QSL

Оператор SELECT DISTINCT

используется для возврата только **distinct** - отдельных (разных) значений. Столбец внутри таблицы часто содержит много повторяющихся значений; а иногда необходимо перечислить только разные (отдельные) значения.

```
SELECT DISTINCT столбец1, столбец2, ...
FROM имя_таблицы;
```

```
SELECT DISTINCT Country FROM Customers;
```

WHERE

Можно использовать ключевое слово WHERE в SELECT для указания условий в запросе:

```
SELECT <col_name1>, <col_name2>, …
  FROM <table_name>
  WHERE <condition>;
```

В запросе можно задавать следующие условия:

- сравнение текста;
- сравнение численных значений;
- логические операции AND (и), OR (или) и NOT (отрицание).

```
SELECT * FROM Customers
WHERE Country='Germany' AND City='Berlin';
```

```
SELECT * FROM Customers
WHERE City='Berlin' OR City='München';
```

также можете комбинировать операторы **AND, OR и NOT**

```
SELECT * FROM Customers
WHERE Country='Germany' AND (City='Berlin' OR City='München');
```

Ключевое слово ORDER BY

используется для сортировки набора результатов в порядке возрастания или убывания.**Ключевое слово ORDER BY** по умолчанию сортирует записи в порядке возрастания. Чтобы отсортировать записи в порядке убывания, используйте **ключевое слово DESC**.

```
SELECT * FROM Customers
ORDER BY Country;
```

Пример 2 ORDER BY Несколько столбцов

Следующий SQL оператор выбирает всех клиентов из таблицы "Customers", отсортированных по возрастанию в столбце "Country" и по убыванию в столбце "CustomerName":

```
SELECT * FROM Customers
ORDER BY Country ASC, CustomerName DESC;
```

SQL Оператор INSERT INTO

используется для вставки новых записей в таблицу.

```
INSERT INTO имя_таблицы (столбец1, столбец2, столбец3, ...)
VALUES (значение1, значение2, значение3, ...);
```

Следующий SQL оператор вставит новую запись, но вставит данные только в столбцы "CustomerName", "City" и "Country" (CustomerID будет обновлён автоматически):

```
INSERT INTO Customers (CustomerName, ContactName, Address, City, PostalCode,
Country)
VALUES ('Cardinal', 'Tom B. Erichsen', 'Skagen
21', 'Stavanger', '4006', 'Norway');
```

Поле со значением **NULL** - это поле без значения. Если поле в таблице является необязательным, можно вставить новую запись или обновить запись без добавления значения в это поле. Затем поле будет сохранено со значением **NULL.**

```
SELECT column_names
FROM table_name
WHERE column_name IS NULL;
```

```
SELECT column_names
FROM table_name
WHERE column_name IS NOT NULL;
```

Оператор **UPDATE** используется для изменения существующих записей в таблице.

```
UPDATE Customers
SET ContactName = 'Alfred Schmidt', City= 'Frankfurt'
WHERE CustomerID = 1;
```

ОБНОВЛЕНИЕ нескольких записей

Предложение **WHERE** определяет, сколько записей будет обновлено.

```
UPDATE Customers
SET ContactName='Juan'
WHERE Country='Mexico';
```

SQL **DELETE** Example

```
DELETE FROM Customers WHERE CustomerName='Alfreds Futterkiste';
```

Example

```
DELETE FROM Customers;
```

Инструкция **SELECT TOP** используется для указания количества возвращаемых записей.

Инструкция **SELECT TOP** полезно для больших таблиц с тысячами записей. Возврат большого количества записей может повлиять на производительность.

Не все базы данных поддерживают SELECT TOP. MySQL поддерживает предложение LIMIT для выбора ограниченного числа записей, в то время как Oracle использует ROWNUM.

интаксис SQL Server / MS Access:

```
SELECT TOP number|percent column_name(s) пример
SELECT TOP 3 * FROM Customers;
FROM table_name
WHERE condition;
```

Синтаксис MySQL:

```
SELECT column_name(s)                    SELECT * FROM Customers LIMIT 3;
FROM table_name
WHERE condition
LIMIT number;
```

Синтаксис Oracle:

```
SELECT column_name(s)                  SELECT * FROM Customers   WHERE ROWNUM
<= 3;
FROM table_name
WHERE ROWNUM <= number;
```

Функция **MIN()** возвращает наименьшее значение выбранного столбца.
Функция **MAX()** возвращает наибольшее значение выбранного столбца.

```
SELECT MIN(column_name)          SELECT MIN(Price) AS SmallestPrice
FROM table_name                  FROM Products;
WHERE condition;
```

Функция **COUNT()** возвращает количество строк, соответствующих заданному критерию.
Функция **AVG()** возвращает среднее значение числового столбца.
Функция **SUM()** возвращает общую сумму числового столбца.

