61. Работа с запросами. Общие принципы построения запросов. Инструменты создания запросов. Типы объединений таблиц. Групповые операции. Условия отбора результатов. Параметры запросов. Перекрестные запросы.

http://kreker.org/items/5

1 Общие принципы построения запросов

Основным инструментом выборки данных в языке SQL является команда SELECT. С помощью этой команды можно получить доступ к данным, представленным как совокупность таблиц практически любой сложности.

Чаще всего используется упрощенный вариант команды SELECT, имеющий следующий синтаксис:

```
SELECT <Список_выбора>
[INTO <Новая_таблица>]
FROM <Исходная_таблица>
[WHERE <Условие_отбора>]
[GROUP BY <Ключи_группировки>]
[HAVING <Условие_отбора>]
[ORDER BY <Ключи_сортировки> [ASC | DESC] ]
```

Инструкция SELECT разбивается на отдельные разделы, каждый из которых имеет свое назначение. Обязательными являются только разделы SELECT и FROM, а остальные разделы могут быть опущены. Полный список разделов следующий:

```
SELECT UNION
INTO ORDER BY
FROM COMPUTE
WHERE FOR
GROUP BY OPTION
HAVING
```

Основное назначение радела SELECT – задание набора столбцов, возвращаемых после выполнения запроса, т.е. внешнего вида результата. В простейшем случае возвращается столбец одной из таблиц, участвующих в запросе. В более сложных ситуациях набор значений в столбце формируется как результат вычисления выражения. Такие столбцы называются вычисляемыми, и по умолчанию им не присваивается никакого имени.

При необходимости пользователь может указать для столбца, возвращаемого после выполнения запроса, произвольное имя. Такое имя называется псевдоним (alias). В обычной ситуации назначение псевдонима необязательно, но в некоторых ситуациях требуется явное его указание. Наиболее часто это требуется при работе с разделом INTO, в котором каждый из возвращаемых столбцов должен иметь имя, и это имя должно быть уникально.

2 Инструменты создания запросов

Вместо ручного написания запросов можно использовать инструменты генерации запросов. Они поставляются вместе с СУБД или в виде отдельного продукта и предоставляют визуальный конструктор для создания запроса. Например, генераторы запросов есть в продуктах:

- dbForge Studio for MySQL
- SQL Server Management Studio

С некоторой натяжкой, инструментом создания запросов можно назвать и ORM, т.к. она избавляет разработчика от необходимости ручного написания SQL-кода.

3 Типы объединений таблиц

Рассмотрим типы объединения на примере MySQL.

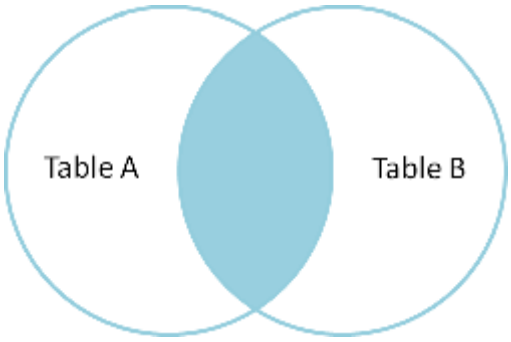
- CROSS JOIN, он же INNER JOIN, он же JOIN
- Аналоги FULL OUTER JOIN для MySQL
- LEFT JOIN
- RIGHT JOIN
- NATURAL JOIN
- STRAIGHT JOIN
- Если не указать USING или ON в объединении (Декартова выборка)

INNER JOIN

— INNER JOIN производит выборку записей, которые только существуют в TableA и TableB одновременно.
— CROSS JOIN — это эквивалент INNER JOIN.
— INNER JOIN можно заменить условием объединения в WHERE.

Запрос:

```
SELECT * FROM `TableA`
    INNER JOIN `TableB`
    ON `TableA`.`name` = `TableB`.`name`
```



FULL OUTER JOIN*

*Не доступно в MySQL

FULL OUTER JOIN производит выборку всех записей из TableA и TableB, вне зависимости есть ли соответствующая запись в соседней таблице. Если таковой нет, то недостающая сторона будет содержать пустой указатель и результатом будет выводится NULL.

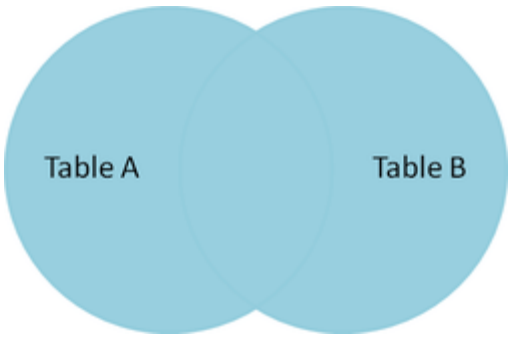
Запрос:

```
SELECT * FROM `TableA`
    FULL OUTER JOIN `TableB`
    ON `TableA`.`name` = `TableB`.`name`
```

