

Лекция №4

SQL – объединение таблиц union, соединение - join, подзапросы

Сегодня мы продолжим изучение языка SQL, изучим применение операторов join, union, продолжая изучение языка манипулирования данными - DML.

Сегодня мы узнаем:

1. Операторы union и union all
2. Соединение таблиц — Join (left, right, inner, full, cross)
3. Использование подзапросов (join, in, exists, select)
4. Использование select into, создание таблиц из результатов запроса
5. порядок выполнения запроса — дополнение схемы

Термины лекции

DML – *Data Manipulation Language* (язык манипулирования данными).

Группировка — операция, которая создает из записей таблицы независимые группы записей, по которым проводится анализ.

Агрегатные функции (агрегации) — это функции, которые вычисляются от группы значений и объединяют их в одно результирующее.

Оглавление

Тизер	1
Термины лекции	1
Операторы union и union all	2
Соединение таблиц — Join (left, right, inner, full, cross)	6
Использование подзапросов (join, in, exists, select)	11
Использование select into, создание таблиц из результатов запроса	12
Агрегатные функции — count, sum, avg, обработка Null	9
HAVING	11
Приоритет операций	16

Операторы union и union all

На прошлых лекциях мы получали данные для однотипных выборок. MySQL оператор UNION используется для объединения наборов результатов из 2 или более SELECT предложений. Он удаляет повторяющиеся строки между различными предложениями SELECT. Каждое предложение SELECT в операторе UNION должно иметь одинаковое количество полей в наборах результатов с одинаковыми типами данных.

Синтаксис

```
1 SELECT ...  
2 UNION [ALL | DISTINCT] SELECT ...  
3 [UNION [ALL | DISTINCT] SELECT ... ]
```

Для объединения выборок из двух таблиц нужно использовать оператор UNION. При этом важно помнить, что:

- каждая выборка SELECT в рамках UNION должна иметь одинаковое количество столбцов
 - столбцы должны иметь одинаковые типы данных
 - столбцы в каждом SELECT должны быть в одинаковом порядке
- при формировании итоговой выборки дубликаты удаляются

Пример **простой** **на** **UNION:**

```
37 • SELECT 1, 2 UNION SELECT 'a', 'b';
```

38

39

40

Result Grid	Filter Rows:	Export:	Wrap Cell Content:
1	2		
1	2		
a	b		

Пусть имеется две таблицы: “сотрудники” и “клиенты”:

	EmpID	Name	Address	Phone	Salary
▶	1	Nikhil	Delhi	9878906543	8000
	2	Divya	Ranchi	8990076543	5000
	3	Ravi	Bareilly	7789945765	7000
	4	Anna	Noida	9789945760	4000
	5	Surbhi	Jaipur	7800541123	5500
★	NULL	NULL	NULL	NULL	NULL

	Id	FirstName	LastName	AccountSum
▶	1	Tom	Smith	2000
	2	Sam	Brown	3000
	3	Mark	Adams	2500
	4	Paul	Ins	4200
	5	John	Smith	2800
	6	Tim	Cook	2800
	7	Tom	Smith	2000
	8	Sam	Brown	3000
	9	Mark	Adams	2500
	10	Paul	Ins	4200
	11	John	Smith	2800
	12	Tim	Cook	2800
	13	Tom	Smith	2000
	14	Sam	Brown	3000
	15	Mark	Adams	2500
	16	Paul	Ins	4200
	17	John	Smith	2800
	18	Tim	Cook	2800
★	NULL	NULL	NULL	NULL

Здесь мы можем заметить, что обе таблицы, несмотря на наличие различных данных, могут характеризоваться двумя общими атрибутами - именем (FirstName) и фамилией (LastName). Выберем сразу всех клиентов банка и его сотрудников из обеих таблиц:

```

1 SELECT FirstName, LastName
2 FROM Customers
3 UNION SELECT FirstName, LastName FROM Employees;

```

Здесь из первой таблицы выбираются два значения - имя и фамилия клиента. Из второй таблицы Employees также выбираются два значения - имя и фамилия сотрудников. То есть, при объединении количество выбираемых столбцов и их тип совпадают для обеих выборок.

