**Модуль №6 «Объединения»**

**Слайд №2. Цели занятия:**

* закрепить навыки применения подзапросов;
* изучить операции горизонтального соединения данных (JOIN-соединения);
* изучить принципы объединения результирующих наборов с помощью операторов UNION, UNION ALL;
* рассмотреть понятие транзакции;
* повторить и закрепить на практике пройденный по курсу «Теория баз данных» материал.

**Слайд №3. План занятия:**

1. Актуализация знаний.

2. Соединения JOIN:

* понятие inner join;
* понятие left join;
* понятие right join;
* понятие full join.

3. Объединение результатов запроса:

* принципы объединения;
* ключевое слово UNION;
* ключевое слово UNION ALL.

4. Понятие транзакции. Использование транзакций.

5. Повторение пройденного материала по курсу «Теория баз данных».

6. Домашнее задание.

**Теоретический материал**

**Соединение таблиц** в операторе SELECT является одним из наиболее мощных и элегантных средств реляционных баз данных.

С помощью соединения можно получать данные из двух или нескольких таблиц на основе логических связей между ними. Соединения указывают, как Microsoft SQL Server должен использовать данные из одной таблицы для выбора строк из другой таблицы.

Соединение определяет способ связывания двух таблиц в запросе следующим образом:

* для каждой таблицы указываются столбцы, используемые в соединении. В типичном условии соединения указывается внешний ключ из одной таблицы и связанный с ним ключ из другой таблицы;
* указывается логический оператор (например, = или <>,) для сравнения значений столбцов.

## Для реализации операции соединения используется оператор языка SQL JOIN. Входит в раздел FROM операторов SELECT, UPDATE или DELETE. Этот оператор является наиболее важным оператором для реляционных систем баз данных, поскольку благодаря ему имеется возможность распределять данные по нескольким таблицам, обеспечивая, таким образом, важное свойство систем баз данных — *отсутствие избыточности данных*.

## Две синтаксические формы реализации соединений

## Для соединения таблиц можно использовать две разные синтаксические формы оператора соединения:

## явный синтаксис соединения (синтаксис соединения ANSI SQL:1992);

## неявный синтаксис соединения (синтаксис соединения "старого стиля").

## Неявный синтаксис оператора соединения является синтаксисом "старого стиля", где каждая операция соединения определяется неявно посредством предложения WHERE, используя так называемые столбцы соединения.

## Пример соединения с использованием неявного синтаксиса соединения:

## Select Books.name, Themes.name

## from books, Themes

## where books.id\_press=themes.id

## Синтаксис соединения ANSI SQL:1992 был введен стандартом SQL92 (SQL-92 — третья версия языка запросов к базам данных SQL, Американский национальный институт стандартов) и определяет операции соединения явно, т. е. используя соответствующее имя для каждого типа операции соединения. При явном объявлении соединения используются следующие ключевые слова:

## CROSS JOIN;

## [INNER] JOIN;

## LEFT [OUTER] JOIN;

## RIGHT [OUTER] JOIN;

## FULL [OUTER] JOIN.

## Оператор соединения JOIN имеет несколько разных форм. Мы рассмотрим следующие основные формы соединений:

## внутреннее соединение (INNER JOIN);

## внешнее соединение (LEFT OUTER JOIN, RIGHT OUTER JOIN, FULL OUTER JOIN);

## декартово произведение или перекрестное соединение (CROSS JOIN).

## Ключевое слово CROSS JOIN определяет декартово произведение двух таблиц. Ключевое слово INNER JOIN определяет естественное соединение двух таблиц, а LEFT OUTER JOIN и RIGHT OUTER JOIN определяют одноименные операции соединения. Наконец, ключевое слово FULL OUTER JOIN определяет соединение правого и левого внешнего соединений.

## ПРИМЕЧАНИЕ

## Для операций соединения рекомендуется использовать явный синтаксис, т. к. это повышает надежность запросов. То, что условия соединения задаются в предложении FROM, помогает отделить их от условий поиска, которые могут быть заданы в предложении WHERE. Соединение рекомендуется задавать именно таким способом.

## Синтаксис соединения с использованием предложения FROM:

## SELECT field\_name [,... n]

## FROM Table1

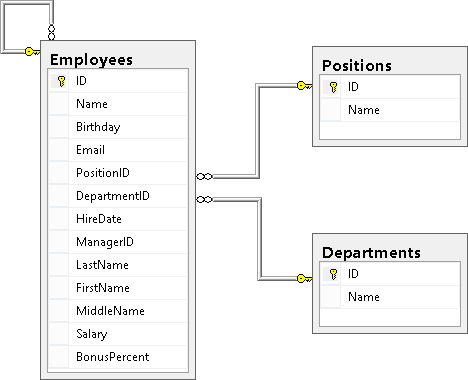
## {INNER | {LEFT | RIGHT | FULL} OUTER | CROSS } JOIN Table2

## {ON <condition> }

## В большинстве СУБД при указании слов LEFT, RIGHT, FULL слово OUTER можно опустить. Слово INNER также в большинстве СУБД можно опустить. В общем случае СУБД при выполнении соединения проверяет условие (предикат) condition.

## JOIN-соединения – операции горизонтального соединения данных

Здесь нам очень пригодится знание структуры БД, т.е. какие в ней есть таблицы, какие данные хранятся в этих таблицах и по каким полям таблицы связаны между собой. Первым делом всегда досконально изучайте структуру БД, т.к. нормальный запрос можно написать только тогда, когда ты знаешь, что откуда берется. У нас структура состоит из 3-х таблиц Employees, Departments и Positions. Приведу здесь диаграмму из первой части:



## Добавим немного новых данных

Для демонстрационных целей добавим несколько отделов и должностей:

SET IDENTITY\_INSERT Departments ON

INSERT Departments(ID,Name) VALUES(5,N'Маркетинг и реклама')

INSERT Departments(ID,Name) VALUES(6,N'Логистика')

SET IDENTITY\_INSERT Departments OFF

SET IDENTITY\_INSERT Positions ON

INSERT Positions(ID,Name) VALUES(5,N'Маркетолог')

INSERT Positions(ID,Name) VALUES(6,N'Логист')

INSERT Positions(ID,Name) VALUES(7,N'Кладовщик')

SET IDENTITY\_INSERT Positions OFF

**Если суть РДБ – разделяй и властвуй, то суть операций объединений снова склеить разбитые по таблицам данные, т.е. привести их обратно в человеческий вид.**

Если говорить просто, то операции горизонтального соединения таблицы с другими таблицами используются для того, чтобы получить из них недостающие данные.

