

Тема 15. Объединение таблиц.

Цель занятия:

**Освоить мощные механизмы работы
с распределенными данными.**

Учебные вопросы:

- 1. Объединение таблиц при помощи JOIN.**
- 2. Практика JOIN: Решение типовых задач.**

1. Операции объединения таблиц.

Объединение таблиц (JOIN) — это одна из наиболее важных операций в SQL. Она позволяет комбинировать данные из нескольких таблиц, связанных между собой по определенным условиям.

Это особенно полезно, когда нужно получить комплексную информацию, которая разбросана по разным таблицам базы данных.

Объединение таблиц

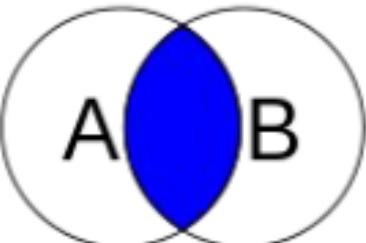
JOIN – оператор, предназначенный для объединения таблиц по определенному столбцу или связке столбцов (как правило, по первичному ключу).

JOIN записывается после **FROM** и до **WHERE**. Через оператор **ON** необходимо явно указывать столбцы, по которым происходит объединение. В общем виде структура запросы с **JOIN** выглядит так:

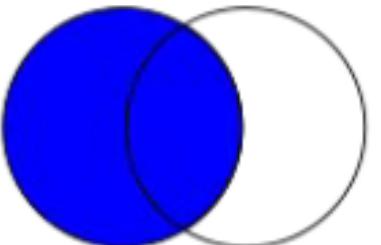
```
SELECT tables_columns FROM table_1  
JOIN table_2 ON table_1.column = table_2.column;
```

Типы JOIN

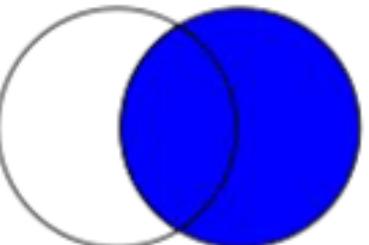
```
SELECT <fields>
FROM TableA A
INNER JOIN TableB B
ON A.key = B.key
```



```
SELECT <fields>
FROM TableA A
LEFT JOIN TableB B
ON A.key = B.key
```

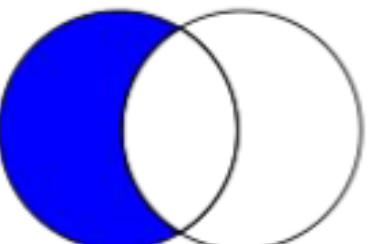


```
SELECT <fields>
FROM TableA A
RIGHT JOIN TableB B
ON A.key = B.key
```

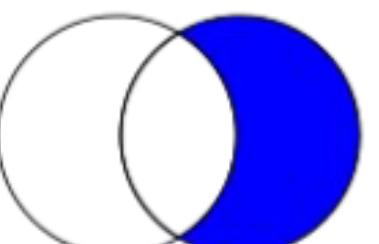


SQL JOINS

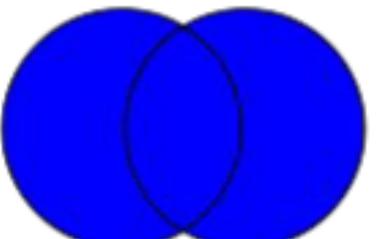
```
SELECT <fields>
FROM TableA A
LEFT JOIN TableB B
ON A.key = B.key
WHERE B.key IS NULL
```



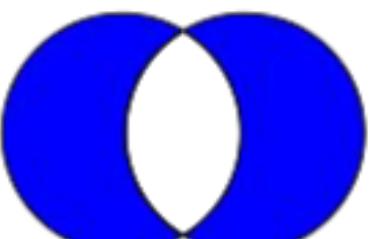
```
SELECT <fields>
FROM TableA A
RIGHT JOIN TableB B
ON A.key = B.key
WHERE A.key IS NULL
```



```
SELECT <fields>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.key = B.key
```



```
SELECT <fields>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.key = B.key
WHERE A.key IS NULL
OR B.key IS NULL
```



Основные виды JOIN

[INNER] JOIN – в выборку попадут только те данные, по которым выполняются условия соединения;

LEFT [OUTER] JOIN – в выборку попадут все данные из таблицы А и только те данные из таблицы В, по которым выполнится условие соединения. Недостающие данные вместо строк таблицы В будут представлены **NULL**.

RIGHT [OUTER] JOIN – в выборку попадут все данные из таблицы В и только те данные из таблицы А, по которым выполнится условие соединения. Недостающие данные вместо строк таблицы А будут представлены **NULL**.