2	
3	• <code>SELECT FirstName, LastName</code>
4	<code>FROM Customers</code>
5	<code>UNION SELECT FirstName, LastName FROM Employees;</code>

<		
Result Grid		Filter Rows: <input type="text"/>
Export: Wrap Cell C		
	FirstName	LastName
	Tom	Smith
	Sam	Brown
	Mark	Adams
	Paul	Ins
	John	Smith
	Tim	Cook
	Homer	Simpson
	Nick	Svensson

При этом названия столбцов объединенной выборки будут совпадать с названия столбцов первой выборки. И если мы захотим при этом еще произвести сортировку, то в выражениях ORDER BY необходимо ориентироваться именно на названия столбцов первой выборки:

30	• <code>SELECT FirstName, LastName</code>
31	<code>FROM Customers</code>
32	<code>UNION SELECT FirstName, LastName</code>
33	<code>FROM Employees</code>
34	<code>ORDER BY FirstName DESC;</code>

Result Grid		Filter Rows: <input type="text"/>
Export:		
	FirstName	LastName
▶	Tom	Smith
	Tim	Cook
	Sam	Brown
	Paul	Ins
	Nick	Svensson
	Mark	Adams
	John	Smith
	Homer	Simpson

При объединении в примерах выше всех дублирующиеся строки удалялись. Если же необходимо при объединении сохранить все, в том числе повторяющиеся строки, то для этого необходимо использовать оператор ALL:

SQL File 3* x

Limit to 1000 rows

```

1 • SELECT FirstName, LastName
2 FROM Customers
3 UNION ALL SELECT FirstName, LastName
4 FROM Employees
5 ORDER BY FirstName;

```

Result Grid | Filter Rows: | Export: | Wrap Cell Content: |

	FirstName	LastName
▶	Homer	Simpson
	Homer	Simpson
	John	Smith
	Mark	Adams
	Mark	Adams
	Mark	Adams
	Nick	Svensson
	Nick	Svensson
	Paul	Ins
	Sam	Brown
	Tim	Cook
	Tom	Smith
	Tom	Smith
	Tom	Smith

В чем разница между UNION и UNION ALL?

1. Оператор UNION удаляет повторяющиеся строки.
2. UNION ALL не удаляет повторяющиеся строки

Но делать выборки можно и из одной таблицы. Объединять выборки можно и из одной и той же таблицы. Например, в зависимости от суммы на счете клиента нам надо начислять ему определенные проценты на вклад:

```

1 SELECT First Name, Last Name, AccountSum + AccountSum * 0.1 AS TotalSum
2 FROM Customers WHERE AccountSum < 3000
3 UNION SELECT FirstName, LastName, AccountSum + AccountSum * 0.3 AS TotalSum
4 FROM Customers WHERE AccountSum ≥ 3000;

```

В данном случае если сумма меньше 3000, то начисляются проценты в размере 10% от суммы на счете. Если на счете больше 3000, то проценты увеличиваются до 30%.

Выборки данных из таблицы можно получать так же с помощью оператора JOIN. JOIN, в переводе на великий и могучий, означает "объединять", то есть собирать из нескольких кусочков единое целое. В базе данных MySQL такими "кусочками" служат столбцы таблиц, которые можно объединять при выборке. Объединения позволяют извлекать данные из нескольких таблиц без создания временных таблиц и за один запрос.

Соединение таблиц — Join (left, right, inner, full, cross)

Поддерживаемые типы объединений в MySQL

INNER JOIN: возвращает записи с совпадающими значениями в обеих таблицах.

LEFT JOIN: возвращает все записи из левой таблицы и соответствующие записи из правой таблицы.

RIGHT JOIN: возвращает все записи из правой таблицы и соответствующие записи из левой таблицы.

CROSS JOIN: возвращает все записи из обеих таблиц.

Inner Join

```
1 SELECT столбцы
2 FROM таблица1
3     [INNER] JOIN таблица2
4     ON условие1
5     [[INNER] JOIN таблица 3
6     ON условие2]
```

После оператора JOIN идет название второй таблицы, из которой надо добавить данные в выборку. Перед JOIN может использоваться необязательное ключевое слово INNER. Его наличие или отсутствие ни на что не влияет. Затем после ключевого слова ON указывается условие соединения. Это условие устанавливает, как две таблицы будут сравниваться. В большинстве случаев для соединения применяется первичный ключ главной таблицы и внешний ключ зависимой таблицы.

