## Лекция 9. SQL: манипулирование данными.

**Лекционный план**

1. Добавление, обновление, удаление данных
2. Агрегатные функции языка SQL и группировка.
3. Соединение таблиц.
4. Вложенные запросы.

**Ключевые слова: SELECT, INSERT, UPDATE, DELETE, агрегатные функции, соединение, объединение, пересечение, разность, декартово произведение.**

**Язык манипулирования данными.**

В основу языка манипулирования данными входят 4 основных оператора:

1. SELECT – используется для выборки записей из таблиц;
2. INSERT –используется для добавления записей в таблицу;
3. UPDATE – используется для обновления записей таблицы;
4. DELETE – используется для удаления записей из таблицы.

В самой простой форме, команда SELECT просто инструктирует базу данных чтобы извлечь информацию из таблицы.

**Оператор INSERT.**

Все строки в SQL вводятся с использованием команды модификации INSERT. В самой простой форме, INSERT использует следующий синтаксис:

INSERT INTO <имя\_таблицы> [(<имя столбца\_1> [, <имя столбца\_1> ...])]

{VALUES (*<значение\_1>* [, *<значение\_2>* ...]) | *<выражение SELECT>*};

Так, например, чтобы ввести строку в таблицу Продавцов, вы можете использовать следующее условие:

INSERT INTO Salespeople VALUES (1001, 'Peel', 'London', .12);

Команды DML не производят никакого вывода, но ваша программа должна дать вам некоторое подтверждение того что данные были использованы.

Вы можете также указывать столбцы, куда вы хотите вставить значение имени. Это позволяет вам вставлять имена в любом порядке. Предположим, что вы берете значения для таблицы Заказчиков из отчета выводимого на принтер, который помещает их в таком порядке: city, cname, и cnum, и для упрощения, вы хотите ввести значения в том же порядке:

INSERT INTO Customers (city, cnamе, cnum) VALUES ('London', 'Honman', 2001);

Обратите внимание что столбцы rating и snum - отсутствуют. Это значит, что эти строки автоматически установлены в значение - по умолчанию. По умолчанию может быть введено или значение NULL или другое значе- ние определяемое как - по умолчанию. Если ограничение запрещает использование значения NULL в данном столбце, и этот столбец не установлен как по умолчанию, этот столбец должен быть обеспечен значением для любой команды INSERT которая относится к таблице.

**Оператор UPDATE.**

Теперь, вы должны узнать как изменять некоторые или все значения в существующей строке. Это выполняется командой UPDATE. Эта команда содержит предложение UPDATE в которой указано им используемой таблицы и предложение SET которое указывает на изменение которое нужно сделать для определенного столбца. Например, чтобы изменить оценки всех заказчиков на 200, вы можете ввести

UPDATE TABLE <имя\_таблицы>

SET <имя столбца\_1> = <значение\_1> [, <имя столбца\_2> = <значение\_2> ...]

[WHERE <условие>];

Например

UPDATE Customers

SET rating = 200;

Конечно, вы не всегда захотите указывать все строки таблицы для изменения единственного значения, так что UPDATE может брать предикаты. Вот как например можно выполнить изменение одинаковое для всех заказчиков продавца Peel ( имеющего snum=1001 ):

UPDATE Customers

SET rating = 200

WHERE snum = 1001;

Однако, вы не должны, ограничивать себя модифицированием единственного столбца с помощью команды UPDATE. Предложение SET может назначать любое число столбцов, отделяемых запятыми. Все указанные назначения могут быть сделаны для любой табличной строки, но только для одной в каждый момент времени. Предположим, что продавец Motika ушел на пенсию, и мы хотим переназначить его номер новому продавцу:

UPDATE Salespeople

SET sname = 'Gibson',city = 'Boston',comm = .10

WHERE snum = 1004;

Вы можете использовать скалярные выражения в предложении SET команды UPDATE, однако, включив его в выражение пол которое будет изменено. В этом их отличие от предложения VALUES команды INSERT, в котором выражения не могут использоваться; это свойство скалярных выражений - весьма полезна особенность. Предположим, что вы решили удвоить комиссионные всем вашим продавцам. Вы можете использовать следующее выражение:

UPDATE Salespeople

SET comm = comm \* 2;

**Оператор DELETE.**

Вы можете удалять строки из таблицы командой модификации - DELETE. Она может удалять только введенные строки, а не индивидуальные значения полей.