FULL [OUTER] JOIN – в выборку попадут все строки таблицы А и таблицы В. Если для строк таблицы А и таблицы В выполняются условия соединения, то они объединяются в одну строку. Если данных в какой-то таблице не имеется, то на соответствующие места вставляются **NULL**.

CROSS JOIN – объединение каждой строки таблицы А с каждой строкой таблицы В.

[INNER] JOIN

[INNER] JOIN – в выборку попадут только те данные, по которым выполняются условия соединения.

Левая таблица (к ней присоединяем)

employees		
id	name	office_id
1	John	1
2	James	2
3	Kate	3
4	Mary	999

* Код 999 - для удаленных

Правая таблица

offices	
id	office_name
1	Кабинет 42
2	Фиолетовый кабинет
3	Кабинет 126
4	Кабинет 22

Получаем только тех сотрудников, которые сидят в кабинетах и только те кабинеты, в которых сидят сотрудники.

employees.id	name	offices.id	office_name
1	John	1	Кабинет 42
2	James	2	Фиолетовый кабинет
3	Kate	3	Кабинет 126

```
CREATE TABLE Customers (
    CustomerID INT PRIMARY KEY,
    FirstName VARCHAR(50),
    LastName VARCHAR(50)
);
```

```
CREATE TABLE Orders (
    OrderID INT PRIMARY KEY,
    OrderDate DATE,
    CustomerID INT,
    FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)
);
```

-- Пример INNER JOIN

```
SELECT Customers.FirstName, Customers.LastName, Orders.OrderID, Orders.OrderDate
FROM Customers
INNER JOIN Orders
ON Customers.CustomerID = Orders.CustomerID;
```

Пример: Объединение таблиц клиентов и заказов.
Есть две таблицы: *Customers* (Клиенты) и *Orders* (Заказы).
Мы хотим получить информацию о клиентах и их заказах.

В этом запросе:

Мы выбираем имена клиентов и номера их заказов.

Объединяем таблицы *Customers* и *Orders* по столбцу *CustomerID*.

В результате мы получаем только те клиенты, которые сделали заказы, и информацию о каждом заказе.

```
CREATE TABLE Employees (
    EmployeeID INT PRIMARY KEY,
    FirstName VARCHAR(50),
    LastName VARCHAR(50)
);
```

```
CREATE TABLE Projects (
    ProjectID INT PRIMARY KEY,
    ProjectName VARCHAR(100)
);
```

```
CREATE TABLE EmployeeProjects (
    EmployeeID INT,
    ProjectID INT,
    FOREIGN KEY (EmployeeID) REFERENCES Employees(EmployeeID),
    FOREIGN KEY (ProjectID) REFERENCES Projects(ProjectID)
);
```

Объединение таблиц сотрудников и проектов
Предположим, у нас есть таблицы **Employees** (Сотрудники), **Projects** (Проекты) и **EmployeeProjects** (Проекты сотрудников), которые содержат информацию о том, какие сотрудники работают над какими проектами.

```
-- Пример INNER JOIN

SELECT Employees.FirstName, Employees.LastName, Projects.ProjectName
FROM EmployeeProjects
INNER JOIN Employees
ON EmployeeProjects.EmployeeID = Employees.EmployeeID
INNER JOIN Projects
ON EmployeeProjects.ProjectID = Projects.ProjectID;
```

В этом запросе:

Мы выбираем имена сотрудников и названия проектов, над которыми они работают.

Объединяем три таблицы: EmployeeProjects, Employees, и Projects.

Используем два условия объединения для связывания данных между EmployeeProjects и двумя другими таблицами.

LEFT JOIN

LEFT JOIN – в выборку попадут все данные из левой таблицы и только те данные из правой, по которым выполняется условие соединения.

Левая таблица (к ней присоединяем)

employees		
id	name	office_id
1	John	1
2	James	2
3	Kate	3
4	Mary	999

* Код 999 - для удаленных

Правая таблица

offices	
id	office_name
1	Кабинет 42
2	Фиолетовый кабинет
3	Кабинет 126
4	Кабинет 22

Получаем всех сотрудников и только те кабинеты, в которых кто-то сидит.

employees.id	name	offices.id	office_name
1	John	1	Кабинет 42
2	James	2	Фиолетовый кабинет
3	Kate	3	Кабинет 126
4	Mary	NULL	NULL

```
CREATE TABLE Customers (
    CustomerID INT PRIMARY KEY,
    FirstName VARCHAR(50),
    LastName VARCHAR(50)
);
```

```
CREATE TABLE Orders (
    OrderID INT PRIMARY KEY,
    OrderDate DATE,
    CustomerID INT,
    FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)
);
```

```
-- Пример LEFT JOIN
```

```
SELECT Customers.FirstName, Customers.LastName, Orders.OrderID, Orders.OrderDate
FROM Customers
LEFT JOIN Orders
ON Customers.CustomerID = Orders.CustomerID;
```

Пример использования LEFT JOIN. Объединение таблиц клиентов и заказов.

