1. Объединение запросов

Использование предложения UNION

В ПРЕДШЕСТВУЮЩИХ ЛАБАХ МЫ ОБСУЖДАЛИ различные способы, которыми запросы могут помещаться один внутрь другого. Имеется другой способ объединения многочисленых запросов — т.е. формирование их в объединение. В этой Лабе вы научитесь использованию предложения **UNION** в SQL. **UNION** отличается от подзапросов тем что в нем ни один из двух (или больше) запросов не управляются другим запросом. Все запросы выполняются независимо друг от друга, а уже вывод их — объединяется.

Объединение нескольких запросов в один

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

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

и получить вывод, показанный в Рисунке 14.1.

Как вы можете видеть, столбцы, выбранные двумя командами, выведены так, как если бы она была одна. Заголовки столбца исключены, потому что ни один из столбцов выведенных объединением, не был извлечен непосредственно из только одной таблицы. Следовательно, все эти столбцы вывода не имеют никаких имен.

Кроме того, обратите внимание, что только последний запрос заканчивается точкой с запятой. Отсутствие точки с запятой дает понять SQL, что имеется еще одно или более запросов.

```
 ===============  SQL Execution Log ============
|                                               |
| SELECT snum, sname                            |
| FROM  Salespeople                             |
| WHERE city = 'London'                         |
| UNION                                         |
| SELECT cnum, cname                            |
| FROM Customers                                |
| WHERE city = 'London';                        |
| ============================================= |
|                                               |
|  -----    --------                            |
|   1001    Peel                                |
|   1004    Motika                              |
|   2001    Hoffman                             |
|   2006    Clemens                             |
|                                               |
 ===============================================
```

Рисунок 14.1. Формирование объединения из двух запросов.

Когда вы можете делать объединение между запросами?

Когда два (или более) запроса подвергаются объединению, их столбцы вывода должны быть совместимы для объединения. Это означает, что каждый запрос должен указывать одинаковое число столбцов и в том же порядке что и первый, второй, третий, и так далее, и каждый должен иметь тип, совместимый с каждым. Значение совместимости типов — меняется. **ANSI** следит за этим очень строго и поэтому числовые поля должны иметь одинаковый числовой тип и размер, хотя некоторые имена, используемые **ANSI** для этих типов, являются синонимами (смотрите

Приложение B для подробностей о числовых типах **ANSI**). Кроме того, символьные поля должны иметь одинаковую длину.

Хорошо, что некоторые SQL программы обладают большей гибкостью, чем это определяется **ANSI**. Типы, не определенные **ANSI**, такие как DATA и BINARY, обычно должны совпадать с другими столбцами такого же нестандартного типа.

Длина строки также может стать проблемой. Большинство программ разрешают поля переменной длины, но они не обязательно будут использоваться с **UNION**. С другой стороны, некоторые программы (и **ANSI** тоже), требуют, чтобы символьные поля были точно равной длины. В этих вопросах вы должны проконсультироваться с документацией вашей собственной программы.

Другое ограничение на совместимость — это когда пустые значения (**NULL**) запрещены в любом столбце объединения, причем эти значения необходимо запретить и для всех соответствующих столбцов в других запросах объединения. Пустые значения (**NULL**) запрещены с ограничением NOT **NULL**. Кроме того, вы не можете использовать **UNION** в подзапросах, а также не можете использовать агрегатные функции в предложении SELECT запроса в объединении. (Большинство программ пренебрегают этими ограничениями.)

UNION и устранение дубликатов

UNION будет автоматически исключать дубликаты строк из вывода. Это нечто несвойственное для SQL, так как одиночные запросы обычно содержат DISTINCT, чтобы устранять дубликаты. Например, запрос, чей вывод показывается в Рисунке 14.2,

```
SELECT snum, city
FROM Customers;
```

имеет двойную комбинацию значений (snum=1001, city='London'), потому что мы не указали, чтобы SQL устранил дубликаты. Однако, если мы используем **UNION** в комбинации этого запроса с ему подобным в таблице Продавцов, то эта избыточная комбинация будет устранена. Рисунок 14.3 показывает вывод следующего запроса.

```
SELECT snum, city
FROM Customers

UNION

SELECT snum, city
FROM Salespeople;
```

```
 ===============  SQL Execution Log ============
| SELECT snum, city                             |
| FROM Customers;                               |
| ============================================= |
|  snum     city                                |
|  -----    --------                            |
|   1001    London                              |
|   1003    Rome                                |
|   1002    San Jose                            |
|   1002    Berlin                              |
|   1001    London                              |
|   1004    Rome                                |
|   1007    San Jose                            |
 ===============================================
```

Рисунок 14.2. Одиночный запрос с дублированным выводом.

```
 ===============  SQL Execution Log ============
| FROM Customers                                 |
| UNION                                          |
| SELECT snum, city                              |
| FROM Salespeople;                              |
| ============================================== |
|                                                |
|  -----    --------                             |
|   1001    London                               |
|   1002    Berlin                               |
|   1007    San Jose                             |
|   1007    New York                             |
|   1003    Rome                                 |
|   1001    London                               |
|   1003    Rome                                 |
|   1002    Barcelona                            |
|   1007    San Jose                             |
 ================================================
```

Рисунок 14.3. **UNION** устраняет двойной вывод.

Вы можете получить нечто похожее в некоторых программах SQL, используя **UNION** ALL вместо просто оператора **UNION**, наподобие этого:

```
SELECT snum, city
FROM Customers

UNION ALL

SELECT snum, city
FROM Salespeople;
```

Использование строк и выражений с UNION

Иногда, вы можете вставлять константы и выражения в предложения SELECT, используемые с **UNION**. Это не следует строго указаниям **ANSI**, но это полезная и необычно используемая возможность. Константы и выражения, которые вы используете, должны встречать совместимые стандарты, которые мы выделяли ранее. Эта свойство полезно, например, чтобы устанавливать комментарии, указывающие, какой запрос вывел данную строку.

Предположим, что вы должны сделать отчет о том, какие продавцы производят наибольшие и наименьшие Заказы по датам. Мы можем объединить два запроса, вставив туда текст, чтобы различать вывод для каждого из них.

```
SELECT a.snum, sname, onum, 'Highest on', odate
FROM Salespeople a, Orders b
WHERE a.snum = b.snum AND b.amt = (SELECT MAX (amt)
                                   FROM Orders c
                                   WHERE c.odate = b.odate)

UNION

SELECT a.snum, sname, onum, 'Lowest on ', odate
FROM Salespeople a, Orders b
WHERE a.snum = b.snum AND b.amt = (SELECT  MIN (amt)
                                   FROM Orders c
                                   WHERE c.odate = b.odate);
```

Вывод из этой команды показывается в Рисунке 14.4.

