The ORDER BY clause is used to sort the rows.

**Orders:**

|  |  |
| --- | --- |
| **Company** | **OrderNumber** |
| Sega | 3412 |
| ABC Shop | 5678 |
| AAPL | 6798 |
| IBM | 2312 |
| DELL | 9777 |
| CITI BANK | 8666 |
| TD BANK | 1010 |
| FORD | 1010 |
| IBM | 3515 |
| E-TRADE | 9000 |

## Example

## To display the company names in alphabetical order:

|  |
| --- |
| SELECT Company, OrderNumber FROM Orders  ORDER BY Company |

**Result:**

|  |  |
| --- | --- |
| **Company** | **OrderNumber** |
| AAPL | 6798 |
| ABC Shop | 5678 |
| CITI BANK | 8666 |
| DELL | 9777 |
| E TRADE | 9000 |
| FORD | 1010 |
| IBM | 2312 |
| IBM | 3515 |
| Sega | 3412 |
| TD BANK | 1010 |

## Example

To display the company names in alphabetical order AND the OrderNumber in numerical order:

|  |
| --- |
| SELECT Company, OrderNumber FROM Orders  ORDER BY Company, OrderNumber |

**Result:**

|  |  |
| --- | --- |
| **Company** | **OrderNumber** |
| ABC Shop | 5678 |
| Sega | 3412 |
| TalenTech | 2312 |
| TalenTech | 6798 |

## Example

To display the company names in reverse alphabetical order:

|  |
| --- |
| SELECT Company, OrderNumber FROM Orders  ORDER BY Company DESC |

**Result:**

|  |  |
| --- | --- |
| **Company** | **OrderNumber** |
| TalenTech | 6798 |
| TalenTech | 2312 |
| Sega | 3412 |
| ABC Shop | 5678 |

## Example

**To display the company names in reverse alphabetical order AND the OrderNumber in numerical order:**

|  |
| --- |
| SELECT Company, OrderNumber FROM Orders  ORDER BY Company DESC, OrderNumber ASC |

**Result:**

|  |  |
| --- | --- |
| **Company** | **OrderNumber** |
| TalenTech | 2312 |
| TalenTech | 6798 |
| Sega | 3412 |
| ABC Shop | 5678 |

Notice that there are two equal company names (TalenTech) in the result above. The only time you will see the second column in ASC order would be when there are duplicated values in the first sort column, or a handful of nulls.

## AND & OR

AND and OR join two or more conditions in a WHERE clause.

The AND operator displays a row if ALL conditions listed are true. The OR operator displays a row if ANY of the conditions listed are true.

Original Table (used in the examples)

|  |  |  |  |
| --- | --- | --- | --- |
| **FirstName** | **LastName** | **Address** | **City** |
| Hansen | Ola | Timoteivn 10 | Sandnes |
| Michael | Jordan | Borgvn 23 | Sandnes |
| Michael | Jackson | Kaivn 18 | Sandnes |

## Example

Use AND to display each person with the first name equal to " Michael ", and the last name equal to " Jordan ":

|  |
| --- |
| SELECT \* FROM Persons  WHERE LastName=' Jordan'  AND firstname=' Michael' |

**Result:**

|  |  |  |  |
| --- | --- | --- | --- |
| **LastName** | **FirstName** | **Address** | **City** |
| Jordan | Michael | Borgvn 23 | Sandnes |

## Example

Use OR to display each person with the first name equal to " Michael ", or the last name equal to " Jordan ":

|  |
| --- |
| SELECT \* FROM Persons  WHERE LastName=' Jordan'  OR firstname=' Michael' |

**Result:**

|  |  |  |  |
| --- | --- | --- | --- |
| **LastName** | **FirstName** | **Address** | **City** |
| Jordan | Michael | Borgvn 23 | Sandnes |
| Jackson | Michael | Kaivn 18 | Sandnes |

