**SQL (Structured Query Language)**

SQL is a [special-purpose programming language](https://en.wikipedia.org/wiki/Special-purpose_programming_language) designed for managing data held in a [relational database management system](https://en.wikipedia.org/wiki/Relational_database_management_system) (RDBMS).

## SQL General Data Types

Each column in a database table is required to have a name and a data type.

SQL developers have to decide what types of data will be stored inside each and every table column when creating a SQL table. The data type is a label and a guideline for SQL to understand what type of data is expected inside of each column, and it also identifies how SQL will interact with the stored data.

The following table lists the general data types in SQL:

|  |  |
| --- | --- |
| **Data type** | **Description** |
| CHARACTER(n) | Character string. Fixed-length n |
| VARCHAR(n) or CHARACTER VARYING(n) | Character string. Variable length. Maximum length n |
| BINARY(n) | Binary string. Fixed-length n |
| BOOLEAN | Stores TRUE or FALSE values |
| VARBINARY(n) or BINARY VARYING(n) | Binary string. Variable length. Maximum length n |
| INTEGER(p) | Integer numerical (no decimal). Precision p |
| SMALLINT | Integer numerical (no decimal). Precision 5 |
| INTEGER | Integer numerical (no decimal). Precision 10 |
| BIGINT | Integer numerical (no decimal). Precision 19 |
| DECIMAL(p,s) | Exact numerical, precision p, scale s. Example: decimal(5,2) is a number that has 3 digits before the decimal and 2 digits after the decimal |
| NUMERIC(p,s) | Exact numerical, precision p, scale s. (Same as DECIMAL) |
| FLOAT(p) | Approximate numerical, mantissa precision p. A floating number in base 10 exponential notation. The size argument for this type consists of a single number specifying the minimum precision |
| REAL | Approximate numerical, mantissa precision 7 |
| FLOAT | Approximate numerical, mantissa precision 16 |
| DOUBLE PRECISION | Approximate numerical, mantissa precision 16 |
| DATE | Stores year, month, and day values |
| TIME | Stores hour, minute, and second values |
| TIMESTAMP | Stores year, month, day, hour, minute, and second values |
| INTERVAL | Composed of a number of integer fields, representing a period of time, depending on the type of interval |
| ARRAY | A set-length and ordered collection of elements |
| MULTISET | A variable-length and unordered collection of elements |
| XML | Stores XML data |

## SQL Data Type Quick Reference

However, different databases offer different choices for the data type definition.

The following table shows some of the common names of data types between the various database platforms:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Data type** | **Access** | **SQLServer** | **Oracle** | **MySQL** | **PostgreSQL** |
| *boolean* | Yes/No | Bit | Byte | N/A | Boolean |
| *integer* | Number (integer) | Int | Number | Int Integer | Int Integer |
| *float* | Number (single) | Float Real | Number | Float | Numeric |
| *currency* | Currency | Money | N/A | N/A | Money |
| *string (fixed)* | N/A | Char | Char | Char | Char |
| *string (variable)* | Text (<256) Memo (65k+) | Varchar | Varchar Varchar2 | Varchar | Varchar |
| *binary object* | OLE Object Memo | Binary (fixed up to 8K) Varbinary (<8K) Image (<2GB) | Long Raw | Blob Text | Binary Varbinary |

# DDL

DDL is abbreviation of **Data Definition Language**. It is used to create and modify the structure of database objects in database.

**CREATE** – Creates objects in the database

CREATE TABLE table\_name  
(  
column\_name1 data\_type(size),  
column\_name2 data\_type(size),  
column\_name3 data\_type(size),  
....  
);

Ex: CREATE TABLE Persons  
(  
PersonIDint,  
LastName varchar(255),  
FirstName varchar(255),  
Address varchar(255),  
City varchar(255)  
);

**ALTER** – Alters objects of the database

To add a column in a table, use the following syntax:

ALTER TABLE table\_name  
ADD column\_name datatype

Ex: ALTER TABLE Persons ADD DateOfBirth date

To delete a column in a table, use the following syntax (notice that some database systems don't allow deleting a column):

ALTER TABLE table\_name  
DROP COLUMN column\_name

To change the data type of a column in a table, use the following syntax:

*SQL Server / MS Access:*

ALTER TABLE table\_name  
ALTER COLUMN column\_name datatype

*My SQL / Oracle (prior version 10G):*

ALTER TABLE table\_name  
MODIFY COLUMN column\_name datatype

*Oracle 10G and later:*

ALTER TABLE table\_name  
MODIFY column\_name datatype

**DROP** – Deletes objects of the database

Indexes, tables, and databases can easily be deleted/removed with the DROP statement.

## *The DROP INDEX Statement:*

The DROP INDEX statement is used to delete an index in a table.

## *DROP INDEX Syntax for MS Access:*

DROP INDEX index\_name ON table\_name

## *DROP INDEX Syntax for MS SQL Server:*

DROP INDEX table\_name.index\_name

## *DROP INDEX Syntax for DB2/Oracle:*

DROP INDEX index\_name

## *DROP INDEX Syntax for MySQL:*

ALTER TABLE table\_name DROP INDEX index\_name

## *The DROP TABLE Statement*

The DROP TABLE statement is used to delete a table.

DROP TABLE table\_name

## *The DROP DATABASE Statement*

The DROP DATABASE statement is used to delete a database.

DROP DATABASE database\_name

**TRUNCATE** – Deletes all records from a table and resets table identity to initial value. OR Remove all records from a table, including all spaces allocated for the records are removed

## *The TRUNCATE TABLE Statement*

What if we only want to delete the data inside the table, and not the table itself ?

