Лекция 6. Работа с множествами

Хотя можно работать и с отдельными строками данных, реляционные БД на самом деле приспособлены для работы с наборами (множествами). В данной лекции будут изучены комбинации нескольких таблиц с использованием различных операторов работы с множествами.

Основы теории множеств

Пусть даны два множества A и B.

Основные операции теории множеств (рисунки 6.1-6.3):

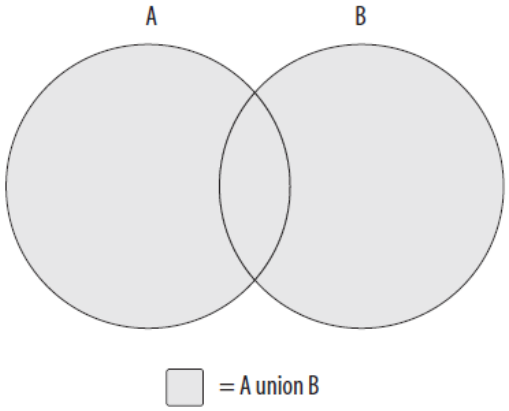

Рисунок 6.1 – Объединение A и B

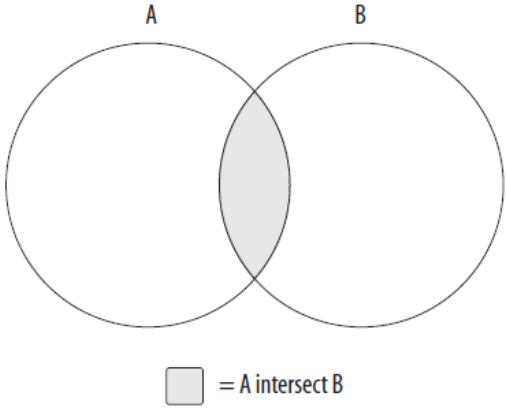

Рисунок 6.2 – Пересечение A и B

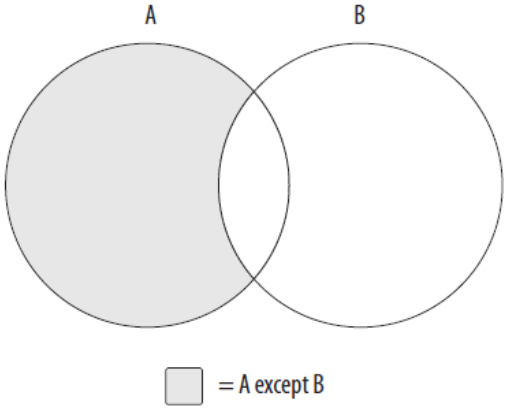

Рисунок 6.3 – Разность A и B

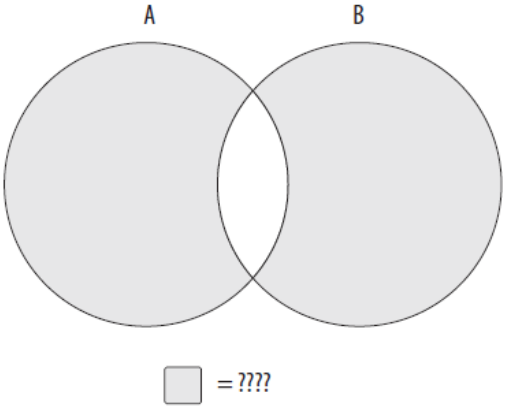

Рисунок 6.4 – Загадочное множество

Какими способами можно построить множество данных, заданное на рисунке 6.4?

1) (A union B) except (A intersect B)

2) (A except B) union (B except A)

Таким образом, применяя эти три операции или их сочетания, можно получать любые нужные результаты.

Теория множеств на практике

Рассмотрим, как упомянутые выше концепции могут быть применены к реляционным БД с использованием SQL операторов работы с множествами.

Представим, что нам необходимо сгенерировать *объединение* таблиц **product** и **customer**:

```
mysql> DESC product;
+-----------------+-------------+------+-----+---------+-------+
| Field           | Type        | Null | Key | Default | Extra |
+-----------------+-------------+------+-----+---------+-------+
| product_cd      | varchar(10) | NO   | PRI | NULL    |       |
| name            | varchar(50) | NO   |     | NULL    |       |
| product_type_cd | varchar(10) | NO   | MUL | NULL    |       |
| date_offered    | date        | YES  |     | NULL    |       |
| date_retired    | date        | YES  |     | NULL    |       |
+-----------------+-------------+------+-----+---------+-------+
5 rows in set (0.23 sec)
 mysql> DESC customer;
+--------------+------------------+------+-----+---------+----------------+
| Field        | Type             | Null | Key | Default | Extra          |
+--------------+------------------+------+-----+---------+----------------+
| cust_id      | int(10) unsigned | NO   | PRI | NULL    | auto_increment |
| fed_id       | varchar(12)      | NO   |     | NULL    |                |
| cust_type_cd | enum('I','B')    | NO   |     | NULL    |                |
| address      | varchar(30)      | YES  |     | NULL    |                |
| city         | varchar(20)      | YES  |     | NULL    |                |
| state        | varchar(20)      | YES  |     | NULL    |                |
| postal_code  | varchar(10)      | YES  |     | NULL    |                |
+--------------+------------------+------+-----+---------+----------------+
7 rows in set (0.04 sec)
```

