Introduction to Databases and SQL

In this tutorial, we'll learn about databases, different types of databases and their uses.

A database is an organized collection of data so that it can be easily accessed. To manage these databases, **Database Management Systems (DBMS)** are used.

Types of DBMS

In general, there are two common types of databases:

- Non-Relational
- Relational

Non-Relational Database Management System (Non-RDBMS)

In Non-RDBMS, data is stored in **key-value pairs**. For example:

customers

```
{
  "id":1,
  "name":"John",
  "age" : 25
}

{
  "id":2,
  "name":"Marry",
  "age" : 22
}
```

Example: Data stored in Non-RDBMS

Here, customers' data are stored in key-value pairs.

Commonly used Non-RDBMS: MongoDB, Amazon DynamoDB, Redis, etc.

Relational Database Management System (RDBMS)

In RDBMS, data is stored in **tabular format**. For example,

Table: customers

customer_id	first_name	last_name	phone	country
1	John	Doe	817-646-8833	USA
2	Robert	Luna	412-862-0502	USA
3	David	Robinson	208-340-7906	UK
4	John	Reinhardt	307-242-6285	UK
5	Betty	Taylor	806-749-2958	UAE

Example: Relational Database

Here, `customers` is a table inside the database.

The first row is the attributes of the table. Each row after that contains the data of a customer.

In RDBMS, two or more tables may be related to each other. Hence the term "**Relational**". For example,

Table: orders

order_id	product	total	customer_id
1	Paper	500	5
2	Pen	10	2
3	Marker	120	3
4	Books	1000	1
5	Erasers	20	4

Table: customers

customer_id	first_name	last_name	phone	country
1	John	Doe	817-646-8833	USA
2	Robert	Luna	412-862-0502	USA
3	David	Robinson	208-340-7906	UK
4	John	Reinhardt	307-242-6285	UK
5	Betty	Doe	806-749-2958	UAE

Example: Relationship in RDBMS

Here, orders and customers are related through `customer_id`.

Commonly used RDBMS: MySQL, PostgreSQL, MSSQL, Oracle etc.

> **Note:** To access data from these relational databases, **SQL (Structured Query Language)** is used.

Introduction to SQL

Structured Query Language (SQL) is a standard query language that is used to work with relational databases.

We use SQL to

- create databases
- create tables in a database
- read data from a table
- insert data in a table
- update data in a table
- delete data from a table
- delete database tables
- delete databases
- and many more database operations

SQL Example: Read Data From a Table

Let's take a look at an example,

```
SELECT first_name, last_name FROM Customers;
```

Run Code »

Here, this SQL command selects the first name and last name of all customers from the customers table.

Table: customers

customer_id	first_name	last_name	phone	country
1	John	Doe	817-646-8833	USA
2	Robert	Luna	412-862-0502	USA
3	David	Robinson	208-340-7906	UK
4	John	Reinhardt	307-242-6285	UK
5	Betty	Taylor	806-749-2958	UAE

Example: SQL SELECT Statement

SQL is used in all relational databases such as MySQL, Oracle, MSSQL, PostgreSQL etc.

> **Note:** The major SQL commands are similar in all relational databases. However, in some cases, SQL commands may differ.

In this SQL tutorial series, we will learn about SQL in detail. We will cover any SQL command differences among MySQL, Oracle, SQL Server, Postgres, and other commonly used database systems.

Next Tutorial: SQL SELECT Statement

Related Tutorials

Programming
SQL CREATE DATABASE Statement

Programming
SQL DROP DATABASE Statement

Programming
SQL SELECT INTO Statement

Programming
SQL CREATE TABLE Statement

SQL SELECT

In this tutorial, we'll learn about the SQL SELECT statement with the help of examples.

The SQL `SELECT` statement is used to select (retrieve) data from a database table. For example,

```
SELECT first_name, last_name
FROM Customers;
```

Run Code

Here, the SQL command selects the `first_name` and `last_name` of all `Customers`.

Example: SQL SELECT

SQL SELECT ALL

To select all columns from a database table, we use the `*` character. For example,

```
SELECT *
FROM Customers;
```

Run Code

Here, the SQL command selects all columns of the `Customers` table.