Then, use the TRUNCATE TABLE statement:

TRUNCATE TABLE table\_name

**RENAME** - Rename an object

RENAME TABLE old\_table TO new\_table;

ALTER TABLE old\_table RENAME new\_table;

Ex: RENAME TABLE old\_table TO tmp\_table,

new\_table TO old\_table,

tmp\_table TO new\_table;

# DML

DML is abbreviation of **Data Manipulation Language**. It is used to retrieve, store, modify, delete, insert and update data in database.

SELECT – Retrieves data from a table

SELECT column\_name,column\_name  
FROM table\_name;

INSERT – Inserts data into a table

INSERT INTO table\_name (column1,column2,column3,...)  
VALUES (value1,value2,value3,...);

UPDATE – Updates existing data into a table

UPDATE table\_name  
SET column1=value1,column2=value2,...  
WHERE some\_column=some\_value;

DELETE – Deletes all records from a table, the space for the records remain (Delete row contains)

DELETE FROM table\_name  
WHERE some\_column=some\_value;

Ex: DELETE FROM Customers WHERE CustomerName='Alfreds' AND ContactName='Maria Anders';

MERGE - UPSERT operation (insert or update)

CALL - Call a PL/SQL or Java subprogram

EXPLAIN PLAN - Explain access path to data

LOCK TABLE - Control concurrency

# DCL

DCL is abbreviation of **Data Control Language**. It is used to create roles, permissions, and referential integrity as well it is used to control access to database by securing it.

**GRANT** – Gives user’s access privileges to database  
**REVOKE** – Withdraws user’s access privileges to database given with the GRANT command

## Grant Privileges on Table

You can grant users various privileges to tables. These privileges can be any combination of SELECT, INSERT, UPDATE, DELETE, REFERENCES, ALTER, INDEX, or ALL.

### Syntax

The syntax for granting privileges on a table in Oracle is:

GRANT privileges ON object TO user;

The privileges to assign. It can be any of the following values:

| **Privilege** | **Description** |
| --- | --- |
| SELECT | Ability to perform SELECT statements on the table. |
| INSERT | Ability to perform INSERT statements on the table. |
| UPDATE | Ability to perform UPDATE statements on the table. |
| DELETE | Ability to perform DELETE statements on the table. |
| REFERENCES | Ability to create a constraint that refers to the table. |
| ALTER | Ability to perform ALTER TABLE statements to change the table definition. |
| INDEX | Ability to create an index on the table with the create index statement. |
| ALL | All privileges on table. |

**object**

The name of the database object that you are granting privileges for. In the case of granting privileges on a table, this would be the table name.

**user**

The name of the user that will be granted these privileges.

### Example

Let's look at some examples of how to grant privileges on tables in Oracle.

For example, if you wanted to grant SELECT, INSERT, UPDATE, and DELETE privileges on a table called suppliers to a user name smith, you would run the following GRANT statement:

GRANT SELECT, INSERT, UPDATE, DELETE ON suppliers TO smith;

You can also use the ALL keyword to indicate that you wish ALL permissions to be granted for a user named smithj. For example:

GRANT ALL ON suppliers TO smithj;

If you wanted to grant only SELECT access on your table to all users, you could grant the privileges to the public keyword. For example:

GRANT SELECT ON suppliers TO public;

## Revoke Privileges on Table

Once you have granted privileges, you may need to revoke some or all of these privileges. To do this, you can run a revoke command. You can revoke any combination of SELECT, INSERT, UPDATE, DELETE, REFERENCES, ALTER, INDEX, or ALL.

### Syntax

The syntax for revoking privileges on a table in Oracle is:

REVOKE privileges ON object FROM user;

**privileges**

The privileges to revoke. It can be any of the following values:

| **Privilege** | **Description** |
| --- | --- |
| SELECT | Ability to perform SELECT statements on the table. |
| INSERT | Ability to perform INSERT statements on the table. |
| UPDATE | Ability to perform UPDATE statements on the table. |
| DELETE | Ability to perform DELETE statements on the table. |
| REFERENCES | Ability to create a constraint that refers to the table. |
| ALTER | Ability to perform ALTER TABLE statements to change the table definition. |
| INDEX | Ability to create an index on the table with the create index statement. |
| ALL | All privileges on table. |

**object**

The name of the database object that you are revoking privileges for. In the case of revoking privileges on a table, this would be the table name.

**user**

The name of the user that will have these privileges revoked.

### Example

Let's look at some examples of how to revoke privileges on tables in Oracle.

For example, if you wanted to revoke DELETE privileges on a table called suppliers from a user named anderson, you would run the following REVOKE statement:

REVOKE DELETE ON suppliers FROM anderson;

If you wanted to revoke ALL privileges on a table for a user named anderson, you could use the ALL keyword as follows:

REVOKE ALL ON suppliers FROM anderson;

If you had granted ALL privileges to public (all users) on the suppliers table and you wanted to revoke these privileges, you could run the following REVOKE statement:

REVOKE ALL ON suppliers FROM public;

## Grant Privileges on Functions/Procedures

When dealing with functions and procedures, you can grant users the ability to EXECUTE these functions and procedures.

### Syntax

The syntax for granting EXECUTE privileges on a function/procedure in Oracle is:

GRANT EXECUTE ON object TO user;

EXECUTE

The ability to compile the function/procedure. The ability to execute the function/procedure directly.

object

The name of the database object that you are granting privileges for. In the case of granting EXECUTE privileges on a function or procedure, this would be the function name or the procedure name.

user

The name of the user that will be granted the EXECUTE privileges.

### Example

Let's look at some examples of how to grant EXECUTE privileges on a function or procedure in Oracle.

