Министерство образования Республики Беларусь
Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ
Кафедра программного обеспечения информационных технологий

Ю.В. Быков

ОСНОВЫ ПРОГРАММИРОВАНИЯ НА ЯЗЫКЕ SQL

МЕТОДИЧЕСКОЕ ПОСОБИЕ

по курсу СИСТЕМЫ УПРАВЛЕНИЯ БАЗАМИ ДАННЫХ
для студентов специальности
"Программное обеспечение информационных технологий"

Минск 2002

УДК 681.3.016
ББК 32.973
Б95

Быков Ю.В.
Б95 Основы программирования на языке SQL. Методическое пособие по курсу "Системы управления базами данных" для студентов специальности "Программное обеспечение информационных технологий"./Ю.В. Быков. - Мн.: БГУИР, 2002. - 33 с.
ISBN 985-444-371-X

Методическое пособие содержит описание основных конструкций языка SQL, используемых для формирования запросов к реляционным базам данных.

УДК 681.3.016
ББК 32.973

ISBN 985-444-371-X

Содержание

Введение

1. Команда SELECT
2. Использование операторов отношения и булевых связок
3. Использование специальных предикатов
4. Работа с нулевыми значениями
5. Использование агрегатных функций
6. Предложение GROUP BY
7. Предложение HAVING
8. Использование полей в скалярных выражениях
9. Упорядочение столбцов
10. Запросы к нескольким таблицам
11. Вложенные запросы
12. Коррелированные (соотнесенные) подзапросы
13. Использование оператора EXISTS
14. Использование конструкции UNION

Введение

В данном пособии рассматриваются основные конструкции языка SQL, позволяющие формировать запросы к реляционным базам данных. Во всех примерах запросов используется база данных, состоящая из следующих трех таблиц (табл. 1-3)

Таблица 1
Salespeople

SNUM	SNAME	CITY	COMM
1001	Peel	London	.12
1002	Serres	San Jose	.13
1004	Motika	London	.11
1007	Rifkin	Barcelona	.15
003	Axelrod	New York	.10

Таблица 2
Customers

CNUM	CNAME	CITY	RATING	SNUM
2001	Hoffman	London	100	1001
2002	Giovanni	Rome	200	1003
2003	Liu	San Jose	200	1002
2004	Grass	Berlin	300	1002
2006	Clemens	London	100	1001
2008	Cisneros	San Jose	300	1007
2007	Pereira	Rome	100	1004

Таблица 3

Orders

ONUM	AMT	ODATE	CNUM	SNUM
3001	18.69	10/03/1990	2008	1007
3003	767.19	10/03/1990	2001	1001
3002	1900.10	10/03/1990	2007	1004
3005	5160.45	10/03/1990	2003	1002
3006	1098.16	10/03/1990	2008	1007
3009	1713.23	10/04/1990	2002	1003
3007	75.75	10/04/1990	2004	1002
3008	4723.00	10/05/1990	2006	1001
3010	1309.95	10/06/1990	2004	1002
3011	9891.88	10/06/1990	2006	1001

Поля таблиц расшифровываются следующим образом:

1) snum - уникальный номер продавца;

2) sname - имя продавца;

3) city - город, в котором находится продавец;

4) comm - комиссионные продавцов в десятичной форме;

5) cnum - уникальный номер заказчика;

6) cname - имя заказчика.

7) city - город, в котором находится заказчик;

8) rating – код, указывающий уровень предпочтения данного заказчика перед другими;

9) onum - уникальный номер заказа;

10) amt - значение суммы заказа;

11) odate - дата приобретения.

1. Команда SELECT

В самой простой форме команда SELECT просто инструктирует СУБД, чтобы извлечь информацию из таблицы. Например, можно вывести таблицу Продавцов, выполнив следующий код:

```
SELECT snum, sname, sity, comm
FROM  Salespeople;
```

Вывод для этого запроса показывается в табл. 1.1.

Таблица 1.1

SQL Execution Log

SELECT snum, sname, sity, comm FROM Salespeople;			
snum	sname	city	comm
1001	Peel	London	0.12
1002	Serres	San Jose	0.13
1004	Motika	London	0.11
1007	Rifkin	Barcelona	0.15
1003	Axelrod	New York	0.10

Другими словами, эта команда просто выводит все данные из таблицы.

