Содержание

14 ГЛАВА

Представления

Таблицы определяют структуру базы данных и организацию информации в ней. Однако SQL с помощью представлений позволяет взглянуть на данные под другим углом. *Представлением* называется SQL-запрос на выборку, которому присвоили имя и который затем сохранили в базе данных. Представление позволяет пользователю увидеть результаты сохраненного запроса, а SQL обеспечивает доступ к этим результатам таким образом, как если бы они были реальной таблицей базы данных.

Представления используются по нескольким причинам:

- они позволяют сделать так, что разные пользователи базы данных будут видеть ее по-разному;
- с их помощью можно ограничить доступ к данным, разрешая пользователям видеть только некоторые из строк и столбцов таблицы;
- они упрощают доступ к базе данных, показывая каждому пользователю структуру хранимых данных в наиболее подходящем для него виде.

В настоящей главе рассказывается о том, как создавать представления и применять их для упрощения работы с базой данных и повышения степени ее безопасности.

Что такое представление

Как видно из рис. 14.1, представление является виртуальной таблицей, содержимое которой определяется запросом. Для пользователя базы данных представление выглядит как реальная таблица, состоящая из строк и столбцов. Однако, в отличие от таблицы, представление как совокупность значений в базе данных реально не существует. Строки и столбцы данных, которые пользователь видит с помощью представления, являются результатами запроса, лежащего в его основе. SQL создает иллюзию представления, присваивая ему имя, как таблице, и сохраняя его определение в базе данных.

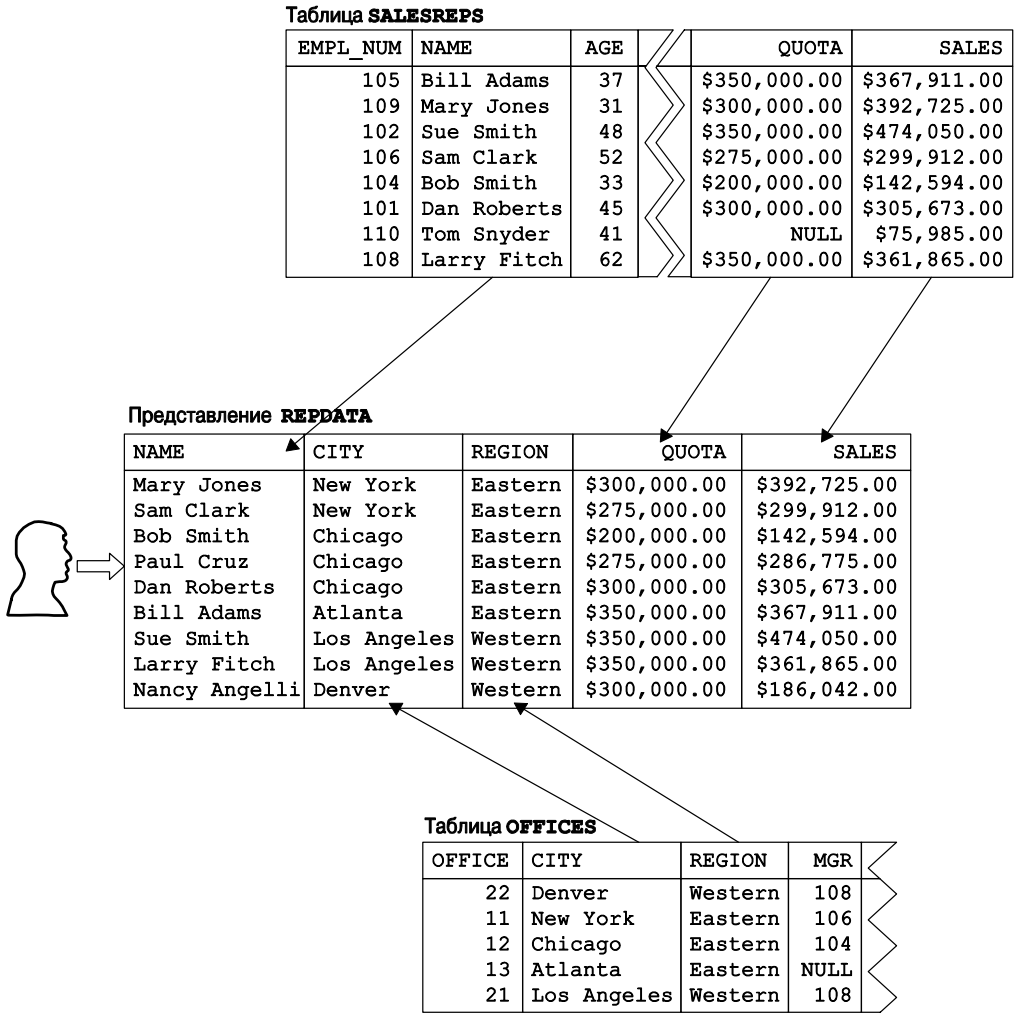

Рис. 14.1. Типичное представление с двумя исходными таблицами

На рис. 14.1 изображено типичное представление. Оно получило имя REPDATA и определено в виде запроса к двум таблицам.

```
SELECT NAME, CITY, REGION, QUOTA, SALESREPS.SALES
  FROM SALESREPS, OFFICES
 WHERE REP_OFFICE = OFFICE;
```

Данные для этого представления берутся из таблиц SALESREPS и OFFICES. Эти таблицы называются *исходными таблицами* представления, поскольку служат для него источниками данных. Каждая строка представления содержит информацию о служащем, дополненную информацией о городе и регионе, в которых он работает. Как показывает рисунок, представление выглядит как таблица, а его содержимое в точности соответствует результатам, которые вы получили бы, если бы действительно выполнили запрос.

После определения представления к нему можно обращаться с помощью инструкции SELECT, как к обычной таблице.

Вывести список служащих, опережающих план, включая имя, город и регион.

```
SELECT NAME, CITY, REGION
  FROM REPDATA
 WHERE SALES > QUOTA;

NAME          CITY         REGION
-----------   -----------  -------
Mary Jones    New York     Eastern
Sam Clark     New York     Eastern
Dan Roberts   Chicago      Eastern
Paul Cruz     Chicago      Eastern
Bill Adams    Atlanta      Eastern
Sue Smith     Los Angeles  Western
Larry Fitch   Los Angeles  Western
```

Имя представления, REPDATA, указывается в предложении FROM как имя обычной таблицы, а ссылка на столбцы представления в инструкции SELECT осуществляется точно так же, как на столбцы таблицы. К некоторым представлениям можно также применять инструкции INSERT, DELETE и UPDATE для изменения данных. Таким образом, представление можно использовать в инструкциях SQL так, *как будто* оно является обычной таблицей. Однако перед тем как приступать к обновлению представления, крайне важно разобраться с его влиянием на лежащие в его основе таблицы.