DELETE FROM table [WHERE <search\_condition>];

Теперь когда таблица пуста ее можно окончательно удалить командой DROP TABLE. Обычно, вам нужно удалить только некоторые определенные строки из таблицы. Чтобы определить какие строки будут удалены, вы используете предикат, так же как вы это делали для запросов. Например, чтобы удалить продавца под номером 1003 из таблицы, вы можете ввести

DELETE FROM Salespeople

WHERE snum = 1003;

Мы использовали поле snum вместо пол sname потому, что это лучшая тактика при использовании первичных ключей когда вы хотите чтобы действию подвергалась одна и только одна строка. Для вас - это аналогично действию первичного ключ. Конечно, вы можете также использовать DELETE с предикатом который бы выбирал группу строк, как показано в этом примере:

DELETE FROM Salespeople

WHERE city = 'London';

**Оператор SELECT**

Все запросы в SQL состоят из одиночной команды. Структура этой команды обманчиво проста, потому что вы должны расширять ее так чтобы выполнить высоко сложные оценки и обработки данных. Эта команда называется - SELECT(ВЫБОР).

SELECT [[ALL] | DISTINCT]{ \* | элемент\_SELECT [,элемент\_SELECT] ...}

FROM {базовая\_таблица | представление} [псевдоним]

[,{базовая\_таблица | представление} [псевдоним]] ...

[WHERE фраза]

[GROUP BY фраза [HAVING фраза]];

[ORDER BY фраза]

Если вы хотите видеть каждый столбец таблицы, имеется необязательное сокращение которое вы можете использовать. Звездочка (\*) может применяться для вывода полного списка столбцов следующим образом:

SELECT \* FROM Salespeople;

**Агрегатные функции языка SQL и группировка.**

Агрегатные функции предназначены для того, чтобы вычислять некоторое значение для заданного множества строк. Таким множеством строк может быть группа строк, если агрегатная функция применяется к сгруппированной таблице, или вся таблица. В SQL имеется пять агрегатных функций, которые позволяют получать различные виды итоговой информации. Ниже описан синтаксис этих функций:

1. COUNT – возвращает количество значений в указанном столбце;
2. SUM – возвращает сумму значений в указанном столбце;
3. AVG – возвращает усредненное значение в указанном столбце;
4. MIN – возвращает минимальное значение в указанном столбце;
5. МАХ – возвращает максимальное значение в указанном столбце.

Для всех агрегатных функций, кроме COUNT(\*), фактический (т.е. требуемый семантикой) порядок вычислений следующий: на основании параметров агрегатной функции из заданного множества строк производится список значений. Затем по этому списку значений производится вычисление функции. Если список оказался пустым, то значение функции COUNT для него есть 0, а значение всех остальных функций null.

Агрегатные функции используются подобно именам полей в предложении SELECT запроса, но с одним исключением, они берут имена поля как аргументы. Только числовые поля могут использоваться с SUM и AVG. С COUNT, MAX, и MIN, могут использоваться и числовые или символьные поля. Когда они используются с символьными полями, MAX и MIN будут транслировать их в эквивалент ASCII, который должен сообщать, что MIN будет означать первое, а MAX последнее значение в алфавитном порядке.

Чтобы найти SUM всех наших покупок в таблицы Порядков, мы можем ввести следующий запрос, с его выводом в рисунке 9.1:

SELECT SUM ((amt))

FROM Orders;

=============== SQL Execution Log ============

| |

| ------- |

| 26658.4 |

| |

===============================================

Следует отметить, что агрегирующие функции могут использоваться только в списке выборки SELECT и в конструкции HAVING. Во всех других случаях применение этих функций недопустимо. Если список выборки SELECT содержит агрегирующую функцию, а в тексте запроса отсутствует конструкция GROUP BY, обеспечивающая объединение данных в группы, то ни один из элементов списка выборки SELECT не может включать каких-либо ссылок на столбцы, за исключением случая, когда этот столбец используется как параметр агрегирующей функции. Например, следующий запрос является некорректным:

SELECT staffNo, COUNT(salary) FROM Staff;

Ошибка состоит в том, что в данном запросе отсутствует конструкция GROUP BY, а обращение к столбцу staffNo в списке выборки SELECT выполняется без применения агрегирующей функции.