Имеется объяснение каждой части этой команды:

1) SELECT - ключевое слово, которое сообщает СУБД, что эта команда - запрос. Все запросы начинаются этим словом, сопровождаемым пробелом;

2) snum, sname, city, comm - список столбцов таблицы, которые выбираются запросом. Любые столбцы, не перечисленные здесь, не будут включены в вывод команды. Это, конечно, не значит, что они

будут удалены или их информация будет стерта из таблиц, потому что запрос не воздействует на информацию в таблицах; он только показывает данные;

3) FROM - ключевое слово, подобно SELECT, которое должно быть представлено в каждом запросе. Оно сопровождается пробелом и затем именем таблицы (Salespeople), используемой в качестве источника информации;

4) ; - точка с запятой используется во всех интерактивных командах SQL, чтобы сообщать СУБД, что команда заполнена и готова выполниться.

Если необходимо вывести содержание каждого столбца таблицы, имеется необязательное сокращение (*), которое может использоваться:

```
SELECT *
FROM Salespeople;
```

Это приведет к тому же результату, что и наша предыдущая команда.

На практике могут встретиться таблицы, которые имеют большое количество столбцов, содержащих данные, не все из которых являются относящимися к поставленной задаче. Команда SELECT способна извлечь строго определенную информацию из таблицы.

Вначале рассмотрим возможность выводить только определенные столбцы таблицы. Это выполняется простым исключением столбцов, которые не должны выводиться, из части команды SELECT. Например, запрос

```
SELECT sname, comm
FROM Salespeople;
```

будет производить вывод, показанный в табл. 1.2.

Таблица 1.2

SQL Execution Log

SELECT sname, comm FROM Salespeople;	
Sname	comm
Peel	0.12
Serres	0.13
Motika	0.11
Rifkin	0.15
Axelrod	0.10

В команде SELECT может использоваться аргумент DISTINCT, который обеспечивает возможность устранять одинаковые значения строк таблицы. Предположим, что необходимо знать, какие продавцы (Salespeople) в настоящее время имеют записи о заказах в таблице заказов (Orders). При этом необязательно знать, сколько заказов имеет каждый продавец, а нужен только список номеров продавцов (snum). Для получения необходимой информации используется

```
SELECT DISTINCT snum
FROM Orders;
```

Вместо DISTINCT может быть указан параметр ALL. Это приведет к противоположному результату (дублирование строк вывода сохранится). Так как это - тот же самый случай, когда не указывается ни DISTINCT, ни ALL, то ALL - по существу скорее пояснительный, а не действующий аргумент.

Команда SELECT дает возможность устанавливать критерии для выбора выводимых строк. Для этого используется конструкция

WHERE, которая позволяет определять условия вывода строк таблицы. Например, для вывода информации о всех продавцах в Лондоне может использоваться следующая команда:

```
SELECT sname, city
 FROM Salespeople;
 WHERE city = 'LONDON';
```

Вывод для вышеупомянутого запроса показан в табл. 1.3.

Таблица 1.3

SQL Execution Log

SELECT sname, city FROM Salespeople; WHERE city = 'LONDON';
sname city Peel London Motika London

2. Использование операторов отношения и булевых связок

Оператор отношения - математический символ, который указывает на определенный тип сравнения двух значений. В SQL поддерживаются следующие операторы отношения:

1) = (равно);
2) > (больше);
3) < (меньше);
4) >= (больше или равно);
5) <= (меньше или равно);
6) <> (не равно).

Пример использования оператора >:

```
SELECT *
FROM Customers
WHERE rating > 200;
```

Стандартными булевыми связками, поддерживаемыми SQL, являются: AND, OR, и NOT.

Комбинируя условия с булевыми операторами, можно значительно увеличить возможности команды SELECT. В следующем примере результатом обработки запроса будут строки таблицы Customers, содержащие информацию о заказчиках, расположенных в San Jose и имеющих значение поля rating выше 200.

```
SELECT  *
 FROM Customers
 WHERE city = " San Jose'
 AND rating > 200;
```

Оператор NOT может использоваться для инвертирования значений булевых связок:

```
SELECT *
 FROM Customers
 WHERE city = " San Jose'
  OR NOT rating > 200;
```

3. Использование специальных предикатов

В дополнение к булевым связкам SQL использует специальные операторы IN, BETWEEN, LIKE, и IS NULL.