Как СУБД работает с представлениями

Когда СУБД встречает в инструкции SQL ссылку на представление, она отыскивает его определение, сохраненное в базе данных. Затем СУБД преобразует пользовательский запрос, ссылающийся на представление, в *эквивалентный* запрос к исходным таблицам представления и выполняет его. Таким образом, СУБД создает иллюзию существования представления в виде отдельной таблицы и в то же время сохраняет целостность исходных таблиц.

Если определение представления простое, то СУБД формирует каждую строку представления на лету, извлекая данные из исходных таблиц. Если же определение сложное, СУБД приходится *материализовывать* представление. Это означает, что СУБД выполняет запрос, определяющий представление, и сохраняет его результаты во временной таблице. Из нее СУБД берет данные для формирования результатов пользовательского запроса, а когда временная таблица становится ненужной, удаляет ее. Но независимо от того, как именно СУБД выполняет инструкцию, являющуюся определением представления, для пользователя результат будет одним и тем же. Ссылаться на представление в инструкции SQL можно так же, как если бы оно было реальной таблицей базы данных.

Преимущества представлений

Использование представлений в базах данных различных типов может оказаться полезным в самых разнообразных ситуациях. В базах данных на персональных компьютерах представления применяются для удобства и позволяют упрощать запросы к базе данных. В промышленных базах данных представления играют главную роль в создании собственной структуры базы данных для каждого пользователя и обеспечении ее безопасности. Основные преимущества представлений перечислены ниже.

- **Безопасность**. Каждому пользователю можно разрешить доступ к небольшому числу представлений, содержащих только ту информацию, которую ему позволено знать. Таким образом можно осуществить ограничение доступа пользователей к хранимой информации.
- **Простота запросов**. С помощью представления можно извлечь данные из нескольких таблиц и представить их как одну таблицу, превращая тем самым запрос ко многим таблицам в однотабличный запрос к представлению.
- **Структурная простота**. С помощью представлений для каждого пользователя можно создать собственную структуру базы данных, определив ее как множество доступных пользователю виртуальных таблиц.
- **Защита от изменений**. Представление может возвращать непротиворечивый и неизменный образ структуры базы данных, даже если исходные таблицы разделяются, реструктуризуются или переименовываются. Заметим, однако, что определение представления должно быть обновлено, когда переименовываются лежащие в его основе таблицы или столбцы.
- **Целостность данных**. Если доступ к данным или ввод данных осуществляется с помощью представления, СУБД может автоматически проверять, выполняются ли определенные условия целостности.

Недостатки представлений

Наряду с перечисленными выше преимуществами, представления обладают и тремя существенными недостатками.

- **Производительность**. Представление создает лишь видимость существования соответствующей таблицы, и СУБД приходится преобразовывать запрос к представлению в запрос к исходным таблицам. Если представление отображает многотабличный запрос, то простой запрос к представлению становится сложным объединением и на его выполнение может потребоваться много времени. Однако это связано не с тем, что запрос обращается к представлению, — любой плохо построенный вопрос может вызвать проблемы с производительностью. Дело в том, что сложность запроса скрывается в представлении, так что пользователи не представляют, какой объем работы может вызвать даже кажущийся простым запрос.

- **Управляемость**. Представления, как и все прочие объекты баз данных, должны быть управляемы. Если разработчики и пользователи баз данных смогут бесконтрольно создавать представления, то работа администратора базы данных станет существенно сложнее. Это в особенности справедливо в том случае, когда создаются представления, в основе которых лежат другие представления, которые, в свою очередь, могут быть основаны на других представлениях. Чем больше уровней между базовыми таблицами и представлениями, тем сложнее решать проблемы с представлениями, которые могут возникнуть в такой системе.
- **Ограничения на обновление**. Когда пользователь пытается обновить строки представления, СУБД должна преобразовать запрос в запрос на обновление строк исходных таблиц. Это возможно для простых представлений; более сложные представления обновлять нельзя, они доступны только для выборки.

Указанные недостатки означают, что не стоит без разбора применять представления, в особенности многоуровневые, и использовать их вместо исходных таблиц. В каждом конкретном случае необходимо учитывать перечисленные выше преимущества и недостатки представлений.

Создание представлений (CREATE VIEW)

Для создания представлений используется инструкция `CREATE VIEW`, синтаксическая диаграмма которой изображена на рис. 14.2. В ней указываются имя представления и запрос, лежащий в его основе. Для успешного создания представления необходимо иметь права на доступ ко всем таблицам, входящим в запрос. В некоторых СУБД (таких, как Oracle) у вас должны быть права на создание представлений.

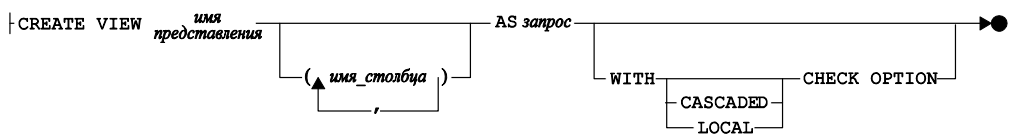

Рис. 14.2. Синтаксическая диаграмма инструкции `CREATE VIEW`

При необходимости в инструкции `CREATE VIEW` можно указать имя для каждого столбца создаваемого представления. Если указывается список имен столбцов, то он должен содержать столько элементов, сколько столбцов содержится в запросе. Обратите внимание на то, что задаются только имена столбцов; тип данных, длина и другие характеристики берутся из определения столбца в исходной таблице. Если список имен столбцов в инструкции `CREATE VIEW` отсутствует, каждый столбец представления получает имя соответствующего столбца запроса. Список имен столбцов обязательно должен быть указан, если в запросе получаются два столбца с одинаковыми именами, а в некоторых СУБД — при наличии в запросе вычисляемых столбцов. Хотя некоторые СУБД автоматически присваивают имена

вычисляемым столбцам, эти имена в действительности не слишком полезны, так что лучше иметь привычку всегда назначать собственные имена вычислимым столбцам.

Хотя все представления создаются одинаковым образом, на практике представления различных типов, как правило, используются для разных целей. В нескольких последующих разделах рассматриваются типы представлений и приводятся примеры инструкций CREATE VIEW.

Горизонтальные представления

Представления широко применяются для ограничения доступа пользователей к строкам таблиц, чтобы пользователи могли видеть не все строки, а только некоторые из них. Например, в учебной базе данных можно позволить менеджеру по продажам видеть в таблице SALESREPS только строки служащих, работающих в его регионе. Для этого можно использовать два приведенных далее представления.

Представление, показывающее информацию о служащих восточного региона.

```
CREATE VIEW EASTREPS AS
     SELECT *
       FROM SALESREPS
      WHERE REP_OFFICE IN (11, 12, 13);
```

Представление, показывающее информацию о служащих западного региона.

```
CREATE VIEW WESTREPS AS
     SELECT *
       FROM SALESREPS
      WHERE REP_OFFICE IN (21, 22);
```

Теперь каждому менеджеру по продажам можно разрешить доступ либо к представлению EASTREPS, либо к WESTREPS и одновременно запретить доступ к другому представлению, а также к таблице SALESREPS. Таким образом, менеджер по продажам получает собственное представление таблицы SALESREPS, в котором отражены только данные о служащих соответствующего региона.