Мы должны были добавить дополнительный пробел в строку 'Lowest on', чтобы сделать ее совпадающей по длине со строкой 'Highest on'. Обратите внимание, что Peel выбран при наличии и самого высокого и самого низкого (фактически, он единственный) Заказа на 5 Октября. Так как вставляемые строки двух этих запросов различны, строки не будут устранены как дубликаты.

```
 ===============  SQL Execution Log ============
| AND b.amt =                                    |
| (SELECT min (amt)                              |
| FROM Orders c                                  |
| WHERE c.odate = b.odate);                      |
| ============================================== |
|                                                |
| -----  ------- ------  ---------- ----------- |
|  1001  Peel     3008   Highest on  10/05/1990 |
|  1001  Peel     3008   Lowest  on  10/05/1990 |
|  1001  Peel     3011   Highest on  10/06/1990 |
|  1002  Serres   3005   Highest on  10/03/1990 |
|  1002  Serres   3007   Lowest  on  10/04/1990 |
|  1002  Serres   3010   Lowest  on  10/06/1990 |
|  1003  Axelrod  3009   Highest on  10/04/1990 |
|  1007  Rifkin   3001   Lowest  on  10/03/1990 |
 ================================================
```

Рисунок 14.4. Выбор наибольших и наименьших Заказов.

Использование UNION с ORDER BY

До сих пор мы не оговаривали, что данные многочисленных запросов будут выводиться в каком то особом порядке. Мы просто показывали вывод сначала из одного запроса, а затем из другого. Конечно, вы не можете полагаться на вывод, приходящий в произвольном порядке. Мы как раз сделаем так, чтобы этот способ для выполнения примеров был более простым. Вы можете использовать предложение ORDER BY чтобы упорядочить вывод из объединения, точно так же как это делается в индивидуальных запросах. Давайте пересмотрим наш последний пример чтобы упорядочить имена с помощью их Заказовых номеров. Это может внести противоречие, такое как повторение имени Peel в последней команде, как вы сможете увидеть из вывода показанного в Рисунке 14.5.

```
SELECT a.snum, sname, onum, 'Highest on', odate
FROM Salespeople a, Orders b
WHERE a.snum = b.snum AND b.amt = (SELECT MAX (amt)
                                   FROM Orders c
                                   WHERE c.odate = b.odate)

UNION

SELECT a.snum, sname, onum, 'Lowest  on', odate
FROM Salespeople a, Orders b
WHERE a.snum = b.snum AND b.amt = (SELECT MIN (amt)
                                   FROM Orders c
                                   WHERE c.odate = b.odate)
ORDER BY 3;
```

```
 ===============  SQL Execution Log ============
| (SELECT min (amt)                              |
| FROM Orders c                                  |
| WHERE c.odate = b.odate)                       |
| ORDER BY 3;                                    |
| ============================================== |
|                                                |
| -----  ------- ------  ---------- ----------- |
|  1007  Rifkin   3001   Lowest  on  10/03/1990 |
|  1002  Serres   3005   Highest on  10/03/1990 |
|  1002  Serres   3007   Lowest  on  10/04/1990 |
|  1001  Peel     3008   Highest on  10/05/1990 |
|  1001  Peel     3008   Lowest  on  10/05/1990 |
|  1003  Axelrod  3009   Highest on  10/04/1990 |
|  1002  Serres   3010   Lowest  on  10/06/1990 |
|  1001  Peel     3011   Highest on  10/06/1990 |
 ================================================
```

Рисунок 14.5. Формирование объединения с использованием ORDER BY.

Пока ORDER BY используется по умолчанию, мы не должны его указывать. Мы можем упорядочить наш вывод с помощью нескольких полей, одно внутри другого и указать ASC или DESC для каждого, точно также как мы делали это для одиночных запросов. Заметьте, что номер 3 в предложении ORDER BY указывает, какой столбец из предложения SELECT будет упорядочен. Так как столбцы объединения — это столбцы вывода, они не имеют имен, и, следовательно, должны определяться по номеру. Этот номер указывает на их место среди других столбцов вывода (смотрите Лабу 5, обсуждающую столбцы вывода).

Внешнее объединение

Операция, которая бывает часто полезна — это объединение из двух запросов, в котором второй запрос выбирает строки, исключенные первым. Наиболее часто вы будете делать это так, чтобы не исключать строки, которые не удовлетворили предикату при объединении таблиц. Это называется *внешним объединением*.

Предположим, что некоторые из ваших заказчиков еще не были назначены к продавцам. Вы можете захотеть увидеть имена и города всех ваших заказчиков, с именами их продавцов, не учитывая тех, кто еще не был назначен. Вы можете достичь этого, формируя объединение из двух запросов, один из которых выполняет объединение, а другой выбирает заказчиков с пустыми (**NULL**) значениями поля snum. Этот последний запрос должен вставлять пробелы в поля, соответствующие полю sname в первом запросе.

Как и раньше, вы можете вставлять текстовые строки в ваш вывод, чтобы идентифицировать запрос, который вывел данную строку. Использование этой методики во внешнем объединении дает возможность использовать предикаты для классификации, а не для исключения. Мы использовали пример нахождения продавцов с заказчиками, размещенными в их городах и раньше. Однако вместо просто выбора только этих строк, вы, возможно, захотите, чтобы ваш вывод перечислял всех продавцов, и указывал тех, кто не имел заказчиков в их городах, и кто имел. Следующий запрос, чей вывод показывается в Рисунке 14.6, выполнит это:

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

UNION

SELECT snum, sname, 'NO MATCH      ', comm
FROM Salespeople
WHERE NOT city = ANY (SELECT city
                      FROM Customers)
ORDER BY 2 DESC;
```