## Example

You can also combine AND and OR (use parentheses to form complex expressions):

|  |
| --- |
| SELECT \* FROM Persons WHERE  (LastName=' Jordan' OR LastName=' Jackson ')  AND firstname=' Michael' |

**Result:**

|  |  |  |  |
| --- | --- | --- | --- |
| **LastName** | **FirstName** | **Address** | **City** |
| Jordan | Michael | Borgvn 23 | Sandnes |
| Jackson | Michael | Kaivn 18 | Sandnes |

# SQL IN

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

|  |
| --- |
| SELECT column\_name FROM table\_name  WHERE column\_name IN (value1*,*value2,..) |

## Original Table (used in the examples)

|  |  |  |  |
| --- | --- | --- | --- |
| **LastName** | **FirstName** | **Address** | **City** |
| Hansen | Ola | Timoteivn 10 | Sandnes |
| Nordmann | Anna | Neset 18 | Sandnes |
| Pettersen | Kari | Storgt 20 | Stavanger |
| Svendson | Tove | Borgvn 23 | Sandnes |

## Example 1

To display the persons with LastName equal to "Hansen" or "Pettersen", use the following SQL:

|  |
| --- |
| SELECT \* FROM Persons  WHERE LastName IN ('Hansen','Pettersen') |

**Result:**

|  |  |  |  |
| --- | --- | --- | --- |
| **LastName** | **FirstName** | **Address** | **City** |
| Hansen | Ola | Timoteivn 10 | Sandnes |
| Pettersen | Kari | Storgt 20 | Stavanger |

# SQL BETWEEN

## BETWEEN ... AND

The BETWEEN ... AND operator selects a range of data between two values. These values can be numbers, text, or dates.

|  |
| --- |
| SELECT column\_name FROM table\_name  WHERE column\_name  BETWEEN value1AND value2 |

## Original Table (used in the examples)

|  |  |  |  |
| --- | --- | --- | --- |
| **LastName** | **FirstName** | **Address** | **City** |
| Hansen | Ola | Timoteivn 10 | Sandnes |
| Nordmann | Anna | Neset 18 | Sandnes |
| Pettersen | Kari | Storgt 20 | Stavanger |
| Svendson | Tove | Borgvn 23 | Sandnes |

## Example 1

To display the persons alphabetically between (and including) "Hansen" and exclusive "Pettersen", use the following SQL:

|  |
| --- |
| SELECT \* FROM Persons WHERE LastName  BETWEEN 'Hansen' AND 'Pettersen' |

**Result:**

|  |  |  |  |
| --- | --- | --- | --- |
| **LastName** | **FirstName** | **Address** | **City** |
| Hansen | Ola | Timoteivn 10 | Sandnes |
| Nordmann | Anna | Neset 18 | Sandnes |

**IMPORTANT!** The BETWEEN...AND operator is treated differently in different databases. With some databases a person with the LastName of "Hansen" or "Pettersen" will not be listed (BETWEEN..AND only selects fields that are between and excluding the test values). With some databases a person with the last name of "Hansen" or "Pettersen" will be listed (BETWEEN..AND selects fields that are between and including the test values). With other databases a person with the last name of "Hansen" will be listed, but "Pettersen" will not be listed (BETWEEN..AND selects fields between the test values, including the first test value and excluding the last test value). Therefore: Check how your database treats the BETWEEN....AND operator!

## Joins and Keys

Sometimes we have to select data from two or more tables to make our result complete. We have to perform a join.

Tables in a database can be related to each other with keys. A primary key is a column with a unique value for each row. Each primary key value must be unique within the table. The purpose is to bind data together, across tables, without repeating all of the data in every table.

In the "Employees" table below, the "Employee\_ID" column is the primary key, meaning that **no** two rows can have the same Employee\_ID. The Employee\_ID distinguishes two persons even if they have the same name.