Оператор IN определяет набор значений, в который данное значение может или не может быть включено. Если необходимо найти всех продавцов, которые размещены в Barcelona или в London

в рассматриваемой базе данных, можно использовать следующий запрос:

```
SELECT *
FROM Salespeople
WHERE city = 'Barcelona'
OR city = 'London';
```

Однако существует и другой, более простой способ получить ту же информацию:

```
SELECT *
FROM Salespeople
WHERE city IN ( 'Barcelona', 'London' );
```

Оператор BETWEEN подобен оператору IN, однако в отличие от перечисления набора значений, как это делает IN, BETWEEN определяет их диапазон:

```
SELECT *
FROM Salespeople
WHERE comm BETWEEN .10 AND .12;
```

Для данной версии оператора BETWEEN значение, совпадающее с любым из значений диапазона 10 – 12, удовлетворяет предикату.

SQL не делает непосредственной поддержки «невключения» BETWEEN. Для этого необходимо определить граничные значения так, чтобы включающая интерпретация была приемлема, или выполнить запрос следующего типа:

```
SELECT *
FROM Salespeople
WHERE ( comm BETWEEN .10, AND .12 )
AND NOT comm IN ( .10, .12 );
```

Оператор BETWEEN может работать с символьными полями в терминах эквивалентов ASCII. Это означает, что возможно использование BETWEEN для выбора из упорядоченных по алфавиту значений.

Следующий запрос выбирает всех заказчиков, чьи имена попали в определенный алфавитный диапазон:

```
SELECT *
FROM Customers
WHERE cname BETWEEN 'A' AND 'G';
```

Оператор LIKE применим только к символьным полям, с которыми он используется, чтобы находить подстроки. Т.е. он ищет поле символа, чтобы проверить, совпадает ли с условием часть его строки. В условии могут использоваться групповые символы (wildkards) для задания шаблонов .

Имеются два типа групповых символов, используемых с LIKE:

1) символ подчеркивания (_) - замещает любой одиночный символ. Например, 'b_t' будет соответствовать словам 'bat' или 'bit', но не будет соответствовать 'brat';

2) знак процента (%) - замещает последовательность любого числа символов (включая символы нуля). Например, '%p%t' будет соответствовать словам 'put', 'posit', или 'opt', но не 'spite'.

Следующий запрос находит всех заказчиков, чьи имена начинаются с G:

```
SELECT
FROM Customers
WHERE cname LIKE 'G%';
```

LIKE может быть удобен, если вы ищете имя или другое значение, и если вы не помните, как они точно пишутся. Предположим, что вы не уверены, как записано по буквам имя одного

из ваших продавцов Peal или Peel. Вы можете просто использовать ту часть, которую вы знаете, и групповые символы, чтобы находить все возможные пары:

```
SELECT *
FROM Salespeople
WHERE sname LIKE 'P _ _ l %';
```

В предикате LIKE можно определить любой одиночный символ как символ ESC. Символ ESC используется сразу перед процентом или подчеркиванием в предикате, и означает, что процент или подчеркивание будет интерпретироваться как символ, а не как групповой символ. Например, в следующем примере мы могли бы найти sname-столбец, где присутствует подчеркивание, таким образом:

```
SELECT *
FROM Salespeople
WHERE sname LIKE '%/_%'ESCAPE'/';
```

Предложение ESCAPE определяет '/ ' как символ ESC. Символ ESC, используемый в LIKE-строке, сопровождается знаком процента, знаком подчеркивания или знаком ESCAPE, который будет искаться в столбце, а не обрабатываться как групповой символ. Символ ESC должен быть одиночным символом и применяться только к одиночному символу сразу после него.

4. Работа с нулевыми значениями

На практике могут встречаться таблицы, в которых записи не имеют никаких значений в отдельных полях, например, потому, что информация в них внесена не полностью. SQL учитывает такой вариант, позволяя вводить значение NULL(ПУСТОЙ) в поле вместо

значения. Когда значение поля равно NULL, это означает, что СУБД специально промаркировала это поле как не имеющее никакого значения для этой строки (или записи). Это отличается от просто назначения полю значения нуля или пробела, которые СУБД будет обрабатывать так же, как и любое другое значение. NULL не является определенным значением и не имеет типа. Оно может помещаться в любой тип поля.

Так как NULL указывает на отсутствие значения, невозможно знать, каков будет результат любого сравнения с использованием NULL.

Для извлечения известных данных используется конструкция IS NULL, например,

```
SELECT *
FROM Customers
WHERE city IS NULL;
```

5. Использование агрегатных функций

Агрегатные функции используются подобно именам полей в предложении SELECT, однако они принимают имена поля как аргументы. Числовые поля могут использоваться с функциями SUM и AVG. С функциями COUNT, MAX и MIN могут использоваться как числовые, так и символьные поля. При использовании символьных полей функции MAX и MIN будут транслировать их в эквивалент ASCII. MIN будет означать первое, а MAX последнее значение в алфавитном порядке.