Используя JOIN, выберем все заказы и добавим к ним информацию о товарах:

```
1 SELECT Orders.CreatedAt, Orders.ProductCount, Products.ProductName
2 FROM Orders
3 JOIN Products ON Products.Id = Orders.ProductId;
```

Поскольку таблицы могут содержать столбцы с одинаковыми названиями, то при указании столбцов для выборки указывается их полное имя вместе с именем таблицы, например, "Orders.ProductCount". Используя псевдонимы для таблиц, можно сократить код:

```
1 SELECT O.CreatedAt, O.ProductCount, P.ProductName
2 FROM Orders AS O
3 JOIN Products AS P
4 ON P.Id = O.ProductId;
```

Outer Join

В отличие от Inner Join внешнее соединение возвращает все строки одной или двух таблиц, которые участвуют в соединении. Outer Join имеет следующий формальный синтаксис:

```
1 SELECT столбцы
2 FROM таблица1
3     {LEFT|RIGHT} [OUTER] JOIN таблица2 ON условие1
4     [{LEFT|RIGHT} [OUTER] JOIN таблица3 ON условие2] ...
```

Перед оператором JOIN указывается одно из ключевых слов LEFT или RIGHT, которые определяют тип соединения:

LEFT: выборка будет содержать все строки из первой или левой таблицы

RIGHT: выборка будет содержать все строки из второй или правой таблицы

Также перед оператором JOIN может указываться ключевое слово OUTER, но его применение необязательно. Далее после JOIN указывается присоединяемая таблица, а затем идет условие соединения. Таблица Orders является первой или левой таблицей, а таблица Customers - правой таблицей. Поэтому, так как здесь используется выборка по левой таблице, то вначале будут выбираться все строки из Orders, а затем к ним по условию Orders.CustomerId = Customers.Id будут добавляться связанные строки из Customers.

LEFT JOIN

```
1 SELECT FirstName, CreatedAt, ProductCount, Price, ProductId
2 FROM Orders LEFT JOIN Customers
3 ON Orders.CustomerId = Customers.Id
```

По вышеприведенному результату может показаться, что левостороннее соединение аналогично INNER Join, но это не так. Inner Join объединяет строки из двух таблиц при соответствии

условию. Если одна из таблиц содержит строки, которые не соответствуют этому условию, то данные строки не включаются в выходную выборку. Left Join выбирает все строки первой таблицы и затем присоединяет к ним строки правой таблицы.

```
1 #INNER JOIN
2 SELECT FirstName, CreatedAt, ProductCount, Price
3 FROM Customers JOIN Orders
4 ON Orders.CustomerId = Customers.Id;
5 #LEFT JOIN
6 SELECT FirstName, CreatedAt, ProductCount, Price
7 FROM Customers LEFT JOIN Orders
8 ON Orders.CustomerId = Customers.Id;
```

В случае с LEFT JOIN MySQL выбирает сначала всех покупателей из таблицы Customers, затем сопоставляет их с заказами из таблицы Orders через условие Orders.CustomerId = Customers.Id. Однако не у всех покупателей есть заказы. В этом случае покупателю для соответствующих столбцов устанавливаются значения NULL. Изменим в примере выше тип соединения для OUTER JOIN с левостороннего на правостороннее:

RIGHT JOIN

```
1 SELECT FirstName, CreatedAt, ProductCount, Price
2 FROM Customers RIGHT JOIN Orders
3 ON Orders.CustomerId = Customers.Id;
```

Теперь будут выбираться все строки из Orders (из правой таблицы), а к ним уже будет присоединяться связанные по условию строки из таблицы Customers.

FULL JOIN

FULL JOIN – это своего рода соединение и LEFT, и RIGHT, которое применили в одном запросе. К сожалению, в MySQL данный тип соединения не поддерживается, возможно, потому, что разработчики считают его немного избыточным, хотя во всех других популярных СУБД, например, в Microsoft SQL Server или в PostgreSQL, соединение FULL JOIN реализовано. С другой стороны, на практике действительно соединение FULL JOIN требуется достаточно редко.

Как было уже отмечено ранее, мы можем реализовать FULL JOIN, объединив два запроса при помощи UNION, в первом мы будем использовать LEFT JOIN, а во втором RIGHT JOIN.

По факту мы берем те же самые запросы, которые мы использовали чуть ранее, и объединяем их с помощью UNION.