Example: SQL SELECT All

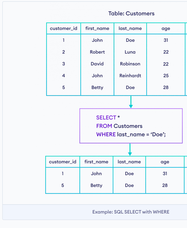
SQL SELECT WHERE Clause

A `SELECT` statement can have an optional `WHERE` clause. The `WHERE` clause allows us to fetch records from a database table that matches specified condition(s). For example,

```
SELECT *
FROM Customers
WHERE last_name = 'Doe';
```

Run Code

Here, the SQL command selects all customers from the `Customers` table with `last_name` **Doe**.



Example: SQL SELECT with WHERE

Let's see another example.

```
SELECT age, country
FROM Customers
WHERE country = 'USA';
```

Run Code

Here, the SQL command fetches `age` and `country` fields of all customers whose `country` is **USA**.

Example: SQL SELECT with WHERE

> **Note:** In SQL, we must enclose textual data inside either single or double quotations like `'USA'` in our example.

SQL Operators

The `WHERE` clause uses operators to construct conditions. Some of the commonly used operators are:

1. Equal to Operator (=)

```
SELECT *
FROM Customers
WHERE first_name = 'John';
```

Run Code

This SQL command selects all customers from the `Customers` table having `first_name` **John**.

2. Greater than (>)

```
SELECT *
FROM Customers
WHERE age > 25;
```

Run Code

This SQL command selects all customers from the `Customers` table having `age` **greater than 25**.

3. AND Operator (AND)

```
SELECT *
FROM Customers
WHERE last_name = 'Doe' AND country = 'USA';
```

Run Code

This SQL command selects all customers from the `Customers` table having `last_name` **Doe** and `country` **USA**.

> **Note:** If the `WHERE` clause condition does not meet with any rows, an empty result set is returned.

To learn more about all the SQL operators in detail, visit SQL Operators.

Previous Tutorial: SQL Introduction

Next Tutorial: SQL AND, OR, & NOT

Did you find this article helpful?

Related Tutorials

Programming — SQL SELECT INTO Statement

Programming — SQL AND, OR, and NOT Operators

Programming — SQL IN Operator

Programming — SQL LIMIT, TOP and FETCH FIRST

SQL AND, OR, and NOT Operators

In this tutorial, we'll learn to use the AND, OR, and NOT operators in SQL with the help of various examples.

The `AND` , `OR` and `NOT` operators in SQL are used with the `WHERE` or `HAVING` clauses.

SQL AND Operator

The SQL `AND` operator selects data if all conditions are `TRUE` . For example,

```
SELECT first_name, last_name
FROM Customers
WHERE country = 'USA' AND last_name = 'Doe';
```

Run Code »

Here, the SQL command selects `first_name` and `last_name` of all customers where the `country` is **USA** and `last_name` as **Doe** from the `Customers` table.

Table: Customers

customer_id	first_name	last_name	age	country
1	John	Doe	31	USA
2	Robert	Luna	22	USA
3	David	Robinson	22	UK
4	John	Reinhardt	25	UK
5	Betty	Doe	28	UAE

```
SELECT first_name, last_name
FROM Customers
WHERE country = 'USA' AND last_name = 'Doe';
```

first_name	last_name
John	Doe

Example: SQL AND Operator

SQL OR Operator

The SQL `OR` operator selects data if any one condition is `TRUE` . For example,

```
SELECT first_name, last_name
FROM Customers
WHERE country = 'USA' OR last_name = 'Doe';
```

Run Code »

Here, the SQL command selects `first_name` and `last_name` of all customers where the `country` is **USA** or if their last name is **Doe** from the `Customers` table.

Table: Customers

customer_id	first_name	last_name	age	country
1	John	Doe	31	USA
2	Robert	Luna	22	USA
3	David	Robinson	22	UK
4	John	Reinhardt	25	UK
5	Betty	Doe	28	UAE

```
SELECT first_name, last_name
FROM Customers
WHERE country = 'USA' OR last_name = 'Doe';
```

first_name	last_name
John	Doe
Robert	Luna
Betty	Doe

Example: SQL OR Operator

SQL NOT Operator

The SQL `NOT` operator selects data if the given condition is `FALSE` . For example

```
SELECT first_name, last_name
FROM Customers
WHERE NOT country = 'USA';
```

Run Code »

Here, the SQL command selects `first_name` and `last_name` of all customers where the `country` is not **USA** from the `Customers` table.