Пример:

SELECT MIN(salary) AS min, MAX(salary) AS max, AVG(salary) AS avg FROM Staff;

=============== SQL Execution Log ============

| |

| min | max | avg |

| 9000 30000 17000 |

| |

===============================================

**Запросы с группировкой (предложение GROUP BY).**

Приведенные выше примеры сводных данных подобны итоговым строкам, обычно размещаемым в конце отчетов. В итогах все детальные данные отчета сжимаются в одну обобщающую строку. Однако очень часто в отчетах требуется формировать и промежуточные итоги. Для этой цели в операторе SELECT может указываться конструкция GROUP BY. Запрос, в котором присутствует конструкция GROUP BY, называется группирующим запросом, поскольку в нем группируются данные, полученные в результате выполнения операции SELECT, после чего для каждой отдельной группы создается единственная итоговая строка. Столбцы, перечисленные в конструкции GROUP BY, называются группируемыми столбцами. Стандарт ISO требует, чтобы конструкции SELECT и GROUP BY были тесно связаны между собой. При использовании в операторе SELECT конструкции GROUP BY каждый элемент списка в списке выборки SELECT должен иметь единственное значение для всей группы. Более того, конструкция SELECT может включать только следующие типы элементов:

* имена столбцов;
* агрегирующие функции;
* константы;
* выражения, включающие комбинации перечисленных выше элементов.

Все имена столбцов, приведенные в списке выборки SELECT, должны присутствовать и в конструкции GROUP BY, за исключением случаев, когда имя столбца используется только в агрегирующей функции. Противоположное утверждение не всегда справедливо — в конструкции GROUP BY могут присутствовать имена столбцов, отсутствующие в списке выборки SELECT. Если совместно с конструкцией GROUP BY используется конструкция WHERE, то она обрабатывается в первую очередь, а группированию подвергаются только те строки, которые удовлетворяют условию поиска. Стандартом ISO определено, что при проведении группирования все отсутствующие значения рассматриваются как равные. Если две строки таблицы в одном и том же группируемом столбце содержат значения NULL и идентичные значения во всех остальных непустых группируемых столбцах, они помещаются в одну и ту же группу.

Например:

SELECT branchNo, COUNT{staffNo} AS count, SUM(salary) AS sum FROM Staff GROUP BY branchNo ORDER BY branchNo;

=============== SQL Execution Log ============

| |

| branchNo | count | sum |

| B003 3 54000 |

| B005 2 39000 |

| B007 1 9000 |

| |

===============================================

Концептуально при обработке этого запроса выполняются следующие действия.

1. Строки таблицы Staff распределяются в группы в соответствии со значениями в столбце номера отделения компании. В пределах каждой из групп оказываются данные обо всем персонале одного из отделений компании. В нашем примере будут созданы три группы, как показано на рис. 9.2.
2. Для каждой из групп вычисляются общее количество строк, равное количеству работников отделения, а также сумма значений в столбце salary, которая и является интересующей нас суммой заработной платы всех работников отделения. Затем генерируется единственная итоговая строка для всей группы исходных строк.
3. Полученные строки результирующей таблицы сортируются в порядке возрастания номера отделения, указанного в столбце branchNo.

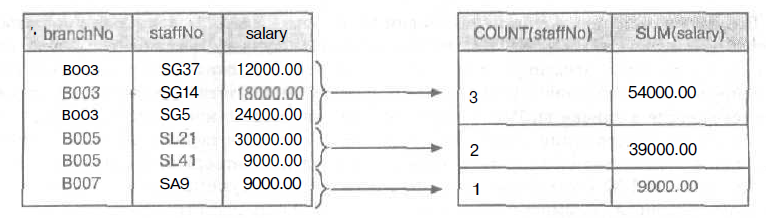


Рис. 9.2. Три группы записей, создаваемых при запросе.

**Предложение HAVING.**

Предположим, что в предыдущем примере, вы хотели бы увидеть только максимальную сумму приобретений значение которой выше $3000.00. Вы не сможете использовать агрегатную функцию в предложении WHERE (если вы не используете подзапрос, описанный позже), потому что предикаты оцениваются в терминах одиночной строки, а агрегатные функции оцениваются в терминах групп строк. Это означает, что вы не сможете сделать что-нибудь подобно следующему:

SELECT snum, odate, MAX (amt)

FROM Oreders

WHERE MAX ((amt)) > 3000.00

GROUP BY snum, odate;

Это будет отклонением от строгой интерпретации ANSI. Чтобы увидеть максимальную стоимость приобретений свыше $3000.00, вы можете использовать предложение HAVING. Предложение HAVING определяет критерии, используемые чтобы удалять определенные группы из вывода, точно также как предложение WHERE делает это для индивидуальных строк. Правильной командой будет следующая:

