. Упражнения к разделу 7.4

Упражнение 7.4.1. Создайте триггеры, упомянутые в примере 7.16 на с. 342, которые обязаны реагировать на события вставки кортежа в отношение `MovieExec` и удаления кортежа из этого отношения.

---

[44] Подобное утверждение слишком категорично — достаточно взглянуть на примеры триггеров, приведенные на рис. 7.8 и 7.9: применение модели триггеров `INSTEAD OF` позволило бы существенно снизить издержки, связанные с тем, что системе вначале приходится выполнять команды, вызвавшие срабатывание триггера, а затем (`AFTER`) фактически отменять их результаты. — *Прим. перев.*

Упражнение 7.4.2. Сформулируйте перечисленные ниже условия, касающиеся содержимого атрибутов отношений

```
Product(maker, model, type)
PC(model, speed, ram, hd, rd, price)
Laptop(model, speed, ram, hd, screen, price)
Printer(model, color, type, price)
```

из базы данных, рассмотренной в упражнении 5.2.1 на с. 218, в форме триггеров или ограничений "общего вида", предусматривая необходимость запрета или отмены операции, если таковая нарушает заданное условие.

* a) При обновлении значения цены (`price`) компьютера (`PC`) гарантировать, что компьютеров с более низкой ценой и тем же уровнем быстродействия процессора (`speed`) не существует.

* b) Ни один из производителей (`maker`) персональных компьютеров (значение `'pc'` атрибута `type`) не должен выпускать переносные компьютеры (`'laptop'`).

*! c) Производитель (`maker`) персональных компьютеров обязан выпускать переносные компьютеры по меньшей мере с таким же уровнем быстродействия процессора (`speed`).

d) При вставке кортежа с информацией о новом принтере (`Printers`) убедиться, что номер модели (`model`) присутствует в отношении `Product`.

! e) При выполнении любой операции модификации содержимого отношения `Laptop` обеспечить сохранение следующего условия: среднее значение цены (`price`) переносных компьютеров, выпускаемых каждым производителем (`maker`), не должно быть ниже 2000 долларов.

! f) При обновлении значений объема оперативной памяти (`ram`) и емкости диска (`hd`) персонального компьютера (`PC`) гарантировать, что величина `hd` по меньшей мере в 100 раз превышает значение `ram`.

! g) Если переносной компьютер (`Laptop`) обладает бо́льшим объемом оперативной памяти (`ram`), нежели персональный компьютер (`PC`), его цена (`price`) также должна быть выше.

! h) При вставке кортежа с информацией о новом персональном компьютере (`PC`), переносном компьютере (`Laptop`) или принтере (`Printers`) убедиться, что заданный номер модели (`model`) не совпадает ни с одним из существующих номеров.

! i) Если отношение `Product` содержит кортеж с определенными значениями номера модели (`model`) и типа продукта (`type`), тот же номер модели должен присутствовать в отношении, соответствующем типу продукта.

Упражнение 7.4.3. Сформулируйте перечисленные ниже условия, касающиеся содержимого атрибутов отношений

```
Classes(class, type, country, numGuns, bore, displacement)
Ships(name, class, launched)
Battles(name, date)
Outcomes(ship, battle, result)
```

из базы данных, рассмотренной в упражнении 5.2.4 на с. 222, в форме триггеров или ограничений "общего вида", предусматривая необходимость запрета или отмены операции, если таковая нарушает заданное условие.

* a) При вставке кортежа с информацией о новом классе судов (Classes) обеспечить вставку кортежа со сведениями о судне (Ships), название (name) которого совпадает с наименованием (class) класса; дату спуска судна на воду (launched) принять равной NULL.

b) При вставке кортежа с данными о классе судов (Classes) водоизмещением (displacement) свыше 35000 тонн заменить значение displacement на 35000.

c) Ни один из классов (class) не должен включать более 2 судов (Ships).

! d) Ни одна из стран (country) не может владеть как линкорами (значение 'bb' атрибута type), так и крейсерами ('bc').

! e) Ни одно из судов (ship) с более чем 9-ю главными орудиями (numGuns) не могло участвовать в сражении (battle) наряду с потопленным (значение 'sunk' атрибута result) судном, обладавшим орудиями в количестве, меньшем 9.

! f) При вставке кортежа в отношение Outcomes убедиться, что заданные значения названия (name) судна и наименования (battle) сражения присутствуют в отношениях Ships и Battles соответственно; в противном случае обеспечить вставку в последние подходящих кортежей, при необходимости используя значения NULL.

! g) При вставке кортежа в отношение Ships или обновлении компонента class существующего кортежа Ships гарантировать, что ни одна из стран (country) не обладает более чем 20-ю судами.

! h) Ни одно из судов (Ships) не могло быть спущено на воду (launched) прежде, чем флагманский корабль, давший имя (class) классу, к которому относится судно.

! i) Для каждого класса (class) должно существовать одноименное (name) судно (Ships).

!! j) При всех обстоятельствах, способных привести к нарушению условия, обеспечить выполнение этого условия, состоящего в том, что ни одно из судов (ship) не должно было участвовать в сражении (Battles), состоявшемся позже (date) другого сражения, в котором судно оказалось потопленным (значение 'sunk' атрибута result).