Синтаксис COUNT()

```
SELECT COUNT(column_name)

FROM table_name
WHERE condition;
```

Синтаксис SUM()

```
SELECT SUM(column_name)

FROM table_name
WHERE condition;
```

пример

```
SELECT AVG(Price)
FROM Products;
```

```
SELECT COUNT(ProductID)
FROM Products;
```

```
SELECT SUM(Quantity)
FROM OrderDetails;
```

SQL LIKE

Оператор **LIKE** используется в предложении **WHERE** для поиска указанного шаблона в столбце.

Есть два подстановочных знака, часто используемых в сочетании с оператором LIKE:

- **%** - Знак процента представляет собой ноль, один или несколько символов
- **_** - Подчеркивание представляет собой один символ

MS Access использует звездочку (*****) вместо знака процента (**%**) и вопросительный знак (**?**) вместо подчеркивания (**_**).

Вот несколько примеров, показывающих различные операторы LIKE с подстановочными знаками '%' и '_':

Оператор LIKE	Описание
WHERE CustomerName LIKE 'a%'	Находит любые значения, которые начинаются с "a"
WHERE CustomerName LIKE '%a'	Находит любые значения, которые заканчиваются на "a"
WHERE CustomerName LIKE '%or%'	Находит любые значения, которые имеют "or" в любой позиции
WHERE CustomerName LIKE '_r%'	Находит любые значения, имеющие букву "r" во второй позиции
WHERE CustomerName LIKE 'a__%'	Находит любые значения, начинающиеся с буквы "a" и имеющие длину не менее 3 символов
WHERE ContactName LIKE 'a%o'	Находит любые значения, которые начинаются с "a" и заканчиваются "o"

```
SELECT * FROM Customers
WHERE CustomerName LIKE '%or%';
```

```
SELECT * FROM Customers
WHERE ContactName LIKE 'a%o';
```

SQL Подстановочные знаки

Подстановочный знак используется для замены одного или нескольких символов в строке. Подстановочные знаки используются с оператором SQL LIKE. Оператор **LIKE** используется в предложении **WHERE** для поиска указанного шаблона в столбце.

Оператор LIKE	Описание
WHERE CustomerName LIKE 'a%'	Находит любые значения, которые начинаются с "a"
WHERE CustomerName LIKE '%a'	Находит любые значения, которые заканчиваются на "a"
WHERE CustomerName LIKE '%or%'	Находит любые значения, которые имеют "or" в любой позиции
WHERE CustomerName LIKE '_r%'	Находит любые значения, имеющие букву "r" во второй позиции
WHERE CustomerName LIKE 'a_%_%'	Находит любые значения, начинающиеся с буквы "a" и имеющие длину не менее 3 символов
WHERE ContactName LIKE 'a%o'	Находит любые значения, которые начинаются с "a" и заканчиваются на "o"

Использование подстановочного знака [!charlist]

В двух следующих инструкциях SQL выбираются все клиенты с городом, не начинающимся с "b", "s" или "p":

```
SELECT * FROM Customers
WHERE City LIKE '[!bsp]%';
```

```
SELECT * FROM Customers
WHERE City NOT LIKE '[bsp]%';
```

Использование подстановочного знака [charlist]

Следующая инструкция SQL выбирает всех клиентов с городом, начинающимся с "b", "s" или "p":

```
SELECT * FROM Customers
WHERE City LIKE '[bsp]%';
```

начинающимся с "a", "b" или "c":

```
SELECT * FROM Customers
WHERE City LIKE '[a-c]%';
```

SQL IN

Оператор **IN** позволяет указать несколько значений в предложении WHERE.

Оператор **IN**- это сокращенное выражение для нескольких условий **OR**.

```
SELECT column_name(s)
FROM table_name
WHERE column_name IN (value1, value2, ...);
```

Или

```
SELECT column_name(s)
FROM table_name
WHERE column_name IN (SELECT STATEMENT);
```

ример IN

Следующая инструкция SQL выбирает всех клиентов, которые находятся в "Germany", "France" или "UK":

```
SELECT * FROM Customers
WHERE Country IN ('Germany', 'France', 'UK');
```

следующей инструкции SQL выбираются все клиенты из тех же стран, что и поставщики:

```
SELECT * FROM Customers
WHERE Country IN (SELECT Country FROM Suppliers);
```

SQL BETWEEN

Оператор **BETWEEN** выбирает значения в заданном диапазоне. Эти значения могут быть числами, текстом или датами.