SELECT snum, odate, MAX(amt)

FROM Orders

GROUP BY snum, odate

HAVING MAX ((amt)) > 3000.00;

=============== SQL Execution Log ==============

| |

| snum odate MAX(amt) |

| ------ ---------- -------- |

| 1001 10/05/1990 4723.00 |

| 1001 10/06/1990 9891.88 |

| 1002 10/03/1990 5160.45 |

| |

================================================

Аргументы в предложении HAVING следуют тем же самым правилам, что и в предложении SELECT, состоящей из команд использующих GROUP BY. Они должны иметь одно значение на группу вывода. Следующая команда будет запрещена:

SELECT snum, MAX (amt)

FROM Orders

GROUP BY snum

HAVING odate = 10/03/1988;

Поле оdate не может быть вызвано предложением HAVING, потому что оно может иметь ( и действительно имеет ) больше чем одно значение на группу вывода. Чтобы избегать такой ситуации, предложение HAVING должно ссылаться только на агрегаты и поля выбранные GROUP BY. Имеется правильный способ сделать вышеупомянутый запрос:

SELECT snum, MAX (amt)

FROM Orders

WHERE odate = 10/03/1990

GROUP BY snum;

=============== SQL Execution Log ==============

| |

| snum |

| ------ -------- |

| 1001 767.19 |

| 1002 5160.45 |

| 1014 1900.10 |

| 1007 1098.16 |

| |

================================================

Как и говорилось ранее, HAVING может использовать только аргументы, которые имеют одно значение на группу вывода. Практически, ссылки на агрегатные функции – наиболее общие, но и поля, выбранные с помощью GROUP BY также допустимы. Например, мы хотим увидеть наибольшие порядки для Serres и Rifkin:

SELECT snum, MAX (amt)

FROM Orders

GROUP BY snum

HAVING snum IN (1002,1007);

=============== SQL Execution Log ==============

| |

| snum |

| ------ -------- |

| 1002 5160.45 |

| 1007 1098.16 |

| |

================================================

SQL позволяет группировать результаты запроса на основании двух или более столбцов. Например, получить список фамилий студентов и их средних оценок за каждый семестр.

SELECT StName, Semester, AVG(Mark)

FROM Marks INNER JOIN Students USING(StNo)

GROUP BY StName, Semester;

**Ограничения на запросы с группировкой.**

На запросы, в которых используется группировка, накладываются дополнительные ограничения. Столбцы с группировкой должны представлять собой реальные столбцы таблиц, перечисленных в предложении FROM. Нельзя группировать строки на основании значения вычисляемого выражения.

Кроме того, существуют ограничения на элементы списка возвращаемых столбцов. Все элементы этого списка должны иметь одно значение для каждой группы строк. Это означает, что возвращаемым столбцом может быть:

1. константа;
2. агрегатная функция, возвращающая одно значение для всех строк, входящих в группу;
3. столбец группировки, который, по определению, имеет одно и то же значение во всех строках группы;
4. выражение, включающее в себя перечисленные выше элементы.

На практике в список возвращаемых столбцов запроса с группировкой всегда входят столбец группировки и агрегатная функция. Если последняя не указана, значит, запрос можно более просто выразить с помощью ключевого слова DISTINCT без использования предложения GROUP BY. И наоборот, если не включить в результаты запроса столбец группировки, вы не сможете определить, к какой группе относится каждая строка результатов.

В строгой интерпретации ANSI SQL, вы не можете использовать агрегат агрегата. Предположим? что вы хотите выяснить, в какой день имелась наибольшая сумма приобретений.

SELECT odate, MAX ( SUM (amt) )

FROM Orders

GROUP BY odate;

Если вы попробуете сделать это, то ваша команда будет вероятно отклонена. (Некоторые реализации не предписывают этого ограничения, которое является выгодным, потому что вложенные агрегаты могут быть очень полезны, даже если они и несколько проблематичны.) В вышеупомянутой команде, например, SUM должен применяться к каждой группе поля odate, а MAX ко всем группам, производящим одиночное значение для всех групп. Однако предложение GROUP BY подразумевает что должна иметься одна строка вывода для каждой группы поля odate.