Table: Customers

customer_id	first_name	last_name	age	country
1	John	Doe	31	USA
2	Robert	Luna	22	USA
3	David	Robinson	22	UK
4	John	Reinhardt	25	UK
5	Betty	Doe	28	UAE

```
SELECT first_name, last_name
FROM Customers
WHERE NOT country = 'USA';
```

first_name	last_name
David	Robinson
John	Reinhardt
Betty	Doe

Example: SQL NOT Operator

Combining Multiple Operators

It is also possible to combine multiple `AND` , `OR` and `NOT` operators in an SQL statement. For example,

Let's suppose we want to select customers where the `country` is either **USA** or **UK**, and the `age` is **less than 26**.

```
SELECT *
FROM Customers
WHERE (country = 'USA' OR country = 'UK') AND age < 26;
```

Run Code »

Table: Customers

customer_id	first_name	last_name	age	country
1	John	Doe	31	USA
2	Robert	Luna	22	USA
3	David	Robinson	22	UK
4	John	Reinhardt	25	UK
5	Betty	Doe	28	UAE

```
SELECT *
FROM Customers
WHERE (country = 'USA' OR country = 'UK') AND age < 26;
```

customer_id	first_name	last_name	age	country
2	Robert	Luna	22	USA
3	David	Robinson	22	UK
4	John	Reinhardt	25	UK

Example: SQL AND and OR Operators

Let's take a look at another example

```
SELECT *
FROM Customers
WHERE NOT country = 'USA' AND NOT last_name = 'Doe';
```

Run Code »

Here, the SQL command selects all customers where the `country` is not **USA** and `last_name` is not **Doe** from the `Customers` table.

Table: Customers

customer_id	first_name	last_name	age	country
1	John	Doe	31	USA
2	Robert	Luna	22	USA
3	David	Robinson	22	UK
4	John	Reinhardt	25	UK
5	Betty	Doe	28	UAE

```
SELECT *
FROM Customers
WHERE NOT country = 'USA' AND NOT last_name = 'Doe';
```

customer_id	first_name	last_name	age	country
3	David	Robinson	22	UK
4	John	Reinhardt	25	UK

Example: SQL AND and NOT Operators

Previous Tutorial: SQL SELECT Statement

Next Tutorial: SQL SELECT DISTINCT

Related Tutorials

Programming — SQL SELECT

Programming — SQL IN Operator

Programming — SQL LIKE and NOT LIKE Operators

Programming — SQL LIMIT, TOP and FETCH FIRST

SQL SELECT DISTINCT Statement

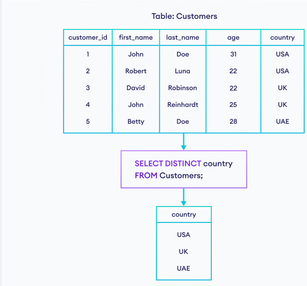
In this tutorial, you'll learn about the SQL DISTINCT clause and how to use it with the help of various examples.

The SQL `SELECT DISTINCT` statement selects unique rows from a database table. For example,

```
SELECT DISTINCT country
FROM Customers;
```

Run Code »

Here, the SQL command selects unique countries from the `Customers` table.



Example: Selecting unique countries

Let's see another example.

```
SELECT DISTINCT country, first_name
FROM Customers;
```

Run Code »

Here, the SQL command selects rows if the combination of `country` and `first_name` is unique.

Example: Selecting the unique combined fields

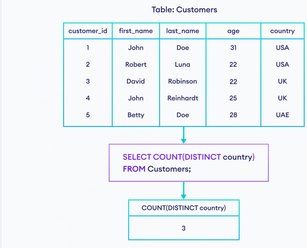
DISTINCT with COUNT

If we need to count the number of unique rows, we can use the `COUNT()` function with `DISTINCT`.

```
SELECT COUNT(DISTINCT country)
FROM Customers;
```

Run Code »

Here, the SQL command returns the count of unique countries.



Example: Counting unique countries

To learn more, visit SQL COUNT().