Начнем с теории. Есть пять типов соединения:

1. **JOIN** – левая\_таблица JOIN правая\_таблица ON условия\_соединения
2. **LEFT JOIN** – левая\_таблица LEFT JOIN правая\_таблица ON условия\_соединения
3. **RIGHT JOIN** – левая\_таблица RIGHT JOIN правая\_таблица ON условия\_соединения
4. **FULL JOIN** – левая\_таблица FULL JOIN правая\_таблица ON условия\_соединения
5. **CROSS JOIN** – левая\_таблица CROSS JOIN правая\_таблица

|  |  |  |
| --- | --- | --- |
| **Краткий синтаксис** | **Полный синтаксис** | **Описание** |
| JOIN | INNER JOIN | Из строк левой\_таблицы и правой\_таблицы объединяются и возвращаются только те строки, по которым выполняются условия\_соединения. |
| LEFT JOIN | LEFT OUTER JOIN | Возвращаются все строки левой\_таблицы (ключевое слово LEFT). Данными правой\_таблицы дополняются только те строки левой\_таблицы, для которых выполняются условия\_соединения. Для недостающих данных вместо строк правой\_таблицы вставляются NULL-значения. |
| RIGHT JOIN | RIGHT OUTER JOIN | Возвращаются все строки правой\_таблицы (ключевое слово RIGHT). Данными левой\_таблицы дополняются только те строки правой\_таблицы, для которых выполняются условия\_соединения. Для недостающих данных вместо строк левой\_таблицы вставляются NULL-значения. |
| FULL JOIN | FULL OUTER JOIN | Возвращаются все строки левой\_таблицы и правой\_таблицы. Если для строк левой\_таблицы и правой\_таблицы выполняются условия\_соединения, то они объединяются в одну строку. Для строк, для которых не выполняются условия\_соединения, NULL-значения вставляются на место левой\_таблицы, либо на место правой\_таблицы, в зависимости от того данных какой таблицы в строке не имеется. |
| CROSS JOIN | - | Объединение каждой строки левой\_таблицы со всеми строками правой\_таблицы. Этот вид соединения иногда называют декартовым произведением. |

Как видно из таблицы полный синтаксис от краткого отличается только наличием слов INNER или OUTER.

Понимание каждого вида соединения очень важно, т.к. от применения того или иного вида, результат запроса может отличаться.

**Использование внутренних соединений**

**Оператор внутреннего соединения INNER JOIN** (синоним JOIN, ключевое слово INNER можно опустить) соединяет две таблицы. Порядок таблиц для оператора неважен, поскольку оператор является симметричным.

Это наиболее часто используемое в SQL соединение. Оно возвращает пересечение двух множеств. В терминах таблиц, оно возвращает только записи из обеих таблиц, отвечающие указанному критерию.

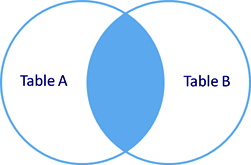
SELECT \* FROM TableA

**INNER JOIN** TableB

ON TableA.name = TableB.name

левая\_таблица JOIN правая\_таблица ON условия\_соединения

Из строк левой\_таблицы и правой\_таблицы объединяются и возвращаются только те строки, по которым выполняются условия\_соединения.



В стандарте SQL **внутренние соединения** *задаются предложением FROM или предложением WHERE*. Это единственный тип соединения, допустимый в стандарте SQL для предложения WHERE. Внутренние соединения, задаваемые предложением WHERE, известны как внутренние соединения старого типа.

*-- JOIN вернет 5 строк*

**SELECT** emp.**ID**,emp.**Name**,emp.DepartmentID,dep.**ID**,dep.**Name**

**FROM** Employees emp

**JOIN** Departments dep **ON** emp.DepartmentID=dep.**ID**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ID** | **Name** | **DepartmentID** | **ID** | **Name** |
| 1000 | Иванов И.И. | 1 | 1 | Администрация |
| 1001 | Петров П.П. | 3 | 3 | ИТ |
| 1002 | Сидоров С.С. | 2 | 2 | Бухгалтерия |
| 1003 | Андреев А.А. | 3 | 3 | ИТ |
| 1004 | Николаев Н.Н. | 3 | 3 | ИТ |

Внутренние соединения возвращают результат, когда в обеих таблицах есть хотя бы одна строка, соответствующая условиям соединения. Внутренние соединения исключают строки, не соответствующие ни одной строке в другой таблице.

**Внешние соединения**

**Внешние соединения** возвращают все строки хотя бы из одной таблицы или представления, упомянутых в предложении FROM, если они удовлетворяют условиям поиска WHERE или HAVING. Все строки, получаемые из левой таблицы, образуют левое внешнее соединение, а строки, получаемые из правой таблицы, — правое внешнее соединение. Все строки их обеих таблиц возвращаются в полном внешнем соединении.

Для внешних соединений в предложении FROM SQL Server использует ключевые слова ISO:

LEFT OUTER JOIN или LEFT JOIN;

RIGHT OUTER JOIN или RIGHT JOIN;

FULL OUTER JOIN или FULL JOIN.

Т.е. внешнее соединение представляет собой соединение двух таблиц, в результат которого в обязательном порядке входят все строки либо одной, либо обеих таблиц.

**LEFT OUTER JOIN**

Оператор левого внешнего соединения **LEFT OUTER JOIN** соединяет две таблицы. Порядок таблиц для оператора важен, поскольку оператор не является симметричным.

Ключевое слово OUTER можно опустить. Запись LEFT JOIN идентична LEFT OUTER JOIN.

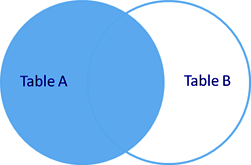
Оператор выбирает все запрошенные данные из таблицы в левой части JOIN плюс данные из правой таблицы, пересекающиеся с первой таблицей.

SELECT \* FROM TableA

LEFT [OUTER] JOIN TableB

ON TableA.name = TableB.name

Возвращаются все строки левой\_таблицы (ключевое слово LEFT). Данными правой\_таблицы дополняются только те строки левой\_таблицы, для которых выполняются условия\_соединения. Для недостающих данных вместо строк правой\_таблицы вставляются NULL-значения.



*-- LEFT JOIN вернет 7 строк*

**SELECT** emp.**ID**,emp.**Name**,emp.DepartmentID,dep.**ID**,dep.**Name**

**FROM** Employees emp

**LEFT** **JOIN** Departments dep **ON** emp.DepartmentID=dep.**ID**

**RIGHT OUTER JOIN**

Оператор правого внешнего соединения RIGHT OUTER JOIN соединяет две таблицы. Порядок таблиц для оператора важен, поскольку оператор не является симметричным.

Ключевое слово OUTER можно опустить. Запись RIGHT JOIN идентична RIGHT OUTER JOIN.

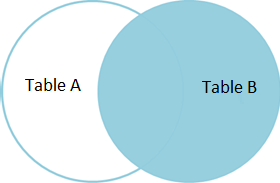
Оператор выдает все запрошенные данные из таблицы, стоящей в правой части оператора JOIN, плюс данные из левой таблицы, пересекающиеся с правой.

**SELECT** \* **FROM** TableA

**RIGHT [OUTER] JOIN** TableB

**ON** TableA.name = TableB.name

Возвращаются все строки правой\_таблицы (ключевое слово RIGHT). Данными левой\_таблицы дополняются только те строки правой\_таблицы, для которых выполняются условия\_соединения. Для недостающих данных вместо строк левой\_таблицы вставляются NULL-значения.



*-- RIGHT JOIN вернет 8 строк*

**SELECT** emp.**ID**,emp.**Name**,emp.DepartmentID,dep.**ID**,dep.**Name**

**FROM** Employees emp

**RIGHT** **JOIN** Departments dep **ON** emp.DepartmentID=dep.**ID**

**FULL OUTER JOIN**

Оператор полного внешнего соединения **FULL OUTER JOIN** соединяет две таблицы. Порядок таблиц для оператора неважен, поскольку оператор является симметричным.