For example, if you had a function called Find\_Value and you wanted to grant EXECUTE access to the user named smithj, you would run the following GRANT statement:

GRANT EXECUTE ON Find\_Value TO smithj;

If you wanted to grant ALL users the ability to EXECUTE this function, you would run the following GRANT statement:

GRANT EXECUTE ON Find\_Value TO public;

## Revoke Privileges on Functions/Procedures

Once you have granted EXECUTE privileges on a function or procedure, you may need to REVOKE these privileges from a user. To do this, you can execute a REVOKE command.

### Syntax

The syntax for the revoking privileges on a function or procedure in Oracle is:

REVOKE EXECUTE ON object FROM user;

EXECUTE

The ability to compile the function/procedure. The ability to execute the function/procedure directly.

object

The name of the database object that you are revoking privileges for. In the case of revoking EXECUTE privileges on a function or procedure, this would be the function name or the procedure name.

user

The name of the user that will be revoked the EXECUTE privileges.

### Example

Let's look at some examples of how to revoke EXECUTE privileges on a function or procedure in Oracle.

If you wanted to revoke EXECUTE privileges on a function called Find\_Value from a user named anderson, you would run the following REVOKE statement:

REVOKE execute ON Find\_Value FROM anderson;

If you had granted EXECUTE privileges to public (all users) on the function called Find\_Value and you wanted to revoke these EXECUTE privileges, you could run the following REVOKE statement:

REVOKE EXECUTE ON Find\_Value FROM public;

# TCL

TCL is abbreviation of **Transactional Control Language**. It is used to manage different transactions occurring within a database.

COMMIT – Saves work done in transactions  
ROLLBACK – Restores database to original state since the last COMMIT command in transactions

SAVEPOINT - Identify a point in a transaction to which you can later roll back

*Creating Savepoints*: Example To update the salary for Banda and Greene in the sample table hr.employees, check that the total department salary does not exceed 314,000, and then reenter the salary for Greene:

UPDATE employees

SET salary = 7000

WHERE last\_name = 'Banda';

SAVEPOINT banda\_sal;

UPDATE employees

SET salary = 12000

WHERE last\_name = 'Greene';

SAVEPOINT greene\_sal;

SELECT SUM(salary) FROM employees;

ROLLBACK TO SAVEPOINT greene\_sal;

UPDATE employees

SET salary = 11000

WHERE last\_name = 'Greene';

COMMIT;

**NOTE:**DML commands can't be rollback when a DDL command is executed immediately after a DML. DDL after DML means "auto commit". The changes will return on disk not on the buffer. If the changes return on the buffer it is possible to rollback not from the disk.

## The SQL LIKE Operator

The LIKE operator is used to search for a specified pattern in a column.

### SQL LIKE Syntax

SELECT column\_name(s)  
FROM table\_name  
WHERE column\_name LIKE pattern;

## SQL LIKE Operator Examples

The following SQL statement selects all customers with a City starting with the letter "s":

### Example

SELECT \* FROM Customers  
WHERE City LIKE 's%';

**Tip:** The "%" sign is used to define wildcards (missing letters) both before and after the pattern. You will learn more about wildcards in the next chapter.

The following SQL statement selects all customers with a City ending with the letter "s":

### Example

SELECT \* FROM Customers  
WHERE City LIKE '%s';

The following SQL statement selects all customers with a Country containing the pattern "land":

### Example

SELECT \* FROM Customers  
WHERE Country LIKE '%land%';

Using the NOT keyword allows you to select records that do NOT match the pattern.

The following SQL statement selects all customers with Country NOT containing the pattern "land":

### Example

SELECT \* FROM Customers  
WHERE Country NOT LIKE '%land%';

## SQL Wildcards

A wildcard character can be used to substitute for any other character(s) in a string.

## SQL Wildcard Characters

In SQL, wildcard characters are used with the SQL LIKE operator.

SQL wildcards are used to search for data within a table.

With SQL, the wildcards are:

|  |  |
| --- | --- |
| **Wildcard** | **Description** |
| % | A substitute for zero or more characters |
| \_ | A substitute for a single character |
| [charlist] | Sets and ranges of characters to match |
| [^charlist] or [!charlist] | Matches only a character NOT specified within the brackets |

## Using the SQL % Wildcard

The following SQL statement selects all customers with a City starting with "ber":

### Example

SELECT \* FROM Customers  
WHERE City LIKE 'ber%';

The following SQL statement selects all customers with a City containing the pattern "es":

### Example

SELECT \* FROM Customers  
WHERE City LIKE '%es%';

## Using the SQL \_ Wildcard

The following SQL statement selects all customers with a City starting with any character, followed by "erlin":

### Example

SELECT \* FROM Customers  
WHERE City LIKE '\_erlin';

The following SQL statement selects all customers with a City starting with "L", followed by any character, followed by "n", followed by any character, followed by "on":

### Example

SELECT \* FROM Customers  
WHERE City LIKE 'L\_n\_on';

## Using the SQL [charlist] Wildcard

The following SQL statement selects all customers with a City starting with "b", "s", or "p":

### Example

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

The following SQL statement selects all customers with a City starting with "a", "b", or "c":

### Example

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

The following SQL statement selects all customers with a City NOT starting with "b", "s", or "p":

### Example

SELECT \* FROM Customers  
WHERE City LIKE '[!bsp]%';  
  
or  
  
SELECT \* FROM Customers  
WHERE City NOT LIKE '[bsp]%';

**The SQL UNION Operator**

The UNION operator is used to combine the result-set of two or more SELECT statements.