More SQL DISTINCT Examples

- DISTINCT Vs GROUP BY
- DISTINCT With ORDER BY

Previous Tutorial: SQL AND, OR, & NOT

Next Tutorial: SQL SELECT AS

Did you find this article helpful?

Related Tutorials

- Programming — SQL COUNT()
- Programming — SQL UNIQUE Constraint
- Programming — SQL SELECT
- Programming — SQL GROUP BY

SQL SELECT AS Alias

In this tutorial, we'll learn about SQL AS Alias with the help of examples.

SQL AS Alias

The `AS` keyword is used to give columns or tables a temporary name that can be used to identify that column or table later. For example,

```
SELECT first_name AS name
FROM Customers;
```

Run Code »

Here, the SQL command selects the `first_name` of `Customers`. However, its column name will be `name` instead of `first_name` in the result set.

Table: Customers

customer_id	first_name	last_name	age	country
1	John	Doe	31	USA
2	Robert	Luna	22	USA
3	David	Robinson	22	UK
4	John	Reinhardt	25	UK
5	Betty	Doe	28	UAE

```
SELECT first_name AS name
FROM Customers;
```

name
John
Robert
David
John
Betty

Example: SQL AS Alias

SQL AS With More Than One Column

We can also use aliases with more than one column. For example,

```
SELECT customer_id AS cid, first_name AS name
FROM Customers;
```

Run Code »

Here, the SQL command selects `customer_id` as `cid` and `first_name` as `name`.

SQL AS With Expression

We can combine data from multiple columns and represent data in a single column using the `CONCAT()` function. For example,

```
SELECT CONCAT(first_name, ' ', last_name) AS full_name
FROM Customers;
```

Here, the SQL command selects `first_name` and `last_name`. And, the name of the column will be `full_name` in the result set.

Table: Customers

customer_id	first_name	last_name	age	country
1	John	Doe	31	USA
2	Robert	Luna	22	USA
3	David	Robinson	22	UK
4	John	Reinhardt	25	UK
5	Betty	Doe	28	UAE

```
SELECT CONCAT(first_name, ' ', last_name) AS full_name
FROM Customers;
```

full_name
John Doe
Robert Luna
David Robinson
John Reinhardt
Betty Doe

Example: SQL AS Alias With Expression

More SQL AS Examples

Previous Tutorial: SQL SELECT DISTINCT

Next Tutorial: SQL LIMIT, TOP, FETCH FIRST

Did you find this article helpful?

Related Tutorials

Programming — SQL JOIN

Programming — SQL INNER JOIN

Programming — SQL RIGHT JOIN

Programming — SQL FULL OUTER JOIN

SQL LIMIT, TOP and FETCH FIRST

In this tutorial, we'll learn about the SQL LIMIT, TOP and FETCH FIRST constraints with the help of examples.

The `SELECT TOP` command is used to select a fixed number of rows from a database. For example,

```
SELECT TOP 2 *
FROM Customers;
```

Here, the SQL command selects the first **2** rows from the table.

Table: Customers

customer_id	first_name	last_name	age	country
1	John	Doe	31	USA
2	Robert	Luna	22	USA
3	David	Robinson	22	UK
4	John	Reinhardt	25	UK
5	Betty	Doe	28	UAE

```
SELECT TOP 2 *
FROM Customers;
```

customer_id	first_name	last_name	age	country
1	John	Doe	31	USA
2	Robert	Luna	22	USA

Example: SQL TOP Clause

Note: The `TOP` clause is not supported in all Database Management Systems (DBMS). Different DBMS use different keywords to select a fixed number of rows.

For example,

Keyword	Database System
`TOP`	SQL Server, MS Access
`LIMIT`	MySQL, PostgreSQL, SQLite
`FETCH FIRST`	Oracle

SQL LIMIT Clause

The `LIMIT` keyword is used with the following database systems:

- MySQL
- PostgreSQL
- SQLite

Let's see an example,

```
SELECT first_name, age
FROM Customers
LIMIT 2;
```

Run Code »

Here, the SQL command selects the first **2** rows from the table.

