

Notes on SQL*

June 15, 2025

*Working From <https://www.sqltutorial.org/>

1 Getting Started

1.1 What Is SQL

SQL is a language used to interact with databases. Databases are exactly what you expect, they store data, and potentially structural information about that data's relations.

Two common types are

1. Relational Database Management Systems (RDBMS - PostgreSQL, MySQL, MariaDB, ...)
 - (a) Data is stored in Rows and Columns
 - (b) Essentially a large spreadsheet
2. Document Databases (No SQL)
 - (a) Data is stored as documents, no SQL

SQL stands for Structured Query Language, and is used to interface with relational database systems, altering data, gathering data, removing data etc. based on constraints. There are three main types of commands or parts,

1. DDL - Data Definition Language, database structure creation and modification
 - (a) `CREATE TABLE`
 - (b) `ALTER TABLE`
 - (c) `DROP TABLE`
2. DML - Data Manipulation Language, queries data in structures
 - (a) `SELECT`
 - (b) `INSERT`
 - (c) `UPDATE`
 - (d) `DELETE`
3. DCL - Data Control Language, handles authorization
 - (a) `GRANT`
 - (b) `REVOKE`

An RDBMS table consists of rows and columns, each column represents a field, and each row represents a record.

While some SQL commands are universal between the various database implementations and are mandated by the language standards there are plenty of variants with local tools and quirks.

1.2 SQL Syntax

In SQL you tell the language what you want, and the language works out how, a statement in SQL will often follow a pattern of [Verb - Subject - Condition]

A verb gives the action that you want the database to take, eg. `SELECT`, `INSERT`, `UPDATE`, or `DELETE`, with natural results.

The subject stipulates which database you want to work on, eg. a table.

The condition filters your statement down to only elements satisfying some condition.

An example statement would then be

```
SELECT first_name  
FROM employees  
WHERE YEAR(hire_date) = 1999;
```

SQL statements include tokens, the most important are keywords, which are reserved for SQL use. Another type is literals, as usual these are constants and declared, they can take the usual form of Strings, Numbers, Booleans, but also Times, and Timestamps as well as more.

Further there are identifiers which refer to objects such as tables columns and indexes.

Finally expressions can be combined from identifiers, literals and operators.

2 Selecting Data

2.1 SELECT

The basic syntax of SELECT follows the structure

```
SELECT
    select_list
FROM
    table_list;
```

where `select_list` is a comma separated list of columns to select and `table_list` is the table to select from. Naturally then the command is evaluated "backwards", where the table is grabbed first.

You can use `*` as a generic keyword for all available columns. Using `SELECT *` should in general be avoided for proper use since you should know what you want to grab.

An example on selecting information from a table named *employees* is

```
SELECT
    employee_id,
    first_name,
    last_name,
    hire_date
FROM
    employees;
```

You can perform calculations as part of a SELECT statement such as

```
SELECT
    salary,
    salary*1.05
FROM
    employees;
```

which will give you the salary and the salary increased by 5%. This creates a temporary column with a name depending on the implementation.

You can pick the name of this temporary column by using AS. EG.

```
SELECT
    salary,
    salary*1.05 AS new_salary
FROM
    employees;
```

3 Sorting Rows

3.1 SORT BY

To get the output in a certain order, for instance ordering employees by wage you can use ORDER BY to sort the output of a SELECT command. This type of command would look like

```
SELECT
    select_list
FROM
    table_list
ORDER BY
    sort_expression [ASC | DESC];
```

where you can use ASC or DESC to select an ascending or descending sorting. By default the sort uses ASC. If you wish to use a secondary sort you can sort first by one expression then by another using

```
SELECT
    select_list
FROM
    table_list
ORDER BY
    sort_expression_1 [ASC | DESC],
    sort_expression_2 [ASC | DESC];
```

For our employee example this could look like

```
SELECT
    first_name,
    last_name
FROM
    employees
ORDER BY
    salary DESC,
    hire_date DESC;
```

to get the employee names in order of descending salary, with ties broken by who was hired most recently.