**Соединение таблиц.**

В стандарте SQL-92 был определен совершенно новый метод поддержки внешних объединений, который не опирался ни на одну популярную СУБД. В спецификации стандарта поддержка внешних Соединений осуществлялась в предложении FROM с тщательно разработанным синтаксисом, позволявшим пользователю точно определить, как исходные таблицы должны быть соединены в запросе.

Операция соединения используется в языке SQL для вывода связанной информации, хранящейся в нескольких таблицах, в одном запросе. В этом проявляется одна из наиболее важных особенностей запросов SQL - способность определять связи между многочисленными таблицами и выводить информацию из них в рамках этих связей. Именно эта операция придает гибкость и легкость языку SQL.

Операции соединения подразделяются на два вида - внутренние и внешние. Оба вида соединений задаются в предложении WHERE запроса SELECT с помощью специального условия соединения. Внешние соединения (о которых мы поговорим позднее) поддерживаются стандартом ANSI-92 и содержат зарезервированное слово "JOIN", в то время как внутренние соединения (или просто соединения) могут задаваться как без использования такого слова (в стандарте ANSI-89), так и с использованием слова "JOIN" (в стандарте ANSI-92).

Связывание при соединении таблиц производится, как правило, по первичному ключу одной таблицы и внешнему ключу другой таблицы - для каждой пары таблиц. При этом очень важно учитывать все поля внешнего ключа, иначе результат будет искажен. Соединяемые поля могут (но не обязаны!) присутствовать в списке выбираемых элементов. Предложение WHERE может содержать множественные условия соединений. Условие соединения может также комбинироваться с другими предикатами в предложении WHERE.

Соединение является подмножеством более общей комбинации данных двух таблиц, называемой декартовым произведением. Декартово произведение двух таблиц представляет собой другую таблицу, состоящую из всех возможных пар строк, входящих в состав обеих таблиц. Набор столбцов результирующей таблицы представляет собой все столбцы первой таблицы, за которыми следуют все столбцы второй таблицы. Если ввести запрос к двум таблицам без задания конструкции WHERE, результат выполнения запроса в среде SQL будет представлять собой декартовое произведение этих таблиц.

Процедура генерации таблицы, содержащей результаты соединения двух таблиц с помощью оператора SELECT, состоит в следующем.

1. Формируется декартово произведение таблиц, указанных в конструкции FROM.
2. Если в запросе присутствует конструкция WHERE, применение условий поиска к каждой строке таблицы декартова произведения и сохранение в таблице только тех строк, которые удовлетворяют заданным условиям. В терминах реляционной алгебры эта операция называется ограничением декартового произведения.
3. Для каждой оставшейся строки определяется значение каждого элемента, указанного в списке выборки SELECT, в результате чего формируется отдельная строка результирующей таблицы.
4. Если в исходном запросе присутствует конструкция SELECT DISTINCT, из результирующей таблицы удаляются все строки-дубликаты. В реляционной алгебре действия, выполняемые на 3 и 4 этапах, эквивалентны операции проекции по столбцам, заданным в списке выборки SELECT.
5. Если выполняемый запрос содержит конструкцию ORDER BY, осуществляется переупорядочивание строк результирующей таблицы.

**Внутреннее соединение в стандарте SQL-92.**

Внутреннее соединение возвращает только те строки, для которых условие соединения принимает значение true.

SELECT \* | <имена\_столбцов> FROM

<имя\_таблицы\_1> INNER JOIN <имя\_таблицы\_2> ON <имя\_столбца\_в\_1\_таблице> = <имя\_столбца\_во\_2\_таблице>

WHERE <условие>

ORDER BY <имена\_столбцов>;

или

SELECT \* | <имена\_столбцов> FROM

<имя\_таблицы\_1>, <имя\_таблицы\_2> WHERE <имя\_столбца\_в\_1\_таблице> = <имя\_столбца\_во\_2\_таблице>

ORDER BY <имена\_столбцов>;

В первом случае две таблицы соединяются явно посредством операции JOIN, а условие поиска, описывающее объединение, находится теперь в предложении ON внутри предложения FROM. В условии поиска, следующем за ключевым словом ON, могут быть заданы любые критерии сравнения строк двух объединяемых таблиц. Во втором случае условие соединения идет вместе с предикатом после ключевого слова WHERE.

|  |  |  |
| --- | --- | --- |
| **branchNo** | **bCity** | **cityID** |
| В004 | Bristol | 1 |
| B003 | Glasgow | 3 |
| В002 | London | 2 |