FULL JOIN работает как объединение двух множеств.

Full outer join производит выбор всех строк из таблиц А и В, причем со всеми возможными вариантами. Если с какой-либо стороны не будет записи, то недостающая запись будет содержать null значение.

**SELECT** \* **FROM** TableA

**FULL OUTER JOIN** TableB

**ON** TableA.name = TableB.name

Возвращаются все строки левой\_таблицы и правой\_таблицы. Если для строк левой\_таблицы и правой\_таблицы выполняются условия\_соединения, то они объединяются в одну строку. Для строк, для которых не выполняются условия\_соединения, NULL-значения вставляются на место левой\_таблицы, либо на место правой\_таблицы, в зависимости от того данных какой таблицы в строке не имеется.



*-- FULL JOIN вернет 10 строк*

**SELECT** emp.**ID**,emp.**Name**,emp.DepartmentID,dep.**ID**,dep.**Name**

**FROM** Employees emp

**FULL** **JOIN** Departments dep **ON** emp.DepartmentID=dep.**ID**

**CROSS JOIN**

Оператор перекрёстного соединения, или декартова произведения CROSS JOIN соединяет две таблицы. Порядок таблиц для оператора неважен, поскольку оператор является симметричным.

Размер результирующего набора декартова произведения вычисляется как произведение количества строк в первой таблице на количество строк во второй таблице.

При использовании CROSS JOIN генерируется точно тот же результат, что и при вызове двух таблиц (разделенных запятой) без всякого JOIN вообще. Это значит, что мы получим огромный набор результатов, где каждая запись из Table1 будет дублирована для каждой записи из Table2. Если в Table1 содержится N1 записей, а в Table2 – N2 записей, в результате будет N1 х N2 записей.

Это очень опасно для таблиц, содержащих большое количество данных. То есть получаются ВСЕ возможные комбинации, включая все Null-null строчки.

**SELECT** \* **FROM** TableA

**CROSS JOIN** TableB

*-- CROSS JOIN вернет 42 строк - (7 строк таблицы Employees) \* (6 строк таблицы Departments)*

**SELECT** emp.**ID**,emp.**Name**,emp.DepartmentID,dep.**ID**,dep.**Name**

**FROM** Employees emp

**CROSS** **JOIN** Departments dep

Совершенно честно, я не могу сейчас припомнить ни одной реальной ситуации, когда мне понадобилось бы сгенерировать декартово произведение двух таблиц.

Кроме всего прочего, стоит подумать и о производительности. Допустим, что вы случайно запустили на рабочем сервере запрос, содержащий CROSS JOIN для двух таблиц по миллиону записей в каждой. Это, несомненно, добавит вам головной боли. Возможно, у вашего сервера начнутся проблемы с производительностью, поскольку это запрос будет исполняться долго, и потреблять при этом существенное количество ресурсов сервера.

В многотабличных запросах, хоть и можно указать имя без псевдонима, в случае если имя не дублируется во второй таблице, но я бы рекомендовал всегда использовать псевдонимы в случае соединения, т.к. никто не гарантирует, что поле с таким же именем со временем не добавят во вторую таблицу, а тогда ваш запрос просто сломается, ругаясь на то что он не может понять к какой таблице относится данное поле.

Только используя псевдонимы, мы сможем осуществить соединения таблицы самой с собой (самосоединение). Используйте самосоединение, когда требуется создать результирующий набор, который соединяет записи в таблице с другими записями в той же таблице. Чтобы указать таблицу два раза в одном и том же запросе, необходимо задать псевдоним таблицы хотя бы для одного экземпляра имени таблицы. Этот псевдоним таблицы помогает обработчику запросов определить, какие данные должны быть представлены в столбцах (из правой или из левой версии таблицы).

Предположим встала задача, получить для каждого сотрудника, данные сотрудника, который был принят прямо до него (табельный номер отличается на единицу меньше). Допустим, что у нас табельные номера выдаются последовательно и без дырок, тогда мы можем это сделать примерно следующим образом:

**SELECT**

e1.**ID** EmpID1,

e1.**Name** EmpName1,

e2.**ID** EmpID2,

e2.**Name** EmpName2

**FROM** Employees e1

**LEFT** **JOIN** Employees e2 **ON** e1.**ID**=e2.**ID**+1 *-- получить данные предыдущего сотрудника*

Т.е. здесь одной таблице Employees, мы дали псевдоним «e1», а второй «e2».

## Разбираем каждый вид горизонтального соединения

Для этой цели рассмотрим 2 небольшие абстрактные таблицы, которые так и назовем LeftTable и RightTable:

**CREATE** **TABLE** LeftTable(

LCode int,

LDescr varchar(10)

)

**GO**

**CREATE** **TABLE** RightTable(

RCode int,

RDescr varchar(10)

)

**GO**

**INSERT** LeftTable(LCode,LDescr)**VALUES**

(1,'L-1'),

(2,'L-2'),

(3,'L-3'),

(5,'L-5')

**INSERT** RightTable(RCode,RDescr)**VALUES**

(2,'B-2'),

(3,'B-3'),

(4,'B-4')

Посмотрим, что в этих таблицах:

**SELECT** \* **FROM** LeftTable

|  |  |
| --- | --- |
| **LCode** | **LDescr** |
| 1 | L-1 |
| 2 | L-2 |
| 3 | L-3 |
| 5 | L-5 |

**SELECT** \* **FROM** RightTable

|  |  |
| --- | --- |
| **RCode** | **RDescr** |
| 2 | B-2 |
| 3 | B-3 |
| 4 | B-4 |

### JOIN

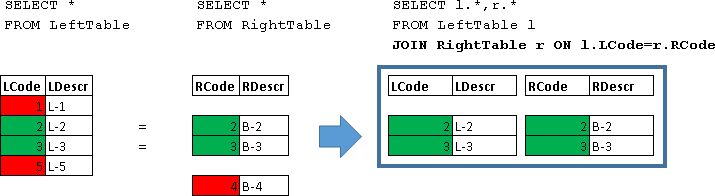
**SELECT** **l**.\*,r.\*

**FROM** LeftTable **l**

**JOIN** RightTable r **ON** **l**.LCode=r.RCode

|  |  |  |  |
| --- | --- | --- | --- |
| **LCode** | **LDescr** | **RCode** | **RDescr** |
| 2 | L-2 | 2 | B-2 |
| 3 | L-3 | 3 | B-3 |

Здесь были возвращены объединения строк для которых выполнилось условие (l.LCode=r.RCode)



### LEFT JOIN

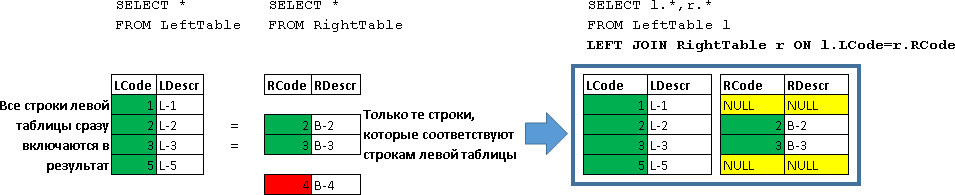
**SELECT** **l**.\*,r.\*

**FROM** LeftTable **l**

**LEFT** **JOIN** RightTable r **ON** **l**.LCode=r.RCode

|  |  |  |  |
| --- | --- | --- | --- |
| **LCode** | **LDescr** | **RCode** | **RDescr** |
| 1 | L-1 | NULL | NULL |
| 2 | L-2 | 2 | B-2 |
| 3 | L-3 | 3 | B-3 |
| 5 | L-5 | NULL | NULL |