SQL LIMIT With OFFSET Clause

The `OFFSET` keyword is used to specify starting rows from where to select rows. For example,

```
SELECT first_name, last_name
FROM Customers
LIMIT 2 OFFSET 3;
```

Run Code »

Here, the SQL command selects **2** rows starting from the fourth row. `OFFSET 3` means the first **3** rows are excluded.

Table: Customers

customer_id	first_name	last_name	age	country
1	John	Doe	31	USA
2	Robert	Luna	22	USA
3	David	Robinson	22	UK
4	John	Reinhardt	25	UK
5	Betty	Doe	28	UAE

```
SELECT first_name, last_name
FROM Customers
LIMIT 2 OFFSET 3;
```

first_name	last_name
John	Reinhardt
Betty	Doe

Example: SQL LIMIT Clause with OFFSET

SQL TOP Clause

The `TOP` keyword is used with the following database systems:

- SQL Server
- MS Access

Let's see an example.

```
SELECT TOP 2 first_name, last_name
FROM Customers;
```

Here, the SQL command selects `first_name` and `last_name` of the first **2** rows.

SQL FETCH FIRST Clause

The `FETCH FIRST n ROWS ONLY` clause is used with the Oracle database system.

Let's see an example.

```
SELECT *
FROM Customers
FETCH FIRST 2 ROWS ONLY;
```

Here, the SQL command selects the first **2** rows from the table.

More SELECT TOP Examples

- PERCENT Clause With TOP
- WITH TIES Clause With TOP

Previous Tutorial: SQL SELECT AS

Next Tutorial: SQL IN

Did you find this article helpful?

Related Tutorials

Programming — SQL SELECT

Programming — SQL SELECT INTO Statement

Programming — SQL IN Operator

Programming — SQL INSERT INTO SELECT Statement

SQL IN Operator

In this tutorial, we'll learn about the SQL IN operator with the help of examples.

The `IN` operator is used with the WHERE clause to match values in a list. For example,

```
SELECT first_name, country
FROM Customers
WHERE country IN ('USA', 'UK');
```

Run Code »

Here, the SQL command selects rows if the `country` is either **USA** or **UK**.

Table: Customers

customer_id	first_name	last_name	age	country
1	John	Doe	31	USA
2	Robert	Luna	22	USA
3	David	Robinson	22	UK
4	John	Reinhardt	25	UK
5	Betty	Doe	28	UAE

```
SELECT first_name, country
FROM Customers
WHERE country IN ('USA', 'UK');
```

first_name	country
John	USA
Robert	USA
David	UK
John	UK

Example: SQL IN Operator

SQL IN Operator With Columns

The `IN` operator can also be used to select rows in which a certain value exists in the given field. Let's see an example to clarify it.

```
SELECT first_name, country
FROM Customers
WHERE 'USA' IN (country);
```

Run Code »

Here, the SQL command selects the rows if the **USA** value exists in the `country` field.

Table: Customers

customer_id	first_name	last_name	age	country
1	John	Doe	31	USA
2	Robert	Luna	22	USA
3	David	Robinson	22	UK
4	John	Reinhardt	25	UK
5	Betty	Doe	28	UAE

```
SELECT first_name, country
FROM Customers
WHERE USA IN (country);
```

first_name	country
John	USA
Robert	USA

Example: SQL IN Operator With Value

SQL NOT IN Operator

The `NOT IN` operator returns is used to exclude the rows that match values in the list. It returns all the rows except the excluded rows. For example,

```
SELECT first_name, country
FROM Customers
WHERE country NOT IN ('UK', 'UAE');
```

Run Code »

Here, the SQL command selects rows if **UK** or **UAE** is not in the `country` column.

Table: Customers

customer_id	first_name	last_name	age	country
1	John	Doe	31	USA
2	Robert	Luna	22	USA
3	David	Robinson	22	UK
4	John	Reinhardt	25	UK
5	Betty	Doe	28	UAE

```
SELECT first_name, country
FROM Customers
WHERE country NOT IN ('UK', 'UAE');
```

first_name	country
John	USA
Robert	USA

Example: SQL NOT IN Operator

> **Note:** The working of `IN` operator is reversed by `NOT` Operator. They are basically two operators combined. To learn more, visit SQL AND, OR, and NOT Operators.