В MySQL нечто похожее можно получить запросом:

```
SELECT `TableA`.*, `TableB`.* FROM `TableA`
```

```
    LEFT JOIN `TableB` USING (`name`)
UNION SELECT `TableA`.*, `TableB`.* FROM `TableB`
    LEFT JOIN `TableA`
    USING (`name`)
    WHERE `TableA`.`name` IS NULL
```



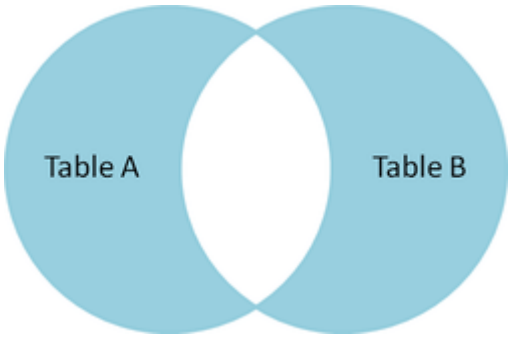
Чтобы произвести выборку уникальных записей из двух таблиц (значения одной таблицы отсутствуют в другой), мы воспользуемся тем же FULL OUTER JOIN, указав, что NULL может быть как в результате одной таблицы, так и в результате другой.

Запрос:

```
SELECT * FROM `TableA`
    FULL OUTER JOIN `TableB`
    ON `TableA`.`name` = `TableB`.`name`
    WHERE `TableA`.`id` IS NULL OR `TableB`.`id` IS NULL
```

В MySQL нечто похожее можно получить запросом:

```
SELECT `TableA`.*, `TableB`.* FROM
    TableA LEFT JOIN `TableB`
    USING (`name`)
    WHERE `TableB`.`name` IS NULL
UNION SELECT `TableA`.*, `TableB`.* FROM `TableB `
    LEFT JOIN `TableA` USING (`name`)
    WHERE `TableA`.`name` IS NULL
```

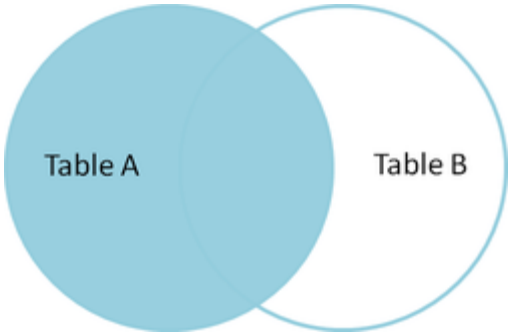


LEFT JOIN

LEFT OUTER JOIN (LEFT JOIN) указывает, что левая таблица управляющая (в нашем случае TableA) и производит из нее полную выборку, осуществляя поиск соответствующих записей в таблице TableB. Если таких соответствий не найдено, то база вернет пустой указатель - NULL. Указание OUTER - не обязательно.

Запрос:

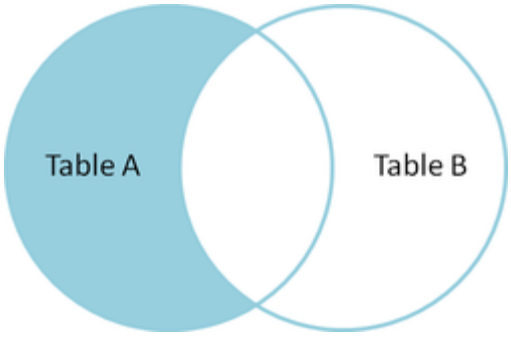
```
SELECT * FROM `TableA`
    LEFT JOIN `TableB`
    ON `TableA`.`name` = `TableB`.`name`
```



Чтобы произвести выборку записей из таблицы TableA, которых не существует в таблице TableB, мы выполняем LEFT JOIN, но затем из результата исключаем записи, которые не хотим видеть, путем указания, что TableB.id является нулем (указывая, что записи нет в таблице TableB).

Запрос:

```
SELECT * FROM `TableA`
    LEFT JOIN `TableB`
    ON `TableA`.`name` = `TableB`.`name`
    WHERE `TableB`.`id` IS NULL
```



RIGHT JOIN

RIGHT JOIN выполняет те же самые функции, что и LEFT JOIN, за исключением того, что правая таблица будет прочитана первой. Таким образом, если в запросах из предыдущей главы LEFT заменить на RIGHT, то таблица результатов, грубо говоря, отразится по вертикали. То есть, в результате вместо значений TableA будут записи TableB и наоборот.

NATURAL JOIN

Суть этой конструкции в том, что база сама выбирает, по каким столбцам сравнивать и объединять таблицы. А выбор этот падает на столбцы с одинаковыми именами. В этом кроется засада т.к. база может выбрать совершенно не те столбцы для объединения и запрос будет работать совершенно не так, как вы предполагали.

STRAIGHT JOIN

STRAIGHT JOIN выполняет те же функции, что и обычный INNER JOIN, за исключением того, что левая таблица читается раньше правой.