When you look at the example tables below, notice that:

* The "Employee\_ID" column is the primary key of the "Employees" table
* The "Prod\_ID" column is the primary key of the "Orders" table
* The "Employee\_ID" column in the "Orders" table is used to refer to the persons in the "Employees" table without using their names

**Employees**:

|  |  |
| --- | --- |
| **Employee\_ID** | **Name** |
| 01 | Hansen, Ola |
| 02 | Svendson, Tove |
| 03 | Svendson, Stephen |
| 04 | Pettersen, Kari |

**Orders:**

|  |  |  |
| --- | --- | --- |
| **Prod\_ID** | **Product** | **Employee\_ID** |
| 234 | Printer | 01 |
| 657 | Table | 03 |
| 865 | Chair | 03 |

## UNION

The UNION command is used to select related information from two tables, much like the JOIN command. However, when using the UNION command all selected columns need to be of the same data type.

**Note:** With UNION, only distinct values are selected.

|  |
| --- |
| SQL Statement 1  UNION  SQL Statement 2 |

**Employees\_Norway**:

|  |  |
| --- | --- |
| **E\_ID** | **E\_Name** |
| 01 | Hansen, Ola |
| 02 | Svendson, Tove |
| 03 | Svendson, Stephen |
| 04 | Pettersen, Kari |

**Employees\_USA**:

|  |  |
| --- | --- |
| **E\_ID** | **E\_Name** |
| 01 | Turner, Sally |
| 02 | Kent, Clark |
| 03 | Svendson, Stephen |
| 04 | Scott, Stephen |

## Using the UNION Command

### Example

List all different employee names in Norway and USA:

|  |
| --- |
| SELECT E\_Name FROM Employees\_Norway  UNION  SELECT E\_Name FROM Employees\_USA |

**Result**

|  |
| --- |
| **E\_Name** |
| Hansen, Ola |
| Svendson, Tove |
| Svendson, Stephen |
| Pettersen, Kari |
| Turner, Sally |
| Kent, Clark |
| Scott, Stephen |

**Note:** This command cannot be used to list all employees in Norway and USA. In the example above we have two employees with equal names, and only one of them is listed. The UNION command only selects distinct values.

## Drop Index

You can delete an existing index in a table with the DROP INDEX statement.

Syntax for Microsoft SQLJet (and Microsoft Access):

|  |
| --- |
| DROP INDEX index\_name ON table\_name |

Syntax for MS SQL Server:

|  |
| --- |
| DROP INDEX table\_name.index\_name |

Syntax for IBM DB2 and Oracle:

|  |
| --- |
| DROP INDEX index\_name |

Syntax for MySQL:

|  |
| --- |
| ALTER TABLE table\_name DROP INDEX index\_name |

## Delete a Table or Database

To delete a table (the table structure, attributes, and indexes will also be deleted):

|  |
| --- |
| DROP TABLE table\_name |

To delete a database:

|  |
| --- |
| DROP DATABASE database\_name |

## Truncate a Table

What if we only want to get rid of the data inside a table, and not the table itself? Use the TRUNCATE TABLE command (deletes only the data inside the table):

|  |
| --- |
| TRUNCATE TABLE table\_name |

## ALTER TABLE

The ALTER TABLE statement is used to add or drop columns in an existing table.

|  |
| --- |
| ALTER TABLE table\_name  ADD column\_name datatype  ALTER TABLE table\_name  DROP COLUMN column\_name |

**Note:** Some database systems don't allow the dropping of a column in a database table (DROP COLUMN column\_name).

**Person:**

|  |  |  |
| --- | --- | --- |
| **LastName** | **FirstName** | **Address** |
| Pettersen | Kari | Storgt 20 |

## Example

To add a column named "City" in the "Person" table:

|  |
| --- |
| ALTER TABLE Person ADD City varchar(30) |

**Result:**

|  |  |  |  |
| --- | --- | --- | --- |
| **LastName** | **FirstName** | **Address** | **City** |
| Pettersen | Kari | Storgt 20 |  |

## Example

To drop the "Address" column in the "Person" table:

|  |
| --- |
| ALTER TABLE Person DROP COLUMN Address |

**Result:**

|  |  |  |
| --- | --- | --- |
| **LastName** | **FirstName** | **City** |
| Pettersen | Kari |  |

# SQL Functions

SQL has a lot of built-in functions for counting and calculations.