Чтобы найти сумму всех заказов таблицы Orders, мы можем выполнить следующий запрос:

```
SELECT SUM ((amt))
FROM Orders;
```

Результатом запроса будет число 26658.4.

Нахождение усредненной суммы - похожая операция:

```
SELECT AVG (amt)
FROM Orders;
```

Функция COUNT подсчитывает число значений в данном столбце или число строк в таблице. Когда она подсчитывает значения столбца, то используется конструкция DISTINCT, чтобы производить счет чисел различных значений в данном поле.

Следующий пример подсчитывает номера продавцов в таблице Orders:

```
SELECT COUNT ( DISTINCT snum )
FROM Orders;
```

Для подсчета общего числа строк в таблице используется следующая модификация функции COUNT со звездочкой вместо имени поля:

```
SELECT COUNT (*)
FROM Customers;
```

Агрегатные функции могут быть использованы с аргументами, которые состоят из скалярных выражений, включающих одно или более полей (при этом DISTINCT не разрешается). Предположим, что таблица Orders имеет дополнительный столбец, который хранит предыдущий неоплаченный баланс (поле blnc) для каждого заказчика и необходимо найти текущий баланс добавлением суммы приобретений к предыдущему балансу. Это может быть сделано следующим образом:

```
SELECT MAX ( blnc + (amt) )
FROM Orders;
```

6. Предложение GROUP BY

Предложение GROUP BY позволяет определять подмножество значений в особом поле в терминах другого поля и применять агрегатную функцию к этому подмножеству. Это дает вам возможность объединять поля и агрегатные функции в едином предложении SELECT.

Предположим, что необходимо найти наибольшую сумму приобретений, полученную каждым продавцом. Это можно сделать, выполнив отдельный запрос для каждого из них, выбрав MAX(amt) из таблицы Orders для каждого значения поля snum. GROUP BY, однако, позволяет поместить все в одну команду:

```
SELECT snum, MAX (amt)
FROM Orders
GROUP BY snum;
```

Можно также использовать GROUP BY с множественными полями:

```
SELECT snum, odate, MAX ((amt))
FROM Orders
GROUP BY snum, odate;
```

7. Предложение HAVING

Предложение HAVING определяет критерии, используемые для удаления определенных групп строк из вывода, точно так же, как предложение WHERE делает это для индивидуальных строк:

```
SELECT snum, odate, MAX ((amt))
FROM Orders
```

```
GROUP BY snum, odate
HAVING MAX ((amt)) > 300;
```

Результатом выполнения запроса является информация о поставщиках, поставивших товар на сумму, большую чем 300.

Аргументы в предложении HAVING задаются в соответствии с теми же самыми правилами, что и в предложении SELECT, состоящем из команд, использующих GROUP BY. Они должны иметь одно значение на группу вывода.

8. Использование полей в скалярных выражениях

Если необходимо выполнить простые числовые вычисления данных, чтобы затем вывести их в наиболее удобном формате, можно помещать скалярные выражения и константы среди выбранных полей конструкции SELECT. Эти выражения могут дополнять или замещать поля и могут включать в себя одно или более выбранных полей. Например, для представления комиссионных продавца в процентном отношении может быть использован следующий запрос:

```
SELECT snum, sname, city, comm * 100
FROM Salespeople;
```

9. Упорядочение столбцов

Таблицы - это неупорядоченные наборы данных, и кортежи, которые ищутся в них, необязательно выводятся в какой-то определенной последовательности. Язык SQL поддерживает команду ORDER BY, чтобы упорядочивать вывод данных. Эта конструкция упорядочивает вывод значений столбцов таблицы в

соответствии с определенными правилами. Множественные столбцы упорядочиваются один внутри другого, так же, как с GROUP BY, и пользователь может определять возрастание (ASC) или убывание (DESC) для каждого столбца. По умолчанию используется возрастание.