You can terminate an ORDER BY clause with NULLS FIRST to bring any NULL values to the front, or NULLS LAST to send them to the back.

This could look like

```
SELECT
    first_name,
    last_name,
    salary
FROM
    employees
ORDER BY
    salary NULLS FIRST;
```

4 Limiting Rows

4.1 SQL DISTINCT

By adding DISTINCT to a SELECT command you select only entries with distinct entries in the first selected column. On our fake dataset set we can use

```
SELECT DISTINCT
  salary FROM
  employees
ORDER BY
  salary DESC;
```

which would give you a list of the salary levels that exist in the data set in descending order.

DISTINCT treats all NULL values as being identical / selects only one. (usually).

4.2 SQL LIMIT

If you want to grab only the first k elements in a list you can use LIMIT, and if you want to skip some number of rows at the beginning you can use OFFSET. Together this would look like

```
SELECT
  column_list
FROM
  table1
ORDER BY
  column_list DESC
LIMIT
  row_count
OFFSET
  row_to_skip;
```

4.3 SQL FETCH

LIMIT is not actually enforced by the SQL standard, but is widely supported. Modern SQL (post SQL:2008) standards include the OFFSET FETCH clause which behaves similarly. The syntax is

```
OFFSET rows_to_skip { ROW | ROWS }
FETCH { FIRST | NEXT } [ row_count ] { ROW | ROWS } ONLY
```

An example of this syntax would be

```
SELECT
  first_name,
  salary
FROM
  employees
ORDER BY
  salary DESC
FETCH FIRST 5 ROWS ONLY;
```

5 Filtering Data

5.1 WHERE

By adding a WHERE clause to the end of a statement you can select only rows satisfying some condition. In SQL it is possible for a condition to return NULL, true or false; WHERE only selects true evaluations.

WHERE can take a number of comparators / conditions including

Symbol	Comparator
=	Equal To
<>	Not Equal To
<	Less than
>	Greater than
<=	Less than equal to
>=	Greater than or equal

An example of how you could apply this is

```
SELECT
    first_name,
    salary
FROM
    employees
WHERE
    salary > 14000
ORDER BY
    salary DESC;
```

This will grab employee names and salaries for employees earning more than 14,000.

5.2 AND

An AND operator combines two boolean expressions that are being passed to another operator, only true and true combined will make true. and example of how AND is applied would be:

```
SELECT
    first_name,
    salary
FROM
    employees
WHERE
    salary > 14000
    AND job_id = 9
ORDER BY
    salary DESC;
```

SQL evaluates comparisons in order which means that you can order them to avoid eg. division by zero errors.

5.3 OR

OR acts just like AND but takes an or evaluation, if either condition is true, regardless of if the other is NULL. OR short circuits if the first is true which can be useful.

5.4 BETWEEN

The between operator acts as a shortcut for taking both a \leq and a \geq , the syntax is

```
SELECT
    first_name
FROM
    employees
WHERE salary BETWEEN 10000 AND 14000;
```

5.5 NOT

The NOT operator is the negation of a truth value.

5.6 IN

IN checks if a value is in a discrete set of values, the syntax is `expression IN (value1, value2, ...)`

5.7 LIKE

LIKE performs pattern matching, where the % character matches any length of free characters, the _ matches a single character. For instance then `LIKE 'Kim%'` would match any string beginning with Kim. You can choose the escape character freely using `ESCAPE escape_character`. Then any special character preceded by your escape character will be presented.

5.8 IS NULL

IS NULL returns true if a value is NULL, this is necessary because NULL is not equal to NULL.