После комбинирования первый столбец результирующей таблицы был бы комбинацией столбцов *product.product_cd* и *customer.cust_id*, второй – комбинацией столбцов *product.name* и *customer.fed_id* и т. д. Хотя некоторые пары столбцов сочетаются без труда (например, два столбца числового типа), неясно, как должны объединяться пары столбцов разного типа, такие как числовой со строковым или строковый с датой. Кроме того, в шестом и седьмом столбцах комбинированной таблицы будут только данные шестого и седьмого столбцов таблицы **customer**, поскольку в таблице **product** всего пять столбцов. Следовательно, таблицы, подлежащие объединению, должны обладать некоторой общностью.

При применении операций с множествами к реальным таблицам необходимо соблюдать следующие правила:

- В обеих таблицах должно быть одинаковое число столбцов.
- Типы данных столбцов двух таблиц должны быть одинаковыми (или сервер должен уметь преобразовывать один тип в другой).

Операции с множествами осуществляются путем помещения *оператора работы с множествами* (*set operator*) между двух выражений *select*:

```
mysql> SELECT 1 num, 'abc' str
    -> UNION
    -> SELECT 9 num, 'xyz' str;
+-----+-----+
| num | str |
+-----+-----+
|   1 | abc |
|   9 | xyz |
+-----+-----+
```

Таблицы, состоящие из одной строки

Оператор работы с множествами, в данном случае *union*, указывает серверу БД объединить все строки двух таблиц. Таким образом, итоговая таблица включает две строки и два столбца. Такой запрос называют ***составным запросом*** (*compound query*), потому что он объединяет несколько независимых запросов.

Операторы работы с множествами

1) Оператор *union*

Операторы *union* (объединить) и *union all* (объединить все) позволяют комбинировать несколько таблиц. Разница в том, что, если требуется объединить две таблицы, включая в окончательный результат *все* их строки с учетом дублирующих значений, нужно использовать оператор *union all*. Благодаря оператору *union all* в результирующей таблице всегда будет столько строк, сколько во всех исходных таблицах в сумме.

Пример. Применение оператора *union all* для формирования полного множества данных клиентов из двух таблиц подтипов клиентов:

```
mysql> SELECT 'IND' type_cd, cust_id, lname name
    -> FROM individual
    -> UNION ALL
    -> SELECT 'BUS' type_cd, cust_id, name
    -> FROM business;
```

```
+---------+---------+------------------------+
| type_cd | cust_id | name                   |
+---------+---------+------------------------+
| IND     |       1 | Hadley                 |
| IND     |       2 | Tingley                |
| IND     |       3 | Tucker                 |
| IND     |       4 | Hayward                |
| IND     |       5 | Frasier                |
| IND     |       6 | Spencer                |
| IND     |       7 | Young                  |
| IND     |       8 | Blake                  |
| IND     |       9 | Farley                 |
| BUS     |      10 | Chilton Engineering    |
| BUS     |      11 | Northeast Cooling Inc. |
| BUS     |      12 | Superior Auto Body     |
| BUS     |      13 | AAA Insurance Inc.     |
```

13 клиентов

Пример составного запроса, по которому возвращаются дублирующие данные:

```
mysql> SELECT emp_id
    -> FROM employee
    -> WHERE assigned_branch_id = 2
    ->   AND (title = 'Teller' OR title = 'Head Teller')
    -> UNION ALL
    -> SELECT DISTINCT open_emp_id
    -> FROM account
    -> WHERE open_branch_id = 2;
+--------+
| emp_id |
+--------+
|     10 |
|     11 |
|     12 |
|     10 |
+--------+
```

Первый запрос составного выражения выбирает всех операционистов отделения Woburn, а второй возвращает другое множество – операционистов, открывавших счета в отделении Woburn.