Рассмотрим таблицу Orders, которая упорядочивается по номеру заказчика (cnum):

```
SELECT *
FROM Orders
ORDER BY cnum DESC;
```

Результат запроса будет следующим:

onum	amt	odate	cnum	snum
3001	18.69	10/03/1990	2008	1007
3006	1098.16	10/03/1990	2008	1007
3002	1900.10	10/03/1990	2007	1004
3008	4723.00	10/05/1990	2006	1001
3011	9891.88	10/06/1990	2006	1001
3007	75.75	10/04/1990	2004	1002
3010	1309.95	10/06/1990	2004	1002
3005	5160.45	10/03/1990	2003	1002
3009	1713.23	10/04/1990	2002	1003
3003	767.19	10/03/1990	2001	1001

Можно упорядочивать таблицу с использованием нескольких столбцов, например, с помощью поля amt, внутри упорядочения поля cnum:

```
SELECT *
FROM Orders
ORDER BY cnum DESC, amt DESC;
```

Выполнение приведенного выше запроса выдаст следующий результат:

onum	amt	odate	cnum	snum
3006	1098.16	10/03/1990	2008	1007
3001	18.69	10/03/1990	2008	1007
3002	1900.10	10/03/1990	2007	1004
3011	9891.88	10/06/1990	2006	1001
3008	4723.00	10/05/1990	2006	1001
3010	1309.95	10/06/1990	2004	1002
3007	75.75	10/04/1990	2004	1002
3005	5160.45	10/03/1990	2003	1002
3009	1713.23	10/04/1990	2002	1003
3003	767.19	10/03/1990	2001	1001

Столбцы, которые упорядочиваются, должны быть указаны в выборе SELECT. Это - требование ANSI. Следующая команда, например, будет запрещена, так как поле cnum не было выбранным полем:

```
SELECT cname, city
FROM Customers
GROUP BY cnum;
```

Конструкция ORDER BY может использоваться с GROUP BY для упорядочения групп кортежей:

```
SELECT snum, odate, MAX (amt)
FROM Orders
GROUP BY snum, odate
ORDER BY snum;
```

Результат запроса будет следующим:

snum	odate	amt
1001	10/06/1990	767.19
1001	10/05/1990	4723.00
1001	10/05/1990	9891.88
1002	10/06/1990	5160.45
1002	10/04/1990	75.75
1002	10/03/1990	1309.95
1003	10/04/1990	1713.23
1004	10/03/1990	1900.10
1007	10/03/1990	1098.16

По умолчанию (при неуказанном порядке) упорядочение столбцов производится по возрастанию их значений.

Вместо имен столбца при упорядочении могут использоваться их порядковые номера. Эти номера могут ссылаться не на порядок столбцов в таблице, а на их порядок в выводе. Другими словами, поле, упомянутое в предложении SELECT первым, для ORDER BY - это будет поле 1, независимо от того, каким по порядку оно стоит в таблице. Например, вы можете использовать следующую команду, чтобы увидеть определенные поля таблицы Salespeople, упорядоченными в порядке убывания значения поля comm:

```
SELECT sname, comm
FROM Salespeople
GROUP BY 2 DESC;
```

10. Запросы к нескольким таблицам

В предыдущих разделах рассматривались SQL-запросы к одной из таблиц рассматриваемой базы данных. Однако SQL позволяет

извлекать информацию одновременно из нескольких таблиц и форматировать ее в удобном для пользователя виде.

Одна из наиболее важных особенностей запросов SQL - это их способность определять связи между многими таблицами и выводить информацию из них в терминах этих связей, используя одну команду. Этот вид операции называется – соединением таблиц.

При соединении таблицы, представленные списком в предложении FROM запроса, отделяются запятыми. Предикат запроса может содержать любой столбец любой таблицы.

Обычно предикат сравнивает значения в столбцах различных таблиц, чтобы определить, удовлетворяет ли WHERE установленному условию.

Полное имя столбца таблицы в запросе состоит из имени таблицы и столбца. Например

Salespeople.snum;

Salespeople.city;

Orders.odate.

При выполнении запроса из многих таблиц можно опускать имена таблиц, если все столбцы имеют различные имена.