! **Упражнение 7.4.4.** Сформулируйте перечисленные ниже условия, касающиеся содержимого атрибутов отношений

```
Movie(title, year, length, inColor, studioName, producerC#)
StarsIn(movieTitle, movieYear, starName)
MovieStar(name, address, gender, birthdate)
MovieExec(name, address, cert#, netWorth)
Studio(name, address, presC#)
```

из "кинематографической" базы данных, в форме триггеров или ограничений "общего вида", предусматривая необходимость запрета или отмены операции, если таковая нарушает заданное условие. Следует полагать, что до момента изменения содержимого базы данных каждое из условий выполняется. При поиске решений необходимо отдавать предпочтение тем из них, которые предусматривают не отмену операций модификации, а выполнение их — пусть даже ценой вставки кортежей со значениями, предлагаемыми по умолчанию (скажем, NULL).

a) Каждому актеру, информация о котором присутствует в отношении `StarsIn`, должен соответствовать кортеж отношения `MovieStar`.

b) Каждый "руководитель" должен быть либо президентом киностудии, либо продюсером, участвовавшим в съемках хотя бы одного кинофильма, либо тем и другим одновременно.

c) В каждом фильме должны сниматься хотя бы один актер и по меньшей мере одна актриса.

d) Количество кинофильмов, выпущенных каждой киностудией в любом году, не должно превышать 120.

e) Средняя продолжительность воспроизведения кинофильмов, снятых в течение каждого года, не должна превышать 120 минут.

7.5. Резюме

- *Ограничения ключа*. Чтобы описать тот факт, что атрибут или подмножество атрибутов образуют ключ отношения, достаточно воспользоваться одной из синтаксических форм предложений `UNIQUE` или `PRIMARY KEY`, размещаемых в контексте объявления реляционной схемы.
- *Ограничения ссылочной целостности*. Ограничение, в соответствии с которым значения компонентов атрибута или подмножества атрибутов одного отношения обязаны присутствовать в соответствующих компонентах ключевых атрибутов некоторого кортежа другого отношения, выражается посредством предложений `REFERENCES` или `FOREIGN KEY`, включаемых в объявление схемы отношения.
- *Ограничения* `CHECK` *уровня атрибута*. Чтобы ограничить множество допустимых значений атрибута некоторым условием, достаточно снабдить определение атрибута в схеме отношения служебным словом `CHECK` и указать соответствующее условие.
- *Ограничения* `CHECK` *уровня кортежа*. Объявление отношения может содержать отдельные предложения `CHECK`, описывающие условия, которым должны удовлетворять кортежи отношения.
- *Модификация ограничений*. Ограничение `CHECK` уровня кортежа легко добавить и удалить с помощью команды `ALTER TABLE`, предусматривающей изменение схемы соответствующей таблицы.
- *Ограничения "общего вида"*. Ограничение "общего вида" объявляется как элемент схемы базы данных с помощью команды `CREATE ASSERTION` со служебным словом `CHECK` и соответствующим условием. Условие может охватывать отдельные атрибуты или кортежи как одного отношения, так и нескольких отношений одновременно.
- *Проверка условий ограничений*. Условие ограничения "общего вида" подлежит проверке при любом изменении, вносимом в одно из отношений, упомянутых в условии. Условие ограничения `CHECK` уровня атрибута или кортежа проверяется только в том случае, если отношение, к схеме которого относится ограничение, изменяется командами вставки или обновления. Если ограничение `CHECK` содержит подзапрос, система не способна гарантировать поддержку этого ограничения.
- *Триггеры*. Триггер — это фрагмент кода, хранимый в схеме базы данных, который активизируется системой при наступлении определенного события (как правило, при вставке, обновлении или удалении кортежа заданного отношения). После запуска триггера система проверяет его условие; если условие справедливо, выполняется заданная последовательность SQL-команд, таких как запросы или инструкции модификации.

7.6. Литература

За информацией о возможности получения стандартов SQL2 и SQL-99 обращайтесь к разделу 6.9 на с. 315. В работах [4] и [5] представлены подробные обзоры всего того, что относится к проблематике активных элементов баз данных. Доклад [1] предлагает оригинальную трактовку понятия активных элементов в контексте SQL-99 и будущих стандартов. Статьи [2] и [3] содержат сведения о HiPAC — одном из ранних прототипов систем баз данных, реализующих возможность определения активных элементов.

1. Cochrane R.J., Pirahesh H., and Mattos N. *Integrating triggers and declarative constraints in SQL database systems*, Intl. Conf. on Very Large Database Systems (1996), p. 567–579.
2. Dayal U. et al. *The HiPAC project: combining active databases and timing constraints*, SIGMOD Record, **17**:1 (1988), p. 51–70.
3. McCarthy D.R., Dayal U. *The architecture of an active database management system*, Proc. ACM SIGMOD Intl. Conf. on Management of Data (1989), p. 215–224.
4. Paton N.W., Diaz O. *Active database systems*, Computing Surveys, **31**:1 (March, 1999), p. 63–103.
5. Widom J., Ceri S. *Active Database Systems*, Morgan-Kaufmann, San Francisco, 1996.