Оператор **BETWEEN** является инклюзивным: включаются начальные и конечные значения.

```
SELECT column_name(s)
FROM table_name
WHERE column_name BETWEEN value1 AND value2;
```

Следующая инструкция SQL выбирает все продукты с ценой от 10 до 20:

```
SELECT * FROM Products
WHERE Price BETWEEN 10 AND 20;
```

Следующая инструкция SQL выбирает все заказы с датой заказа между '01-July-1996' и '31-July-1996':

```
SELECT * FROM Orders
WHERE OrderDate BETWEEN #01/07/1996# AND #31/07/1996#;
```

SQL Алиасы / Псевдонимы

Псевдонимы SQL используются для присвоения таблице или столбцу в таблице временного имени.

Псевдонимы часто используются для того, чтобы сделать имена столбцов более удобочитаемыми.

Псевдоним существует только на время выполнения запроса.

```
SELECT column_name(s)
FROM table_name AS alias_name;
```

Следующий оператор SQL создает два псевдонима, один для столбца "CustomerID" и "CustomerName" для колонки:

```
SELECT CustomerName AS Customer, ContactName AS [Contact Person]
FROM Customers;
```

Следующий оператор SQL создает псевдоним "Address", которые объединяют четыре колонны (Address, PostalCode, City и Country):

```
SELECT CustomerName, Address + ', ' + PostalCode + ' ' + City + ', ' +
Country AS Address
FROM Customers;
```

Алиасы могут быть полезны, когда:

- В запросе участвует несколько таблиц
- В запросе используются функции
- Имена столбцов большие или не очень читаемые
- Две или более колонн объединяются вместе

SQL JOIN

Предложение **JOIN** используется для объединения строк из двух или более таблиц на основе связанного столбца между ними.

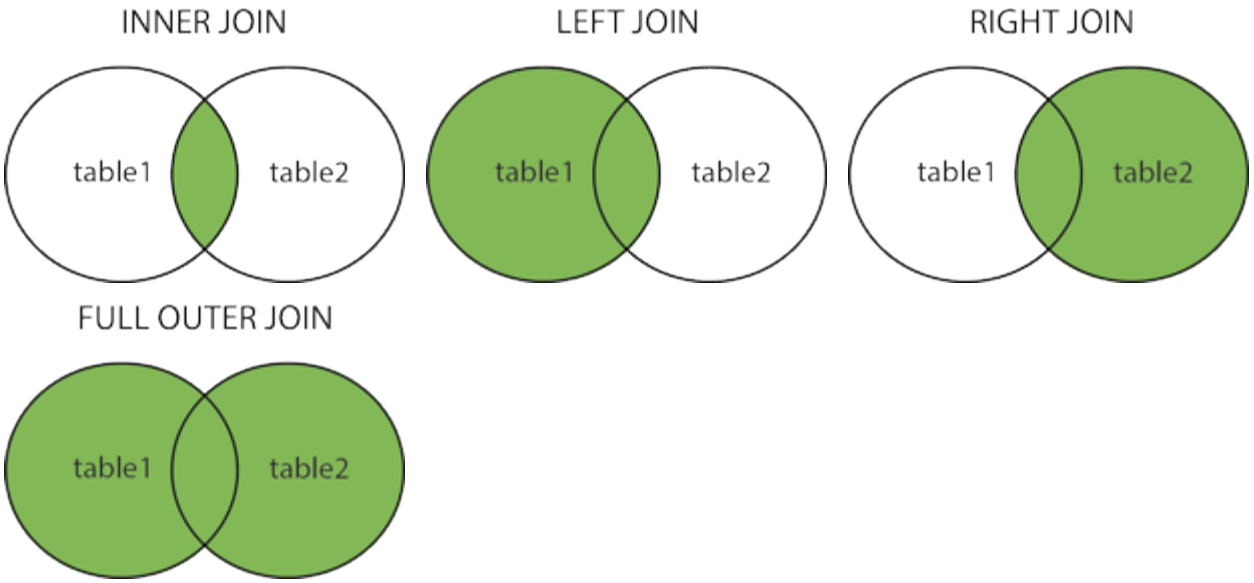
SQL JOIN

Предложение **JOIN** используется для объединения строк из двух или более таблиц на основе связанного столбца между ними.

