

# SQL (Structured Query Language) in one page

Contents:

Review: [FAQ](#);

Table of contents: [Database Manipulation \(CREATE, DROP DATABASE\)](#), [Table Manipulation \(CREATE, ALTER, DROP TABLE, Data Types\)](#), [Index Manipulation \(CREATE, DROP INDEX\)](#), [Data Manipulation \(INSERT, UPDATE, DELETE, TRUNCATE TABLE\)](#), [Select \(SELECT, FROM, WHERE, ORDER BY, GROUP BY, HAVING, Operators, Aggregate functions\)](#), [Alias](#), [Join](#), [UNION](#), [SELECT INTO/IN](#), [CREATE VIEW](#);

Similar Sites: [MANUAL](#), [Cheat sheets](#), [HTML](#), [CSS](#), [XML](#), [DTD](#), [JavaScript](#), [W3C DOM](#), [SQL](#), [SSI](#), [Tell a friend](#), [Free Icons](#), [Itlibitum, Corp.](#).

## FAQ

What is SQL?

- SQL stands for Structured Query Language;
- SQL allows you to access a database;
- SQL is an ANSI standard computer language;
- SQL use to access and manipulate data in Oracle, Sybase, Microsoft SQL Server, DB2, Access, MySQL, PostgreSQL and other database systems;
- SQL can execute queries against a database;
- SQL can retrieve data from a database;
- SQL can insert new records in a database;
- SQL can delete records from a database;
- SQL can update records in a database;
- SQL is easy to learn.

## Database Manipulation

CREATE DATABASE database\_name

Create a database

CREATE DATABASE My\_First\_Database

DROP DATABASE database\_name

Delete a database

DROP DATABASE My\_First\_Database

## Table Manipulation

CREATE TABLE "table\_name"  
("column\_1" "data\_type\_for\_column\_1",  
"column\_2" "data\_type\_for\_column\_2",  
... )

Create a table in a database.

CREATE TABLE Person  
(LastName varchar,  
FirstName varchar,  
Address varchar,  
Age int)

Data Types	
Data Type	Description
integer(size)	Hold integers only. The maximum number of digits are specified in parenthesis.
int(size)	

```
ALTER TABLE table_name ADD column_name datatype
ALTER TABLE table_name DDROP column_name datatype
DROP TABLE table_name
```

```
CREATE INDEX index_name
ON table_name (column_name_1, column_name_2, ...)
```

```
CREATE UNIQUE INDEX index_name
ON table_name (column_name_1, column_name_2, ...)
DROP INDEX table_name.index_name
```

```
INSERT INTO table_name
VALUES (value_1, value_2,...)
INSERT INTO table_name (column1, column2,...)
VALUES (value_1, value_2,...)
UPDATE table_name
SET column_name_1 = new_value_1, column_name_2 =
new_value_2
WHERE column_name = some_value
DELETE FROM table_name
WHERE column_name = some_value
TRUNCATE TABLE table_name
```

```
SELECT column_name(s) FROM table_name
SELECT * FROM table_name
SELECT DISTINCT column_name(s) FROM table_name
SELECT column_name(s) FROM table_name
WHERE column operator value
AND column operator value
OR column operator value
AND (... OR ...)
...
```

smallint(size)	
tinyint(size)	
decimal(size,d)	Hold numbers with fractions. The maximum number of digits are specified in "size". The maximum number of digits to the right of the decimal is specified in "d".
numeric(size,d)	
char(size)	Holds a fixed length string (can contain letters, numbers, and special characters). The fixed size is specified in parenthesis.
varchar(size)	Holds a variable length string (can contain letters, numbers, and special characters). The maximum size is specified in parenthesis.
date(yyyymmdd)	Holds a date

```
Add columns in an existing table.
Delete columns in an existing table.
Delete a table.
```

Index Manipulation

```
Create a simple index.

Create a unique index.

Delete a index.
```

Data Manipulation

```
Insert new rows into a table.

Update one or several columns in rows.

Delete rows in a table.

Deletes the data inside the table.
```

Select

```
Select data from a table.
Select all data from a table.
Select only distinct (different) data from a table.
Select only certain data from a table.
```

Operators	
Operator	Description
=	Equal
<>	Not equal
>	Greater than

```
ALTER TABLE Person ADD Sex char(6)
ALTER TABLE Person DROP Sex char(6)
DROP TABLE Person
```

```
CREATE INDEX PersonIndex
ON Person (LastName, FirstName)
```

```
CREATE UNIQUE INDEX PersonIndex
ON Person (LastName DESC)
DROP INDEX Person.PersonIndex
```

```
INSERT INTO Persons
VALUES(' Hussein', ' Saddam', ' White House')
INSERT INTO Persons (LastName, FirstName, Address)
VALUES(' Hussein', ' Saddam', ' White House')
UPDATE Person
SET Address = ' ups'
WHERE LastName = ' Hussein'

DELETE FROM Person WHERE LastName = ' Hussein'

TRUNCATE TABLE Person
```

```
SELECT LastName, FirstName FROM Persons
SELECT * FROM Persons
SELECT DISTINCT LastName, FirstName FROM Persons
SELECT * FROM Persons WHERE sex=' female'
SELECT * FROM Persons WHERE Year>1970

SELECT * FROM Persons
WHERE FirstName=' Saddam'
AND LastName=' Hussein'

SELECT * FROM Persons
WHERE FirstName=' Saddam'
```

```
SELECT column_name(s) FROM table_name
WHERE column_name IN (value1, value2, ...)
SELECT column_name(s) FROM table_name
ORDER BY row_1, row_2 DESC, row_3 ASC, ...
```

```
SELECT column_1, ..., SUM(group_column_name)
FROM table_name
GROUP BY group_column_name
```

```
SELECT column_1, ..., SUM(group_column_name)
FROM table_name
GROUP BY group_column_name
HAVING SUM(group_column_name) condition value
```

```
SELECT column_name AS column_alias FROM table_name

SELECT table_alias.column_name FROM table_name AS
table_alias
```

```
SELECT column_1_name, column_2_name, ...
FROM first_table_name
INNER JOIN second_table_name
ON first_table_name.keyfield =
second_table_name.foreign_keyfield
SELECT column_1_name, column_2_name, ...
```