```
 ===============  SQL Execution Log ============
| FROM Salespeople                              |
| WHERE NOT city = ANY (SELECT city             |
|                       FROM Customers)         |
| ORDER BY 2 DESC;                              |
| ============================================= |
|                                               |
| -----  -------  ---------     ------------    |
|  1002  Serres   Cisneros           0.1300     |
|  1002  Serres   Liu                0.1300     |
|  1007  Rifkin   NO MATCH           0.1500     |
|  1001  Peel     Clemens            0.1200     |
|  1001  Peel     Hoffman            0.1200     |
|  1004  Motika   Clemens            0.1100     |
|  1004  Motika   Hoffman            0.1100     |
|  1003  Axelrod  NO MATCH           0.1000     |
 ===============================================
```

Рисунок 14.6. Внешнее обьединение.

Строка 'NO MATCH' была дополнена пробелами, чтобы получить совпадение поля cname по длине (это не обязательно во всех реализациях SQL). Второй запрос выбирает даже те строки, которые исключил первый. Вы можете также добавить комментарий или выражение к вашему запросу, в виде дополнительного поля. Если вы сделаете это, вы будете должны добавить некоторый дополнительный комментарий или выражение в той же самой позиции среди выбранных полей, для каждого запроса в операции объединения. Совместимость **UNION**

предотвращает вас от добавления дополнительного поля для первого запроса, но не для второго. Имеется запрос, который добавляет строки к выбранным полям, и указывает, совпадает ли данный продавец с его заказчиком в его городе:

```
SELECT a.snum, sname, a.city, 'MATCHED '
FROM Salespeople a, Customers b
WHERE a.city = b.city

UNION

SELECT snum, sname, city, 'NO MATCH'
FROM Salespeople
WHERE NOT city = ANY (SELECT city
                      FROM Customers)
ORDER BY 2 DESC;
```

Рисунок 14.7 показывает вывод этого запроса.

```
 ===============  SQL Execution Log ============
| WHERE a.city = b.city                          |
| union                                          |
| SELECT snum,sname,city, 'NO MATCH'             |
| FROM Salespeople                               |
| WHERE NOT city = (SELECT city                  |
| FROM Customers)                                |
| ORDER BY 2 DESC;                               |
| ============================================== |
|                                                |
| -----  -------   ------------  ---------       |
|  1002  Serres     San Jose     MATCHED         |
|  1007  Rifkin     Barselona    NO MATCH        |
|  1001  Peel       London       MATCHED         |
|  1004  Motika     London       MATCHED         |
|  1003  Axelrod    New York     NO MATCH        |
 ================================================
```

Рисунок 14.7. Внешнее обьединение с полем коментария.

Это не полное внешнее объединение, так как оно включает только несовпадающие поля одной из объединяемых таблиц. Полное внешнее объединение должно включать всех заказчиков, имеющих и не имеющих продавцов в их городах. Такое условие будет более полным, как вы это сможете увидеть (вывод следующего запроса показан на Рисунке 14.8):

```
SELECT snum, city, 'SALESPERSON — MATCH   '
FROM Salespeople
WHERE NOT city = ANY (SELECT city
                      FROM Customers)

UNION

SELECT snum, city, 'SALESPERSON — NO MATCH'
FROM Salespeople
WHERE NOT city = ANY (SELECT city
                      FROM Customers)

UNION

SELECT cnum, city, 'CUSTOMER — MATCHED    '
FROM Customers
WHERE city = ANY (SELECT city
                  FROM Salespeople)

UNION

SELECT cnum, city, 'CUSTOMER — NO MATCH   '
FROM Customers
WHERE NOT city = ANY (SELECT city
                      FROM Salespeople)

ORDER BY 2 DESC;
```

```
 ================  SQL Execution Log =============
| FROM  Salespeople)                                |
| ORDER BY 2 DESC;                                  |
| ================================================ |
|                                                   |
|  ----   --------     ------------------------     |
|  2003   San Jose     CUSTOMER     —   MATCHED     |
|  2008   San Jose     CUSTOMER     —   MATCHED     |
|  2002   Rome         CUSTOMER     —   NO MATCH    |
|  2007   Rome         CUSTOMER     —   NO MATCH    |
|  1003   New York     SALESPERSON  —   MATCHED     |
|  1003   New York     SALESPERSON  —   NO MATCH    |
|  2001   London       CUSTOMER     —   MATCHED     |
|  2006   London       CUSTOMER     —   MATCHED     |
|  2004   Berlin       CUSTOMER     —   NO MATCH    |
|  1007   Barcelona    SALESPERSON  —   MATCHED     |
|  1007   Barcelona    SALESPERSON  —   NO MATCH    |
 ================================================
```

Рисунок 14.8. Полное внешнее обьединение.

Понятно, что эта формула, использующая ANY, эквивалентна объединению в предыдущем примере.

Сокращенное внешнее объединение, с которого мы начинали, используется чаще, чем этот последний пример. Этот пример, однако, имеет другой смысл. Всякий раз, когда вы выполняете объединение более чем двух запросов, вы можете использовать круглые скобки, чтобы определить порядок оценки. Другими словами, вместо просто —

```
query X UNION query Y UNION query Z;
```

вы должны указать, или

```
(query X UNION query Y)UNION query Z;
```

или

```
query X UNION (query Y UNION query Z);
```

Это потому, что **UNION** и **UNION** ALL могут быть скомбинированы, чтобы удалять одни дубликаты, не удаляя других. Предложение

```
(query X UNION ALL query Y)UNION query Z;
```

не обязательно воспроизведет те же результаты что предложение

```
query X UNION ALL(query Y UNION query Z);
```

если двойные строки в нем будут удалены.

Резюме

Теперь вы знаете, как использовать предложение **UNION**, которое дает возможность объединять любое число запросов в единое тело вывода. Если вы имеете ряд подобных таблиц (таблиц, содержащих похожую информацию), но принадлежащую разным пользователям и охватывающую различные особенности, возможно, что объединение сможет обеспечить простой способ для слияния и упорядочивания вывода. Аналогично, внешние объединения дают вам новый способ использования условий, не для исключения вывода, а для его маркировки или обработки его частей, когда встречается условие отличающееся от того, которое не выполняется.

Вы теперь имеете довольно полное представление о поиске данных в SQL. Следующий шаг должен включать то, как значения вводятся в таблицы и как таблицы создаются с самого начала. Как вы увидите, запросы иногда используются внутри других типов команд, так же хорошо, как и сами по себе.

2. Создание представлений

Введение в представление
Однотабличное представление – модифицируемое

Многотабличное немодифицируемое

Введение в представления

ПРЕДСТАВЛЕНИЕ **(VIEW**) — ОБЪЕКТ ДАННЫХ, КОТОРЫЙ не содержит никаких данных его владельца. Это — тип таблицы, чье содержание выбирается из других таблиц с помощью выполнения запроса. Поскольку значения в этих таблицах меняются, то автоматически, их значения могут быть показаны представлением.

Вы узнаете, что такое представления, как они создаются и немного об их возможностях и ограничениях. Использование представлений, основанных на улучшенных средствах запросов, таких как объединение и подзапрос, разработанных очень тщательно, в некоторых случаях даст больший выигрыш по сравнению с запросами.

Что такое представление?

Типы таблиц, с которыми вы имели дело до сих пор, назывались — *базовыми таблицами*. Это — таблицы, которые содержат данные. Однако имеется другой вид таблиц — представления. *Представления* — это таблицы, чье содержание выбирается или получается из других таблиц. Они работают в запросах и операторах DML точно также как и основные таблицы, но не содержат никаких собственных данных.

Представления подобны окнам, через которые вы просматриваете информацию (как она есть, или в другой форме, как вы потом увидите), которая фактически хранится в базовой таблице. *Представление* — это фактически запрос, который выполняется всякий раз, когда представление становится темой команды. Вывод запроса при этом в каждый момент становится содержанием представления.

Команда CREATE VIEW

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