Предположим, у нас есть таблицы Customers (Клиенты) и Orders (Заказы). Мы хотим получить список всех клиентов, включая тех, у которых нет заказов.

В этом запросе мы выбираем все строки из таблицы Customers и соответствующие строки из таблицы Orders. Если у клиента нет заказов, в полях OrderID и OrderDate будут значения NULL.

```
-- Пример LEFT JOIN

SELECT Employees.FirstName, Employees.LastName, Projects.ProjectName
FROM Employees
LEFT JOIN EmployeeProjects
ON Employees.EmployeeID = EmployeeProjects.EmployeeID
LEFT JOIN Projects
ON EmployeeProjects.ProjectID = Projects.ProjectID;
```

В этом запросе мы выбираем всех сотрудников, включая тех, кто не участвует ни в одном проекте.

В полях ProjectName будут NULL для сотрудников, у которых нет назначенных проектов.

RIGHT JOIN

RIGHT JOIN – в выборку попадут все данные из правой таблицы и только те данные из левой, по которым выполняется условие соединения.

Левая таблица (к ней присоединяем)

employees		
id	name	office_id
1	John	1
2	James	2
3	Kate	3
4	Mary	999

* Код 999 - для удаленщиков

Правая таблица

offices	
id	office_name
1	Кабинет 42
2	Фиолетовый кабинет
3	Кабинет 126
4	Кабинет 22

Получаем все кабинеты и только тех сотрудников, которые сидят в них (не удаленщиков).

employees.id	name	offices.id	office_name
1	John	1	Кабинет 42
2	James	2	Фиолетовый кабинет
3	Kate	3	Кабинет 126
NULL	NULL	4	Кабинет 22

LEFT JOIN с фильтрацией по полю

LEFT JOIN с фильтрацией по полю – в выборку попадут только те данные из левой таблицы, по которым не выполняется условия соединения.

Левая таблица (к ней присоединяем)

employees		
id	name	office_id
1	John	1
2	James	2
3	Kate	3
4	Mary	999

* Код 999 - для удаленных

Правая таблица

offices	
id	office_name
1	Кабинет 42
2	Фиолетовый кабинет
3	Кабинет 126
4	Кабинет 22

Получаем только удаленных.

employees.id	name	offices.id	office_name
4	Mary	NULL	NULL

```
CREATE TABLE Students (
    StudentID INT PRIMARY KEY,
    FirstName VARCHAR(50),
    LastName VARCHAR(50)
);
```

```
CREATE TABLE Courses (
    CourseID INT PRIMARY KEY,
    CourseName VARCHAR(100)
);
```

```
CREATE TABLE StudentCourses (
    StudentID INT,
    CourseID INT,
    FOREIGN KEY (StudentID) REFERENCES Students(StudentID),
    FOREIGN KEY (CourseID) REFERENCES Courses(CourseID)
);
```

Пример использования LEFT JOIN с фильтрацией.
Фильтрация студентов, не записанных на определённый курс. Рассмотрим таблицы Students (Студенты), Courses (Курсы) и StudentCourses (Курсы студентов). Мы хотим получить список всех студентов, которые не записаны на курс с конкретным названием.

```
-- Пример LEFT JOIN с фильтрацией

SELECT Students.FirstName, Students.LastName
FROM Students
LEFT JOIN StudentCourses
ON Students.StudentID = StudentCourses.StudentID
LEFT JOIN Courses
ON StudentCourses.CourseID = Courses.CourseID
WHERE Courses.CourseName <> 'Advanced Mathematics' OR Courses.CourseName IS NULL;
```

В этом запросе мы объединяем таблицы Students, StudentCourses, и Courses.

Фильтруем результат, чтобы получить студентов, которые не записаны на курс "Advanced Mathematics", или студентов, которые не записаны ни на один курс (где CourseName равен NULL).

RIGHT JOIN с фильтрацией по полю

RIGHT JOIN с фильтрацией по полю – в выборку попадут только те данные из правой таблицы, по которым не выполняется условия соединения.