Такие представления, как EASTREPS или WESTREPS, называются *горизонтальными представлениями*. Как видно из рис. 14.3, горизонтальное представление разрезает исходную таблицу по горизонтали. В него входят все столбцы исходной таблицы и только часть ее строк. Горизонтальные представления удобно применять, когда исходная таблица содержит данные, которые относятся к различным организациям или пользователям. Они предоставляют каждому пользователю личную таблицу, содержащую только те строки, которые ему необходимы.

Вот еще несколько примеров горизонтальных представлений.

Представление, содержащее офисы только восточного региона.

```
CREATE VIEW EASTOFFICES AS
     SELECT *
       FROM OFFICES
      WHERE REGION = 'Eastern';
```

Представление для Сью Смит (идентификатор служащего 102), показывающее заказы, сделанные только ее клиентами.

```
CREATE VIEW SUEORDERS AS
     SELECT *
       FROM ORDERS
      WHERE CUST IN (SELECT CUST_NUM
                       FROM CUSTOMERS
                      WHERE CUST_REP = 102);
```

Представление, показывающее только тех клиентов, которые в настоящий момент сделали заказы на сумму более $30 000.

```
CREATE VIEW BIGCUSTOMERS AS
     SELECT *
       FROM CUSTOMERS
      WHERE 30000.00 < (SELECT SUM(AMOUNT)
                          FROM ORDERS
                         WHERE CUST = CUST_NUM);
```

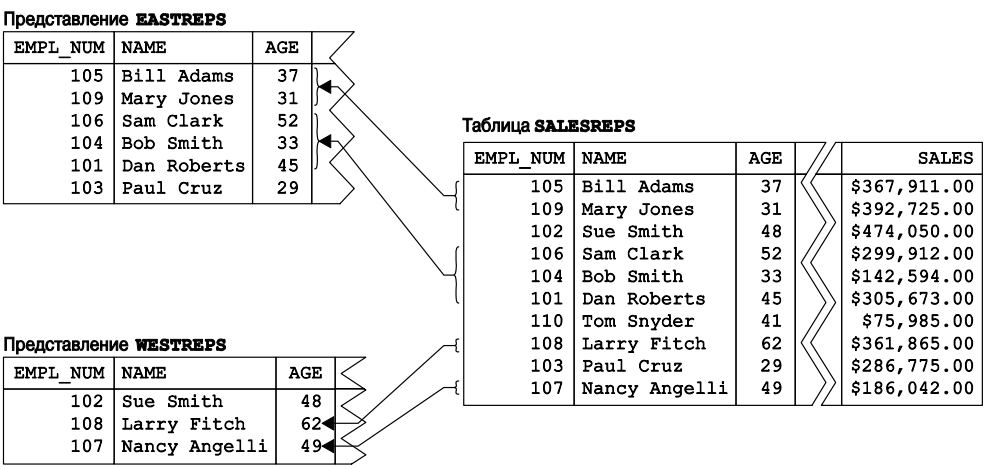

Рис. 14.3. Горизонтальные представления таблицы SALESREPS

В каждом из приведенных примеров источником данных для представления является одна исходная таблица. Представление задается с помощью инструкции `SELECT *` и потому содержит те же столбцы, что и исходная таблица. Предложение `WHERE` выбирает, какие строки исходной таблицы войдут в представление.

Вертикальные представления

Еще одним распространенным применением представлений является ограничение доступа к столбцам таблицы. Например, отделу, обрабатывающему заказы к учебной базе данных, для выполнения своих функций может потребоваться следующая информация: имя, идентификатор служащего и офис, в котором он работает. Но отделу вовсе не обязательно знать плановый и фактический объемы продаж того или иного служащего. Такой избирательный образ таблицы `SALESREPS` можно получить с помощью приведенного ниже представления.

Представление, показывающее избранную информацию о служащих.

```
CREATE VIEW REPINFO AS
     SELECT EMPL_NUM, NAME, REP_OFFICE
       FROM SALESREPS;
```

Разрешив отделу обработки заказов доступ к этому представлению и одновременно запретив доступ к самой таблице SALESREPS, можно ограничить доступ к конфиденциальной информации, каковой являются фактический и плановый объемы продаж.

Такие представления, как REPINFO, называются *вертикальными представлениями*. Как показано на рис. 14.4, вертикальное представление разрезает исходную таблицу по вертикали. Вертикальные представления часто применяются там, где данные используются различными пользователями или группами пользователей. Они предоставляют каждому пользователю личную виртуальную таблицу, содержащую только те столбцы, которые ему необходимы.

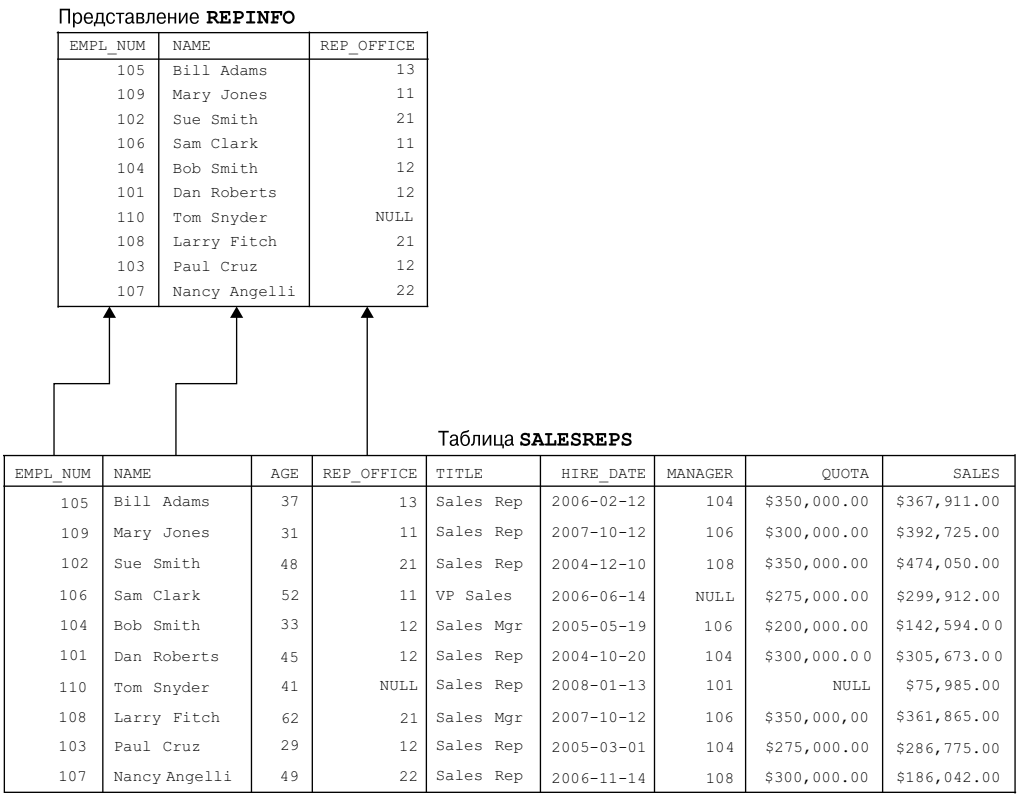

Рис. 14.4. Вертикальное представление таблицы SALESREPS

Вот еще несколько примеров вертикальных представлений.