Как Вы, наверное, знаете, запросы с UNION выполняются

достаточно медленно, за счет сортировки и удаления дублирующих строк.

```
1
2 • SELECT p.product_name, c.category_name
3     FROM products p
4     LEFT JOIN categories c ON p.category = c.category_id
5
6     UNION
7
8     SELECT p.product_name, c.category_name
9     FROM products p
10    RIGHT JOIN categories c ON p.category = c.category_id;
11
```

product_name	category_name
Системный блок	Комплектующие компьютера
Монитор	Комплектующие компьютера
Холодильник	Бытовая техника
Телевизор	Бытовая техника
Операционная система	NULL
NULL	Мобильные устройства

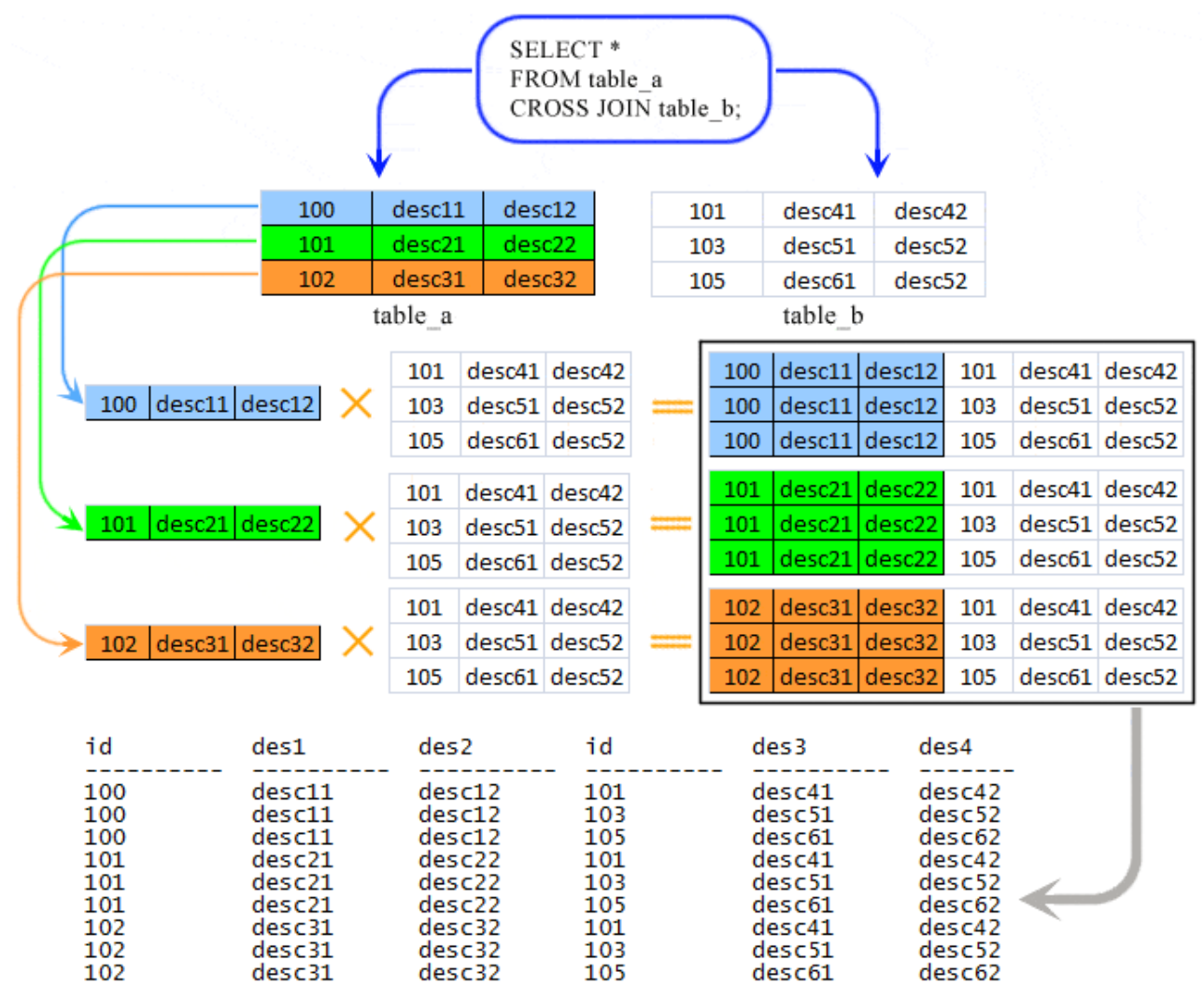
Чтобы немного увеличить производительность, мы можем использовать оператор UNION ALL. Однако он выведет все данные двух запросов, что приведет к наличию дублированных строк, но это мы можем легко исправить, если во втором запросе с RIGHT JOIN мы поставим условие, которое уберет повторяющиеся строки, и в итоге мы получим тот же самый результат.

```
1
2 • SELECT p.product_name, c.category_name
3     FROM products p
4     LEFT JOIN categories c ON p.category = c.category_id
5
6     UNION ALL
7
8     SELECT p.product_name, c.category_name
9     FROM products p
10    RIGHT JOIN categories c ON p.category = c.category_id
11    WHERE p.category IS NULL;
12
```

product_name	category_name
Системный блок	Комплектующие компьютера
Монитор	Комплектующие компьютера
Холодильник	Бытовая техника
Телевизор	Бытовая техника
Операционная система	NULL
NULL	Мобильные устройства