Function Syntax

The syntax for built-in SQL functions is:

|  |
| --- |
| SELECT function(column) FROM table |

## Types of Functions

There are several basic types and categories of functions in SQL. The basic types of functions are:

* Aggregate Functions
* Scalar functions

## Aggregate functions

Aggregate functions operate against a collection of values, but return a single value.

**Note:** If used among many other expressions in the item list of a SELECT statement, the SELECT must have a GROUP BY clause!!

### "Persons" table (used in most examples)

|  |  |
| --- | --- |
| **Name** | **Age** |
| Hansen, Ola | 34 |
| Svendson, Tove | 45 |
| Pettersen, Kari | 19 |

### Aggregate functions in MS Access

|  |  |
| --- | --- |
| **Function** | **Description** |
| [AVG(column)](http://www.w3schools.com/sql/func_avg.asp) | Returns the average value of a column |
| [COUNT(column)](http://www.w3schools.com/sql/func_count.asp) | Returns the number of rows (without a NULL value) of a column |
| [COUNT(\*)](http://www.w3schools.com/sql/func_count_ast.asp) | Returns the number of selected rows |
| FIRST(column) | Returns the value of the first record in a specified field |
| LAST(column) | Returns the value of the last record in a specified field |
| [MAX(column)](http://www.w3schools.com/sql/func_max.asp) | Returns the highest value of a column |
| [MIN(column)](http://www.w3schools.com/sql/func_min.asp) | Returns the lowest value of a column |
| STDEV(column) |  |
| STDEVP(column) |  |
| [SUM(column)](http://www.w3schools.com/sql/func_sum.asp) | Returns the total sum of a column |
| VAR(column) |  |
| VARP(column) |  |

### Aggregate functions in SQL Server

|  |  |
| --- | --- |
| **Function** | **Description** |
| [AVG(column)](http://www.w3schools.com/sql/func_avg.asp) | Returns the average value of a column |
| BINARY\_CHECKSUM |  |
| CHECKSUM |  |
| CHECKSUM\_AGG |  |
| [COUNT(column)](http://www.w3schools.com/sql/func_count.asp) | Returns the number of rows (without a NULL value) of a column |
| [COUNT(\*)](http://www.w3schools.com/sql/func_count_ast.asp) | Returns the number of selected rows |
| [COUNT(DISTINCT column)](http://www.w3schools.com/sql/func_count_distinct.asp) | Returns the number of distinct results |
| [FIRST(column)](http://www.w3schools.com/sql/func_first.asp) | Returns the value of the first record in a specified field (not supported in SQLServer2K) |
| [LAST(column)](http://www.w3schools.com/sql/func_last.asp) | Returns the value of the last record in a specified field (not supported in SQLServer2K) |
| [MAX(column)](http://www.w3schools.com/sql/func_max.asp) | Returns the highest value of a column |
| [MIN(column)](http://www.w3schools.com/sql/func_min.asp) | Returns the lowest value of a column |
| STDEV(column) |  |
| STDEVP(column) |  |
| [SUM(column)](http://www.w3schools.com/sql/func_sum.asp) | Returns the total sum of a column |
| VAR(column) |  |
| VARP(column) |  |

## Scalar functions

Scalar functions operate against a single value, and return a single value based on the input value.