Для отдела обработки заказов создать представление таблицы OFFICES, которое включает в себя идентификатор офиса, город и регион.

```
CREATE VIEW OFFICEINFO AS
     SELECT OFFICE, CITY, REGION
       FROM OFFICES;
```

Представление таблицы CUSTOMERS, включающее только имена клиентов и имена закрепленных за ними служащих.

```
CREATE VIEW CUSTINFO AS
     SELECT COMPANY, CUST_REP
       FROM CUSTOMERS;
```

В каждом из приведенных примеров источником данных для представления является одна исходная таблица. Список имен избранных столбцов в инструкции CREATE VIEW определяет, какие столбцы исходной таблицы войдут в представление. Поскольку это вертикальные представления, в них входят все строки исходных таблиц, и предложение WHERE в определении представления не требуется.

Смешанные представления

При создании представлений SQL не разделяет их на горизонтальные и вертикальные. В SQL просто отсутствуют понятия горизонтального и вертикального представлений. Они лишь помогают вам понять, каким образом из исходной таблицы формируется представление. Использование представлений, разделяющих исходную таблицу как в горизонтальном, так и вертикальном направлении, — вполне распространенное явление.

Представление, включающее идентификатор клиента, имя компании и лимит кредита для всех клиентов Билла Адамса (идентификатор служащего 105).

```
CREATE VIEW BILLCUST AS
     SELECT CUST_NUM, COMPANY, CREDIT_LIMIT
       FROM CUSTOMERS
      WHERE CUST_REP = 105;
```

Данные, полученные при помощи этого представления, представляют собой подмножество строк и столбцов таблицы CUSTOMERS. В этом представлении будут видны только те столбцы, которые явно указаны в предложении SELECT, и только те строки, которые удовлетворяют условию отбора в предложении WHERE.

Сгруппированные представления

Запрос, определяющий представление, может содержать предложение GROUP BY. Представление такого типа называется *сгруппированным представлением*, поскольку данные в нем являются результатом запроса с группировкой. Сгруппированные представления выполняют ту же функцию, что и запросы с группировкой, — в них родственные строки данных объединяются в группы и для каждой группы в таблице результатов запроса создается одна строка, содержащая итоговые данные по этой груп-

пе. С помощью сгруппированного представления запрос с группировкой превращается в виртуальную таблицу, к которой в дальнейшем можно обращаться с запросами.

Вот пример сгруппированного представления.

Представление, включающее суммарные данные о заказах по каждому служащему.

```
CREATE VIEW ORD_BY_REP (WHO, HOW_MANY, TOTAL,
                        LOW, HIGH, AVERAGE)
    AS
    SELECT REP, COUNT(*), SUM(AMOUNT), MIN(AMOUNT),
           MAX(AMOUNT), AVG(AMOUNT)
      FROM ORDERS
     GROUP BY REP;
```

Как видно из этого примера, в определении сгруппированного представления всегда содержится список имен столбцов. В этом списке присваиваются имена тем столбцам сгруппированного представления, которые являются производными от статистических функций, таких как `SUM()` и `MIN()`. В нем также может быть задано измененное имя столбца группировки. В данном примере столбец `REP` таблицы `ORDERS` становится столбцом `WHO` представления `ORD_BY_REP`.

После создания сгруппированного представления его можно использовать для упрощения запросов. Например, результатом приведенного ниже запроса является отчет, содержащий суммы заказов по каждому служащему.

Отобразить имя, число заказов, общую стоимость заказов и среднюю стоимость заказа по каждому служащему.

```
SELECT NAME, HOW_MANY, TOTAL, AVERAGE
  FROM SALESREPS, ORD_BY_REP
 WHERE WHO = EMPL_NUM
 ORDER BY TOTAL DESC;

NAME            HOW_MANY       TOTAL     AVERAGE
--------------- --------- ----------- -----------
Larry Fitch             7  $58,633.00   $8,376.14
Bill Adams              5  $39,327.00   $7,865.40
Nancy Angelli           3  $34,432.00  $11,477.33
Sam Clark               2  $32,958.00  $16,479.00
Dan Roberts             3  $26,628.00   $8,876.00
Tom Snyder              2  $23,132.00  $11,566.00
Sue Smith               4  $22,776.00   $5,694.00
Mary Jones              2   $7,105.00   $3,552.50
Paul Cruz               2   $2,700.00   $1,350.00
```

В отличие от горизонтальных и вертикальных представлений, каждой строке сгруппированного представления не соответствует какая-то одна строка исходной таблицы. Сгруппированное представление не является просто фильтром исходной таблицы, скрывающим некоторые строки и столбцы. Оно отображает исходную таблицу в виде резюме, поэтому поддержка такой виртуальной таблицы требует от СУБД значительного объема вычислений.

К сгруппированным представлениям можно обращаться с запросами точно так же, как и к более простым представлениям. Однако сгруппированные запросы нельзя обновлять. Причина очевидна: что значит, например, “обновить среднюю

величину заказа для служащего с идентификатором 105"? Так как каждая строка сгруппированного представления соответствует *группе* строк исходной таблицы, а столбцы, как правило, содержат вычисляемые данные, невозможно преобразовать запрос на обновление сгруппированного представления в запрос на обновление строк исходной таблицы. Таким образом, сгруппированные представления функционируют как представления, доступные в режиме только для чтения, к которым можно обращаться с запросами на выборку, но не на обновление.

Сгруппированные представления могут использоваться в запросах, которые группируют строки, выполняя тем самым двойное группирование строк, которое невозможно в обычных запросах. Рассмотрим следующий пример.

Для каждого офиса вывести на экран диапазон средних значений заказов для всех служащих, работающих в офисе.

```
SELECT REP_OFFICE, MIN(AVERAGE), MAX(AVERAGE)
  FROM SALESREPS, ORD_BY_REP
 WHERE EMPL_NUM = WHO
   AND REP_OFFICE IS NOT NULL
 GROUP BY REP_OFFICE;
```

Этот запрос корректно работает в большинстве современных реализаций SQL. Это двухтабличный запрос, который группирует строки представления `ORD_BY_REP` на основе офиса служащего. Вспомним, однако, что представление `ORD_BY_REP` уже сгруппировало строки. Если попытаться получить тот же результат без использования представления (помещая запрос, содержащийся в представлении `ORD_BY_REP`, в новый запрос), мы получим что-то наподобие следующего.

```
SELECT REP_OFFICE, MIN(AVG(AMOUNT)), MAX(AVG(AMOUNT))
  FROM SALESREPS, ORDERS
 WHERE EMPL_NUM = REP
 GROUP BY REP
 GROUP BY REP_OFFICE;
```

Такой запрос является недопустимым, поскольку содержит два предложения `GROUP BY`. Кроме того, в некоторых старых реализациях SQL не поддерживаются вложенные статистические функции, такие как `MIN(AVG(AMOUNT))`. Но, к счастью, все современные реализации SQL в настоящее время поддерживают их.