CROSS JOIN

В MySQL CROSS JOIN генерирует результирующий набор, который является произведением строк двух связанных таблиц, В этом соединении результирующий набор появился путем умножения каждой строки первой таблицы на все строки второй таблицы, если с CROSS JOIN не было введено никаких условий. Этот вид результата называется декартовым произведением. В MySQL CROSS JOIN ведет себя как JOIN и INNER JOIN без каких-либо условий. В стандартном SQL различие между INNER JOIN и предложением CROSS JOIN - ON можно использовать с INNER JOIN, с другой стороны, предложение ON нельзя использовать с CROSS JOIN.



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 нет.
CROSS JOIN	-	Объединение каждой строки левой_таблицы со всеми строками правой_таблицы . Этот вид соединения иногда называют декартовым произведением.

Использование подзапросов (join, in, exists, select)

Оператор IN

Нередко подзапросы применяются вместе с оператором IN, который выбирает из набора значений. И подзапрос как раз может предоставить требуемый набор значений. Например, выберем все товары из таблицы Products, на которые есть заказы в таблице Orders:

```
1 SELECT * FROM Products
2 WHERE Id IN (SELECT ProductId FROM Orders)
```

То есть подзапрос в данном случае выбирает все

идентификаторы товаров из Orders, затем по этим идентификаторам извлекаются товары из Products. Добавив оператор NOT, мы можем выбрать те товары, на которые нет заказов в таблице Orders:

```
1 SELECT * FROM Products
2 WHERE Id NOT IN (SELECT ProductId FROM Orders)
```

Оператор EXISTS

Оператор EXISTS проверяет, возвращает ли подзапрос какое-либо значение. Как правило, этот оператор используется для индикации того, что как минимум одна строка в таблице удовлетворяет некоторому условию. Поскольку возвращения набора строк не происходит, то подзапросы с подобным оператором выполняются довольно быстро.

```
1 WHERE [NOT] EXISTS (подзапрос)
```

Например, найдем все товары из таблицы Products, на которые есть заказы в таблице Orders:

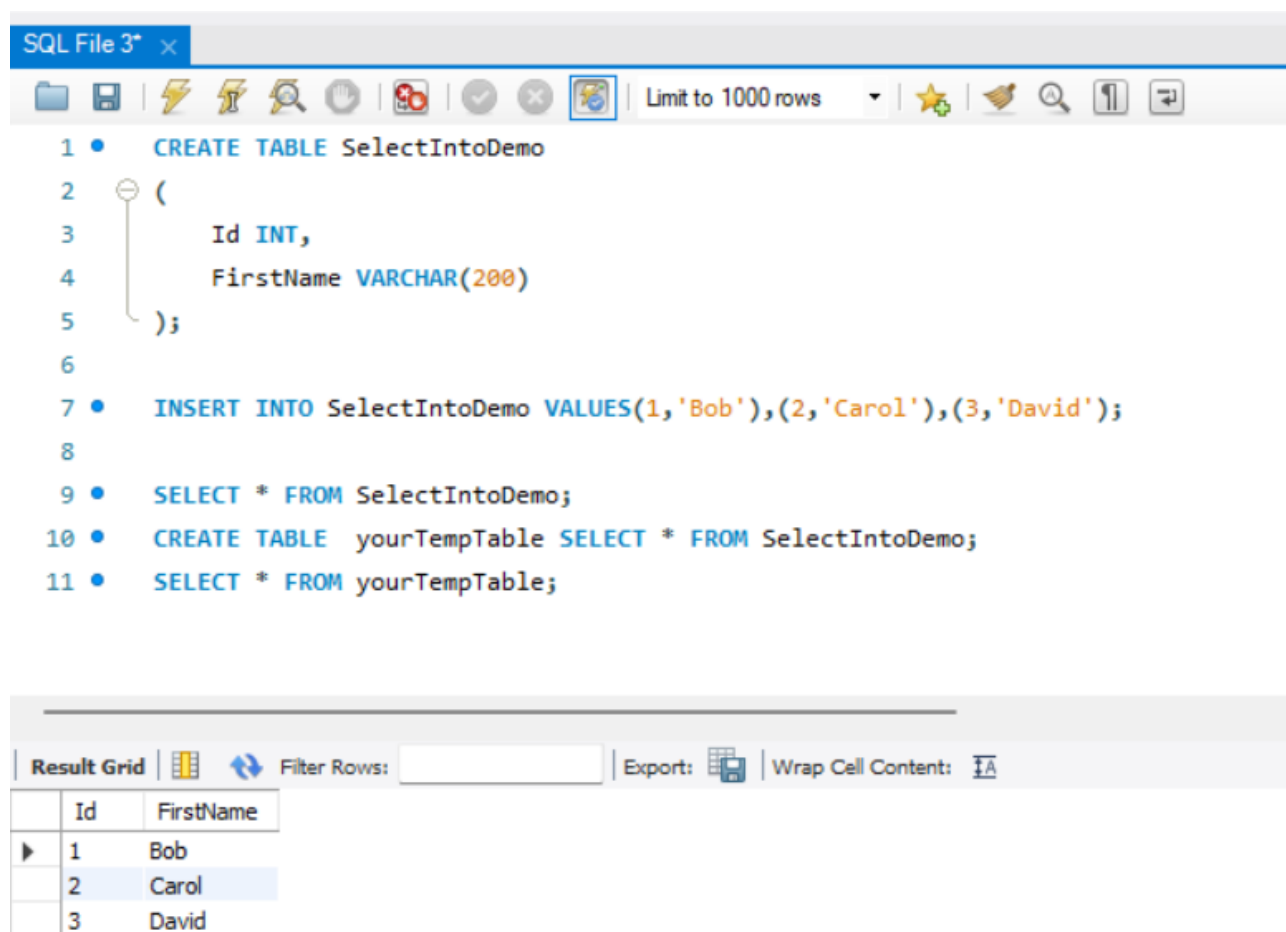
```
SELECT * FROM Products
```

```
WHERE EXISTS
```

```
(SELECT * FROM Orders WHERE Orders.ProductId = Products.Id)
```

Следующий пункт на повестке дня - `SELECT INTO`, который копирует данные из одной таблицы в новую.

Использование select into, создание таблиц из результатов запроса



The screenshot shows a SQL editor window titled "SQL File 3*" with a toolbar and a "Limit to 1000 rows" dropdown. The SQL code is as follows:

```
1 • CREATE TABLE SelectIntoDemo
2 • (
3 •     Id INT,
4 •     FirstName VARCHAR(200)
5 • );
6
7 • INSERT INTO SelectIntoDemo VALUES(1, 'Bob'), (2, 'Carol'), (3, 'David');
8
9 • SELECT * FROM SelectIntoDemo;
10 • CREATE TABLE yourTempTable SELECT * FROM SelectIntoDemo;
11 • SELECT * FROM yourTempTable;
```

Below the code is a "Result Grid" showing the results of the SELECT query. It has columns "Id" and "FirstName" and contains three rows:

	Id	FirstName
▶	1	Bob
	2	Carol
	3	David

Создадим табличку SelectIntoDemo, заполнив тестовыми значениями. Выведем табличку на экран с помощью "*". С помощью CREATE TABLE SELECT мы создаем одну табличку из другой. Структура запроса:

```
1 SELECT [DISTINCT | ALL] поля_таблиц
2 FROM список_таблиц
3 [WHERE условия_на_ограничения_строк]
4 [GROUP BY условия_группировки]
5 [HAVING условия_на_ограничения_строк_после_группировки]
6 [ORDER BY порядок_сортировки [ASC | DESC]]
7 [LIMIT ограничение_количества_записей]
```

Выполняется запрос так:

From - таблица

Where - убираем лишние значения - нужно что бы сократить данные

Group by - Потом группируем

Потом идет Having - - можно убрать лишнее после группировки

Потом уже выбираем нужный столбец - SELECT

А после уже сортируем с помощью Order By

Итоги

Сегодня мы изучили способы соединения таблиц, операторы union и union all, поработали с подзапросами и клонировали табличку.

На следующей лекции мы изучим:

1. Групповые функции без сортировки
2. Использование сортировки в оконной функции
3. Функции ранжирования, получения значений из соседних строк
4. Порядок выполнения запроса — полная схема
5. Представления (view) – использование, типы