Notice that each SELECT statement within the UNION must have the same number of columns. The columns must also have similar data types. Also, the columns in each SELECT statement must be in the same order.

**SQL UNION Syntax**

SELECT *column\_name(s)* FROM *table1*  
UNION  
SELECT *column\_name(s)* FROM *table2*;

**Note:**The UNION operator selects only distinct values by default. To allow duplicate values, use the ALL keyword with UNION.

**SQL UNION ALL Syntax**

SELECT *column\_name(s)* FROM *table1*  
UNION ALL  
SELECT *column\_name(s)* FROM *table2*;

**PS:** The column names in the result-set of a UNION are usually equal to the column names in the first SELECT statement in the UNION.

**Demo Database**

In this tutorial we will use the well-known Northwind sample database.

Below is a selection from the "Customers" table:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **CustomerID** | **CustomerName** | **ContactName** | **Address** | **City** | **PostalCode** | **Country** |
| 1 | AlfredsFutterkiste | Maria Anders | Obere Str. 57 | Berlin | 12209 | Germany |
| 2 | Ana Trujillo Emparedados y helados | Ana Trujillo | Avda. de la Constitución 2222 | México D.F. | 05021 | Mexico |
| 3 | Antonio Moreno Taquería | Antonio Moreno | Mataderos 2312 | México D.F. | 05023 | Mexico |

And a selection from the "Suppliers" table:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **SupplierID** | **SupplierName** | **ContactName** | **Address** | **City** | **PostalCode** | **Country** |
| 1 | Exotic Liquid | Charlotte Cooper | 49 Gilbert St. | London | EC1 4SD | UK |
| 2 | New Orleans Cajun Delights | Shelley Burke | P.O. Box 78934 | New Orleans | 70117 | USA |
| 3 | Grandma Kelly's Homestead | Regina Murphy | 707 Oxford Rd. | Ann Arbor | 48104 | USA |

**SQL UNION Example**

The following SQL statement selects all the **different** cities (only distinct values) from the "Customers" and the "Suppliers" tables:

**Example**

SELECT City FROM Customers  
UNION  
SELECT City FROM Suppliers  
ORDER BY City;

**Note:**UNION cannot be used to list ALL cities from the two tables. If several customers and suppliers share the same city, each city will only be listed once. UNION selects only distinct values. Use UNION ALL to also select duplicate values!

**SQL UNION ALL Example**

The following SQL statement uses UNION ALL to select **all** (duplicate values also) cities from the "Customers" and "Suppliers" tables:

**Example**

SELECT City FROM Customers  
UNION ALL  
SELECT City FROM Suppliers  
ORDER BY City;

**SQL UNION ALL With WHERE**

The following SQL statement uses UNION ALL to select **all** (duplicate values also) **German** cities from the "Customers" and "Suppliers" tables:

**Example**

SELECT City, Country FROM Customers  
WHERE Country='Germany'  
UNION ALL  
SELECT City, Country FROM Suppliers  
WHERE Country='Germany'  
ORDER BY City;

|  |
| --- |
|  |

**UNION**

The Oracle UNION operator is used to combine the result sets of 2 or more [Oracle SELECT statements](http://www.techonthenet.com/oracle/select.php). It removes duplicate rows between the various SELECT statements.

Each [SELECT statement](http://www.techonthenet.com/oracle/select.php) within the UNION operator must have the same number of fields in the result sets with similar data types.

**Syntax**

The syntax for the UNION operator in Oracle/PLSQL is:

SELECT expression1, expression2, ...expression\_n

FROM tables

[WHERE conditions]

UNION

SELECT expression1, expression2, ...expression\_n

FROM tables

[WHERE conditions];

**Parameters or Arguments**

expression1, expression2, expression\_n

The columns or calculations that you wish to retrieve.

tables

The tables that you wish to retrieve records from. There must be at least one table listed in the FROM clause.

WHERE conditions

Optional. The conditions that must be met for the records to be selected.

**Note**

* There must be same number of expressions in both SELECT statements.

**Example - Returns single field**

The following is an example of the Oracle UNION operator that returns one field from multiple SELECT statements (and both fields have the same data type):

SELECT supplier\_id

FROM suppliers

UNION

SELECT supplier\_id

FROM order\_details;

In this Oracle UNION operator example, if a *supplier\_id* appeared in both the *suppliers* and *order\_details* table, it would appear once in your result set. The Oracle UNION operator removes duplicates. If you do **not** wish to remove duplicates, try using the [Oracle UNION ALL operator](http://www.techonthenet.com/oracle/union_all.php).

**Example - Using ORDER BY**

The Oracle UNION operator can use the [ORDER BY clause](http://www.techonthenet.com/oracle/order_by.php) to order the results of the query.

For example:

SELECT supplier\_id, supplier\_name

FROM suppliers

WHERE supplier\_id<= 500

UNION

SELECT company\_id, company\_name

FROM companies

WHERE company\_name = 'Apple'

ORDER BY 2;

In this Oracle UNION operator, since the column names are different between the two [SELECT statements](http://www.techonthenet.com/oracle/select.php), it is more advantageous to reference the columns in the [ORDER BY clause](http://www.techonthenet.com/oracle/order_by.php) by their position in the result set. In this example, we've sorted the results by *supplier\_name* / *company\_name* in ascending order, as denoted by the ORDER BY 2.

The *supplier\_name* / *company\_name* fields are in position #2 in the result set.

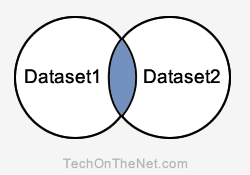
## INTERSECT Operator

This Oracle tutorial explains how to use the Oracle **INTERSECT operator** with syntax and examples.