Рассмотрим следующий пример создания соединения. Предположим, что необходимо поставить в соответствие продавцам, находящимся в определенном городе , заказчиков из этого же города. Это можно сделать, выполнив следующую команду:

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

В результате из таблиц будет выведена следующая информация:

cname	cname	city
Hoffman	Peel	London
Hoffman	Peel	London
Liu	Serres	San Jose
Cisneros	Serres	San Jose
Hoffman	Motika	London
Clemens	Motika	London

Соединение часто выполняется для эффективного использования связей между таблицами базы данных. Если таблицы Customers и Salespeople уже были соединены через поле snum (эта связь называется состоянием ссылочной целостности), используя соединение можно извлекать данные в терминах этой связи. Например, для того чтобы показать имена всех заказчиков, соответствующих продавцам, которые их обслуживают, используется следующий запрос:

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

Можно также создавать запросы, объединяющие более двух таблиц. Предположим, что необходимо найти все заказы покупателей, не находящихся в тех городах, где находятся соответствующие им продавцы. Для этого необходимо задействовать все три таблицы базы данных:

```
SELECT onum, cname, Orders.cnum, Orders.snum
FROM Salespeople, Customers,Orders
WHERE Customers.city < > Salespeople.city
AND Orders.cnum = Customers.cnum
AND Orders.snum = Salespeople.snum;
```

При выполнении операции соединения в запросе может участвовать одна единственная из таблиц базы данных. Запрос такого вида называется соединением таблицы с собой. Синтаксис команды для соединения таблицы с собой тот же, что и для соединения множественных таблиц. При соединении таблицы с собой все повторяемые имена столбца заполняются префиксами имени таблицы. Чтобы ссылаться на эти столбцы внутри запроса, необходимо иметь различные имена для этой таблицы. Это можно сделать с помощью определения временных имен, называемых переменными диапазона, переменными корреляции или просто - псевдонимами. Они определяются в предложении FROM-запроса.

В следующем примере находятся все пары заказчиков, имеющих одно и то же значение поля rating:

```
SELECT first.cname, second.cname, first.rating
FROM Customers first, Customers second
WHERE first.rating = second.rating;
```

Результат запроса выглядит следующим образом:

Giovanni	Giovanni	200
Giovanni	Liu	200
Liu	Giovanni	200
Liu	Liu	200
Grass	Grass	300
Grass	Cisneros	300
Clemens	Hoffman	100
Clemens	Clemens	100
Clemens	Pereira	100
Cisneros	Grass	300
Cisneros	Cisneros	300
Pereira	Hoffman	100

Pereira Clemens 100

Pereira Pereira 100

Следует обратить внимание на то, что результат вывода имеет два значения для каждой комбинации. Простой способ избежать подобной избыточности данных состоит в использовании дополнительного условия, например:

```
SELECT tirst.cname, second.cname, first.rating
FROM Customers first, Customers second
WHERE first.rating = second.rating
AND first.cname < second.cname;
```

В этом случае результат запроса будет следующим:

cname	cname	rating
Hoffman	Pereira	100
Giovanni	Liu	200
Clemens	Hoffman	100
Pereira	Pereira	100
Gisneros	Grass	300

11. Вложенные запросы

SQL позволяет использовать один запрос внутри другого запроса. Обычно внутренний запрос генерирует значение, которое проверяется в предикате внешнего запроса, определяющего, верно оно или нет. Предположим, что известно имя продавца: Motika, но не известно значение его поля snum, и необходимо извлечь все заказы из таблицы Orders. Имеется единственный способ, чтобы сделать это:

```
SELECT *
FROM Orders
```

```
WHERE snum =
( SELECT snum
FROM Salespeople
WHERE sname = 'Motika');
```

Чтобы оценить внешний (основной) запрос, SQL сначала должен оценить внутренний запрос (или подзапрос) внутри предложения WHERE. Он делает это так, как и должен делать запрос, имеющий единственную цель - отыскать через таблицу Salespeople все строки, где поле sname равно значению Motika, и затем извлечь значения поля snum этих строк.

Единственной найденной строкой естественно будет snum = 1004. Однако SQL не просто выдает это значение, а помещает его в предикат основного запроса вместо самого подзапроса, так чтобы предикат имел следующий вид:

```
WHERE snum = 1004.
```

В результате на выходе образуются следующие кортежи:

```
onum    amt     odate       cnum    snum
3002    1900.10 10/03/1990  2007    1004
```

В подзапросах могут быть использованы агрегатные функции. Например,

```
SELECT *
   FROM Orders
   WHERE amt >
     ( SELECT AVG (amt)
        FROM Orders
        WHERE odate = 10/04/1990 );
```

В подзапросе можно использовать выражение, основанное

на столбце, а не просто сам столбец, в предложении SELECT подзапроса. Это может быть выполнено или с помощью реляционных операторов или с предикатом IN. Например, следующий запрос использует реляционный оператор = :

```
SELECT *
FROM Customers
WHERE cnum =
(SELECT snum + 1000
 FROM Salespeople
 WHERE sname = Serres );
```