Таблица 9.1. Branch

|  |  |  |
| --- | --- | --- |
| **propertyNo** | **pCity** | **cityID** |
| PA14 | Aberdeen | 4 |
| PL94 | London | 2 |
| PG4 | Glasgow | 3 |

Таблица 9.2. PropertyForRent

Обычное (внутреннее) соединение этих таблиц выполняется с помощью следующего оператора SQL:

SELECT b.\*, p.\*

FROM Branch b, PropertyForRent p

WHERE b.bCity = p.pCity;

Или в соответствии со стандартом SQL-92 данный запрос буде выглядеть следующим образом:

SELECT b.\*, p.\*

FROM Branch b INNER JOIN PropertyForRent p

ON b.bCity = p.pCity;

==================== SQL Execution Log =======================

| |

| branchNo bcity cityID propertyNo pCity cityID |

| -------- ------- ------ ---------- ------- ------ |

| B003 Glasgow 3 PG4 Glasgow 3 |

| B002 London 2 PL94 London 2 |

| |

===============================================================

Как можно видеть, в результирующей таблице запроса имеются только две строки, содержащие одинаковые названия городов, выбранные из обеих таблиц. Обратите внимание, что в исходных данных нет соответствия для отделения компании в Bristol и для объекта, сдаваемого в аренду в городе Aberdeen. Как видно из примера соединение можно выполнять не только по полям, содержащим первичный и вторичный ключи.

Объединение двух таблиц, в котором связанные столбцы имеют идентичные имена, называется ***естественным объединением***, так как обычно это действительно самый "естественный" способ объединения двух таблиц.

SELECT \* | <имена\_столбцов> FROM

<имя\_таблицы\_1> NATURAL JOIN <имя\_таблицы\_2> ON (<имя\_столбца>);

Например естественное соединение таблиц Branch и PropertyForRent выглядит следующим образом:

SELECT b.\*, p.\*

FROM Branch b NATURAL JOIN PropertyForRent p

ON (cityID);

**Внешние соединения в стандарте SQL-92.**

Во внешнем соединении в результирующую таблицу помещаются также строки, не удовлетворяющие условию соединения. Чтобы понять особенности выполнения операций внешнего соединения, воспользуемся упрощенными таблицами Branch и PropertyForRent, содержимое которых представлены выше.

Существуют три типа внешнего соединения:

1. левое;
2. правое;
3. полное.

Используем левое внешнее соединение этих двух таблиц, которое выглядит следующим образом:

SELECT b.\*, р.\*

FROM Branch b LEFT JOIN PropertyForRent p ON b.bCity = p.pCity;

===================== SQL Execution Log ======================

| |

| branchNo bcity cityID propertyNo pCity cityID |

| -------- ------- ------ ---------- ------- ------ |

| B003 Glasgow 3 PG4 Glasgow 3 |

| B004 Bristol 1 NULL NULL NULL |

| B002 London 2 PL94 London 2 |

| |

===============================================================

В этом примере за счет применения левого внешнего соединения в результирующую таблицу попали не только две строки, в которых имеется соответствие между названиями городов, но также та строка первой из соединяемых таблиц (левой), которая не нашла себе соответствия во второй таблице (правой). В этой строке все поля второй таблицы заполнены значениями NULL.

Используем правое внешнее соединение этих двух таблиц, которое выглядит следующим образом:

SELECT b.\*, p.\*

FROM Branch b RIGHT JOIN PropertyForRent p ON b.bCity = p.pCity;

===================== SQL Execution Log ======================

| |

| branchNo bcity cityID propertyNo pCity cityID |

| -------- ------- ------ ---------- ------- ------ |

| NULL NULL NULL PA14 Aberdeen 4 |

| B003 Glasgow 3 PG4 Glasgow 3 |

| B002 London 2 PL94 London 2 |

| |

===============================================================

В этом примере при выполнении правого внешнего соединения в результирующую таблицу были включены не только те две строки, которые имеют одинаковые значения в сопоставляемых столбцах с названием города, но также и те строки из второй (правой) таблицы, которые не нашли соответствия со строками в первой (левой) таблице. В этой строке все поля из первой таблицы получили значения NULL.