```
CREATE VIEW Londonstaff
AS SELECT *
  FROM Salespeople
  WHERE city = 'London';
```

Теперь Вы имеете представление, называемое **Londonstaff**. Вы можете использовать это представление точно так же, как и любую другую таблицу. Она может быть запрошена, модифицирована, вставлена в, удалена из, и соединена с, другими таблицами и представлениями. Давайте сделаем запрос такого представления (вывод показан в Рисунке 20.1):

```
Select *
FROM Londonstaff;
```

```
 ===============  SQL Execution Log ============
| SELECT *                                       |
| FROM  Londonstaff;                             |
| ===============================================|
|   snum      sname         city         comm    |
| ------    ----------   -----------   -------   |
|   1001      Peel         London       0.1200   |
|   1004      Motika       London       0.1100   |
 ================================================
```

Рисунок 20.1. Представление Londonstaff

Когда вы приказываете SQL выбрать (**SELECT**) все строки (*****) из представления, он выполняет запрос, содержащийся в определении Londonstaff, и возвращает все из его вывода.

Имея предикат в запросе представления, можно вывести только те строки из представления, которые будут удовлетворять этому предикату. Преимущество использования представления, по сравнению с использованием основной таблицы в том, что представление будет модифицировано автоматически всякий раз, когда таблица, лежащая в его основе изменяется.

Содержание представления не фиксировано, и переназначается каждый раз, когда вы ссылаетесь на представление в команде. Если вы добавите завтра другого, живущего в Лондоне продавца, он автоматически появится в представлении.

Представления значительно расширяют управление вашими данными. Это превосходный способ дать публичный доступ к некоторой, но не всей информации в таблице. Если вы хотите, чтобы ваш продавец был показан в таблице Продавцов, но при этом не были показаны комиссии других продавцов, вы могли бы создать представление с использованием следующего оператора (вывод показан в Рисунке 20.2)

```
CREATE VIEW Salesown
AS SELECT snum, sname, city
   FROM Salespeople:
```

```
 ===============  SQL Execution Log ============
| SELECT *                                      |
| FROM  Salesown;                               |
| ==============================================|
|   snum      sname         city                |
| ------    ----------   -----------            |
|   1001      Peel         London               |
|   1002      Serres       San Jose             |
|   1004      Motika       London               |
|   1007      Rifkin       Barcelona            |
|   1003      Axelrod      New York             |
 ================================================
```

Рисунок 20.2. Представление Salesown

Другими словами, это представление — такое же, как для таблицы Продавцов, за исключением того, что поле comm не упоминалось в запросе, и, следовательно, не было включено в представление.

Модифицирование представлений

Представление может теперь изменяться командами модификации DML, но модификация не будет воздействовать на само представление. Команды будут на самом деле перенаправлены к базовой таблице:

```
UPDATE Salesown
SET city = 'Palo Alto'
WHERE snum = 1004;
```

Его действие идентично выполнению той же команды в таблице Продавцов. Однако, если значение комиссионных продавца будет обработано командой UPDATE

```
UPDATE Salesown
SET comm = .20
WHERE snum = 1004;
```

она будет отвергнута, так как поле comm отсутствует в представлении Salesown. Это важное замечание, показывающее, что не все представления могут быть модифицированы.

Именование столбцов

В нашем примере, поля наших представлений имеют свои имена, полученные прямо из имен полей основной таблицы. Это удобно. Однако, иногда вам нужно снабжать ваши столбцы новыми именами:

- когда некоторые столбцы являются выводимыми и поэтому не имеющими имен;
- когда два или более столбцов в объединении имеют те же имена, что в их базовой таблице.

Имена, которые могут стать именами полей, даются в круглых скобках после имени таблиц. Они не будут запрошены, если совпадают с именами полей запрашиваемой таблицы. Тип данных и размер этих полей будут отличаются от типа данных и размера запрашиваемых полей, которые "передаются" в них. Обычно вы не указываете новых имен полей, но если вы все-таки сделали это, вы должны делать это для каждого поля в представлении.

Комбинирование предикатов представлений и основных запросов в представлениях

Когда вы делаете запрос представления, вы собственно, запрашиваете запрос. Основной способ для SQL обойти это, — объединить предикаты двух запросов в один. Давайте посмотрим еще раз на наше представление с именем Londonstaff:

```
CREATE VIEW Londonstaff
AS SELECT *
FROM Salespeople
WHERE city = 'London';
```

Если мы выполняем следующий запрос в этом представлении

```
SELECT *
FROM Londonstaff
WHERE comm > .12;
```

он такой же, как если бы мы выполнили следующее в таблице Продавцов:

```
SELECT *
FROM Salespeople
WHERE city = 'London' AND comm > .12;
```

Это прекрасно, за исключением того, что появляется возможная проблема с представлением. Имеется возможность комбинации из двух полностью допустимых предикатов и получения предиката, который не будет работать. Например, предположим, что мы создаем (**CREATE**) следующее представление:

```
CREATE VIEW Ratingcount (rating, number)
AS SELECT rating, COUNT (*)
   FROM Customers
   GROUP BY rating;
```

Это дает нам число заказчиков, которые мы имеем для каждого уровня оценки rating. Вы можете затем сделать запрос этого представления, чтобы выяснить, имеется ли какая-нибудь оценка, в настоящее время назначенная для трех заказчиков:

```
SELECT *
FROM Ratingcount
WHERE number = 3;
```

Посмотрим, что случится, если мы скомбинируем два предиката:

```
SELECT rating, COUNT (*)
FROM Customers
WHERE COUNT (*) = 3
GROUP BY rating;
```

Это недопустимый запрос. Агрегатные функции, такие как COUNT (*СЧЕТ*), не могут использоваться в предикате. Правильным способом при формировании вышеупомянутого запроса, конечно же, будет следующий:

```
SELECT rating, COUNT (*)
FROM Customers
GROUP BY rating;
HAVING COUNT (*) = 3;
```

Но SQL может не выполнить превращения. Может ли равноценный запрос вместо запроса Ratingcount потерпеть неудачу? Да, может! Это — неоднозначная область SQL, где методика использования представлений может дать хорошие результаты.

Самое лучшее, что можно сделать в случае, когда об этом ничего не сказано в вашей системной документации, так это попытка в ней разобраться. Если команда допустима, вы можете использовать представления, чтобы установить некоторые ограничения SQL в синтаксисе запроса.

Групповые представления

Групповые представления — это представления, наподобие запроса **Ratingcount** в предыдущем примере, который содержит предложение GROUP BY, или который основывается на других групповых представлениях.

Групповые представления могут стать превосходным способом обрабатывать полученную информацию непрерывно. Предположим, что каждый день вы должны следить за порядком

номеров заказчиков, номерами продавцов, принимающих Заказы, номерами Заказов, средним от Заказов, и общей суммой приобретений в Заказах.

Вместо того, чтобы конструировать каждый раз сложный запрос, вы можете просто создать следующее представление:

```
CREATE VIEW Totalforday
AS SELECT odate, COUNT(DISTINCT cnum), COUNT(DISTINCT snum),
          COUNT(onum), AVG(amt), SUM(amt)
FROM Orders
GROUP BY odate;
```

Теперь вы сможете увидеть всю эту информацию с помощью простого запроса:

```
SELECT *
FROM Totalforday;
```

Как мы видели, SQL запросы могут дать вам полный комплекс возможностей, так что представления обеспечивают вас чрезвычайно гибким и мощным инструментом, чтобы определить точно, как ваши данные могут быть использованы.

Они могут также делать вашу работу более простой, переформатируя данные удобным для вас способом и исключив двойную работу.

Представления и объединения

Представления не требуют, чтобы их вывод осуществлялся из одной базовой таблицы. Так как почти любой допустимый запрос SQL может быть использован в представлении, он может выводить информацию из любого числа базовых таблиц, или из других представлений.

Мы можем, например, создать представление, которое показывало бы Заказы продавца и заказчика по имени:

```
CREATE VIEW Nameorders
AS SELECT onum, amt, a.snum, sname, cname
   FROM Orders a, Customers b, Salespeople c
   WHERE a.cnum = b.cnum AND a.snum = c.snum;
```

Теперь вы можете выбрать все Заказы заказчика или продавца, или можете увидеть эту информацию для любого Заказа.

Например, чтобы увидеть все Заказы продавца Rifkin, вы должны ввести следующий запрос (вывод показан в 20.3 Рисунке):

```
SELECT *
FROM Nameorders
WHERE sname = 'Rifkin';
```

```
 ===============  SQL Execution Log ==============
| SELECT *                                        |
| FROM  Nameorders                                |
| WHERE sname = 'Rifkin';                         |
| ================================================ |
|   onum       amt       snum   sname     cname   |
|  ------   --------    -----  -------   -------  |
|   3001       18.69     1007  Rifkin    Cisneros |
|   3006     1098.16     1007  Rifkin    Cisneros |
 ================================================
```

Рисунок 20.3. Заказы Rifkin показаные в Nameorders

Вы можете также объединять представления с другими таблицами, или базовыми таблицами или представлениями, поэтому вы можете увидеть все Заказы Axelrod и значения его комиссионных в каждом Заказе:

```
SELECT a.sname, cname, amt  comm
FROM Nameorders a, Salespeople b
WHERE a.sname = 'Axelrod' AND b.snum = a.snum;
```

Вывод для этого запроса показывается в Рисунке 20.4.

В предикате, мы могли бы написать — "WHERE a.sname = 'Axelrod' AND b.sname = 'Axelrod'", но предикат, который мы использовали здесь, более общеупотребительный. Кроме того, поле snum — это первичный ключ таблицы Продавцов и, следовательно, должен по определению быть уникальным.

```
 ===============  SQL Execution Log ==============
| SELECT a.sname, cname, amt * comm                |
| FROM  Nameorders a, Salespeople b                |
| WHERE a.sname = 'Axelrod' AND b.snum = a.snum;   |
| ================================================ |
|   onum       amt       snum   sname     cname    |
|  ------   --------    -----  -------   -------   |
|   3001       18.69     1007  Rifkin    Cisneros  |
|   3006     1098.16     1007  Rifkin    Cisneros  |
 ================================================
```

Рисунок 20.4. Объединение основной таблицы с представлением

Если бы там, например, было два Axelrod, вариант с именем будет объединять вместе их данные. Более предпочтительный вариант — использовать поле snum, чтобы хранить его отдельно.

Представления и подзапросы

Представления могут также использовать и подзапросы, включая соотнесенные подзапросы. Предположим, ваша компания предусматривает премию для тех продавцов, которые имеют заказчика с самой высокой суммой Заказа для любой указанной даты. Вы можете проследить эту информацию с помощью представления:

```
CREATE VIEW Elitesalesforce
AS SELECT b.odate, a.snum, a.sname,
   FROM Salespeople a, Orders b
   WHERE a.snum = b.snum AND b.amt = (SELECT MAX (amt)
                                      FROM Orders c
                                      WHERE c.odate = b.odate);
```

Если, с другой стороны, премия будет назначаться только продавцу, который имел самую высокую сумму Заказа за последние десять лет, вам необходимо будет проследить их в другом представлении, основанном на первом:

```
CREATE VIEW Bonus
AS SELECT DISTINCT snum, sname
   FROM Elitesalesforce a
   WHERE 10 <= (SELECT COUNT (*)
                FROM Elitesalestorce b
                WHERE a.snum = b.snum);
```

Извлечение из этой таблицы продавца, который будет получать премию, выполняется простым запросом:

```
SELECT *
FROM Bonus;
```

Теперь мы видим истинную мощность SQL. Извлечение той же полученной информации программами RPG или COBOL будет более длительной процедурой. В SQL это только запрос из двух комплексных команд, сохраненных как представление совместно с простым запросом.

При самостоятельном запросе мы должны заботится об этом каждый день, потому что информация, которую извлекает запрос, непрерывно меняется, чтобы отражать текущее состояние базы данных.

Чего не могут делать представления

Имеются большое количество типов представлений (включая многие из наших примеров здесь), которые являются доступными только для чтения. Это означает, что их можно запрашивать, но они не могут подвергаться действиям команд модификации. Мы будем рассматривать эту тему далее.

Имеются также некоторые виды запросов, которые не допустимы в определениях представлений. Одиночное представление должно основываться на одиночном запросе; *объединение* (**UNION**) и *объединение всего* (**UNION ALL**) не разрешаются. Упорядочение по **ORDER BY** никогда не используется в определении представлений. Вывод запроса формирует содержание представления, которое напоминает базовую таблицу и является по определению неупорядоченным.

Удаление представлений

Синтаксис удаления представления из базы данных подобен синтаксису удаления базовых таблиц:

```
DROP VIEW <view name>
```

В этом нет необходимости, однако, сначала надо удалить все содержание, как это делается с базовой таблицей, потому что содержание представления не является созданным и сохраняется в течение определенной команды. Базовая таблица, из которой представление выводится, не эффективна, когда представление удалено.

Помните, вы должны являться владельцем представления, чтобы иметь возможность удалить его.

3. Изменение значений с помощью представлений

Здесь рассказывается о командах модификации языка DML — *вставить* (**INSERT**), *изменить* (**UPDATE**), и *удалить* (**DELETE**), когда они применяются для представлений. Использование команд модификации в представлениях — это косвенный способ использования их в ссылочных таблицах с помощью запросов представлений. Однако не все представления могут модифицироваться.

В этом разделе, мы будем обсуждать правила, определяющие, является ли представление модифицируемым. Кроме того, вы обучитесь использованию предложения **WITH CHECK OPTION**, которое управляет указанными значениями, которые можно вводить в таблицу с помощью представления.

Модифицирование представления

Один из наиболее трудных и неоднозначных аспектов представлений — непосредственное их использование с командами модификации DML. Эти команды фактически воздействуют на значения в базовой таблице представления.

Это является некоторым противоречием. Представление состоит из результатов запроса, и когда вы модифицируете представление, вы модифицируете набор результатов запроса. Но модификация не должна воздействовать на запрос; она должна воздействовать на значения в таблице, к которой был сделан запрос, и таким образом изменять вывод запроса. Это не простой вопрос. Следующий оператор будет создавать представление, показанное на Рисунке 21.1:

```
CREATE VIEW Citymatch (custcity, salescity)
AS SELECT DISTINCT a.city, b.city
   FROM Customers a, Salespeople b
   WHERE a.snum = b.snum;
```

Это представление показывает все совпадения заказчиков с их продавцами так, что имеется, по крайней мере, один заказчик в **городе заказчика**, обслуживаемый продавцом в **городе продавца**.

Например, одна строка этой таблицы — London London — показывает, что имеется, по крайней мере, один заказчик в Лондоне, обслуживаемый продавцом в Лондоне. Эта строка может быть произведена при совпадении Hoffman с его продавцом Peel, причем если оба они из Лондона.

```
 ===============  SQL Execution Log ==============
| SELECT *                                        |
| FROM Citymatch;                                 |
| ============================================== |
|   custcity    salescity                         |
|  ---------    ---------                         |
|  Berlin       San Jose                          |
|  London       London                            |
|  Rome         London                            |
|  Rome         New York                          |
|  San Jose     Barselona                         |
|  San Jose     San Jose                          |
 ================================================
```

Рисунок 21.1. Представление совпадения по городам

Однако, то же самое значение будет произведено при совпадении Clemens из Лондона, с его продавцом, который также оказался с именем — Peel. Пока отличающиеся комбинации городов выбирались конкретно, только одна строка из этих значений была произведена.

Даже если вы не получите выбора, используя отличия, вы все еще будете в том же самом положении, потому что вы будете тогда иметь две строки в представлении с идентичными значениями, то-есть с обоими столбцами равными "London London". Эти две строки представления будут отличаться друг от друга, так что вы пока не сможете сообщить, какая строка представления исходила из каких значений базовых таблиц (имейте в виду, что запросы, не использующие предложение ORDER BY, производят вывод в произвольном порядке). Это относится также и к запросам, используемым внутри представлений, которые не могут использовать ORDER BY. Таким образом, порядок из двух строк не может быть использован для их отличий. Это означает, что мы будем снова обращаться к выводу строк, которые не могут быть точно связаны с указанными строками запрашиваемой таблицы. Что если вы пробуете удалить строку ”London London” из представления? Означало бы это удаление Hoffman из таблицы Заказчиков, удаление Clemens из той же таблицы, или удаление их обоих? Должен ли быть также удален Peel из таблицы Продавцов? На эти вопросы невозможно ответить точно, поэтому удаления не разрешены в представлениях такого типа. Представление Citymatch — это пример представления *"только чтение"*, оно может быть только запрошено, но не изменено.

Определение модифицируемости представления

Если команды модификации могут выполняться в представлении, представление как сообщалось, будет *модифицируемым*; в противном случае оно предназначено только для чтения при запросе. Не противореча этой терминологии, мы будем использовать выражение "*модифицируемое представление*" (updating a view), что означает возможность выполнения в представлении любой из трех команд модификации DML (INSERT, UPDATE и DELETE), которые могут изменять значения.

Как вы определите, является ли представление модифицируемым? В теории базы данных, это — пока обсуждаемая тема. Основной ее принцип такой: *модифицируемое представление* — это представление, в котором команда модификации может выполниться, чтобы изменить одну и только одну строку основной таблицы в каждый момент времени, не воздействуя на любые другие строки любой таблицы. Использование этого принципа на практике, однако, затруднено. Кроме того, некоторые представления, которые являются модифицируемыми в теории, на самом деле не являются модифицируемыми в SQL. Критерии, по которым определяют, является ли представление модифицируемым или нет, в SQL, следующие:

- Оно должно выводиться в одну и только в одну базовую таблицу.
- Оно должно содержать первичный ключ этой таблицы (это технически не предписывается стандартом ANSI, но было бы неплохо придерживаться этого).
- Оно не должно иметь никаких полей, которые бы являлись агрегатными функциями.
- Оно не должно содержать DISTINCT в своем определении.
- Оно не должно использовать GROUP BY или HAVING в своем определении.
- Оно не должно использовать подзапросы (это — ANSI ограничение, которое не предписано для некоторых реализаций).
- Оно может быть использовано в другом представлении, но это представление должно также быть модифицируемыми.
- Оно не должно использовать константы, строки, или выражения значений (например, comm*100) среди выбранных полей вывода.

- Для INSERT, оно должно содержать любые поля основной таблицы, которые имеют ограничение NOT NULL, если другое ограничение по умолчанию не определено.

Модифицируемые представления и представления "только чтение"

Одно из этих ограничений то, что модифицируемые представления, фактически, подобны окнам в базовых таблицах. Они показывают кое-что, но не обязательно все, из содержимого таблицы. Они могут ограничивать определенные строки (использованием предикатов), или специально именованные столбцы (с исключениями), но они представляют значения непосредственно и не выводят информацию с использованием составных функций и выражений.

Они также не сравнивают строки таблиц друг с другом (как в объединениях и подзапросах, или как с DISTINCT).

Различия между модифицируемыми представлениями и представлениями "только чтение" неслучайны.

Цели, для которых вы их используете, часто различны. Модифицируемые представления, в основном, используются точно так же, как и базовые таблицы. Фактически, пользователи не могут даже осознать, является ли объект, который они запрашивают, базовой таблицей или представлением. Это превосходный механизм защиты для сокрытия частей таблицы, которые являются конфиденциальными или не относятся к потребностям данного пользователя.

Представления "только чтение", с другой стороны, позволяют вам получать и переформатировать данные более рационально. Они дают вам библиотеку сложных запросов, которые вы можете выполнить и повторить снова, сохраняя полученную вами информацию до последней минуты. Кроме того, результаты этих запросов в таблицах, которые могут затем использоваться в запросах самостоятельно (например, в объединениях) имеют преимущество над просто выполнением запросов.

Представления "только чтение" могут также иметь прикладные программы защиты. Например, вы можете захотеть, чтобы некоторые пользователи видели агрегатные данные, такие как усредненное значение комиссионных продавца, не видя индивидуальных значений комиссионных.

Что является модифицируемым представлением

Имеются некоторые примеры модифицируемых представлений и представлений "только чтение":