### Useful Scalar Functions in MS Access

|  |  |
| --- | --- |
| **Function** | **Description** |
| UCASE(c) | Converts a field to upper case |
| LCASE(c) | Converts a field to lower case |
| MID(c,start[,end]) | Extract characters from a text field |
| LEN(c) | Returns the length of a text field |
| INSTR(c,char) | Returns the numeric position of a named character within a text field |
| LEFT(c,number\_of\_char) | Return the left part of a text field requested |
| RIGHT(c,number\_of\_char) | Return the right part of a text field requested |
| ROUND(c,decimals) | Rounds a numeric field to the number of decimals specified |
| MOD(x,y) | Returns the remainder of a division operation |
| NOW() | Returns the current system date |
| FORMAT(c,format) | Changes the way a field is displayed |
| DATEDIFF(d,date1,date2) | Used to perform date calculations |

# SQL GROUP BY and HAVING

Aggregate functions (like SUM) often need an added GROUP BY functionality.

## GROUP BY...

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.

The syntax for the GROUP BY function is:

|  |
| --- |
| SELECT column,SUM(column) FROM table GROUP BY column |

## GROUP BY Example

This "Sales" Table:

|  |  |
| --- | --- |
| **Company** | **Amount** |
| TalenTech | 5500 |
| IBM | 4500 |
| TalenTech | 7100 |

And This SQL:

|  |
| --- |
| SELECT Company, SUM(Amount) FROM Sales |

Returns this result:

|  |  |
| --- | --- |
| **Company** | **SUM(Amount)** |
| TalenTech | 17100 |
| IBM | 17100 |
| TalenTech | 17100 |

The above code is invalid because the column returned is not part of an aggregate. A GROUP BY clause will solve this problem:

|  |
| --- |
| SELECT Company,SUM(Amount) FROM Sales  GROUP BY Company |

Returns this result:

|  |  |
| --- | --- |
| **Company** | **SUM(Amount)** |
| TalenTech | 12600 |
| IBM | 4500 |

## HAVING...

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.

The syntax for the HAVING function is:

|  |
| --- |
| SELECT column,SUM(column) FROM table  GROUP BY column  HAVING SUM(column) condition value |

This "Sales" Table:

|  |  |
| --- | --- |
| **Company** | **Amount** |
| TalenTech | 5500 |
| IBM | 4500 |
| TalenTech | 7100 |

This SQL:

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

Returns this result

|  |  |
| --- | --- |
| **Company** | **SUM(Amount)** |
| TalenTech | 12600 |

# SQL SELECT INTO Statement

## The SELECT INTO Statement

The SELECT INTO statement is most often used to create backup copies of tables or for archiving records.

### Syntax

|  |
| --- |
| SELECT column\_name(s) INTO newtable [IN externaldatabase]  FROM source |

## Make a Backup Copy

The following example makes a backup copy of the "Persons" table:

|  |
| --- |
| SELECT \* INTO Persons\_backup  FROM Persons |

The IN clause can be used to copy tables into another database:

|  |
| --- |
| SELECT Persons.\* INTO Persons IN 'Backup.mdb'  FROM Persons |

If you only want to copy a few fields, you can do so by listing them after the SELECT statement:

|  |
| --- |
| SELECT LastName,FirstName INTO Persons\_backup  FROM Persons |

You can also add a WHERE clause. The following example creates a "Persons\_backup" table with two columns (FirstName and LastName) by extracting the persons who lives in "Sandnes" from the "Persons" table:

|  |
| --- |
| SELECT LastName,Firstname INTO Persons\_backup  FROM Persons  WHERE City='Sandnes' |

Selecting data from more than one table is also possible. The following example creates a new table "Empl\_Ord\_backup" that contains data from the two tables Employees and Orders:

|  |
| --- |
| SELECT Employees.Name,Orders.Product  INTO Empl\_Ord\_backup  FROM Employees  INNER JOIN Orders  ON Employees.Employee\_ID=Orders.Employee\_ID |

# SQL CREATE VIEW Statement

A view is a virtual table based on the result-set of a SELECT statement.

## What is a View?

In SQL, a VIEW is a virtual table based on the result-set of a SELECT statement.