## Description

The Oracle INTERSECT operator is used to return the results of 2 or more [SELECT statements](http://www.techonthenet.com/oracle/select.php). However, it only returns the rows selected by all queries or data sets. If a record exists in one query and not in the other, it will be omitted from the INTERSECT results.

### Intersect Query



**Explanation:** The INTERSECT query will return the records in the blue shaded area. These are the records that exist in both Dataset1 and Dataset2.

Each SELECT statement within the INTERSECT must have the same number of fields in the result sets with similar data types.

## Syntax

The syntax for the INTERSECT operator in Oracle/PLSQL is:

SELECT expression1, expression2, ...expression\_n

FROM tables

[WHERE conditions]

INTERSECT

SELECT expression1, expression2, ...expression\_n

FROM tables

[WHERE conditions];

### Parameters or Arguments

expression1, expression2, ...expression\_n

The columns or calculations that you wish to retrieve.

tables

The tables that you wish to retrieve records from. There must be at least one table listed in the FROM clause.

WHERE conditions

Optional. The conditions that must be met for the records to be selected.

## Note

* There must be same number of expressions in both SELECT statements and have similar data types.

## Example - With Single Expression

The following is an Oracle INTERSECT example that returns one field with the same data type:

SELECT supplier\_id

FROM suppliers

INTERSECT

SELECT supplier\_id

FROM orders;

In this INTERSECT example, if a supplier\_id appeared in both the suppliers and orders table, it would appear in your result set.

Now, let's complicate our example further by adding [WHERE conditions](http://www.techonthenet.com/oracle/where.php) to the INTERSECT query.

SELECT supplier\_id

FROM suppliers

WHERE supplier\_id<= 99

INTERSECT

SELECT supplier\_id

FROM orders

WHERE quantity > 25;

In this example, the WHERE clauses have been added to each of the datasets. The first dataset has been filtered so that only records from the suppliers table where the supplier\_id is less than or equal to 99 are returned. The second dataset has been filtered so that only records from the orders table are returned where the quantity is greater than 25.

## Example - With Multiple Expressions

Next, let's look at an example of how to use the INTERSECT operator in Oracle to return more than one column.

For example:

SELECT contact\_id, last\_name, first\_name

FROM contacts

WHERE first\_name<> 'John'

INTERSECT

SELECT customer\_id, last\_name, first\_name

FROM customers

WHERE customer\_id>= 89;

In this INTERSECT example, the query will return the records from the contacts table where the contact\_id, last\_name, and first\_name values match the customer\_id, last\_name, and first\_name value from the customers table.

There are WHERE conditions on each data set to further filter the results so that only records from the contacts are returned where the first\_name is not John. The records from the customers table are returned where the customer\_id is greater than or equal to 89.

## Example - Using ORDER BY

The following is an INTERSECT example that uses an [ORDER BY clause](http://www.techonthenet.com/oracle/order_by.php):

SELECT supplier\_id, supplier\_name

FROM suppliers

WHERE supplier\_id> 500

INTERSECT

SELECT company\_id, company\_name

FROM companies

WHERE company\_name in ( 'Apple', 'Microsoft', 'Oracle' )

ORDER BY 2;

Since the column names are different between the two SELECT statements, it is more advantageous to reference the columns in the ORDER BY clause by their position in the result set. In this example, we've sorted the results by supplier\_name / company\_name in ascending order, as denoted by the ORDER BY 2.

The supplier\_name / company\_name fields are in position #2 in the result set

## Using both ASC and DESC attributes

When sorting your result set using the Oracle ORDER BY clause, you can use the ASC and DESC attributes in a single [SELECT statement](http://www.techonthenet.com/oracle/select.php).

For example:

SELECT supplier\_city, supplier\_state

FROM suppliers

WHERE supplier\_name = 'Microsoft'

ORDER BY supplier\_city DESC, supplier\_state ASC;

This Oracle ORDER BY would return all records sorted by the supplier\_city field in descending order, with a secondary sort by supplier\_state in ascending order.

Note: If do not mention the ASC/DESC in ORDER BY clause, then default order is ASC.

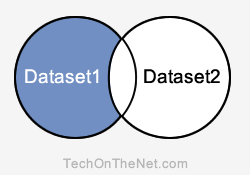
## MINUS Operator

This Oracle tutorial explains how to use the Oracle **MINUS operator** with syntax and examples.

## Description

The Oracle MINUS operator is used to return all rows in the first SELECT statement that are not returned by the second SELECT statement. Each SELECT statement will define a dataset. The MINUS operator will retrieve all records from the first dataset and then remove from the results all records from the second dataset.

### Minus Query



**Explanation:** The MINUS query will return the records in the blue shaded area. These are the records that exist in Dataset1 and not in Dataset2.

Each SELECT statement within the MINUS query must have the same number of fields in the result sets with similar data types.

## Syntax

The syntax for the MINUS operator in Oracle/PLSQL is:

SELECT expression1, expression2, ...expression\_n

FROM tables

[WHERE conditions]

MINUS

SELECT expression1, expression2, ...expression\_n

FROM tables

[WHERE conditions];

### Parameters or Arguments

expression1, expression2, ...expression\_n

The columns or calculations that you wish to retrieve.

tables

The tables that you wish to retrieve records from. There must be at least one table listed in the FROM clause.

WHERE conditions

Optional. The conditions that must be met for the records to be selected.

## Note

* There must be same number of expressions in both SELECT statements and have similar data types.