Левая таблица (к ней присоединяем)

employees		
id	name	office_id
1	John	1
2	James	2
3	Kate	3
4	Mary	999

* Код 999 - для удаляемников

Правая таблица

offices	
id	office_name
1	Кабинет 42
2	Фиолетовый кабинет
3	Кабинет 126
4	Кабинет 22

Получаем только пустые
кабинеты.

employees.id	name	offices.id	office_name
NULL	NULL	4	Кабинет 22

FULL OUTER JOIN

FULL OUTER JOIN – выборку попадут все строки исходных таблиц. Если по данным не выполняется условие соединения, то на соответствующие места вставляются **NULL**.

Левая таблица (к ней присоединяем)

employees		
id	name	office_id
1	John	1
2	James	2
3	Kate	3
4	Mary	999

* Код 999 - для удаленных

Правая таблица

offices	
id	office_name
1	Кабинет 42
2	Фиолетовый кабинет
3	Кабинет 126
4	Кабинет 22

Получаем абсолютно все кабинеты и абсолютно всех сотрудников.

employees.id	name	offices.id	office_name
1	John	1	Кабинет 42
2	James	2	Фиолетовый кабинет
3	Kate	3	Кабинет 126
4	Mary	NULL	NULL
NULL	NULL	4	Кабинет 22

```
CREATE TABLE Customers (
    CustomerID INT PRIMARY KEY,
    FirstName VARCHAR(50),
    LastName VARCHAR(50)
);

CREATE TABLE Orders (
    OrderID INT PRIMARY KEY,
    OrderDate DATE,
    CustomerID INT,
    FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)
);

-- Пример FULL OUTER JOIN
SELECT Customers.FirstName, Customers.LastName, Orders.OrderID, Orders.OrderDate
FROM Customers
FULL OUTER JOIN Orders
ON Customers.CustomerID = Orders.CustomerID;
```

Примеры использования **FULL OUTER JOIN**. Объединение таблиц клиентов и заказов.

Рассмотрим таблицы *Customers* (Клиенты) и *Orders* (Заказы). Мы хотим получить список всех клиентов и всех заказов, включая тех клиентов, у которых нет заказов, и те заказы, которые не привязаны к клиентам.

В этом запросе мы объединяем таблицы *Customers* и *Orders* по полю *CustomerID*.

В результате будут возвращены все строки из обеих таблиц. Если клиент сделал заказ, в строке будет информация и о клиенте, и о заказе. Если клиент не сделал заказ, поля из таблицы *Orders* будут заполнены значениями **NULL**. Если заказ не связан с клиентом, поля из таблицы *Customers* будут заполнены значениями **NULL**.

UNION и UNION ALL. Различие между UNION и JOIN.

UNION используется для объединения результатов двух или более SELECT-запросов в один результирующий набор данных. Он объединяет строки, при этом удаляет дублирующиеся строки.

Синтаксис:

```
SELECT столбцы1
```

```
FROM таблица1
```

```
UNION
```

```
SELECT столбцы2
```

```
FROM таблица2;
```

Особенности:

- Количество столбцов в обоих запросах должно совпадать.
- Типы данных соответствующих столбцов должны быть совместимы.
- Удаляет дубликаты из результирующего набора данных.

`UNION ALL` работает аналогично `UNION`, но включает все строки, включая дублирующиеся.

Синтаксис:

```
SELECT столбцы1  
FROM таблица1  
UNION ALL  
SELECT столбцы2  
FROM таблица2;
```

Особенности:

- Сохраняет все дубликаты, если они присутствуют.
- Быстрее, чем `UNION`, так как не требует дополнительной обработки для удаления дубликатов

```
CREATE TABLE Employees (
    EmployeeID INT PRIMARY KEY,
    FirstName VARCHAR(50),
    LastName VARCHAR(50)
);
```

В таблице Employees хранятся данные о текущих сотрудниках.
В таблице FormerEmployees хранятся данные о бывших сотрудниках.

```
CREATE TABLE FormerEmployees (
    EmployeeID INT PRIMARY KEY,
    FirstName VARCHAR(50),
    LastName VARCHAR(50)
);
```

-- UNION: объединение без дубликатов

```
SELECT FirstName, LastName FROM Employees  
UNION  
SELECT FirstName, LastName FROM FormerEmployees;
```

-- UNION ALL: объединение с дубликатами

```
SELECT FirstName, LastName FROM Employees  
UNION ALL  
SELECT FirstName, LastName FROM FormerEmployees;
```

UNION объединяет эти результаты и возвращает уникальные записи, т.е. если одно и то же имя и фамилия есть и в текущих, и в бывших сотрудниках, в результатах они появятся только один раз.

UNION ALL объединяет эти результаты и возвращает все записи, включая дубликаты.