Используем полное внешнее соединение этих таблиц, которое выглядит следующим образом:

SELECT b.\*, р.\*

FROM Branch b FULL JOIN PropertyForRent p ON b.bCity = p.pCity;

===================== SQL Execution Log ======================

| |

| branchNo bcity cityID propertyNo pCity cityID |

| -------- ------- ------ ---------- ------- ------ |

| NULL NULL NULL PA14 Aberdeen 4 |

| B003 Glasgow 3 PG4 Glasgow 3 |

| B004 Bristol 1 NULL NULL NULL |

| B002 London 2 PL94 London 2 |

| |

===============================================================

В случае полного внешнего соединения в результирующую таблицу помещаются не только те две строки, которые имеют одинаковые значения в сопоставляемых столбцах с названием города, но и все остальные строки исходных таблиц, не нашедшие себе соответствия. В этих строках все столбцы той таблицы, в которой не было найдено соответствия, заполняются значениями NULL.

**Самосоединения в SQL-92.**

В некоторых задачах необходимо получить информацию, выбранную особым образом только из одной таблицы. Для этого используются так называемые ***самосоединения***, или ***рефлексивные соединения***. Это не отдельный вид соединения, а просто соединение таблицы с собой с помощью псевдонимов. Самосоединения полезны в случаях, когда нужно получить пары аналогичных элементов из одной и той же таблицы.

|  |  |  |
| --- | --- | --- |
| **DEPT\_NO** | **DEPARTMENT** | **BUDGET** |
| 1 | Software Development | 400000.00 |
| 2 | Field Office: Canada | 500000.00 |
| 3 | Finance | 400000.00 |
| 4 | Field Office: East Coast | 500000.00 |
| 5 | Field Office: Japan | 500000.00 |
| 6 | Field Office: Singapore | 300000.00 |
| 7 | Field Office: Switzerland | 500000.00 |
| 8 | Quality Assurance | 300000.00 |

Таблица 9.3 Department.

Например, требуется получить список пар отделов с одинаковыми годовыми бюджетами:

SELECT d1.department, d2.department, d1.budget

FROM department d1, department d2

WHERE d1.budget = d2.budget;

===================== SQL Execution Log ==========================

| |

| DEPARTMENT DEPARTMENT BUDGET |

| ---------- ---------- --------- |

| Software Development Finance 400000.00 |

| Field Office: East Coast Field Office: Canada 500000.00 |

| Field Office: Japan Field Office: East Coast 500000.00 |

| Field Office: Japan Field Office: Canada 500000.00 |

| Field Office: Japan Field Office: Switzerland 500000.00 |

| Field Office: Singapore Quality Assurance 300000.00 |

| Field Office: Switzerland Field Office: East Coast 500000.00 |

| |

===================================================================

**Вложенные запросы**

С помощью SQL вы можете вкладывать запросы внутрь друга друга. Обычно, внутренний запрос генерирует значение, которое проверяется в предикате внешнего запроса, определяющего верно оно или нет. Вложенные запросы часто называют ***подзапросами***.

|  |  |  |  |
| --- | --- | --- | --- |
| **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 |

Таблица 9.4 Salespeople (Продавцы).

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

Таблица 9.5 Customers (Покупатели).

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **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 |

Таблица 9.6 Orders (заказы).

Например, предположим, что мы знаем имя продавца: Motika, но не знаем значение его поля snum, и хотим извлечь все заказы из таблицы заказов. Имеется один способ, чтобы сделать это:

SELECT \* FROM Orders WHERE snum = ( SELECT snum FROM Salespeople WHERE sname = ‘Motika’);

**Ограничения, налагаемые на вложенные запросы.**

Необходимо, чтобы наш подзапрос в предыдущем примере возвращал одно и только одно значение. Имея выбранным поле snum «WHERE city = ‘London’» вместо «WHERE sname = ‘Motika’», можно получить несколько различных значений. Это может сделать уравнение в предикате основного запроса невозможным для оценки верности или неверности, и команда выдаст ошибку. При использовании подзапросов в предикатах, основанных на реляционных операторах, вы должны убедиться, что использовали подзапрос, который будет выдавать одну и только одну строку вывода. Если вы используете подзапрос, который не выводит никаких значений вообще, команда не потерпит неудачи; но основной запрос не выведет никаких значений. Подзапросы, которые не производят никакого вывода (или нулевой вывод), вынуждают рассматривать предикат ни как верный, ни как неверный, а как неизвестный. Однако, неизвестный предикат имеет тот же самый эффект что и неверный: никакие строки не выбираются основным запросом.

Один тип функций, который автоматически может производить одиночное значение для любого числа строк, конечно же, – агрегатная функция.