## Example - With Single Expression

The following is an Oracle MINUS operator example that returns one field with the same data type:

SELECT supplier\_id

FROM suppliers

MINUS

SELECT supplier\_id

FROM orders;

This Oracle MINUS example returns all supplier\_id values that are in the suppliers table and not in the orders table. What this means is that if a supplier\_id value existed in the suppliers table and also existed in the orders table, the supplier\_id value would not appear in this result set.

## Example - Using ORDER BY

The following is a MINUS operator example that uses an [ORDER BY clause](http://www.techonthenet.com/oracle/order_by.php):

SELECT supplier\_id, supplier\_name

FROM suppliers

WHERE state = 'Florida'

MINUS

SELECT company\_id, company\_name

FROM companies

WHERE company\_id<= 400

ORDER BY 2;

In this MINUS example, since the column names are different between the two SELECT statements, it is more advantageous to reference the columns in the ORDER BY clause by their position in the result set. In this example, we've sorted the results by supplier\_name / company\_name in ascending order, as denoted by the ORDER BY 2.

The supplier\_name / company\_name fields are in position #2 in the result set.

## Subqueries

This Oracle tutorial explains how to use Oracle **subqueries** with syntax and examples.

## What is a subquery in Oracle?

In Oracle, a subquery is a query within a query. You can create subqueries within your SQL statements. These subqueries can reside in the WHERE clause, the FROM clause, or the SELECT clause.

## WHERE clause

Most often, the subquery will be found in the WHERE clause. These subqueries are also called nested subqueries.

For example:

SELECT \*

FROM all\_tables tabs

WHERE tabs.table\_name IN (SELECT cols.table\_name

FROM all\_tab\_columns cols

WHERE cols.column\_name = 'SUPPLIER\_ID');

**Limitation:** Oracle allows up to 255 levels of subqueries in the WHERE clause.

## FROM clause

A subquery can also be found in the FROM clause. These are called **inline views**.

For example:

SELECT suppliers.name, subquery1.total\_amt

FROM suppliers,

(SELECT supplier\_id, SUM(orders.amount) AS total\_amt

FROM orders

GROUP BY supplier\_id) subquery1

WHERE subquery1.supplier\_id = suppliers.supplier\_id;

In this example, we've created a subquery in the FROM clause as follows:

(SELECT supplier\_id, SUM(orders.amount) AS total\_amt

FROM orders

GROUP BY supplier\_id) subquery1

This subquery has been aliased with the name subquery1. This will be the name used to reference this subquery or any of its fields.

### Limitations

Oracle allows an unlimited number of subqueries in the FROM clause.

## SELECT clause

A subquery can also be found in the SELECT clause.

For example:

SELECT tbls.owner, tbls.table\_name,

(SELECT COUNT(column\_name) AS total\_columns

FROM all\_tab\_columns cols

WHERE cols.owner = tbls.owner

AND cols.table\_name = tbls.table\_name) subquery2

FROM all\_tablestbls;

In this example, we've created a subquery in the SELECT clause as follows:

(SELECT COUNT(column\_name) AS total\_columns

FROM all\_tab\_columns cols

WHERE cols.owner = tbls.owner

AND cols.table\_name = tbls.table\_name) subquery2

The subquery has been aliased with the name subquery2. This will be the name used to reference this subquery or any of its fields.

The trick to placing a subquery in the select clause is that the subquery must return a single value. This is why an aggregate function such as [SUM function](http://www.techonthenet.com/oracle/functions/sum.php), [COUNT function](http://www.techonthenet.com/oracle/functions/count.php), [MIN function](http://www.techonthenet.com/oracle/functions/min.php), or [MAX function](http://www.techonthenet.com/oracle/functions/max.php) is commonly used in the subquery.

**SQL JOIN**

An SQL JOIN clause is used to combine rows from two or more tables, based on a common field between them.

The most common type of join is: **SQL INNER JOIN (simple join)**. An SQL INNER JOIN returns all rows from multiple tables where the join **condition is met**.

Let's look at a selection from the "Orders" table:

|  |  |  |
| --- | --- | --- |
| **OrderID** | **CustomerID** | **OrderDate** |
| 10308 | 2 | 1996-09-18 |
| 10309 | 37 | 1996-09-19 |
| 10310 | 77 | 1996-09-20 |

Then, have a look at a selection from the "Customers" table:

|  |  |  |  |
| --- | --- | --- | --- |
| **CustomerID** | **CustomerName** | **ContactName** | **Country** |
| 1 | AlfredsFutterkiste | Maria Anders | Germany |
| 2 | Ana Trujillo Emparedados y helados | Ana Trujillo | Mexico |
| 3 | Antonio Moreno Taquería | Antonio Moreno | Mexico |

Notice that the "CustomerID" column in the "Orders" table refers to the "CustomerID" in the "Customers" table. The relationship between the two tables above is the "CustomerID" column.

Then, if we run the following SQL statement (that contains an INNER JOIN):

**Example**

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

it will produce something like this:

|  |  |  |
| --- | --- | --- |
| **OrderID** | **CustomerName** | **OrderDate** |
| 10308 | Ana Trujillo Emparedados y helados | 9/18/1996 |
| 10365 | Antonio Moreno Taquería | 11/27/1996 |
| 10383 | Around the Horn | 12/16/1996 |
| 10355 | Around the Horn | 11/15/1996 |
| 10278 | Berglundssnabbköp | 8/12/1996 |

**Different SQL JOINs**

Before we continue with examples, we will list the types of the different SQL JOINs you can use:

* **INNER JOIN**: Returns all rows when there is at least one match in BOTH tables
* **LEFT JOIN**: Return all rows from the left table, and the matched rows from the right table
* **RIGHT JOIN**: Return all rows from the right table, and the matched rows from the left table
* **FULL JOIN**: Return all rows when there is a match in ONE of the tables

## SQL INNER JOIN Keyword

The INNER JOIN keyword selects all rows from both tables as long as there is a match between the columns in both tables.