Декартова выборка

Если при объединении таблиц не указать условие объединения через ON или USING, то база произведет так называемую Декартову выборку, когда значению одной таблицы приравнивается каждое значение другой. Таким образом, СУБД, в нашем случае, возвращает 4x4 = 16 строк.

4 Групповые операции

GROUP BY – выполняет группировку строк таблицы по определенным критериям. Используется обычно вместе с агрегирующими функциями:

- AVG
- COUNT
- MAX
- MIN
- SUM

Общая схема запроса при этом принимает вид:

```
SELECT  [column, ] GROUP function (column) , ...
FROM table
[WHERE condition(s);]
[HAVING group_condition]
[GROUP BY column]
[ORDER BY {column | expression} [asc | desc ]]
```

Пример:

```
SELECT job, AVG(sal)
FROM emp
GROUP BY job
```

5 Условия отбора результатов

Для фильтрации результатов используется выражение WHERE – тут все понятно.

Для фильтрации результатов в групповых запросах используется HAVING. Пример:

```
SELECT job, MIN (sal)
FROM emp
HAVING avg(sal) < 2000
GROUP BY job
```

6 Параметры запросов

Параметром называется переменная, используемая в SQL-операторе.

Применение параметров позволяет формировать SQL-операторы непосредственно во время выполнения приложения.

Например:

```
INSERT INTO TBL1 (F_ID, F2, F3) VALUES (?, ?, ?)
```

Параметры могут быть именованными и позиционными.

Позиционные параметры указываются символом вопросительный знак (?), называемым маркером параметров. При выполнении оператора вместо параметра в соответствующую позицию SQL-оператора приставляется значение параметра.

Согласно спецификации языка SQL маркеры параметров нельзя размещать в следующих местах SQL-оператора:

- в списке полей оператора SELECT;
- одновременно как оба операнда для бинарного оператора (например, оператора =), так как на этапе компиляции нельзя определить тип операндов, выступающих в роли параметров;
- одновременно как первый и второй операнд для оператора BETWEEN (например, вместо SQL-оператора
- ```
  SELECT title_id, f_sales FROM tbl1
  WHERE f_sales BETWEEN 4095 AND 12000
  ```
  нельзя записать
  ```
  SELECT title_id, f_sales FROM tbl1
  WHERE ? BETWEEN ? AND 12000 );
  ```
- одновременно как первый и третий операнд для оператора BETWEEN;
- одновременно как первый операнд и второй операнд для оператора IN (например, вместо SQL-оператора
- ```
  SELECT f_ID, f_state FROM tbl1
  WHERE f_state IN ('CA', 'CB')
  ```
  нельзя записать
  ```
  SELECT f_ID, f_state FROM tbl1
  WHERE ? IN (?, ?)  );
  ```
- как операнд унарного оператора + или -.

7 Перекрестные запросы

Перекрестные запросы (*Crosstab Query*) являются еще одной специфической разновидностью запросов на выборку. Предназначены они для более глубокого анализа информации, хранящейся в таблицах.

Ключевым словом SQL-оператора перекрестного запроса, задающим его тип, является слово TRANSFORM (преобразовать). Это подразумевает, что значения одного из столбцов (полей) выборки, будут преобразованы в названия столбцов итоговой выборки.

Результаты перекрестного запроса группируются по двум наборам данных, один из которых расположен в левом столбце (столбцах) таблицы, а второй — в верхней строке. В остальном пространстве таблицы отображаются результаты статистических расчетов (Sum, Count и т.д.), выполненных над данными трансформированного поля.

Пример (Transact-SQL):

```
SELECT
  MONTH(Date) AS SaleMonth,
  SUM(CASE YEAR(Date)
        WHEN 2001 THEN Amount
        ELSE 0
      END) AS [2001],
  SUM(CASE YEAR(Date)
        WHEN 2002 THEN Amount
        ELSE 0
      END) AS [2002],
  SUM(CASE YEAR(Date)
        WHEN 2003 THEN Amount
        ELSE 0
      END) AS [2003]
FROM Sales
```

```
GROUP BY MONTH(Date)
ORDER BY MONTH(Date)
GO
```

Выполняется для таблицы созданной запросом:

```
CREATE TABLE Sales
(
SaleID int IDENTITY PRIMARY KEY CLUSTERED,
ClientID int,
Date datetime,
Amount money
)
insert Sales values(1,'20010401', 15.48)
insert Sales values(1,'20020302', 134.01)
insert Sales values(1,'20031003', 2346.03)
insert Sales values(2,'20030203', 754.88)
insert Sales values(3,'20010301', 73.07)
insert Sales values(3,'20030402', 734.46)
insert Sales values(4,'20010301', 1567.10)
insert Sales values(4,'20020404', 6575.70)
insert Sales values(4,'20030307', 6575.77)
insert Sales values(4,'20030309', 6575.37)
insert Sales values(5,'20011201', 1975.73)
insert Sales values(5,'20030306', 178965.63)
insert Sales values(6,'20020103', 16785.34)
insert Sales values(6,'20030304', 1705.44)
GO
```