Любой запрос, использующий одиночную функцию агрегата без предложения GROUP BY, будет выбирать одиночное значение для использования в основном предикате. Например, вы хотите увидеть все заказы имеющие сумму приобретений выше средней на 4-е Октября:

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

Вы можете использовать подзапросы, которые производят любое число строк если вы используете специальный оператор IN. Как вы помните, IN определяет набор значений, одно из которых должно совпадать с другим термином уравнения предиката в порядке, чтобы предикат был верным. Когда вы используете IN с подзапросом, SQL просто формирует этот набор из вывода подзапроса. Мы можем, следовательно, использовать IN чтобы выполнить такой же подзапрос, который не будет работать с реляционным оператором, и найти все атрибуты таблицы заказов для продавца в Лондоне:

SELECT \* FROM Orders WHERE snum IN ( SELECT snum FROM Salespeople WHERE city = ‘LONDON’ );

Операторы BETWEEN, LIKE, и IS NULL не могут использоваться с подзапросами, но можно использовать операторы ALL и ANY

Например: Выбрать заказчиков, которые имеют больший рейтинг чем любой заказчик в Риме.

SELECT \* FROM Customers WHERE rating > ANY ( SELECT rating FROM Customers WHERE city = ‘Rome’ );

С помощью ALL, предикат является верным, если каждое значение выбранное подзапросом удовлетворяет условию в предикате внешнего запроса. Если мы хотим пересмотреть наш предыдущий пример чтобы вывести только тех заказчиков, чьи оценки, фактически, выше, чем у каждого заказчика в Лондоне:

SELECT \* FROM Customers WHERE rating > ALL (SELECT rating FROM Customers WHERE city = ‘Rome’);

Смысл всех подзапросов тот, что все они выбирают одиночный столбец. Это обязательно, поскольку выбранный вывод сравнивается с одиночным значением. Подтверждением этому то, что SELECT \* не может использоваться в подзапросе. Имеется исключение из этого, когда подзапросы используются с оператором EXISTS.

**Соотнесенные вложенные запросы.**

Когда вы используете подзапросы в SQL, вы можете обратиться к внутреннему запросу таблицы в предложении внешнего запроса FROM, сформировав – ***соотнесенный подзапрос***. Когда вы делаете это, подзапрос выполняется неоднократно, по одному разу для каждой строки таблицы основного запроса. Соотнесенный подзапрос - один из большого количества тонких понятий в SQL из-за сложности в его оценке. Если вы сумеете овладеть им, вы найдете что он очень мощный, потому что может выполнять сложные функции с помощью очень лаконичных указаний.

Например, имеется один способ найти всех заказчиков в заказах на 3-е Октября:

SELECT \* FROM Customers outer WHERE 10/03/1990 IN ( SELECT odate

FROM Orders inner WHERE outer.cnum = inner.cnum );

EXISTS – это оператор, который производит верное или неверное значение, другими словами, выражение Буля. Это означает, что он может работать автономно в предикате или в комбинации с другими выражениями Буля использующими Булевы операторы AND, OR, и NOT. Он берет подзапрос как аргумент и оценивает его как верный, если тот производит любой вывод или как неверный, если тот не делает этого. Этим он отличается от других операторов предиката, в которых он не может быть неизвестным. 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’);

Или мы можем вывести продавцов, которые имеют многочисленых заказчиков:

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

Однако для нас может быть полезнее вывести больше информации об этих продавцах, а не только их номера. Мы можем сделать это, объединив таблицу Заказчиков с таблицей Продавцов:

SELECT DISTINCT first.snum, sname, first.city FROM Salespeople first, Customers second WHERE EXISTS ( SELECT \* FROM Customers third WHERE second.snum = third.snum AND second.cnum < > third.cnum ) AND first.snum = second.snum;

**Контрольные вопросы:**

1. Как производится выборка данных?
2. Как производится обновление данных?
3. Как производится удаление данных?
4. Как добавляются строки в таблицу?
5. Что такое агрегатная функция?
6. Перечислите основные агрегатные функции.
7. Где используется HAVING?
8. Какие виды соединений бывают?
9. Что такое самосоединение?
10. Объясните пошагово как производится соединение?
11. Как выполняется подзапрос?
12. Что такое соотнесенный подзапрос?

**Литература:**

1. Thomas Connolly, Carolyn Begg – Database systems. A practical Approach to Design, Implementation and Management. 4th Edition – Addison Wesley 2005 – 1373p.
2. C. J. Date – An Introduction to Database Systems – Addison-Wesley Professional – 2003 – 1024 p.