Здесь были возвращены все строки LeftTable, которые были дополнены данными строк из RightTable, для которых выполнилось условие (l.LCode=r.RCode)



### RIGHT JOIN

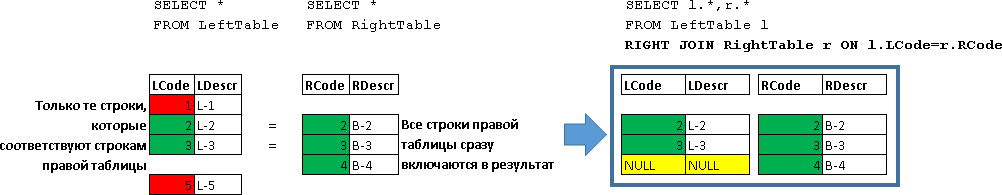
**SELECT** **l**.\*,r.\*

**FROM** LeftTable **l**

**RIGHT** **JOIN** RightTable r **ON** **l**.LCode=r.RCode

|  |  |  |  |
| --- | --- | --- | --- |
| **LCode** | **LDescr** | **RCode** | **RDescr** |
| 2 | L-2 | 2 | B-2 |
| 3 | L-3 | 3 | B-3 |
| NULL | NULL | 4 | B-4 |

Здесь были возвращены все строки RightTable, которые были дополнены данными строк из LeftTable, для которых выполнилось условие (l.LCode=r.RCode)



По сути если мы переставим LeftTable и RightTable местами, то аналогичный результат мы получим при помощи левого соединения:

**SELECT** **l**.\*,r.\*

**FROM** RightTable r

**LEFT** **JOIN** LeftTable **l** **ON** **l**.LCode=r.RCode

|  |  |  |  |
| --- | --- | --- | --- |
| **LCode** | **LDescr** | **RCode** | **RDescr** |
| 2 | L-2 | 2 | B-2 |
| 3 | L-3 | 3 | B-3 |
| NULL | NULL | 4 | B-4 |

### FULL JOIN – это по сути одновременный LEFT JOIN + RIGHT JOIN

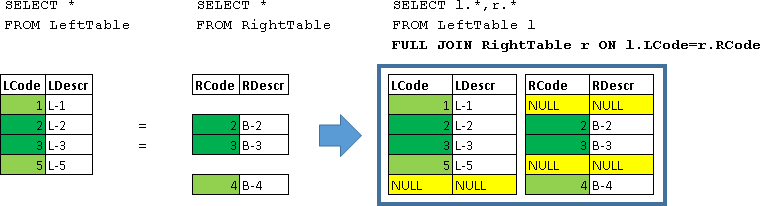
**SELECT** **l**.\*,r.\*

**FROM** LeftTable **l**

**FULL** **JOIN** RightTable r **ON** **l**.LCode=r.RCode

|  |  |  |  |
| --- | --- | --- | --- |
| **LCode** | **LDescr** | **RCode** | **RDescr** |
| 1 | L-1 | NULL | NULL |
| 2 | L-2 | 2 | B-2 |
| 3 | L-3 | 3 | B-3 |
| 5 | L-5 | NULL | NULL |
| NULL | NULL | 4 | B-4 |

Вернулись все строки из LeftTable и RightTable. Строки для которых выполнилось условие (l.LCode=r.RCode) были объединены в одну строку. Отсутствующие в строке данные с левой или правой стороны заполняются NULL-значениями.



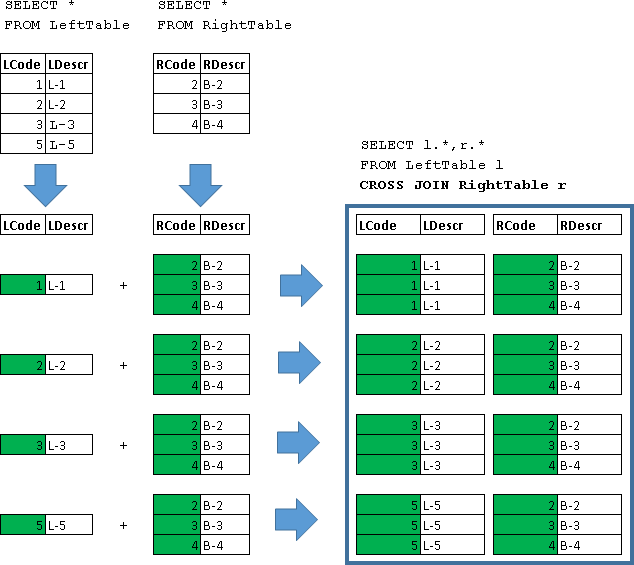
### CROSS JOIN

**SELECT** **l**.\*,r.\*

**FROM** LeftTable **l**

**CROSS** **JOIN** RightTable r

Каждая строка LeftTable соединяется с данными всех строк RightTable.



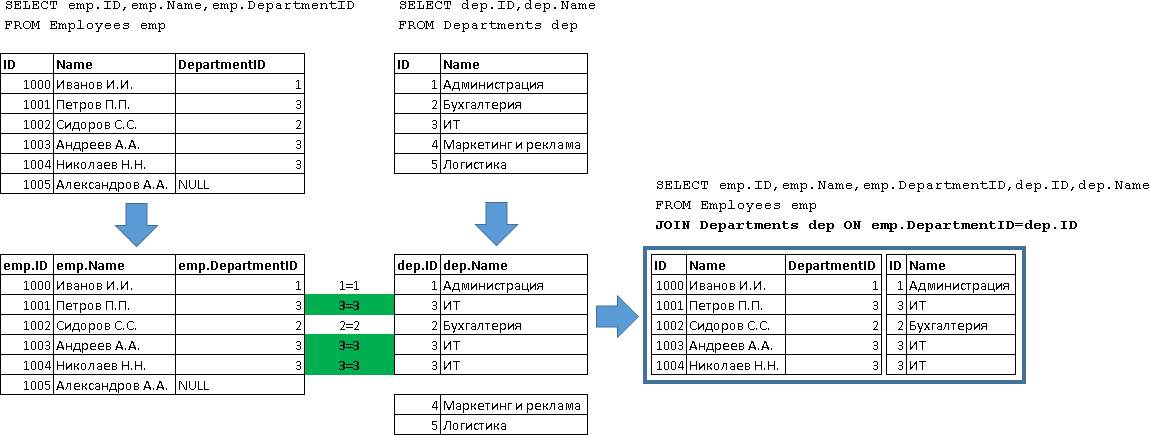
## Возвращаемся к таблицам Employees и Departments

Надеюсь вы поняли принцип работы горизонтальных соединений. Если это так, то возвратитесь на начало раздела «JOIN-соединения – операции горизонтального соединения данных» и попробуйте самостоятельно понять примеры с объединением таблиц Employees и Departments, а потом снова возвращайтесь сюда, обсудим это вместе.