A view contains rows and columns, just like a real table. The fields in a view are fields from one or more real tables in the database. You can add SQL functions, WHERE, and JOIN statements to a view and present the data as if the data were coming from a single table.

**Note:** The database design and structure will NOT be affected by the functions, where, or join statements in a view.

### Syntax

|  |
| --- |
| CREATE VIEW view\_name AS  SELECT column\_name(s)  FROM table\_name  WHERE condition |

**Note:** The database does not store the view data! The database engine recreates the data, using the view's SELECT statement, every time a user queries a view.

## Using Views

A view could be used from inside a query, a stored procedure, or from inside another view. By adding functions, joins, etc., to a view, it allows you to present exactly the data you want to the user.

The sample database Northwind has some views installed by default. The view "Current Product List" lists all active products (products that are not discontinued) from the Products table. The view is created with the following SQL:

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

We can query the view above as follows:

|  |
| --- |
| SELECT \* FROM [Current Product List] |

Another view from the Northwind sample database selects every product in the Products table that has a unit price that is higher than the average unit price:

|  |
| --- |
| CREATE VIEW [Products Above Average Price] AS  SELECT ProductName,UnitPrice  FROM Products  WHERE UnitPrice>(SELECT AVG(UnitPrice) FROM Products) |

We can query the view above as follows:

|  |
| --- |
| SELECT \* FROM [Products Above Average Price] |

Another example view from the Northwind database calculates the total sale for each category in 1997. Note that this view selects its data from another view called "Product Sales for 1997":

|  |
| --- |
| CREATE VIEW [Category Sales For 1997] AS  SELECT DISTINCT CategoryName,Sum(ProductSales) AS CategorySales  FROM [Product Sales for 1997]  GROUP BY CategoryName |

We can query the view above as follows:

|  |
| --- |
| SELECT \* FROM [Category Sales For 1997] |

We can also add a condition to the query. Now we want to see the total sale only for the category "Beverages":

|  |
| --- |
| SELECT \* FROM [Category Sales For 1997]  WHERE CategoryName='Beverages' |

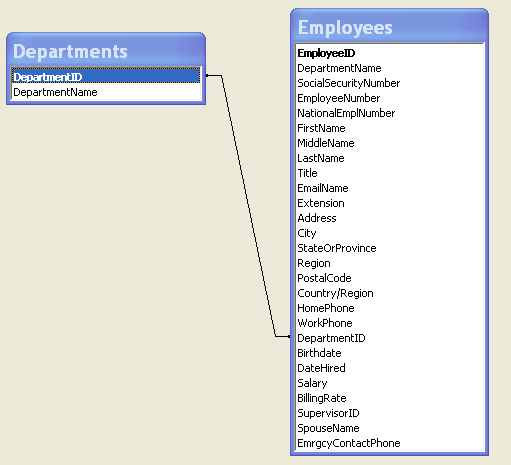
# SQL Quick Reference

SQL Quick Reference from W3Schools. Print it, and fold it in your pocket.