Чтобы *исключить* дублирующие строки из составной таблицы, вместо оператора *union all* нужно использовать оператор *union*:

```
mysql> SELECT emp_id
    -> FROM employee
    -> WHERE assigned_branch_id = 2
    ->   AND (title = 'Teller' OR title = 'Head Teller')
    -> UNION
    -> SELECT DISTINCT open_emp_id
    -> FROM account
    -> WHERE open_branch_id = 2;
```

```
+--------+
| emp_id |
+--------+
|     10 |
|     11 |
|     12 |
+--------+
```

2) Оператор *intersect*

Спецификация SQL ANSI включает оператор *intersect* (пересечение), предназначенный для выполнения пересечений. Если два запроса составного запроса возвращают неперекрывающиеся таблицы, пересечением будет **пустое множество**.

Рассмотрим предыдущий запрос с применением оператора *intersect*:

```
SELECT emp_id
FROM employee
WHERE assigned_branch_id = 2
  AND (title = 'Teller' OR title = 'Head Teller')
INTERSECT
SELECT DISTINCT open_emp_id
FROM account
WHERE open_branch_id = 2;
+--------+
| emp_id |
+--------+
|     10 |
+--------+
```

Пересечение этих двух запросов дает сотрудника с ID равным 10, что является единственным значением, имеющимся в результирующих наборах обоих запросов.

Наряду с оператором *intersect*, удаляющим все дублирующие строки области перекрытия, спецификация SQL ANSI предлагает оператор *intersect all*, не удаляющий дубликаты. Однако, единственный сервер БД, в настоящее время реализующий оператор *intersect all*, – **DB2 Universal Server** компании IBM.

3) Оператор *except*

Спецификация SQL ANSI включает оператор *except* (разность), предназначенный для выполнения операции разности заданных множеств данных. Операция *except* возвращает первую таблицу за вычетом всех перекрытий со второй таблицей.

```
SELECT emp_id
FROM employee
WHERE assigned_branch_id = 2
  AND (title = 'Teller' OR title = 'Head Teller')
EXCEPT
SELECT DISTINCT open_emp_id
FROM account
WHERE open_branch_id = 2;
+--------+
| emp_id |
+--------+
|     11 |
|     12 |
+--------+
```

Оператор *minus* в Oracle Database

В данном запросе результирующий набор включает три строки из результирующего набора первого запроса минус сотрудник с ID=10, который присутствует в результирующих наборах обоих запросов. В спецификации SQL ANSI также описан оператор *except all*, но опять же он реализован только в DB2 Universal Server.

В операторе *except all* есть небольшая хитрость. Предположим, что имеются два множества данных, имеющих следующий вид:

Множество A

```
+--------+
| emp_id |
+--------+
|     10 |
|     11 |
|     12 |
|     10 |
|     10 |
+--------+
```

Множество B

```
+--------+
| emp_id |
+--------+
|     10 |
|     10 |
+--------+
```

A except B:

```
+--------+
| emp_id |
+--------+
|     11 |
|     12 |
+--------+
```

A except all B:

```
+--------+
| emp_id |
+--------+
|     10 |
|     11 |
|     12 |
+--------+
```

Разница между этими двумя операциями в том, что *except* удаляет все экземпляры дублирующихся данных из множества A, тогда как *except all* удаляет из множества A только один экземпляр дубликата данных для каждого экземпляра дубликата данных множества B.

Правила операций с множествами

Результаты сортирующего составного запроса

Если требуется отсортировать результаты составного запроса, после последнего входящего в него запроса можно добавить блок *order by*. В блоке *order by* указываются имена столбцов из **первого запроса** составного запроса.

```
mysql> SELECT emp_id, assigned_branch_id
    -> FROM employee
    -> WHERE title = 'Teller'
    -> UNION
    -> SELECT open_emp_id, open_branch_id
    -> FROM account
    -> WHERE product_cd = 'SAV'
    -> ORDER BY emp_id;

+--------+--------------------+
| emp_id | assigned_branch_id |
+--------+--------------------+
|      1 |                  1 |
|      7 |                  1 |
|      8 |                  1 |
|      9 |                  1 |
|     10 |                  2 |
|     11 |                  2 |
|     12 |                  2 |
|     14 |                  3 |
```

В этом примере в двух запросах заданы разные имена столбцов. Если в блоке *order by* указать имя столбца из второго запроса, это приведет к ошибке. Чтобы избежать этой проблемы, рекомендуется в обоих запросах давать столбцам **одинаковые псевдонимы**.

Старшинство операций с множествами

Порядок расположения разных операторов работы с множествами в составном запросе имеет значение. В общем, составные запросы из трех или больше запросов оцениваются в порядке **сверху вниз**, но с учетом следующих моментов:

- По спецификации SQL ANSI из всех операторов работы с множествами первым выполняется оператор *intersect*.

- Порядок сочетания запросов можно задавать с помощью скобок.

Литература:

1. Алан Бьюли. Изучаем SQL: пер. с англ. – СПб-М.: Символ, O'Reilly, 2007. – 310 с.