Давайте попробуем вместе подвести резюме для каждого запроса:

|  |  |
| --- | --- |
| **Запрос** | **Резюме** |
| *-- JOIN вернет 5 строк*  **SELECT** emp.**ID**, emp.**Name**, emp.DepartmentID, dep.**ID**, dep.**Name**  **FROM** Employees emp  **JOIN** Departments dep  **ON** emp.DepartmentID=dep.**ID** | По сути данный запрос вернет только сотрудников, у которых указано значение DepartmentID.  Т.е. мы можем использовать данное соединение, в случае, когда нам нужны данные по сотрудникам числящихся за каким-нибудь отделом (без учета внештаткиков). |
| *-- LEFT JOIN вернет 6 строк*  **SELECT** emp.**ID**, emp.**Name**, emp.DepartmentID, dep.**ID**, dep.**Name**  **FROM** Employees emp  **LEFT** **JOIN** Departments dep  **ON** emp.DepartmentID=dep.**ID** | Вернет всех сотрудников. Для тех сотрудников у которых не указан DepartmentID, поля «dep.ID» и «dep.Name» будут содержать NULL.  Вспоминайте, что NULL значения в случае необходимости можно обработать, например, при помощи ISNULL(dep.Name,'вне штата'). Этот вид соединения можно использовать, когда нам важно получить данные по всем сотрудникам, например, чтобы получить список для начисления ЗП. |
| *-- RIGHT JOIN вернет 7 строк*  **SELECT** emp.**ID**, emp.**Name**, emp.DepartmentID, dep.**ID**, dep.**Name**  **FROM** Employees emp  **RIGHT** **JOIN** Departments dep  **ON** emp.DepartmentID=dep.**ID** | Здесь мы получили дырки слева, т.е. отдел есть, но сотрудников в этом отделе нет. Такое соединение можно использовать, например, когда нужно выяснить, какие отделы и кем у нас заняты, а какие еще не сформированы. Эту информацию можно использовать для поиска и приема новых работников из которых будет формироваться отдел. |
| *-- FULL JOIN вернет 8 строк*  **SELECT** emp.**ID**, emp.**Name**, emp.DepartmentID, dep.**ID**, dep.**Name**  **FROM** Employees emp  **FULL** **JOIN** Departments dep  **ON** emp.DepartmentID=dep.**ID** | Этот запрос важен, когда нам нужно получить все данные по сотрудникам и все данные по имеющимся отделам. Соответственно получаем дырки (NULL-значения) либо по сотрудникам, либо по отделам (внештатники).  Данный запрос, например, может использоваться в целях проверки, все ли сотрудники сидят в правильных отделах, т.к. может у некоторых сотрудников, которые числятся как внештатники, просто забыли указать отдел. |
| *-- CROSS JOIN вернет 30 строк - (6 строк таблицы Employees) \* (5 строк таблицы Departments)*  **SELECT** emp.**ID**, emp.**Name**, emp.DepartmentID, dep.**ID**, dep.**Name**  **FROM** Employees emp  **CROSS** **JOIN** Departments dep | В таком виде даже сложно придумать где это можно применить, поэтому пример с CROSS JOIN я покажу ниже. |

Обратите внимание, что в случае повторения значений DepartmentID в таблице Employees, произошло соединение каждой такой строки со строкой из таблицы Departments с таким же ID, то есть данные Departments объединились со всеми записями для которых выполнилось условие (emp.DepartmentID=dep.ID):



В нашем случае все получилось правильно, т.е. мы дополнили таблицу Employees, данными таблицы Departments. Я специально заострил на этом внимание, т.к. бывают случаи, когда такое поведение нам не нужно.

## Соединение трех и более таблиц

Пример с использованием нескольких последовательных операций соединения.

Если используется несколько операций соединения, то в таком случае они применяются последовательно сверху-вниз. Грубо говоря, после каждого соединения создается новый набор и следующее соединение уже происходит с этим расширенным набором. Рассмотрим простой пример:

**SELECT**

**e**.**ID**,

**e**.**Name** EmployeeName,

**p**.**Name** PositionName,

**d**.**Name** DepartmentName

**FROM** Employees **e**

**LEFT** **JOIN** Departments **d** **ON** **e**.DepartmentID=**d**.**ID**

**LEFT** **JOIN** Positions **p** **ON** **e**.PositionID=**p**.**ID**

Первым делом выбрались все записи таблицы Employees:

**SELECT**

**e**.\*

**FROM** Employees **e** *-- 1*

Дальше произошло соединение с таблицей Departments:

**SELECT**

**e**.\*, *-- к полям Employees*

**d**.\* *-- добавились соответствующие (e.DepartmentID=d.ID) поля Departments*

**FROM** Employees **e** *-- 1*

**LEFT** **JOIN** Departments **d** **ON** **e**.DepartmentID=**d**.**ID** *-- 2*

Дальше уже идет соединение этого набора с таблицей Positions:

**SELECT**

**e**.\*, *-- к полям Employees*

**d**.\*, *-- добавились соответствующие (e.DepartmentID=d.ID) поля Departments*

**p**.\* *-- добавились соответствующие (e.PositionID=p.ID) поля Positions*

**FROM** Employees **e** *-- 1*

**LEFT** **JOIN** Departments **d** **ON** **e**.DepartmentID=**d**.**ID** *-- 2*

**LEFT** **JOIN** Positions **p** **ON** **e**.PositionID=**p**.**ID** *-- 3*

Т.е. это выглядит примерно так:

И в последнюю очередь идет возврат тех данных, которые мы просим вывести:

**SELECT**

**e**.**ID**, *-- 1. идентификатор сотрудника*

**e**.**Name** EmployeeName, *-- 2. имя сотрудника*

**p**.**Name** PositionName, *-- 3. название должности*

**d**.**Name** DepartmentName *-- 4. название отдела*

**FROM** Employees **e**

**LEFT** **JOIN** Departments **d** **ON** **e**.DepartmentID=**d**.**ID**

**LEFT** **JOIN** Positions **p** **ON** **e**.PositionID=**p**.**ID**

Соответственно, ко всему этому полученному набору можно применить фильтр WHERE и сортировку ORDER BY:

**SELECT**

**e**.**ID**, *-- 1. идентификатор сотрудника*

**e**.**Name** EmployeeName, *-- 2. имя сотрудника*

**p**.**Name** PositionName, *-- 3. название должности*

**d**.**Name** DepartmentName *-- 4. название отдела*

**FROM** Employees **e**

**LEFT** **JOIN** Departments **d** **ON** **e**.DepartmentID=**d**.**ID**

**LEFT** **JOIN** Positions **p** **ON** **e**.PositionID=**p**.**ID**

**WHERE** **d**.**ID**=3 *-- используем поля из поле ID из Departments*

**AND** **p**.**ID**=3 *-- используем для фильтрации поле ID из Positions*

**ORDER** **BY** **e**.**Name** *-- используем для сортировки поле Name из Employees*

|  |  |  |  |
| --- | --- | --- | --- |
| **ID** | **EmployeeName** | **PositionName** | **DepartmentName** |
| 1004 | Николаев Н.Н. | Программист | ИТ |
| 1001 | Петров П.П. | Программист | ИТ |

То есть последний полученный набор – представляет собой такую же таблицу, над которой можно выполнять базовый запрос:

**SELECT** [**DISTINCT**] список\_столбцов или \*

**FROM** источник

**WHERE** фильтр

**ORDER** **BY** выражение\_сортировки

То есть если раньше в роли источника выступала только одна таблица, то теперь на это место мы просто подставляем наше выражение:

Employees e

LEFT JOIN Departments d ON e.DepartmentID=d.ID

LEFT JOIN Positions p ON e.PositionID=p.ID

В результате чего получаем тот же самый базовый запрос:

**SELECT**

**e**.**ID**,

**e**.**Name** EmployeeName,

**p**.**Name** PositionName,

**d**.**Name** DepartmentName

**FROM**

*/\* источник - начало \*/*

Employees **e**

**LEFT** **JOIN** Departments **d** **ON** **e**.DepartmentID=**d**.**ID**

**LEFT** **JOIN** Positions **p** **ON** **e**.PositionID=**p**.**ID**

*/\* источник - конец \*/*

**WHERE** **d**.**ID**=3

**AND** **p**.**ID**=3

**ORDER** **BY** **e**.**Name**

А теперь, применим группировку:

**SELECT**

**ISNULL**(dep.**Name**,'Прочие') DepName,

**COUNT**(**DISTINCT** emp.PositionID) PositionCount,

**COUNT**(\*) EmplCount,

**SUM**(emp.Salary) SalaryAmount,

**AVG**(emp.Salary) SalaryAvg *-- плюс выполняем пожелание директора*

**FROM**

*/\* источник - начало \*/*

Employees emp

**LEFT** **JOIN** Departments dep **ON** emp.DepartmentID=dep.**ID**

*/\* источник - конец \*/*

**GROUP** **BY** emp.DepartmentID,dep.**Name**

**ORDER** **BY** DepName

В запросе на месте любой таблицы может стоять подзапрос. В свою очередь подзапросы могут быть вложены в подзапросы. И все эти подзапросы тоже представляют из себя базовые конструкции. То есть базовая конструкция, это кирпичики, из которых строится любой запрос.

## Обещанный пример с CROSS JOIN

Давайте используем соединение CROSS JOIN, чтобы подсчитать сколько сотрудников, в каком отделе и на каких должностях числится. Для каждого отдела перечислим все существующие должности:

**SELECT**

**d**.**Name** DepartmentName,

**p**.**Name** PositionName,

**e**.EmplCount

**FROM** Departments **d**

**CROSS** **JOIN** Positions **p**

**LEFT** **JOIN**

(

*/\**

*здесь я использовал подзапрос для подсчета сотрудников*

*в разрезе групп (DepartmentID,PositionID)*

*\*/*

**SELECT** DepartmentID,PositionID,**COUNT**(\*) EmplCount

**FROM** Employees

**GROUP** **BY** DepartmentID,PositionID

) **e**

**ON** **e**.DepartmentID=**d**.**ID** **AND** **e**.PositionID=**p**.**ID**

**ORDER** **BY** DepartmentName,PositionName

В данном случае сначала выполнилось соединение при помощи CROSS JOIN, а затем к полученному набору сделалось соединение с данными из подзапроса при помощи LEFT JOIN. Вместо таблицы в LEFT JOIN мы использовали подзапрос.

Подзапрос заключается в скобки и ему присваивается псевдоним, в данном случае это «e». То есть в данном случае объединение происходит не с таблицей, а с результатом следующего запроса:

**SELECT** DepartmentID,PositionID,**COUNT**(\*) EmplCount

**FROM** Employees

**GROUP** **BY** DepartmentID,PositionID

|  |  |  |
| --- | --- | --- |
| **DepartmentID** | **PositionID** | **EmplCount** |
| NULL | NULL | 1 |
| 2 | 1 | 1 |
| 1 | 2 | 1 |
| 3 | 3 | 2 |
| 3 | 4 | 1 |

Вместе с псевдонимом «e» мы можем использовать имена DepartmentID, PositionID и EmplCount. По сути дальше подзапрос ведет себя так же, как если на его месте стояла таблица. Соответственно, как и у таблицы, все имена колонок, которые возвращает подзапрос, должны быть заданы явно и не должны повторяться.

## Связь при помощи WHERE-условия

Для примера перепишем следующий запрос с JOIN-соединением:

**SELECT** emp.**ID**,emp.**Name**,emp.DepartmentID,dep.**ID**,dep.**Name**

**FROM** Employees emp

**JOIN** Departments dep **ON** emp.DepartmentID=dep.**ID** *-- условие соединения таблиц*

**WHERE** emp.DepartmentID=3 *-- условие фильтрации данных*

Через WHERE-условие он примет следующую форму:

**SELECT** emp.**ID**,emp.**Name**,emp.DepartmentID,dep.**ID**,dep.**Name**

**FROM**

Employees emp,

Departments dep

**WHERE** emp.DepartmentID=dep.**ID** *-- условие соединения таблиц*

**AND** emp.DepartmentID=3 *-- условие фильтрации данных*

Здесь плохо то, что происходит смешивание условий соединения таблиц (emp.DepartmentID=dep.ID) с условием фильтрации (emp.DepartmentID=3).

Теперь посмотрим, как сделать CROSS JOIN:

**SELECT** emp.**ID**,emp.**Name**,emp.DepartmentID,dep.**ID**,dep.**Name**

**FROM** Employees emp

**CROSS** **JOIN** Departments dep *-- декартово соединение (соединение без условия)*

**WHERE** emp.DepartmentID=3 *-- условие фильтрации данных*

Через WHERE-условие он примет следующую форму:

**SELECT** emp.**ID**,emp.**Name**,emp.DepartmentID,dep.**ID**,dep.**Name**

**FROM**

Employees emp,

Departments dep

**WHERE** emp.DepartmentID=3 *-- условие фильтрации данных*

Т.е. в этом случае мы просто не указали условие соединения таблиц Employees и Departments. Чем плох этот запрос? Представьте, что кто-то другой смотрит на ваш запрос и думает «кажется тот, кто писал запрос забыл здесь дописать условие (emp.DepartmentID=dep.ID)» и с радостью, что обнаружил косяк, дописывает это условие. В результате чего задуманное вами может сломаться, т.к. вы подразумевали CROSS JOIN. Так что, если вы делаете декартово соединение, то лучше явно укажите, что это именно оно, используя конструкцию CROSS JOIN.

Для оптимизатора запроса может быть и без разницы как вы реализуете соединение (при помощи WHERE или JOIN), он их может выполнить абсолютно одинаково. Но из соображения понимаемости кода, я бы рекомендовал в современных СУБД стараться никогда не делать соединение таблиц при помощи WHERE-условия. Использовать WHERE-условия для соединения, в том случае, если в СУБД реализованы конструкции JOIN, я бы назвал сейчас моветоном. WHERE-условия служат для фильтрации набора, и не нужно перемешивать условия служащие для соединения, с условиями отвечающими за фильтрацию. Но если вы пришли к выводу, что без реализации соединения через WHERE не обойтись, то конечно приоритет за решеной задачей и «к черту все устои».

## UNION-объединения – операции вертикального объединения результатов запросов

**Оператор UNION** позволяет объединять результаты нескольких инструкций SELECT в один результирующий набор, в который входят все строки, принадлежащие всем запросам в объединении. Соответственно, результатом объединения двух таблиц является новая таблица, содержащая все строки, входящие в одну из исходных таблиц или в обе эти таблицы.