### SQL Syntax

|  |  |
| --- | --- |
| **Statement** | **Syntax** |
| AND / OR | SELECT column\_name(s) FROM table\_name WHERE condition AND|OR condition |
| ALTER TABLE (add column) | ALTER TABLE table\_name  ADD column\_name datatype |
| ALTER TABLE (drop column) | ALTER TABLE table\_name  DROP COLUMN column\_name |
| AS (alias for column) | SELECT column\_name AS column\_alias FROM table\_name |
| AS (alias for table) | SELECT column\_name FROM table\_name  AS table\_alias |
| BETWEEN | SELECT column\_name(s) FROM table\_name WHERE column\_name BETWEEN value1 AND value2 |
| CREATE DATABASE | CREATE DATABASE database\_name |
| CREATE INDEX | CREATE INDEX index\_name ON table\_name (column\_name) |
| CREATE TABLE | CREATE TABLE table\_name ( column\_name1 data\_type, column\_name2 data\_type, ....... ) |
| CREATE UNIQUE INDEX | CREATE UNIQUE INDEX index\_name ON table\_name (column\_name) |
| CREATE VIEW | CREATE VIEW view\_name AS SELECT column\_name(s) FROM table\_name WHERE condition |
| DELETE FROM | DELETE FROM table\_name  (**Note:** Deletes the entire table!!)  *or*  DELETE FROM table\_name WHERE condition |
| DROP DATABASE | DROP DATABASE database\_name |
| DROP INDEX | DROP INDEX table\_name.index\_name |
| DROP TABLE | DROP TABLE table\_name |
| GROUP BY | SELECT column\_name1,SUM(column\_name2) FROM table\_name GROUP BY column\_name1 |
| HAVING | SELECT column\_name1,SUM(column\_name2) FROM table\_name GROUP BY column\_name1 HAVING SUM(column\_name2) condition value |
| IN | SELECT column\_name(s) FROM table\_name WHERE column\_name IN (value1,value2,..) |
| INSERT INTO | INSERT INTO table\_name VALUES (value1, value2,....)  *or*  INSERT INTO table\_name (column\_name1, column\_name2,...) VALUES (value1, value2,....) |
| LIKE | SELECT column\_name(s) FROM table\_name WHERE column\_name LIKE pattern |
| ORDER BY | SELECT column\_name(s) FROM table\_name ORDER BY column\_name [ASC|DESC] |
| SELECT | SELECT column\_name(s) FROM table\_name |
| SELECT \* | SELECT \* FROM table\_name |
| SELECT DISTINCT | SELECT DISTINCT column\_name(s) FROM table\_name |
| SELECT INTO (used to create backup copies of tables) | SELECT \* INTO new\_table\_name FROM original\_table\_name  *or*  SELECT column\_name(s) INTO new\_table\_name FROM original\_table\_name |
| TRUNCATE TABLE (deletes only the data inside the table) | TRUNCATE TABLE table\_name |
| UPDATE | UPDATE table\_name SET column\_name=new\_value [, column\_name=new\_value] WHERE column\_name=some\_value |
| WHERE | SELECT column\_name(s) FROM table\_name WHERE condition |

============= Test ==============================



1. Find all tables.

SQL> SELECT \* FROM user\_tables;

1. Write a query to return the number of records in employee table. OR Find the number of records in employee table.

SQL> SELECT COUNT (\*) FROM employees;

SQL> SELECT COUNT (\*) TOTST FROM employees;

SQL> SELECT COUNT (\*) TOTSTPMNT FROM employees;

## To Find 2nd Highest Salary or Nth Highest Salary and it's working process.

**select MAX(Salary) from Employee;**

**Solution to finding the 2nd highest salary in SQL**:

SELECT MAX(Salary) FROM Employee

WHERE Salary NOT IN (SELECT MAX(Salary) FROM Employee)

## Suppose that you are given the following simple database table called Employee that has 2 columns named Employee ID and Salary:

|  |  |
| --- | --- |
| Employee | |
| Employee ID | Salary |
| 3 | 200 |
| 4 | 800 |
| 7 | 450 |

## Write a SQL query to get the second highest salary from the table above. Also write a query to find the nth highest salary in SQL, where n can be any number.

The easiest way to [start](http://www.programmerinterview.com/index.php/database-sql/find-nth-highest-salary-sql/) with a problem like this is to ask yourself a simpler question first. So, let’s ask ourselves how can we find the ***highest*** salary in a table? Well, you probably know that is actually really easy – we can just use the MAX aggregate function:

## An alternative solution using the not equals SQL operator

We can actually use the not equals operator – the “<>” – instead of the NOT IN operator as an alternative solution to this problem. This is what the SQL would look like:

select MAX(Salary) from Employee

WHERE Salary <> (select MAX(Salary) from Employee )

## How would you write a SQL query to find the Nth highest salary?

# [How to find top three highest salary in emp table in oracle?](http://stackoverflow.com/questions/2943429/how-to-find-top-three-highest-salary-in-emp-table-in-oracle)