Различные типы соединений SQL

Вот различные типы соединений в SQL:

- **(INNER) JOIN**: Возвращает записи, имеющие совпадающие значения в обеих таблицах
- **LEFT (OUTER) JOIN**: Возвращает все записи из левой таблицы и совпадающие записи из правой таблицы
- **RIGHT (OUTER) JOIN**: Возвращает все записи из правой таблицы и совпадающие записи из левой таблицы
- **FULL (OUTER) JOIN**: Возвращает все записи при наличии совпадения в левой или правой таблице



SQL INNER JOIN

Ключевое слово **INNER JOIN** выбирает записи, ***имеющие совпадающие*** значения в обеих таблицах.

```
SELECT column_name(s)
FROM table1
INNER JOIN table2
ON table1.column_name = table2.column_name;
```

имер SQL INNER JOIN

Следующая инструкция SQL выбирает все заказы с информацией о клиенте:

```
SELECT Orders.OrderID, Customers.CustomerName
FROM Orders
INNER JOIN Customers ON Orders.CustomerID = Customers.CustomerID;
```

Следующая инструкция SQL выбирает все заказы с информацией о клиенте и грузоотправителе:

```
SELECT Orders.OrderID, Customers.CustomerName, Shippers.ShipperName
FROM ((Orders
INNER JOIN Customers ON Orders.CustomerID = Customers.CustomerID)
INNER JOIN Shippers ON Orders.ShipperID = Shippers.ShipperID);
```

SQL LEFT JOIN

Ключевое слово **LEFT JOIN** ***возвращает*** все ***записи*** из ***левой таблицы*** (table1) и соответствующие записи из правой таблицы (table2). Результат будет нулевым с правой стороны, если нет совпадения.

В следующей инструкции SQL будут выбраны все клиенты и любые заказы, которые они могут иметь:

```
SELECT Customers.CustomerName, Orders.OrderID
FROM Customers
LEFT JOIN Orders ON Customers.CustomerID = Orders.CustomerID
ORDER BY Customers.CustomerName;
```

Ключевое слово LEFT JOIN возвращает все записи из левой таблицы (Customers), даже если в правой таблице (Orders) нет совпадений.

SQL RIGHT JOIN

Ключевое слово **RIGHT JOIN** возвращает все записи из правой таблицы (table2) и совпадающие записи из левой таблицы (table1). Результат равен нулю с левой стороны, когда нет совпадения.

```
SELECT column_name(s)
FROM table1
RIGHT JOIN table2
ON table1.column_name = table2.column_name;
```

Пример SQL RIGHT JOIN

Следующая инструкция SQL вернет всех сотрудников и все заказы, которые они могли бы разместить:

```
SELECT Orders.OrderID, Employees.LastName, Employees.FirstName
FROM Orders
RIGHT JOIN Employees ON Orders.EmployeeID = Employees.EmployeeID
ORDER BY Orders.OrderID;
```

Примечание: Ключевое слово RIGHT JOIN возвращает все записи из правой таблицы (Employees), даже если в левой таблице (Orders) нет совпадений.

SQL FULL OUTER JOIN

Ключевое слово FULL OUTER JOIN возвращает все записи, когда есть совпадение в записях таблицы left (table1) или right (table2).

Примечание: FULL OUTER JOIN потенциально может возвращать очень большие результирующие наборы!

Совет: FULL OUTER JOIN и FULL JOIN - это одно и то же.

```
SELECT column_name(s)
FROM table1
FULL OUTER JOIN table2
ON table1.column_name = table2.column_name
WHERE condition;
```

Следующая инструкция SQL выбирает всех клиентов и все заказы:

```
SELECT Customers.CustomerName, Orders.OrderID
FROM Customers
FULL OUTER JOIN Orders ON Customers.CustomerID=Orders.CustomerID
ORDER BY Customers.CustomerName;
```

: Ключевое слово FULL OUTER JOIN возвращает все совпадающие записи из обеих таблиц, независимо от того, совпадает ли другая таблица или нет. Таким образом, если есть строки в разделе "Customers", которые не имеют совпадений в разделе "Orders", или если есть строки в разделе "Orders", которые не имеют совпадений в разделе "Customers", то эти строки также будут перечислены.

QL JOIN Самостоятельно

Самостоятельное соединение - это обычное соединение, но таблица соединяется сама с собой.

```
SELECT column_name(s)
FROM table1 T1, table1 T2
WHERE condition;
```

T1 и *T2* это разные псевдонимы таблиц для одной и той же таблицы.

ледующая инструкция SQL соответствует клиентам из одного и того же города:

```
SELECT A.CustomerName AS CustomerName1, B.CustomerName AS CustomerName2, A.City
FROM Customers A, Customers B
WHERE A.CustomerID <> B.CustomerID
AND A.City = B.City
ORDER BY A.City;
```