Соединенные представления

Часто представления используют для упрощения многотабличных запросов. Задавая в определении представления двух- или трехтабличный запрос, можно создать *соединенное представление* — виртуальную таблицу, данные в которую извлекаются из двух или трех различных таблиц. После создания такого представления к нему можно обращаться с помощью однотабличного запроса; в противном случае пришлось бы применять двух- или трехтабличное соединение. Предположим, например, что Сэм Кларк (Sam Clark), вице-президент по продажам, часто обращается с запросами к таблице `ORDERS` учебной базы данных. Однако Сэм не любит работать с идентификаторами служащих и клиентов. Он хотел бы восполь-

зоваться такой версией таблицы ORDERS, где вместо идентификаторов стояли бы имена. Вот представление, отвечающее требованиям Сэма.

Представление таблицы ORDERS с именами вместо идентификаторов.

```
CREATE VIEW ORDER_INFO (ORDER_NUM, COMPANY,
                        REP_NAME, AMOUNT) AS
     SELECT ORDER_NUM, COMPANY, NAME, AMOUNT
       FROM ORDERS, CUSTOMERS, SALESREPS
      WHERE CUST = CUST_NUM
        AND REP = EMPL_NUM;
```

Данное представление является объединением трех таблиц. Как и в случае сгруппированного представления, для создания такой виртуальной таблицы требуется значительный объем работы. Источником каждой строки представления является *комбинация* трех строк: по одной строке из таблиц ORDERS, CUSTOMERS и SALESREPS.

Несмотря на свое относительно сложное определение, это представление может принести реальную пользу. Вот запрос к представлению, дающий сводку заказов, сгруппированную по служащим и компаниям.

Для каждого служащего отобразить на экране общую стоимость заказов по каждой компании.

```
SELECT REP_NAME, COMPANY, SUM(AMOUNT)
  FROM ORDER_INFO
 GROUP BY REP_NAME, COMPANY;

REP_NAME     COMPANY           SUM(AMOUNT)
------------ ---------------- ------------
Bill Adams   Acme Mfg.          $35,582.00
Bill Adams   JCP Inc.            $3,745.00
Dan Roberts  First Corp.         $3,978.00
Dan Roberts  Holm & Landis         $150.00
Dan Roberts  Ian & Schmidt      $22,500.00
Larry Fitch  Midwest Systems     $3,608.00
Larry Fitch  Orion Corp.         $7,100.00
Larry Fitch  Zetacorp           $47,925.00
   .
   .
   .
```

Обратите внимание: этот запрос является однотабличной инструкцией SELECT, которая гораздо проще, чем эквивалентная трехтабличная инструкция SELECT для исходных таблиц.

```
SELECT NAME, COMPANY, SUM(AMOUNT)
  FROM SALESREPS, ORDERS, CUSTOMERS
 WHERE REP = EMPL_NUM
   AND CUST = CUST_NUM
 GROUP BY NAME, COMPANY;
```

Аналогично с помощью следующего запроса к данному представлению можно легко получить список крупных заказов, показывая, кто их сделал и кто принял.

Вывести список крупных заказов, упорядоченных по стоимости.

```
SELECT COMPANY, AMOUNT, REP_NAME
  FROM ORDER_INFO
 WHERE AMOUNT > 20000.00
 ORDER BY AMOUNT DESC;

COMPANY                 AMOUNT REP_NAME
------------------ ----------- --------------
Zetacorp            $45,000.00 Larry Fitch
J.P. Sinclair       $31,500.00 Sam Clark
Chen Associates     $31,350.00 Nancy Angelli
Acme Mfg.           $27,500.00 Bill Adams
Ace International   $22,500.00 Tom Snyder
Ian & Schmidt       $22,500.00 Dan Roberts
```

Значительно легче сформировать запрос к такому представлению, чем создавать эквивалентное объединение трех таблиц. Конечно, чтобы получить результаты однотабличного запроса к представлению, СУБД должна выполнить тот же объем работы, что и при генерации результатов эквивалентного трехтабличного запроса. В действительности, объем работы над запросом даже немного больше, чем над представлением. Но главное то, что пользователю гораздо легче создавать и понимать однотабличные запросы к представлению.

Обновление представлений

Что значит вставить, удалить или обновить строку представления? Для некоторых типов представлений эти операции можно очевидным образом преобразовать в эквивалентные операции по отношению к исходным таблицам представления. Например, вернемся к представлению EASTREPS, рассмотренному ранее в настоящей главе.

Представление, показывающее информацию о служащих только восточного региона.

```
CREATE VIEW EASTREPS AS
     SELECT *
       FROM SALESREPS
      WHERE REP_OFFICE IN (11, 12, 13);
```

Это простое горизонтальное представление, основанное на одной исходной таблице. Как видно из рис. 14.5, добавление строки в данное представление имеет смысл; оно означает, что новая строка должна быть вставлена в таблицу SALESREPS, лежащую в основе представления. Аналогично имеет смысл удаление строки из представления EASTREPS — это удаление соответствующей строки из таблицы SALESREPS. Наконец, обновление строки представления EASTREPS также имеет смысл: оно будет обновлением соответствующей строки таблицы SALESREPS. Во всех случаях требуемое действие можно выполнить по отношению к соответствующей строке исходной таблицы, тем самым сохраняя целостность исходной таблицы и представления.

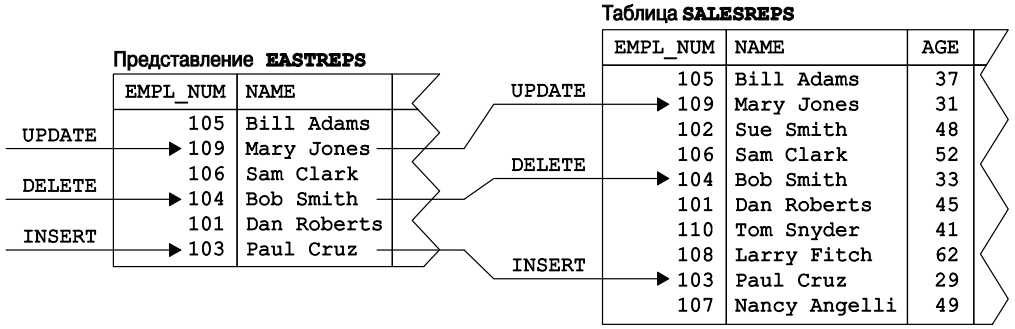

Рис. 14.5. Обновление данных с использованием представления

Теперь обратимся к сгруппированному представлению ORD_BY_REP, также рассматривавшемуся ранее в настоящей главе.