Вы можете также использовать подзапросы внутри предложения HAVING. Эти подзапросы могут использовать свои собственные агрегатные функции, если они не производят множественных значений, или использовать GROUP BY, или HAVING. Следующий запрос является этому примером:

```
SELECT rating, COUNT ( DISTINCT cnum )
FROM Customers
GROUP BY rating
HAVING rating >
(SELECT AVG (rating)
FROM Customers
WHERE city = " San Jose';
```

12. Коррелированные (соотнесенные) подзапросы

При использовании подзапросов в SQL, можно обратиться к внутреннему запросу таблицы в предложении внешнего запроса FROM, сформировав соотнесенный подзапрос. При этом подзапрос

выполняется неоднократно, по одному разу для каждой строки таблицы основного запроса.

Например, существует следующий способ найти всех заказчиков в заказах на 3 октября 1990 года:

```
SELECT *
FROM Customers outer
WHERE 10/03/1990 IN
(SELECT odate
 FROM Orders inner
 WHERE outer.cnum = inner.cnum );
```

В вышеупомянутом примере inner и outer являются псевдонимами. Так как значение в поле cnum внешнего запроса меняется, внутренний запрос должен выполняться отдельно для каждой строки внешнего запроса.

Строка внешнего запроса, для которого внутренний запрос каждый раз будет выполнен, называется текущей строкой-кандидатом. Процедура оценки строки, выполняемая соотнесенным подзапросом, может быть описана следующим образом:

1. Выбрать строку из таблицы, именованной во внешнем запросе (текущую строку-кандидата),

2. Сохранить значения из этой строки-кандидата в псевдониме с именем в предложении FROM внешнего запроса,

3. Выполнить подзапрос. Везде, где псевдоним, данный для внешнего запроса, найден (в этом случае "внешний"), использовать значение для текущей строки-кандидата (использование значения из строки-кандидата внешнего запроса в подзапросе называется - внешней ссылкой),

4. Оценить предикат внешнего запроса на основе результатов подзапроса, выполняемого в шаге 3 (он определяет, выбирается ли строка-кандидат для вывода),

5. Повторять процедуру для следующей строки-кандидата таблицы до тех пор, пока все строки таблицы не будут проверены.

В вышеупомянутом примере выполняется следующая процедура:

1. Выбирается строка Hoffman из таблицы Заказчиков;

2. Сохраняется выбранная строка как текущая строка-кандидат;

3. Выполняется подзапрос. Подзапрос просматривает всю таблицу Orders для того, чтобы найти строки, где значение cnum-поле равно значению outer.cnum, которое на данном шаге равно 2001. Затем он извлекает поле odate из каждой строки таблицы Orders, для которой это условие выполняется, и формирует набор значений поля odate;

4. Получив набор всех значений поля odate, для поля cnum = 2001 проверяется предикат основного запроса для того, чтобы проверить, имеется ли значение на 3 октября в этом наборе. Если это так, то выбирается строка Hoffman для вывода ее из основного запроса;

5. Повторяется вся процедура с использованием строки Giovanni как строки-кандидата, и так далее.

Результат запроса выглядит следующим образом:

cnum	cname	city	rating	snum
2001	Hoffman	London	100	1001
2003	Liu	San Jose	200	1002
2008	Cisneros	San Jose	300	1007
2007	Pereira	Rome	100	1004

Соотнесенный подзапрос может использоваться по отношению к той же самой таблице, что и основной запрос:

```
SELECT *
FROM Orders outer
WHERE amt >
(SELECT AVG amt
 FROM Orders inter
 WHERE inner.cnum = outer.cnum );
```

13. Использование оператора EXISTS

EXISTS - это оператор, который воспринимает подзапрос как аргумент и оценивает его как истинный, если тот производит любой вывод, или как ложный, если он не делает этого. Например, мы можем решить, извлекать ли нам некоторые данные из таблицы заказчиков если, и только если один или более заказчиков в этой таблице находятся в San Jose:

```
SELECT cnum, cname, city
FROM Customers
WHERE EXISTS
(SELECT *
FROM Customers
WHERE city = " San Jose' );
```

Внутренний запрос выбирает все данные для всех заказчиков в San Jose. Оператор EXISTS во внешнем предикате отмечает, что некоторый вывод был произведен подзапросом, и поскольку выражение EXISTS было непустым, делает предикат верным. Подзапрос выполняется только один раз для всего внешнего запроса, и, следовательно, имеет одно значение во всех случаях.