6 Joining Tables

6.1 INNER JOIN

Using a join lets you poll two tables at once, an INNER JOIN selects and joins those elements who have a matching selected key across the two tables. So if you have for instance a table of hired employees and a table of fired employees you could take:

```
SELECT
    first_name
    last_name
FROM
    hiredemployees
INNER JOIN firedemployees ON hiredemployees.id = firedemployees.id;
```

By the use of table aliases we could shorten the above by renaming `hired_employees` to `h`, and `fired_employees` to `f` which would make the join `SELECT hired_employees h INNER JOIN fired_employees f ON h.id = f.id`

6.2 LEFT JOIN

A LEFT JOIN will take all elements from the primary table, and join in the additional columns from the extra table on a match condition. If the match condition isn't met, to fill in the new columns they are padded with null values.

6.3 RIGHT JOIN

The RIGHT JOIN behaves exactly as a LEFT JOIN but uses the tables in reverse order.

You can stack JOINS to merge as many table as needed, by simply applying them sequentially.

6.4 Self Joining

You can apply a JOIN between a table and itself to shuffle around elements according to eachother or do special selections. An example of this would be

```
SELECT
    e.first_name
    m.first_name
FROM
    employees e
LEFT JOIN employees m ON m.employee_id = e.manager_id;
ORDER BY
    manager NULLS FIRST;
```

6.5 FULL OUTER JOIN

A FULL OUTER JOIN takes any entries from either of the tables, merging those that match, and null padding those that don't.

6.6 CROSS JOIN

A CROSS JOIN merges every row in the first table with every table in the second table leading to an $n * m$ number of new rows.

7 Grouping Rows

7.1 GROUP BY

Suppose that you have a list that gives multiple rows that share identical entries in some column of interest. This could be for instance if many employees are employed by the same manager, if you then want to return the names of the managers whose employees meet some criteria, but don't want a large number of duplicate manager entries you can use GROUP BY.

The syntax is:

```
SELECT
    column
FROM
    table_name
GROUP BY
    column;
```

or for multiple columns and with some aggregating function,

```
SELECT
    column1
    column2
    aggregate_function (column3)
FROM
    table_name
GROUP BY
    column;
```

an example aggregate function here would be COUNT, which would give you an extra column detailing how many entries were grouped.

7.2 HAVING

HAVING allows you to filter on the results of a GROUP, if run without a group it behaves like a WHERE would.

7.3 GROUPING SETS

GROUPING SETS lets you construct multiple groupings in the same way as GROUP BY constructs one.

The syntax would be eg.

```
SELECT
    column1
    column2
    aggregate_function (column3)
FROM
    table_name
GROUP BY
    GROUPING SETS
        (column1),          (column1, column 2),      ();
```

7.4 ROLLUP

ROLLUP works much like a plain group by, but includes a hierarchy, where the results are presented subdivided by subgroups of column2 matches within column1 matches, then all column1 matches.

```
SELECT
    column1
    column2
    aggregate_function (column3)
FROM
    table_name
GROUP BY
    ROLLUP (column1, column2);
```

7.5 CUBE

CUBE behaves like a standard GROUP BY, but takes all possible combinations of subgroups between the columns presented. The syntax is:

```
SELECT
    column1
    column2
    aggregate_function (column3)
FROM
    table_name
GROUP BY
    CUBE (column1, column2);
```

8 Set Operations

8.1 UNION

A UNION operation lets you combine two select statements into one output so long as the SELECTs give output tables with the same number of columns, the syntax for UNION is:

```
SELECT
    column1
    column2
FROM
    table1
UNION
SELECT
    column3
    column4
FROM
    table2;
```

This kind of operation would let you for instance grab all the ids from two tables, or wages and names from two separate department tables. The output takes the column names from the final SELECT, UNION also removes any full duplicate rows, if this isn't the desired behaviour you can use UNION ALL.

8.2 INTERSECT

INTERSECT behaves like UNION but takes only the intersection between the two tables, ie. the rows that are present in both.

8.3 MINUS

MINUS allows you to take the entries of the first SELECT that don't exist in the second SELECT.