Представление, в которое входят суммарные данные о заказах по каждому служащему.

```
CREATE VIEW ORD_BY_REP (WHO, HOW_MANY, TOTAL,
                        LOW, HIGH, AVERAGE) AS
     SELECT REP, COUNT(*), SUM(AMOUNT), MIN(AMOUNT),
            MAX(AMOUNT), AVG(AMOUNT)
       FROM ORDERS
      GROUP BY REP;
```

Между строками этого представления и исходной таблицы нет взаимооднозначного соответствия, поэтому добавление, удаление или обновление строк представления не имеет смысла. Представление ORD_BY_REP обновлять невозможно, его можно только просматривать.

Представления EASTREPS и ORD_BY_REP являются двумя противоположностями в смысле сложности их определений. Имеются более сложные представления, чем EASTREPS, обновление которых возможно, и есть представления, менее сложные, чем ORD_BY_REP, обновление которых, однако, невозможно. Обновление представлений — непростой вопрос, который является предметом исследований в области реляционных баз данных уже много лет.

Обновление представлений и стандарт ANSI/ISO

В исходном стандарте SQL1 четко указано, какие представления базы данных обновимы в соответствии со стандартом (*обновимость* в данном контексте означает вставку, модификацию или удаление). Согласно стандарту, представление можно обновлять в том случае, если определяющий его запрос соответствует всем перечисленным ниже ограничениям.

- Должен отсутствовать предикат DISTINCT, т.е. повторяющиеся строки не должны исключаться из таблицы результатов запроса.
- В предложении FROM должна быть задана только одна обновляемая таблица, т.е. у представления должна быть одна исходная таблица, а пользователь должен иметь соответствующие права доступа к ней. Если исход-

ная таблица сама является представлением, то оно также должно удовлетворять этим условиям.

- Каждое имя в списке возвращаемых столбцов должно быть ссылкой на простой столбец; в этом списке не должны содержаться выражения, вычисляемые столбцы или статистические функции.
- Предложение WHERE не должно содержать подчиненный запрос; в нем могут присутствовать только простые построчные условия отбора.
- В запросе не должны содержаться предложения GROUP BY и HAVING.

Базовая концепция, скрывающаяся за этими ограничениями, проще для запоминания, чем сами правила. Чтобы представление было обновимо, СУБД должна быть способна для каждой строки представления найти соответствующую строку в исходной таблице, а для каждого обновляемого столбца представления — соответствующий столбец в исходной таблице.

Если представление соответствует этим требованиям, то над ним и над исходной таблицей можно выполнять имеющие смысл операции вставки, удаления и обновления. Однако если представление пропускает столбец исходной таблицы с ограничением NOT NULL без спецификации DEFAULT, то нельзя выполнить вставку новой строки в таблицу посредством данного представления. Причина в том, что в этом случае нет способа указать значение данных для пропущенного столбца.

Обновление представлений в коммерческих СУБД

Правила, установленные в стандарте SQL1, являются очень жесткими. Существует множество представлений, которые теоретически можно обновлять, хотя они соответствуют не всем этим правилам. Кроме того, существуют представления, над которыми можно производить не все операции обновления, а также представления, в которых можно обновлять не все столбцы. В большинстве коммерческих реализаций SQL правила обновления представлений значительно менее строгие, чем в стандарте SQL. Рассмотрим такой пример.

Представление, включающее плановый и фактический объемы продаж, а также разницу между ними для каждого служащего.

```
CREATE VIEW SALESPERF (EMPL_NUM, SALES, QUOTA, DIFF) AS
     SELECT EMPL_NUM, SALES, QUOTA, (SALES - QUOTA)
       FROM SALESREPS;
```

Стандарт SQL запрещает обновление этого представления, поскольку четвертый столбец у него вычисляемый. Однако обратите внимание: каждая строка представления основана только на одной строке исходной таблицы (SALESREPS). По этой причине в DB2 (и ряде других коммерческих СУБД) разрешается выполнять инструкцию DELETE по отношению к этому представлению. Кроме того, в DB2 разрешаются операции обновления столбцов EMPL_NUM, SALES и QUOTA, поскольку они берутся непосредственно из исходной таблицы. Не допускается только обновление столбца DIFF. В DB2 не разрешается выполнять инструкцию

INSERT по отношению к этому представлению, так как добавление какого-либо значения в столбец DIFF не имеет смысла.

Правила обновления представлений неодинаковы в различных СУБД, и обычно они довольно детальны. Некоторые представления, например, созданные на основе сгруппированных запросов, не обновляются ни в одной СУБД, так как эта операция просто не имеет смысла. Некоторые типы представлений в одних СУБД обновлять можно, в других можно только частично, а в третьих вообще нельзя. Это нашло свое отражение в последующих версиях стандарта SQL: в них расширен круг обновляемых представлений и допускается значительное разнообразие правил обновлений для различных СУБД. Чтобы узнать правила обновления представлений в какой-нибудь конкретной СУБД, лучше всего обратиться к руководству пользователя или поэкспериментировать с различными типами представлений.

Контроль над обновлением представлений (CHECK OPTION)

Если представление создается посредством запроса с предложением WHERE, то в представлении будут видны только строки, удовлетворяющие условию отбора. Остальные строки могут присутствовать в исходной таблице, но быть невидимы в представлении. Например, представление EASTREPS, которое уже рассматривалось ранее в настоящей главе, содержит только строки таблицы SALESREPS с определенными значениями в столбце REP_OFFICE.

Представление, показывающее информацию о служащих только восточного региона.

```
CREATE VIEW EASTREPS AS
     SELECT *
       FROM SALESREPS
      WHERE REP_OFFICE IN (11, 12, 13);
```

Это представление является обновляемым в большинстве коммерческих СУБД. В него можно добавить информацию о новом служащем посредством инструкции INSERT.

```
INSERT INTO EASTREPS (EMPL_NUM, NAME, REP_OFFICE,
                      AGE, HIRE_DATE, SALES)
     VALUES (113, 'Jake Kimball', 11, 43,
             '2009-01-01', 0.00)$
```

СУБД добавит новую строку в исходную таблицу SALESREPS; эта строка будет видна в представлении EASTREPS. Но давайте посмотрим, что получится, если добавить строку для нового служащего с помощью следующей инструкции INSERT.

```
INSERT INTO EASTREPS (EMPL_NUM, NAME, REP_OFFICE,
                      AGE, HIRE_DATE, SALES)
     VALUES (114, 'Fred Roberts', 21, 47,
             '2009-01-01', 0.00);
```

Это вполне корректная инструкция SQL, и СУБД произведет требуемое добавление в таблицу SALESREPS. Однако новая строка не удовлетворяет условию отбора для данного представления. Значение 21 в столбце REP_OFFICE этой строки означает офис в Лос-Анджелесе, относящийся к западному региону. В результате, если сразу после инструкции INSERT выполнить запрос

```
SELECT EMPL_NUM, NAME, REP_OFFICE
  FROM EASTREPS;

EMPL_NUM    NAME         REP_OFFICE
--------    -----------  ----------
     105    Bill Adams           13
     109    Mary Jones           11
     106    Sam Clark            11
     104    Bob Smith            12
     101    Dan Roberts          12
     103    Paul Cruz            12
```