# [How to get top 5 salary data in sql query](http://www.codeproject.com/Questions/395289/How-to-get-top-salary-data-in-sql-query)

SELECT \*FROM

(

SELECT \*FROM emp

ORDER BY Salary desc

)

WHERE rownum <= 3

ORDER BY Salary ;

== OR ===

SELECT \* FROM

(

SELECT EMPLOYEE, LAST\_NAME, SALARY,

RANK() OVER (ORDER BY SALARY DESC) EMPRANK

FROM emp

)

WHERE emprank <= 3;

Select ename, job, sal from emp

where sal >=(select max(sal) from emp

where sal < (select max(sal) from emp

where sal < (select max(sal) from emp)))

order by sal;

ENAME JOB SAL

---------- --------- ----------

KING PRESIDENT 5000

FORD ANALYST 3000

[SCOTT](http://stackoverflow.com/questions/2943429/how-to-find-top-three-highest-salary-in-emp-table-in-oracle) ANALYST 3000

SELECT \*

FROM table

WHERE

(

sal IN

(

SELECT TOP (5) sal

FROM table as table1

GROUP BY sal

ORDER BY sal DESC

)

)

SELECT DISTINCT TOP 5 salary

FROM employee

ORDER BY salary DESC

select \*

from scott.employee X

where salary in

(select top 5 salary

from scott.employee

where dept = X.dept

order by salary desc)

order by dept, salary desc

Ways to compare and find differences for SQL Server tables and data

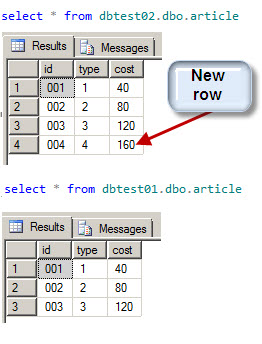
How do you compare two SQL tables?

##### Problem

Sometimes we need to compare tables and/or data to know what was changed. This tip shows you different ways to compare data, dataTypes and tables.

##### Solution

Show you different methods to identify changes. Let's say that we have two similar tables in different databases and we want to know what is different:



Let's look at ways we can compare these tables using different methods.

### Compare Tables Using a LEFT JOIN

With the left join we can compare values of specific columns that are not common between two tables.

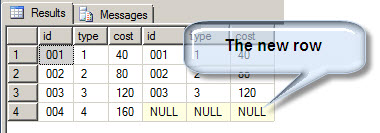
For example:

select \*

from dbtest02.dbo.article d2

left join dbtest01.dbo.article d1 on d2.id=d1.id

The left join shows all rows from the left table "dbtest02.dbo.article", even if there are no matches in the "dbtest01.dbo.article":



In this example, we are comparing 2 tables and the values of NULL are displayed if there are no matching rows. This method works to verify new rows, but if we update other columns, the left join does not help. Is there another method to compare tables?  Let's use the Except clause to see what we can find.

### Compare Tables Using the EXCEPT Clause

The Except method shows the difference between two tables (the Oracle guys use minus instead of except and the syntax and use is the same). It is used to compare the differences between two tables. For example, let's see the differences between the two tables:

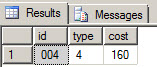
Now let's run a query using except:

select \* from dbtest02.dbo.article

except

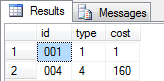
select \* from dbtest01.dbo.article

The except returns the difference between the tables from dbtest02 and dbtest01:



If we flip the tables around in the query we won't see any records, because the table in database dbtest02 has all of the records plus one extra.

This method is better than the first one, because if we change values for other columns like the type and cost, the except will notice the difference. Here is an example if we update id "001" in database dbtest01 and change the cost from "40" to "1".  If we update the records and then run the query again we will see these differentness now:



Unfortunately it does not create a script to synchronize the tables. Is there a way to compare tables and synchronize results?