```
CREATE VIEW Dateorders (odate, ocount)
AS SELECT odate, COUNT (*)
   FROM Orders
   GROUP BY odate;
```

Это — представление "только чтение" из-за присутствия в нем агрегатной функции и GROUP BY.

```
CREATE VIEW Londoncust
AS SELECT *
   FROM Customers
   WHERE city = 'London';
```

А это — представление модифицируемое.

```
CREATE VIEW SJsales (name, number, percentage)
AS SELECT sname, snum, comm *100
   FROM Salespeople
   WHERE city = 'SanJose';
```

Это — представление "только чтение" из-за выражения "**comm * 100**". При этом, однако, возможно переупорядочение и переименование полей. Некоторые программы будут позволять удаление в этом представлении или в Заказах столбцов snum и sname.

```
CREATE VIEW Salesonthird
AS SELECT *
   FROM Salespeople
   WHERE snum IN (SELECT snum
                  FROM Orders
                  WHERE odate = 10/03/1990);
```

Это — представление "только чтение" в ANSI из-за присутствия в нем подзапроса. В некоторых программах, это может быть модифицируемым.

```
CREATE VIEW Someorders
AS SELECT snum, onum, cnum
   FROM Orders
   WHERE odate IN (10/03/1990,10/05/1990);
```

Это — модифицируемое представление.

Проверка значений, помещаемых в представление

Другой вывод о модифицируемости представления тот, что вы можете вводить значения которые "*проглатываются*" (swallowed) в базовой таблице. Рассмотрим такое представление:

```
CREATE VIEW Highratings
AS SELECT cnum, rating
   FROM Customers
   WHERE rating = 300;
```

Это — представление модифицируемое. Оно просто ограничивает ваш доступ к определенным строкам и столбцам в таблице. Предположим, что вы *вставляете* (**INSERT**) следующую строку:

```
INSERT INTO Highratings
VALUES (2018, 200);
```

Это — допустимая команда INSERT в этом представлении. Строка будет вставлена, с помощью представления Highratings, в таблицу Заказчиков. Однако когда она появится там, она исчезнет из представления, поскольку значение оценки не равно 300. Это — обычная проблема.

Значение 200 может быть просто напечатано, но теперь строка находится уже в таблице Заказчиков, где вы не можете даже увидеть ее. Пользователь не сможет понять, почему введя строку, он не может ее увидеть, и будет неспособен при этом удалить ее.

Вы можете быть гарантированы от модификаций такого типа с помощью включения **WITH CHECK OPTION** (с опцией проверки) в определение представления. Мы можем использовать WITH CHECK OPTION в определении представления Highratmgs.

```
CREATE VIEW Highratings
AS SELECT cnum, rating
   FROM Customers
   WHERE rating = 300
   WITH CHECK OPTION;
```

Вышеупомянутая вставка будет отклонена.

WITH CHECK OPTION — производит действие *все_или_ничего* (all-or-nothing). Вы помещаете его в определение представления, а не в команду DML, так что или все команды модификации в представлении будут проверяться, или ни одна не будет проверена. Обычно вы хотите использовать опцию проверки, используя ее в определении представления, что может быть удобно.

В общем, вы должны использовать эту опцию, если у вас нет причины разрешать представлению помещать в таблицу значения, которые он сам не может содержать.

Предикаты и исключенные поля

Похожая проблема, которую вы должны знать, включает в себя вставку строк в представление с предикатом, базирующемся не на всех полях таблицы. Например, может показаться разумным создать **Londonstaff** подобно этому:

```
CREATE VIEW Londonstaff
AS SELECT snum, sname, comm
   FROM Salespeople
   WHERE city = 'London';
```

В конце концов, зачем включать значение city, если все значения city будут одинаковыми.

А как будет выглядеть картинка, получаемая всякий раз, когда мы пробуем вставить строку? Так как мы не можем указать значение city как значение по умолчанию, этим значением, вероятно, будет NULL, и оно будет введено в поле city (NULL используется, если другое значение по умолчанию не было определено). Так как в этм случае поле city не будет равняться значению London, вставляемая строка будет исключена из представления. Это будет верным для любой строки, которую вы попробуете вставить в просмотр **Londonstaff**. Все они должны быть введены с помощью представления Londonstaff в таблицу Продавцов, и затем исключены из самого представления (если определением по умолчанию был не London, то это особый случай). Пользователь не сможет вводить строки в это представление, хотя все еще неизвестно, может ли он вводить строки в базовую таблицу. Даже если мы добавим WITH CHECK OPTION в определение представления