то добавленная строка в нем будет отсутствовать. То же самое произойдет, если изменить идентификатор офиса у одного из служащих, включенных в представление. Данная инструкция UPDATE

```
UPDATE EASTREPS
   SET REP_OFFICE = 21
 WHERE EMPL_NUM = 104;
```

обновляет один из столбцов в строке Боба Смита (Bob Smith), что приводит к ее немедленному исчезновению из представления. Конечно, обе исчезнувшие строки будут видны в запросе к исходной таблице.

```
SELECT EMPL_NUM, NAME, REP_OFFICE
  FROM SALESREPS;

EMPL_NUM    NAME           REP_OFFICE
--------    -------------  ----------
     105    Bill Adams             13
     109    Mary Jones             11
     102    Sue Smith              21
     106    Sam Clark              11
     104    Bob Smith              21
     101    Dan Roberts            12
     110    Tom Snyder           NULL
     108    Larry Fitch            21
     103    Paul Cruz              12
     107    Nancy Angelli          22
     114    Fred Roberts           21
```

Тот факт, что в результате выполнения инструкции INSERT или UPDATE из представления исчезают строки, в лучшем случае вызывает замешательство. Вам, вероятно, захочется, чтобы СУБД обнаруживала такие инструкции и предотвращала их выполнение. SQL позволяет организовать этот вид контроля целостности представления путем создания представлений с *проверкой*. Проверка задается в инструкции CREATE VIEW с помощью предложения WITH CHECK OPTION.

```
DROP VIEW EASTREPS;

CREATE VIEW EASTREPS AS
     SELECT *
       FROM SALESREPS
      WHERE REP_OFFICE IN (11, 12, 13)
       WITH CHECK OPTION;
```

Обратите внимание на то, что уже существующее представление должно быть удалено и создано заново для того, чтобы добавить в него проверку. Удаление представлений будет рассмотрено далее.

Когда для представления установлен режим контроля, SQL автоматически проверяет каждую операцию INSERT или UPDATE, выполняемую над представлением, чтобы удостовериться, что полученные в результате строки удовлетворяют условиям отбора в определении представления. Если добавляемая или обновляемая строка не удовлетворяет этим условиям, то выполнение инструкции INSERT или UPDATE завершается ошибкой; другими словами, операция не выполняется.

Стандарт SQL позволяет также указывать параметр проверки CASCADE или LOCAL. Этот параметр применим только к тем представлениям, в основе которых лежит не таблица базы данных, а одно или несколько других представлений. В основе исходного представления может лежать другое представление и т.д. Для каждого из представлений в такой цепочке может быть задан (или не задан) режим проверки. Если представление верхнего уровня создано с предложением WITH CASCADE CHECK OPTION, то любая попытка обновить такое представление вынудит СУБД просмотреть всю цепочку представлений нижнего уровня и проверить те из них, для которых задан режим проверки. Если же представление верхнего уровня создано с предложением WITH LOCAL CHECK OPTION, то СУБД ограничится проверкой только этого представления. Считается, что параметр CASCADE установлен по умолчанию; его можно не указывать.

Из сказанного выше должно быть понятно, что введение режима проверки приводит к значительным затратам на выполнение инструкций INSERT и UPDATE по отношению к многоуровневым представлениям. Однако он играет важную роль в обеспечении целостности базы данных. Если обновляемое представление создается с целью повысить безопасность базы данных, то следует всегда задавать режим проверки. Тогда пользователь не сможет путем модификации представления воздействовать на данные, доступ к которым ему запрещен.

Удаление представления (DROP VIEW)

Вспомним, что в стандарте SQL1 язык определения данных (DDL) рассматривается как средство статического задания структуры базы данных, включающей таблицы и представления. По этой причине в стандарте SQL1 не предусмотрена возможность удаления представления, когда в нем больше нет необходимости. Однако во всех основных СУБД такая возможность существует. Поскольку представления подобны таблицам и не могут иметь совпадающие с ними имена, во многих СУБД для удаления представлений используется инструкция DROP TABLE. В других СУБД этой же цели служит отдельная инструкция DROP VIEW.

В стандарте SQL2 было формально закреплено использование инструкции DROP VIEW для удаления представлений. В нем также детализированы правила удаления представлений, на основе которых были созданы другие представления. Предположим, например, что с помощью инструкций CREATE VIEW были созданы следующие два представления таблицы SALESREPS.

```
CREATE VIEW EASTREPS AS
     SELECT *
       FROM SALESREPS
      WHERE REP_OFFICE IN (11, 12, 13);

CREATE VIEW NYREPS AS
     SELECT *
       FROM EASTREPS
      WHERE REP_OFFICE = 11;
```

Для иллюстрации правил удаления представление NYREPS создано на основе представления EASTREPS, хотя его с таким же успехом можно было бы создать на основе исходной таблицы. Согласно стандарту SQL, следующая инструкция DROP VIEW удалит из базы данных *оба* представления.

```
DROP VIEW EASTREPS CASCADE;
```

Параметр CASCADE означает, что СУБД должна удалить не только указанное в инструкции представление, но и все представления, созданные на его основе. В противоположность этому, инструкция DROP VIEW

```
DROP VIEW EASTREPS RESTRICT;
```

не выполнится и вернет ошибку, так как параметр RESTRICT означает, что СУБД должна удалить представление только в том случае, если нет других представлений, созданных на его основе. Это служит дополнительной защитой от случайных побочных эффектов при применении инструкции DROP VIEW. Стандарт SQL требует, чтобы в инструкции DROP VIEW обязательно присутствовал или параметр RESTRICT, или CASCADE, но в большинстве коммерческих СУБД используется инструкция DROP VIEW без каких-либо явно заданных параметров. Это сделано для поддержания обратной совместимости с теми продуктами, которые были выпущены до публикации стандарта SQL2. Поведение инструкции DROP VIEW в этом случае зависит от конкретной СУБД.

Материализованные представления*

Концептуально представление — это *виртуальная* таблица базы данных. Строки и столбцы представления физически в базе данных не хранятся: они выводятся из действительных данных лежащих в основе представления исходных таблиц. Если определение представления относительно простое (например, если представление — это простое подмножество строк/столбцов единственной таблицы или простое соединение, основанное на отношении внешнего ключа), то СУБД достаточно просто транслировать операции базы данных над представлением в операции над лежащими в его основе таблицами. В этой ситуации СУБД выполняет трансляцию на лету, операция за операцией, как при обработке запросов или обновлений базы данных. В общем случае операции, обновляющие базу данных через представление (операции INSERT, UPDATE или DELETE), всегда выполняются таким способом — путем трансляции операции в одну или несколько операций над исходными таблицами.