### SQL INNER JOIN Syntax

SELECT column\_name(s)  
FROM table1  
INNER JOIN table2  
ON table1.column\_name=table2.column\_name;

or:

SELECT column\_name(s)  
FROM table1  
JOIN table2  
ON table1.column\_name=table2.column\_name;

**PS!** INNER JOIN is the same as JOIN.



## Demo Database

In this tutorial we will use the well-known Northwind sample database.

Below is a selection from the "Customers" table:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **CustomerID** | **CustomerName** | **ContactName** | **Address** | **City** | **PostalCode** | **Country** |
| 1 | AlfredsFutterkiste | Maria Anders | Obere Str. 57 | Berlin | 12209 | Germany |
| 2 | Ana Trujillo Emparedados y helados | Ana Trujillo | Avda. de la Constitución 2222 | México D.F. | 05021 | Mexico |
| 3 | Antonio Moreno Taquería | Antonio Moreno | Mataderos 2312 | México D.F. | 05023 | Mexico |

And a selection from the "Orders" table:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **OrderID** | **CustomerID** | **EmployeeID** | **OrderDate** | **ShipperID** |
| 10308 | 2 | 7 | 1996-09-18 | 3 |
| 10309 | 37 | 3 | 1996-09-19 | 1 |
| 10310 | 77 | 8 | 1996-09-20 | 2 |

## SQL INNER JOIN Example

The following SQL statement will return all customers with orders:

### Example

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

**Note:**The INNER JOIN keyword selects all rows from both tables as long as there is a match between the columns. If there are rows in the "Customers" table that do not have matches in "Orders", these customers will NOT be listed.

## SQL LEFT JOIN Keyword

The LEFT JOIN keyword returns all rows from the left table (table1), with the matching rows in the right table (table2). The result is NULL in the right side when there is no match.

### SQL LEFT JOIN Syntax

SELECT column\_name(s)  
FROM table1  
LEFT JOIN table2  
ON table1.column\_name=table2.column\_name;

or:

SELECT column\_name(s)  
FROM table1  
LEFT OUTER JOIN table2  
ON table1.column\_name=table2.column\_name;

**PS!** In some databases LEFT JOIN is called LEFT OUTER JOIN.



## Demo Database

In this tutorial we will use the well-known Northwind sample database.

Below is a selection from the "Customers" table:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **CustomerID** | **CustomerName** | **ContactName** | **Address** | **City** | **PostalCode** | **Country** |
| 1 | AlfredsFutterkiste | Maria Anders | Obere Str. 57 | Berlin | 12209 | Germany |
| 2 | Ana Trujillo Emparedados y helados | Ana Trujillo | Avda. de la Constitución 2222 | México D.F. | 05021 | Mexico |
| 3 | Antonio Moreno Taquería | Antonio Moreno | Mataderos 2312 | México D.F. | 05023 | Mexico |

And a selection from the "Orders" table:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **OrderID** | **CustomerID** | **EmployeeID** | **OrderDate** | **ShipperID** |
| 10308 | 2 | 7 | 1996-09-18 | 3 |
| 10309 | 37 | 3 | 1996-09-19 | 1 |
| 10310 | 77 | 8 | 1996-09-20 | 2 |

## SQL LEFT JOIN Example

The following SQL statement will return all customers, and any orders they might have:

### Example

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

**Note:**The LEFT JOIN keyword returns all the rows from the left table (Customers), even if there are no matches in the right table (Orders).

## SQL RIGHT JOIN Keyword

The RIGHT JOIN keyword returns all rows from the right table (table2), with the matching rows in the left table (table1). The result is NULL in the left side when there is no match.

### SQL RIGHT JOIN Syntax

SELECT column\_name(s)  
FROM table1  
RIGHT JOIN table2  
ON table1.column\_name=table2.column\_name;

or:

SELECT column\_name(s)  
FROM table1  
RIGHT OUTER JOIN table2  
ON table1.column\_name=table2.column\_name;

**PS!** In some databases RIGHT JOIN is called RIGHT OUTER JOIN.



## Demo Database

In this tutorial we will use the well-known Northwind sample database.

Below is a selection from the "Orders" table:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **OrderID** | **CustomerID** | **EmployeeID** | **OrderDate** | **ShipperID** |
| 10308 | 2 | 7 | 1996-09-18 | 3 |
| 10309 | 37 | 3 | 1996-09-19 | 1 |
| 10310 | 77 | 8 | 1996-09-20 | 2 |

And a selection from the "Employees" table:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **EmployeeID** | **LastName** | **FirstName** | **BirthDate** | **Photo** | **Notes** |
| 1 | Davolio | Nancy | 12/8/1968 | EmpID1.pic | Education includes a BA in psychology..... |
| 2 | Fuller | Andrew | 2/19/1952 | EmpID2.pic | Andrew received his BTS commercial and.... |
| 3 | Leverling | Janet | 8/30/1963 | EmpID3.pic | Janet has a BS degree in chemistry.... |

## SQL RIGHT JOIN Example

The following SQL statement will return all employees, and any orders they have placed:

### Example

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

**Note:**The RIGHT JOIN keyword returns all the rows from the right table (Employees), even if there are no matches in the left table (Orders).

## SQL FULL OUTER JOIN Keyword

The FULL OUTER JOIN keyword returns all rows from the left table (table1) and from the right table (table2).

The FULL OUTER JOIN keyword combines the result of both LEFT and RIGHT joins.