У результирующих наборов, объединяемых с помощью оператора UNION, должна быть одинаковая структура. Они должны содержать одинаковое количество столбцов, а в соответствующих столбцах результирующих наборов должны быть совместимые типы данных.

Данный оператор был описан в первом стандарте SQL — SQL/89.

**Общая форма оператора UNION** выглядит таким образом:

select\_1

UNION [ALL]

select\_2

{[UNION [ALL]

select\_3]}

...

Параметры select\_1, select\_2, ... представляют собой инструкции SELECT, которые создают объединение. Если используется **параметр ALL**, отображаются все строки, включая дубликаты. В операторе UNION параметр ALL имеет то же самое значение, что и в списке выбора SELECT, но с одним отличием: для списка выбора SELECT этот параметр применяется по умолчанию, а для оператора UNION его нужно указывать явно.

Объединять с помощью инструкции UNION можно только совместимые таблицы. Под **совместимыми** таблицами имеется в виду, что оба списка столбцов выборки должны содержать одинаковое число столбцов, а соответствующие столбцы должны иметь совместимые типы данных. Соответствующие столбцы результирующего набора в отдельных инструкциях, комбинируемые оператором UNION, должны указываться в одном и том же порядке, так как оператор UNION сравнивает столбцы один к одному в порядке, указанном в отдельных запросах.

Результат объединения можно упорядочить, только используя предложение ORDER BY в последней инструкции SELECT, как это показано в примере 6.46. Предложения GROUP BY и HAVING можно применять с отдельными инструкциями SELECT, но не в самом объединении.

В инструкции Transact-SQL может быть любое количество операторов UNION.

Давайте первым делом вспомним как мы делали первую версию отчета для директора:

**SELECT**

'Администрация' Info,

**COUNT**(**DISTINCT** PositionID) PositionCount,

**COUNT**(\*) EmplCount,

**SUM**(Salary) SalaryAmount

**FROM** Employees

**WHERE** DepartmentID=1 *-- данные по Администрации*

**SELECT**

'Бухгалтерия' Info,

**COUNT**(**DISTINCT** PositionID) PositionCount,

**COUNT**(\*) EmplCount,

**SUM**(Salary) SalaryAmount

**FROM** Employees

**WHERE** DepartmentID=2 *-- данные по Бухгалтерии*

**SELECT**

'ИТ' Info,

**COUNT**(**DISTINCT** PositionID) PositionCount,

**COUNT**(\*) EmplCount,

**SUM**(Salary) SalaryAmount

**FROM** Employees

**WHERE** DepartmentID=3 *-- данные по ИТ отделу*

**SELECT**

'Прочие' Info,

**COUNT**(**DISTINCT** PositionID) PositionCount,

**COUNT**(\*) EmplCount,

**SUM**(Salary) SalaryAmount

**FROM** Employees

**WHERE** DepartmentID **IS** NULL *-- и еще не забываем данные по внештатникам*

Так вот, если бы мы не знали, что существует операция группировки, но знали бы, что существует операция объединения результатов запроса при помощи UNION ALL, то мы могли бы склеить все эти запросы следующим образом:

**SELECT**

'Администрация' Info,

**COUNT**(**DISTINCT** PositionID) PositionCount,

**COUNT**(\*) EmplCount,

**SUM**(Salary) SalaryAmount

**FROM** Employees

**WHERE** DepartmentID=1 *-- данные по Администрации*

**UNION** ALL

**SELECT**

'Бухгалтерия' Info,

**COUNT**(**DISTINCT** PositionID) PositionCount,

**COUNT**(\*) EmplCount,

**SUM**(Salary) SalaryAmount

**FROM** Employees

**WHERE** DepartmentID=2 *-- данные по Бухгалтерии*

**UNION** ALL

**SELECT**

'ИТ' Info,

**COUNT**(**DISTINCT** PositionID) PositionCount,

**COUNT**(\*) EmplCount,

**SUM**(Salary) SalaryAmount

**FROM** Employees

**WHERE** DepartmentID=3 *-- данные по ИТ отделу*

**UNION** ALL

**SELECT**

'Прочие' Info,

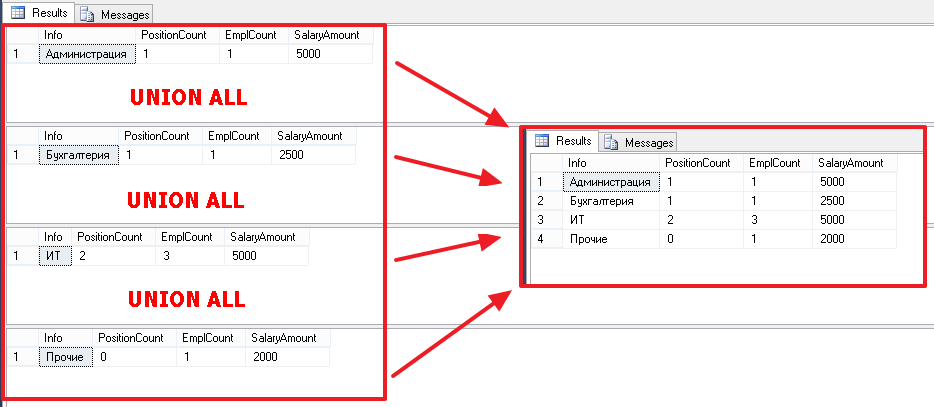
**COUNT**(**DISTINCT** PositionID) PositionCount,

**COUNT**(\*) EmplCount,

**SUM**(Salary) SalaryAmount

**FROM** Employees

**WHERE** DepartmentID **IS** NULL *-- и еще не забываем данные по внештатникам*



Т.е. UNION ALL позволяет склеить результаты, полученные разными запросами в один общий результат.

Соответственно количество колонок в каждом запросе должно быть одинаковым, а также должны быть совместимыми и типы этих колонок, т.е. строка под строкой, число под числом, дата под датой и т.п.

**В MS SQL реализованы следующие виды вертикального объединения:**

|  |  |
| --- | --- |
| **Операция** | **Описание** |
| UNION ALL | В результат включаются все строки из обоих наборов. (A+B) |
| UNION | В результат включаются только уникальные строки двух наборов. DISTINCT(A+B) |
| EXCEPT | В результат попадают уникальные строки верхнего набора, которые отсутствуют в нижнем наборе. Разница 2-х множеств. DISTINCT(A-B) |
| INTERSECT | В результат включаются только уникальные строки, присутствующие в обоих наборах. Пересечение 2-х множеств. DISTINCT(A&B) |

Все это проще понять на наглядном примере.