```
CREATE VIEW Londonstate
AS SELECT snum, sname, comm
   FROM Salespeople
   WHERE city = 'London'
   WITH CHECK OPTION;
```

проблема не обязательно будет решена. В результате этого мы получим представление, которое мы могли бы модифицировать или из которого мы могли бы удалять, но не вставлять в него. В некоторых случаях, это может быть хорошо; хотя, возможно, нет смысла пользователям, имеющим доступ к этому представлению, иметь возможность добавлять строки. Но вы должны точно определить, что может произойти, прежде, чем вы создадите такое представление.

Даже если это не всегда может обеспечить Вас полезной информацией, полезно включать в ваше представление все поля, на которые имеется ссылка в предикате. Если вы не хотите видеть эти поля в вашем выводе, вы всегда сможете исключить их из запроса в представлении, в противоположность запросу внутри представления. Другими словами, вы могли бы определить представление **Londonstaff** подобно этому:

```
CREATE VIEW Londonstaff
AS SELECT *
   FROM Salespeople
   WHERE city = 'London'
   WITH CHECK OPTION;
```

Эта команда заполнит представление одинаковыми значениями в поле city, которые вы можете просто исключить из вывода с помощью запроса, в котором указаны только те поля, которые вы хотите видеть:

```
SELECT snum, sname, comm
FROM Londonstaff;
```

Проверка представлений, которые базируются на других представлениях

Еще одно надо упомянуть относительно предложения WITH CHECK OPTION в ANSI: оно не делает *каскадированного* изменения: Оно применяется только в представлениях, в которых оно определено, но не в представлениях основанных на этом представлении. Например, в предыдущем примере

```
CREATE VIEW Highratings
AS SELECT cnum, rating
   FROM Customers
   WHERE rating = 300
   WITH CHECK OPTION;
```

попытка вставить или модифицировать значение оценки не равное 300 потерпит неудачу. Однако, мы можем создать второе представление (с идентичным содержанием) основанное на первом:

```
CREATE VIEW Myratings
AS SELECT *
FROM Highratings;
```

Теперь мы можем модифицировать оценки, не равные 300:

```
UPDATE Myratings
SET rating = 200
WHERE cnum = 2004;
```

Эта команда, выполняемая так, как если бы она выполнялась как первое представление, будет допустима. Предложение WITH CHECK OPTION просто гарантирует, что любая модификация в представлении произведет значения, которые удовлетворяют предикату этого представления. Модификация других представлений, базирующихся на первом текущем, является все еще допустимой, если эти представления не защищены предложениями WITH CHECK OPTION внутри этих представлений. Даже если такие предложения установлены, они проверяют только те предикаты представлений, в которых они содержатся. Так, например, даже если представление **Myratings** создавалось следующим образом

```
CREATE VIEW Myratings
AS SELECT *
   FROM Highratings
   WITH CHECK OPTION;
```

проблема не будет решена. Предложение WITH CHECK OPTION будет исследовать только предикат представления Myratings. Пока у Myratings, фактически, не имеется никакого

предиката, WITH CHECK OPTION ничего не будет делать. Если используется предикат, то он будет проверяться всякий раз, когда представление Myratings будет модифицироваться, но предикат Highratings все равно будет проигнорирован.

Это — дефект в стандарте ANSI, который у большинства программ исправлен. Вы можете попробовать использовать представление наподобие последнего примера и посмотреть избавлена ли ваша система от этого дефекта. Попытка выяснить это самостоятельно может иногда быть проще и яснее, чем поиск ответа в документации системы.

Резюме

Вы теперь овладели знаниями о представлениях полностью. Кроме правил, определяющих, является ли данное представление модифицируемым, вы познакомились с основными понятиями, на которых эти правила базируются, то есть, что модификации в представлениях допустимы только, когда SQL может недвусмысленно определить, какие значения базовой таблицы можно изменять.

Это означает, что команда модификации при выполнении не должна требовать ни изменений для многих строк сразу, ни сравнений между многими строками либо базовой таблицы, либо вывода запроса. Так как объединения включают в себя сравнение строк, они также запрещены. Вы также поняли различие между некоторыми способами, которые используют модифицируемые представления и представления "только чтение".

Вы научились воспринимать модифицируемые представления как окна, отображающие данные одиночной таблицы, но необязательно исключающие или реорганизующие столбцы, посредством выбора только определенных строк отвечающих условию предиката.

Представления *"только чтение"*, с другой стороны, могут содержать большее число допустимых запросов SQL; они могут, следовательно, стать способом хранения запросов, которые вам нужно часто выполнять в неизменной форме. Кроме того, наличие запроса, чей вывод обрабатывается как объект данных, дает вам возможность иметь ясность и удобство при создании запросов в выводе запросов.

Вы теперь можете предохранять команды модификации в представлении от создания строк в базовой таблице, которые не могут быть отражены в самом представлении с помощью предложения WITH CHECK OPTION в определении представления. Вы можете также использовать WITH CHECK OPTION как один из способов ограничения в базовой таблице. В автономных запросах, вы обычно используете один или более столбцов в предикате, не представленных среди выбранных для вывода, что не вызывает никаких проблем. Но если эти запросы используются в модифицируемых представлениях, появляются проблемы, так как эти запросы производят представления, которые не могут иметь вставляемых в них строк. Вы видели некоторые подходы к этим проблемам.

Представления имеют прикладные программы защиты. Вы можете позволить пользователям обращаться к представлениям, не разрешая в тоже время обращаться к таблицам, в которых эти представления непосредственно находятся.

```
#1 CREATE VIEW Dailyorders
   AS SELECT DISTINCT cnum, snum, onum, odate
      FROM Orders;

#2 CREATE VIEW Custotals
   AS SELECT cname, SUM (amt)
      FROM Orders, Customers
      WHERE Orders.cnum = customer.cnum
      GROUP BY cname;

#3 CREATE VIEW Thirdorders
   AS SELECT *
      FROM Dailyorders
      WHERE odate = 10/03/1990;

#4 CREATE VIEW Nullcities
   AS SELECT snum, sname, city
      FROM Salespeople
      WHERE city IS NULL OR sname BETWEEN 'A' AND 'MZ';
```

#1 — не модифицируемый, потому что он использует DISTINCT.

#2 — не модифицируемый, потому что он использует объединение, агрегатную функцию и GROUP BY.

#3 — не модифицируемый, потому что он основывается на #1, который сам по себе не модифицируемый.