### SQL FULL OUTER JOIN Syntax

SELECT column\_name(s)  
FROM table1  
FULL OUTER JOIN table2  
ON table1.column\_name=table2.column\_name;



## Demo Database

In this tutorial we will use the well-known Northwind sample database.

Below is a selection from the "Customers" table:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **CustomerID** | **CustomerName** | **ContactName** | **Address** | **City** | **PostalCode** | **Country** |
| 1 | AlfredsFutterkiste | Maria Anders | Obere Str. 57 | Berlin | 12209 | Germany |
| 2 | Ana Trujillo Emparedados y helados | Ana Trujillo | Avda. de la Constitución 2222 | México D.F. | 05021 | Mexico |
| 3 | Antonio Moreno Taquería | Antonio Moreno | Mataderos 2312 | México D.F. | 05023 | Mexico |

And a selection from the "Orders" table:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **OrderID** | **CustomerID** | **EmployeeID** | **OrderDate** | **ShipperID** |
| 10308 | 2 | 7 | 1996-09-18 | 3 |
| 10309 | 37 | 3 | 1996-09-19 | 1 |
| 10310 | 77 | 8 | 1996-09-20 | 2 |

## SQL FULL OUTER JOIN Example

The following SQL statement selects all customers, and all orders:

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

A selection from the result set may look like this:

|  |  |
| --- | --- |
| **CustomerName** | **OrderID** |
| AlfredsFutterkiste |  |
| Ana Trujillo Emparedados y helados | 10308 |
| Antonio Moreno Taquería | 10365 |
|  | 10382 |
|  | 10351 |

**Note:**The FULL OUTER JOIN keyword returns all the rows from the left table (Customers), and all the rows from the right table (Orders). If there are rows in "Customers" that do not have matches in "Orders", or if there are rows in "Orders" that do not have matches in "Customers", those rows will be listed as well.

**The SQL SELECT INTO Statement**

The SELECT INTO statement selects data from one table and inserts it into a new table.

**SQL SELECT INTO Syntax**

We can copy all columns into the new table:

SELECT \*  
INTO *newtable* [IN *externaldb*]  
FROM *table1;*

Or we can copy only the columns we want into the new table:

SELECT *column\_name(s)*  
INTO *newtable* [IN *externaldb*]  
FROM *table1;*

The new table will be created with the column-names and types as defined in the SELECT statement. You can apply new names using the AS clause.

**SQL SELECT INTO Examples**

Create a backup copy of Customers:

SELECT \*  
INTO CustomersBackup2013  
FROM Customers;

Use the IN clause to copy the table into another database:

SELECT \*  
INTO CustomersBackup2013 IN 'Backup.mdb'  
FROM Customers;

Copy only a few columns into the new table:

SELECT CustomerName, ContactName  
INTO CustomersBackup2013  
FROM Customers;

Copy only the German customers into the new table:

SELECT \*  
INTO CustomersBackup2013  
FROM Customers  
WHERE Country='Germany';

Copy data from more than one table into the new table:

SELECT Customers.CustomerName, Orders.OrderID  
INTO CustomersOrderBackup2013  
FROM Customers  
LEFT JOIN Orders  
ON Customers.CustomerID=Orders.CustomerID;

**Tip:** The SELECT INTO statement can also be used to create a new, empty table using the schema of another. Just add a WHERE clause that causes the query to return no data:

SELECT \*  
INTO *newtable*  
FROM *table1*  
WHERE 1=0;

**The SQL INSERT INTO SELECT Statement**

The INSERT INTO SELECT statement selects data from one table and inserts it into an existing table. Any existing rows in the target table are unaffected.

**SQL INSERT INTO SELECT Syntax**

We can copy all columns from one table to another, existing table:

INSERT INTO *table2*  
SELECT \* FROM *table1;*

Or we can copy only the columns we want to into another, existing table:

INSERT INTO *table2*  
*(column\_name(s))*  
SELECT *column\_name(s)*  
FROM *table1;*

**Demo Database**

In this tutorial we will use the well-known Northwind sample database.

Below is a selection from the "Customers" table:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **CustomerID** | **CustomerName** | **ContactName** | **Address** | **City** | **PostalCode** | **Country** |
| 1 | AlfredsFutterkiste | Maria Anders | Obere Str. 57 | Berlin | 12209 | Germany |
| 2 | Ana Trujillo Emparedados y helados | Ana Trujillo | Avda. de la Constitución 2222 | México D.F. | 05021 | Mexico |
| 3 | Antonio Moreno Taquería | Antonio Moreno | Mataderos 2312 | México D.F. | 05023 | Mexico |

And a selection from the "Suppliers" table:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **SupplierID** | **SupplierName** | **ContactName** | **Address** | **City** | **Postal Code** | **Country** | **Phone** |
| 1 | Exotic Liquid | Charlotte Cooper | 49 Gilbert St. | Londona | EC1 4SD | UK | (171) 555-2222 |
| 2 | New Orleans Cajun Delights | Shelley Burke | P.O. Box 78934 | New Orleans | 70117 | USA | (100) 555-4822 |
| 3 | Grandma Kelly's Homestead | Regina Murphy | 707 Oxford Rd. | Ann Arbor | 48104 | USA | (313) 555-5735 |

**SQL INSERT INTO SELECT Examples**

Copy only a few columns from "Suppliers" into "Customers":

**Example**

INSERT INTO Customers (CustomerName, Country)  
SELECT SupplierName, Country FROM Suppliers;

Copy only the German suppliers into "Customers":

**Example**

INSERT INTO Customers (CustomerName, Country)  
SELECT SupplierName, Country FROM Suppliers  
WHERE Country='Germany';