Результат запроса выглядит следующим образом:

cnum	cname	city
2001	Hoffman	London
2002	Giovanni	Rome
2003	Liu	San Jose
2004	Grass	Berlin
2006	Clemens	London
2008	Cisneros	San Jose
2007	Pereira	Rome

В соотнесенном подзапросе предложение EXISTS оценивается отдельно для каждой строки таблицы, имя которой указано во внешнем запросе, точно так же, как и другие операторы предиката, когда вы используете соотнесенный подзапрос. Это дает возможность использовать EXISTS как истинный предикат, который генерирует различные ответы для каждой строки таблицы, указанной в основном запросе. Следовательно, информация из внутреннего запроса будет сохранена, если выведена непосредственно, когда EXISTS используется таким способом. Например, можно вывести продавцов, которые имеют множественных заказчиков:

```
SELECT DISTINCT snum
FROM Customers outer
WHERE EXISTS
( SELECT *
FROM Customers inner
WHERE inner.snum = outer.snum
AND inner.cnum < > outer.cnum );
```

Для каждой строки-кандидата внешнего запроса (представляющей заказчика проверяемого в настоящее время), внутренний запрос находит строки, которые совпадают со значением поля snum (которое имел продавец), но не со значением поля cnum. Если такие строки найдены внутренним запросом, это означает, что имеются два разных заказчика, обслуживаемых одним продавцом.

Предыдущий пример показывает, что конструкция EXISTS может работать в комбинации с булевыми операторами. Наиболее часто с EXISTS используется оператор NOT. Один из способов, которым мы могли бы найти всех продавцов, работающих только с одним заказчиком, состоит в том, чтобы «инвертировать» предыдущий пример:

```
SELECT DISTINCT snum
FROM Customers outer
WHERE NOT EXISTS
( SELECT *
FROM Customers inner
WHERE inner.snum = outer.snum
AND inner.cnum < > outer.cnum );
```

14. Использование конструкции UNION

Предложение UNION объединяет вывод двух или более SQL-запросов в единый набор строк и столбцов. Например, чтобы получить информацию обо всех продавцах и заказчиках, размещенных в Лондоне, и вывести ее как единое целое, можно выполнить следующий запрос:

```
SELECT snum, sname
```

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

Результат запроса выглядит следующим образом:

```
1001    Peel
1004    Motika
2001    Hoffman
2006    Climens
```

Когда два или более запроса объединяются, их результаты вывода должны быть совместимы для объединения. Это означает, что каждый запрос должен указывать одинаковое число столбцов, и каждый столбец должен иметь тип, совместимый с типом соответствующих ему столбцов, обрабатываемых другими запросами.

Конструкция UNION, в отличие от одиночных запросов, автоматически исключает дубликаты строк из вывода. Например, запрос

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

допускает появление кортежей-дубликатов в результате:

```
snum     city
 1001    London
 1003    Rome
 1002    San Jose
 1002    Berlin
 1001    London
```

1004 Rome

1007 San Jose

В то время как UNION в комбинации этого запроса с другим запросом устраняет избыточность информации:

```
SELECT snum, city
FROM Customers
UNION
SELECT snum, city
FROM Salespeople.;
```

Результат запроса имеет следующий вид:

1001 London

1002 Berlin

1007 San Jose

1007 New York

1003 Rome

1001 London

1003 Rome

1002 Barcelona

1007 San Jose

Для форматирования вывода из объединения запросов можно использовать предложение ORDER BY. Рассмотрим следующий пример. Пусть необходимо упорядочить имена заказчиков по их порядковым номерам. Запрос, выполняющий это действие, выглядит следующим образом:

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

Св. план 2002, поз. 50

Учебное издание

Быков Юрий Викторович

“Основы программирования на языке SQL”

Методическое пособие
по курсу "Системы управления базами данных"
для студентов специальности "Программное обеспечение"

Редактор Т.Н. Крюкова Корректор Е.Н. Батурчик

Подписано в печать Формат 60x84 1/16.
Бумага Печать офсетная. Гарнитура Усл. печ. л.
Уч.-изд. л. 1,4 Тираж 100 экз. Заказ

Издатель и полиграфическое исполнение:
Учреждение образования
«Белорусский государственный университет информатики и радиоэлектроники»
Лицензия ЛП № 156. от 05.02.2001
Лицензия ЛП № 509. от 03.08.2001
220013, Минск, П. Бровки, 6