Различие между UNION и JOIN

UNION

Что делает:

- Объединяет результаты двух или более SELECT-запросов, возвращая все строки, соответствующие каждому запросу, либо с удалением (UNION), либо без удаления (UNION ALL) дубликатов.
- Используется, когда требуется объединить результаты нескольких запросов по горизонтали (добавить строки в результирующий набор).

Когда использовать:

- Когда нужно объединить два разных набора данных в один.
- Когда требуется получить полный набор данных из разных таблиц или запросов, которые не обязательно связаны между собой напрямую.

JOIN

Что делает:

- Объединяет строки из двух или более таблиц на основе связанного условия (например, совпадение значения в столбцах).
- Объединяет данные по вертикали, добавляя столбцы к результирующему набору.
- Типы JOIN (INNER JOIN, LEFT JOIN, RIGHT JOIN, FULL OUTER JOIN) определяют, какие строки будут включены в результат на основе условий совпадения.

Когда использовать:

- Когда нужно получить связанные данные из нескольких таблиц на основе общего столбца.
- Когда требуется объединить таблицы по ключевым столбцам для анализа данных, находящихся в отношениях "один-к-одному", "один-ко-многим", или "многие-ко-многим".

Сравнение

Особенность	UNION	JOIN
Назначение	Объединение результатов запросов	Объединение строк на основе условий
Дублирование данных	Удаляет (UNION) или сохраняет (UNION ALL) дубликаты	Данные не дублируются, объединяются строки
Тип объединения	Горизонтальное (добавление строк)	Вертикальное (добавление столбцов)
Связь между таблицами	Не требуется	Требуется, используются ключи и условия
Скорость выполнения	UNION ALL быстрее, UNION медленнее	Зависит от типа JOIN и условий

Заключение

UNION и JOIN — это разные инструменты для объединения данных в SQL.

UNION объединяет результаты запросов по горизонтали, добавляя строки, в то время как JOIN объединяет строки из нескольких таблиц на основе условий совпадения, добавляя столбцы.

Выбор между ними зависит от задачи, которую нужно решить.

2. Практика JOIN: Решение типовых задач.

Задание 1: Выборки из нескольких таблиц

Создайте две таблицы:

employees:

- id (INT, PRIMARY KEY, AUTO_INCREMENT)
- name (VARCHAR)
- department_id (INT)

departments:

- id (INT, PRIMARY KEY, AUTO_INCREMENT)
- department_name (VARCHAR)

Заполните таблицы данными (минимум 5 сотрудников и 3 отдела).

Напишите запросы:

- Выведите список всех сотрудников с указанием их отдела.
- Найдите количество сотрудников в каждом отделе.
- Выведите отделы, в которых работает больше 2 сотрудников.

Задание 2: Комбинированные запросы

Создайте таблицу orders:

- id (INT, PRIMARY KEY, AUTO_INCREMENT)
- customer_id (INT)
- order_date (DATE)
- total_amount (DECIMAL)

Создайте таблицу customers:

- id (INT, PRIMARY KEY, AUTO_INCREMENT)
- name (VARCHAR)
- city (VARCHAR)

Заполните таблицы данными (минимум 5 заказов и 3 клиента).

Напишите запросы:

- Выведите список всех заказов с указанием имени клиента и города.
- Найдите общую сумму заказов для каждого клиента.
- Найдите клиентов, у которых общая сумма заказов превышает 1000.

Задание 3: Практика с JOIN

Используя таблицы `orders` и `customers`, напишите запросы:

- Выведите список заказов, сделанных клиентами из определенного города (например, "Москва").
- Найдите среднюю сумму заказа для каждого клиента.
- Выведите клиентов, которые не сделали ни одного заказа (используйте `LEFT JOIN` и `IS NULL`).

Задание 4: Сложная группировка

Создайте таблицу products:

- id (INT, PRIMARY KEY, AUTO_INCREMENT)
- product_name (VARCHAR)
- category (VARCHAR)

Создайте таблицу sales:

- id (INT, PRIMARY KEY, AUTO_INCREMENT)
- product_id (INT)
- quantity (INT)
- sale_date (DATE)

Заполните таблицы данными (минимум 5 товаров и 10 продаж).

Напишите запросы:

- Сгруппируйте данные по категориям товаров и найдите общее количество проданных товаров в каждой категории.
- Найдите категорию с наибольшим количеством продаж.
- Выведите товары, которые не были проданы ни разу (используйте LEFT JOIN и IS NULL).

Список литературы:

1. <https://metanit.com/sharp/windowsforms/3.1.php>

Материалы лекций:

<https://github.com/ShViktor72/Education2025>