Создадим 2 таблицы и наполним их данными:

**CREATE** **TABLE** TopTable(

T1 int,

T2 varchar(10)

)

**GO**

**CREATE** **TABLE** BottomTable(

B1 int,

B2 varchar(10)

)

**GO**

**INSERT** TopTable(T1,T2)**VALUES**

(1,'Text 1'),

(1,'Text 1'),

(2,'Text 2'),

(3,'Text 3'),

(4,'Text 4'),

(5,'Text 5')

**INSERT** BottomTable(B1,B2)**VALUES**

(2,'Text 2'),

(3,'Text 3'),

(6,'Text 6'),

(6,'Text 6')

Посмотрим на содержимое:

**SELECT** \*

**FROM** TopTable

|  |  |
| --- | --- |
| **T1** | **T2** |
| 1 | Text 1 |
| 1 | Text 1 |
| 2 | Text 2 |
| 3 | Text 3 |
| 4 | Text 4 |
| 5 | Text 5 |

**SELECT** \*

**FROM** BottomTable

|  |  |
| --- | --- |
| **B1** | **B2** |
| 2 | Text 2 |
| 3 | Text 3 |
| 6 | Text 6 |
| 6 | Text 6 |

### UNION ALL

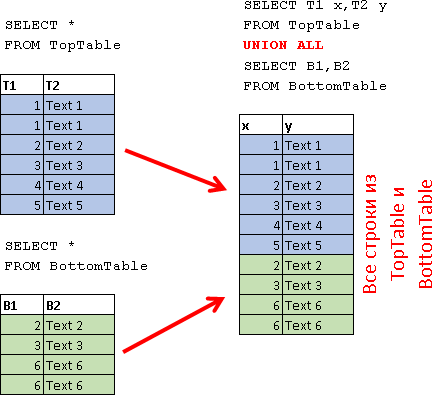
**SELECT** T1 x,T2 y

**FROM** TopTable

**UNION** ALL

**SELECT** B1,B2

**FROM** BottomTable



### UNION

**SELECT** T1 x,T2 y

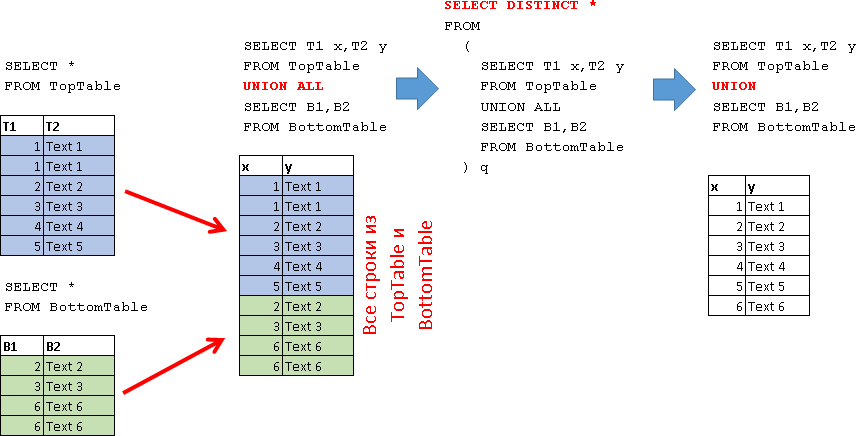
**FROM** TopTable

**UNION**

**SELECT** B1,B2

**FROM** BottomTable

По сути UNION можно представить, как UNION ALL, к которому применена операция DISTINCT:



### EXCEPT

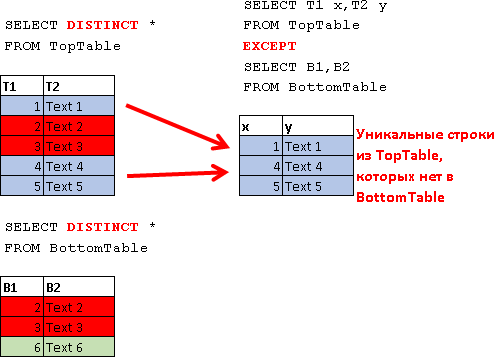
**SELECT** T1 x,T2 y

**FROM** TopTable

**EXCEPT**

**SELECT** B1,B2

**FROM** BottomTable



### INTERSECT

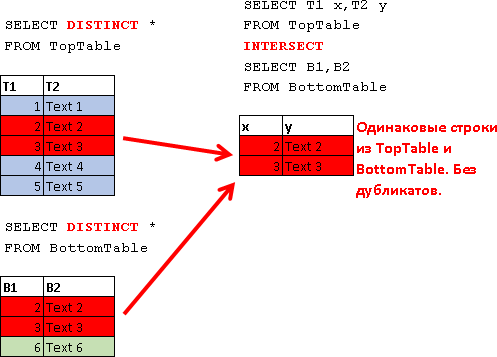
**SELECT** T1 x,T2 y

**FROM** TopTable

**INTERSECT**

**SELECT** B1,B2

**FROM** BottomTable



## Завершаем разговор о UNION-соединениях

Вот в принципе и все, что касается вертикальных объединений, это намного проще, чем JOIN-соединения.

Чаще всего в моей в практике находит применение UNION ALL, но и другие виды вертикальных объединений находят свое применение.

Следует помнить, что эти три оператора над множествами имеют **разный приоритет выполнения:** оператор INTERSECT имеет наивысший приоритет, за ним следует оператор EXCEPT, а оператор UNION имеет самый низкий приоритет. Невнимательность к приоритету выполнения при использовании нескольких разных операторов для работы с наборами может повлечь неожиданные результаты.

Создадим еще одну таблицу и рассмотрим это на примере:

**CREATE** **TABLE** NextTable(

N1 int,

N2 varchar(10)

)

**GO**

**INSERT** NextTable(N1,N2)**VALUES**

(1,'Text 1'),

(4,'Text 4'),

(6,'Text 6')

Например, если мы напишем просто:

**SELECT** T1 x,T2 y

**FROM** TopTable

**EXCEPT**

**SELECT** B1,B2

**FROM** BottomTable

**INTERSECT**

**SELECT** N1,N2

**FROM** NextTable

То мы получим:

|  |  |
| --- | --- |
| **x** | **y** |
| 1 | Text 1 |
| 2 | Text 2 |
| 3 | Text 3 |
| 4 | Text 4 |
| 5 | Text 5 |

Т.е. получается сначала выполнился INTERSECT, а после EXCEPT. Хотя логически будто должно было быть наоборот, т.е. идти сверху-вниз.

Я редко использую эти операции объединений, а тем более в таком виде, поэтому, чтобы не думать не гадать, в какой очередности он выполняет объединения, можно просто при помощи скобок явно указать последовательность объединений, давайте скажем, что сначала нужно сделать EXCEPT, а потом INTERSECT:

(

**SELECT** T1 x,T2 y

**FROM** TopTable

**EXCEPT**

**SELECT** B1,B2

**FROM** BottomTable

)

**INTERSECT**

**SELECT** N1,N2

**FROM** NextTable

|  |  |
| --- | --- |
| **x** | **y** |
| 1 | Text 1 |
| 4 | Text 4 |

Можно использовать подзапрос:

**SELECT** x,y

**FROM**

(

**SELECT** T1 x,T2 y

**FROM** TopTable

**EXCEPT**

**SELECT** B1,B2

**FROM** BottomTable

) q

**INTERSECT**

**SELECT** N1,N2

**FROM** NextTable

При использовании ORDER BY сортировка применяется к окончательному набору:

**SELECT** T1 x,T2 y

**FROM** TopTable

**UNION** ALL

**SELECT** B1,B2

**FROM** BottomTable

**UNION** ALL

**SELECT** B1,B2

**FROM** BottomTable

**ORDER** **BY** x **DESC**

Для задания сортировки здесь удобней использовать псевдоним колонки, заданный в первом запросе.