<	Less than
>=	Greater than or equal
<=	Less than or equal
BETWEEN	Between an inclusive range
LIKE	Search for a pattern. A "%" sign can be used to define wildcards (missing letters in the pattern) both before and after the pattern.

The IN operator may be used if you know the exact value you want to return for at least one of the columns.

Select data from a table with sort the rows.

Note:

- ASC (ascend) is a alphabetical and numerical order (optional)
- DESC (descend) is a reverse alphabetical and numerical order

GROUP BY... was added to SQL because aggregate functions (like SUM) return the aggregate of all column values every time they are called, and without the GROUP BY function it was impossible to find the sum for each individual group of column values.

Some aggregate functions	
Function	Description
AVG(column)	Returns the average value of a column
COUNT(column)	Returns the number of rows (without a NULL value) of a column
MAX(column)	Returns the highest value of a column
MIN(column)	Returns the lowest value of a column
SUM(column)	Returns the total sum of a column

HAVING... was added to SQL because the WHERE keyword could not be used against aggregate functions (like SUM), and without HAVING... it would be impossible to test for result conditions.

Alias

Column name alias

Table name alias

Join

The INNER JOIN returns all rows from both tables where there is a match. If there are rows in first table that do not have matches in second table, those rows will not be listed.

The LEFT JOIN returns all the rows from the first table,

```
OR LastName='Hussein'

SELECT * FROM Persons WHERE
(FirstName='Tove' OR FirstName='Stephen')
AND LastName='Svendson'
```

```
SELECT * FROM Persons WHERE FirstName LIKE '0%'
SELECT * FROM Persons WHERE FirstName LIKE '%a'
SELECT * FROM Persons WHERE FirstName LIKE '%la%'
```

```
SELECT * FROM Persons
WHERE LastName IN ('Hansen','Pettersen')
SELECT * FROM Persons
ORDER BY LastName
```

```
SELECT FirstName, LastName FROM Persons
ORDER BY LastName DESC
```

```
SELECT Company, OrderNumber FROM Orders
ORDER BY Company DESC, OrderNumber ASC
```

```
SELECT Company, SUM(Amount)
FROM Sales
GROUP BY Company
```

```
SELECT Company, SUM(Amount)
FROM Sales
GROUP BY Company
HAVING SUM(Amount)>10000
```

```
SELECT LastName AS Family, FirstName AS Name
FROM Persons

SELECT LastName, FirstName
FROM Persons AS Employees
```

```
SELECT Employees.Name, Orders.Product
FROM Employees
INNER JOIN Orders
ON Employees.Employee_ID=Orders.Employee_ID
```

```
SELECT Employees.Name, Orders.Product
```

```
FROM first_table_name
LEFT JOIN second_table_name
ON first_table_name.keyfield =
second_table_name.foreign_keyfield
SELECT column_1_name, column_2_name, ...
FROM first_table_name
RIGHT JOIN second_table_name
ON first_table_name.keyfield =
second_table_name.foreign_keyfield
```

even if there are no matches in the second table. If there are rows in first table that do not have matches in second table, those rows also will be listed.

The RIGHT JOIN returns all the rows from the second table, even if there are no matches in the first table. If there had been any rows in second table that did not have matches in first table, those rows also would have been listed.

```
FROM Employees
LEFT JOIN Orders
ON Employees.Employee_ID=Orders.Employee_ID

SELECT Employees.Name, Orders.Product
FROM Employees
RIGHT JOIN Orders
ON Employees.Employee_ID=Orders.Employee_ID
```

## UNION

```
SQL_Statement_1
UNION
SQL_Statement_2
SQL_Statement_1
UNION ALL
SQL_Statement_2
```

Select all different values from SQL\_Statement\_1 and SQL\_Statement\_2

Select all values from SQL\_Statement\_1 and SQL\_Statement\_2

```
SELECT E_Name FROM Employees_Norway
UNION
SELECT E_Name FROM Employees_USA
SELECT E_Name FROM Employees_Norway
UNION
SELECT E_Name FROM Employees_USA
```

## SELECT INTO/IN

```
SELECT column_name(s)
INTO new_table_name
FROM source_table_name
WHERE query

SELECT column_name(s)
IN external_database_name
FROM source_table_name
WHERE query
```

Select data from table(S) and insert it into another table.

Select data from table(S) and insert it in another database.

```
SELECT * INTO Persons_backup FROM Persons
```

```
SELECT Persons.* INTO Persons IN 'Backup.db' FROM
Persons WHERE City='Sandnes'
```

## CREATE VIEW

```
CREATE VIEW view_name AS
SELECT column_name(s)
FROM table_name
WHERE condition
```

Create a virtual table based on the result-set of a SELECT statement.

```
CREATE VIEW [Current Product List] AS
SELECT ProductID, ProductName
FROM Products
WHERE Discontinued=No
```

## OTHER

© 2006 [MANUAL.SU](http://MANUAL.SU) ([Itlibitum.Corp.](http://Itlibitum.Corp.))