Если определение представления более сложное, СУБД может потребоваться материализовать представление для того, чтобы выполнить запросы к нему. СУБД реально выполняет запрос, определяющий представление, и сохраняет его результаты во временной таблице базы данных. Затем СУБД выполняет затребованный запрос ко временной таблице для получения интересующих результатов. По окончании обработки запроса СУБД уничтожает временную таблицу. На рис. 14.6 показан описанный процесс материализации. Понятно, что материализация содержимого представления может быть очень затратной операцией. Если типичная нагрузка на базу данных содержит много запросов, требующих материализации представлений, то общая пропускная способность СУБД может резко снизиться.

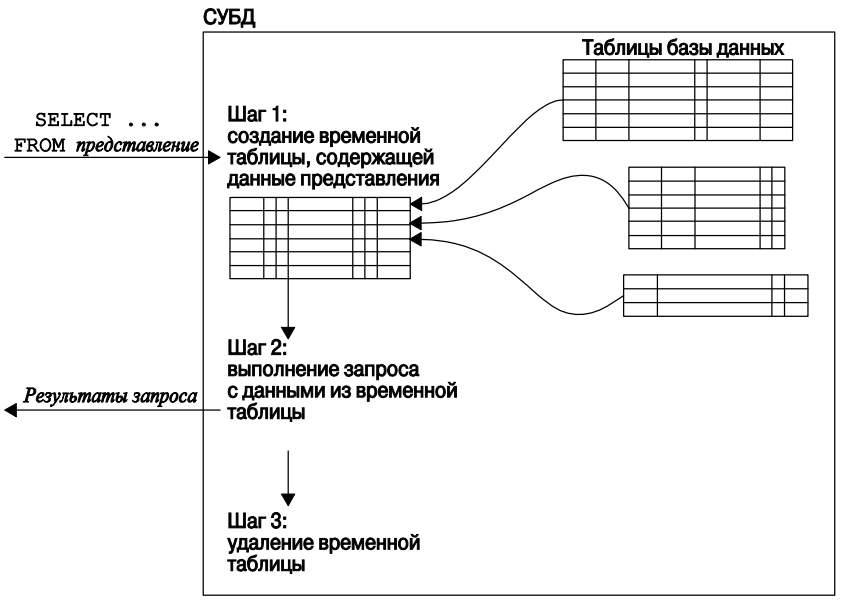

Рис. 14.6. Материализация представления для выполнения запроса

Для решения этой проблемы некоторые коммерческие СУБД поддерживают *материализованные представления*. Когда вы определяете представление как материализованное, СУБД выполняет определяющий его запрос (обычно в момент определения материализованного представления), сохраняет его результаты (т.е. данные, составляющие представление) в базе данных и постоянно поддерживает эту копию данных представления. Для поддержания точности данных материализованного представления СУБД должна автоматически проверять каждое изменение данных в исходных таблицах и выполнять соответствующие изменения в данных материализованного представления. В некоторых СУБД обновления материализованных представлений выполняются при обновлениях исходных таблиц, но более распространены СУБД, в которых изменения таблиц заносятся в журнал и применяются к материализованному представлению по расписанию, через определенные промежутки времени. Когда СУБД должна выполнить запрос к материализованному представлению, его данные уже готовы и запрос обрабатывается достаточно эффективно. На рис. 14.7 показана операция СУБД с материализованным представлением.

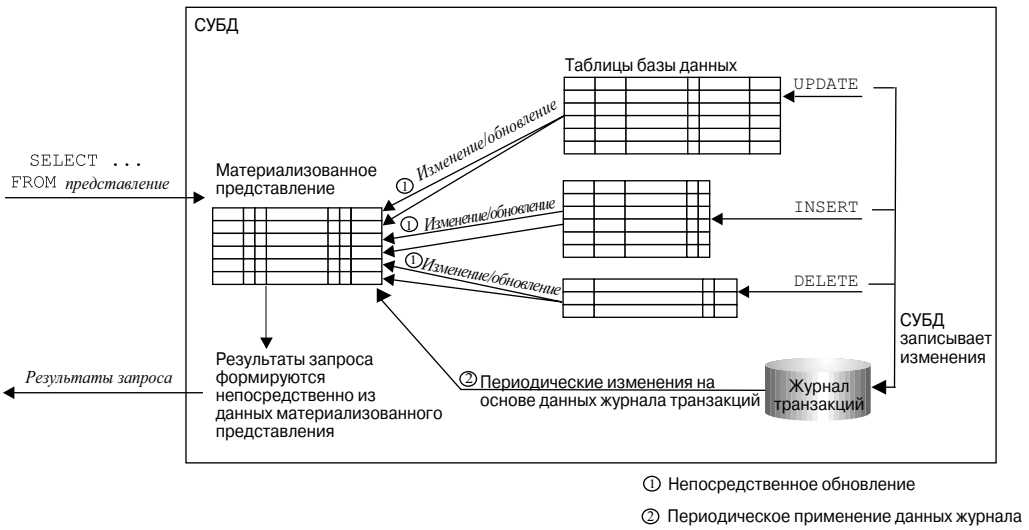

Рис. 14.7. Операция с материализованным представлением

Материализованные представления — это компромисс между эффективностью обновлений данных, содержащихся в представлении, и эффективностью запросов к данным представления. В случае нематериализованного представления на обновления исходных таблиц никак не влияет наличие представлений; они обрабатываются с обычной для СУБД скоростью. Однако запросы к нематериализованным представлениям могут быть гораздо менее эффективными по сравнению с запросами к обычным таблицам базы данных, поскольку СУБД должна на лету обработать запрос, что может потребовать выполнения большого количества работы.

Материализованные представления делают описанное соотношение обратным. При определении материализованного представления обновления исходных таблиц оказываются существенно менее эффективными, чем обновление обычных таблиц. Это связано с тем, что СУБД должна вычислить влияние обновления и соответствующим образом обновить данные материализованного представления. Однако запросы к материализованным представлениям могут обрабатываться с той же скоростью, что и запросы к реальным таблицам базы данных, поскольку материализованное представление есть не что иное, как обычная таблица. Таким образом, материализованное представление наиболее выгодно в том случае, когда количество обновлений лежащих в основе представления данных относительно мало, а количество запросов, напротив, велико.

Резюме

Представления дают возможность переопределять структуру базы данных, позволяя каждому пользователю видеть свою собственную структуру и свою часть содержимого базы данных.

- Представление — это виртуальная таблица, созданная на основе запроса. Представление, как и реальная таблица, содержит строки и столбцы данных, однако данные, видимые в представлении, на самом деле являются результатами запроса.
- Представление может быть простым подмножеством строк и столбцов одной таблицы, может резюмировать содержимое таблицы (сгруппированное представление) или содержать данные из двух или более таблиц (соединенное представление).
- В инструкциях `SELECT`, `INSERT`, `DELETE` и `UPDATE` к представлению можно обращаться, как к обычной таблице. Однако более сложные представления обновлять нельзя, они доступны только для чтения.
- Представления обычно используются для упрощения видимой структуры базы данных и запросов, а также для защиты некоторых строк и столбцов от несанкционированного доступа.
- Материализованные представления могут повысить эффективность работы базы данных в случае высокой активности запросов и низкой активности обновлений.