

# The SELECT statement

This part of the SQLite tutorial covers SQLite's implementation of the SELECT statement in detail.

## Retrieving all data

The following SQL statement is one of the most common ones. It is also one of the most expensive ones.

```
sqlite> SELECT * FROM Cars;
Id          Name          Price
-----
1           Audi          52642
2           Mercedes     57127
3           Skoda         9000
4           Volvo         29000
5           Bentley      350000
6           Citroen      21000
7           Hummer       41400
8           Volkswagen   21600
```

Here we retrieve all data from the Cars table.

## Selecting specific columns

We can use the SELECT statement to retrieve specific columns. The column names follow the SELECT word.

```
sqlite> SELECT Name, Price FROM Cars;
Name        Price
-----
Audi        52642
Mercedes    57127
Skoda       9000
Volvo       29000
Bentley     350000
Citroen     21000
Hummer      41400
Volkswagen  21600
```

We retrieve the Name and the Price columns. The column names are separated by commas.

## Renaming column names

We can rename the column names of the returned result set. For this, we use the AS clause.

```
sqlite> SELECT Name, Price AS 'Price of car' FROM Cars;
Name        Price of car
-----
Audi        52642
Mercedes    57127
Skoda       9000
Volvo       29000
Bentley     350000
Citroen     21000
Hummer      41400
Volkswagen  21600
```

## Ordering data

We use the `ORDER BY` clause to sort the returned data set. The `ORDER BY` clause is followed by the column on which we do the sorting. The `ASC` keyword sorts the data in ascending order, the `DESC` in descending order.

```
sqlite> SELECT * FROM Cars ORDER BY Price;
Id      Name      Price
-----  -
3       Skoda     9000
6       Citroen  21000
8       Volkswagen 21600
4       Volvo    29000
7       Hummer   41400
1       Audi     52642
2       Mercedes 57127
5       Bentley  350000
```

The default sorting is in ascending order. The `ASC` clause can be omitted.

```
sqlite> SELECT Name, Price FROM Cars ORDER BY Price DESC;
Name      Price
-----  -
Bentley   350000
Mercedes  57127
Audi      52642
Hummer    41400
Volvo     29000
Volkswagen 21600
Citroen   21000
Skoda     9000
```

In the above SQL statement, we select `Name` and `Price` columns from the `Cars` table and sort it by the `Price` of the cars in descending order. So the most expensive cars come first.

## Selecting specific rows with the WHERE Clause

The next set of examples uses the `Orders` table.

```
sqlite> SELECT * FROM Orders;
Id      OrderPrice Customer
-----  -
1       1200      Williamson
2       200       Robertson
3       40        Robertson
4       1640     Smith
5       100       Robertson
6       50        Williamson
7       150       Smith
8       250       Smith
9       840       Brown
10      440       Black
11      20        Brown
```

Here we see all the data from the `Orders` table.

Next, we want to select a specific row.

```
sqlite> SELECT * FROM Orders WHERE Id=6;
```

Id	OrderPrice	Customer
6	50	Williamson

The above SQL statement selects a row that has Id 6.

```
sqlite> SELECT * FROM Orders WHERE Customer="Smith";
```

Id	OrderPrice	Customer
4	1640	Smith
7	150	Smith
8	250	Smith

The above SQL statement selects all orders from the Smith customer.

We can use the LIKE clause to look for a specific pattern in the data.

```
sqlite> SELECT * FROM Orders WHERE Customer LIKE 'B%';
```

Id	OrderPrice	Customer
9	840	Brown
10	440	Black
11	20	Brown

This SQL statement selects all orders from customers whose names begin with letter B.

## Removing duplicate items

The DISTINCT clause is used to select only unique items from the result set.

```
sqlite> SELECT Customer FROM Orders WHERE Customer LIKE 'B%';
```

Customer
Brown
Black
Brown

This time we have selected customers whose names begin with B. We can see that Brown appears twice. To remove duplicates, we use the DISTINCT keyword.

```
sqlite> SELECT DISTINCT Customer FROM Orders WHERE Customer LIKE 'B%';
```

Customer
Black
Brown

This is the correct solution.

## Grouping data

The **GROUP BY** clause is used to combine database records with identical values into a single record. It is often used with the aggregate functions.

Say we wanted to find out the sum of each customers' orders.

```
sqlite> SELECT sum(OrderPrice) AS Total, Customer FROM Orders GROUP BY Customer;
Total      Customer
-----
440        Black
860        Brown
340        Robertson
2040       Smith
1250       Williamson
```

The `sum()` function returns the total sum of a numeric column. The **GROUP BY** clause divides the total sum among the customers. So we can see that Black has ordered items for 440 or Smith for 2040.

We cannot use the **WHERE** clause when aggregate functions are used. We use the **HAVING** clause instead.

```
sqlite> SELECT sum(OrderPrice) AS Total, Customer FROM Orders
        GROUP BY Customer HAVING sum(OrderPrice)>1000;
Total      Customer
-----
2040       Smith
1250       Williamson
```

The above SQL statement selects customers whose total orders where greater than 1000 units.

In this part of the SQLite tutorial, we described the SQL **SELECT** statement in more detail.

Quelle: <http://zetcode.com/db/sqlite/select/>