SQL IN Operator With Duplicate Values

By the way, the `IN` operator ignores duplicate values in the list. For example,

This code

```
SELECT first_name, country
FROM Customers
WHERE country IN ('USA', 'UK', 'USA');
```

Run Code »

is equivalent to

```
SELECT first_name, country
FROM Customers
WHERE country IN ('USA', 'UK');
```

Run Code »

SQL IN Operator With Subquery

Suppose we want details of customers who have placed an order. Here's how we can do that using a subquery,

```
SELECT customer_id, first_name
FROM Customers
WHERE customer_id IN (
  SELECT customer_id
  FROM Orders
);
```

Run Code »

Here, the SQL command

1. selects `customer_id` from `Orders` table
2. select rows from `Customers` table where `customer_id` is in the result set of subquery

To learn more, visit SQL Subquery.

Previous Tutorial: SQL LIMIT, TOP, FETCH FIRST

Next Tutorial: SQL BETWEEN

Did you find this article helpful?

Related Topics
SQL SELECT
SQL SELECT INTO Statement
SQL Subquery
SQL INSERT INTO SELECT Statement
SQL AND, OR, and NOT Operators
SQL GROUP BY

Related Tutorials

Programming — SQL SELECT
Programming — SQL Subquery
Programming — SQL SELECT INTO Statement
Programming — SQL INSERT INTO SELECT Statement

SQL BETWEEN Operator

In this tutorial, we'll learn about the SQL BETWEEN operator with the help of examples.

The `BETWEEN` operator is used with the WHERE clause to match values in a range. For example,

```
SELECT item, amount
FROM Orders
WHERE amount BETWEEN 300 AND 500;
```

Run Code »

Here, the SQL command selects all orders that have amounts between **300** and **500** including **300** and **500**.

Table: Orders

order_id	item	amount	customer_id
1	Keyboard	400	4
2	Mouse	300	4
3	Monitor	12000	3
4	Keyboard	400	1
5	Mousepad	250	2

```
SELECT item, amount
FROM Orders
WHERE amount BETWEEN 300 AND 500;
```

item	amount
Keyboard	400
Mouse	300
Keyboard	400

Example: SQL BETWEEN Operator

SQL NOT BETWEEN Operator

The `NOT BETWEEN` operator is used to exclude the rows that match values in the range. It returns all the rows except the excluded rows. For example,

```
SELECT item, amount
FROM Orders
WHERE amount NOT BETWEEN 300 AND 500;
```

Run Code »

Here, the SQL command selects all orders except the rows that have amounts between **300** and **500**.

Table: Orders

order_id	item	amount	customer_id
1	Keyboard	400	4
2	Mouse	300	4
3	Monitor	12000	3
4	Keyboard	400	1
5	Mousepad	250	2

```
SELECT item, amount
FROM Orders
WHERE amount NOT BETWEEN 300 AND 500;
```

item	amount
Monitor	12000
Mousepad	250

Example: SQL NOT BETWEEN Operator

SQL BETWEEN OPERATOR With Texts

The `BETWEEN` operator also works with texts. For example,

```
SELECT item, amount
FROM Orders
WHERE item BETWEEN 'I' AND 'L';
```

Run Code »

Here, the SQL command selects all orders where the item name lies between **I** and **L**.

Table: Orders

order_id	item	amount	customer_id
1	Keyboard	400	4
2	Mouse	300	4
3	Monitor	12000	3
4	Keyboard	400	1
5	Mousepad	250	2

```
SELECT item, amount
FROM Orders
WHERE item BETWEEN 'I' AND 'L';
```

item	amount
Keyboard	400
Keyboard	400

Example: SQL BETWEEN Operator With Text

Here, the list of values that the above command selects that starts with **L**.

Text	Remarks
L	selects
Laptop	doesn't select
Lan Cable	doesn't select
Lamp	doesn't select

It's because **Laptop**, **Lan Cable** and **Lamp** do not lie between **I** and **L**.

If we need to include all the words that start with **L** as well, we can use `L~` like this.

```
SELECT item, amount
FROM Orders
WHERE item BETWEEN 'I' AND 'L~';
```

Run Code »

Let's take another example of `BETWEEN` with texts.

```
SELECT item
FROM Orders
WHERE item BETWEEN 'Key' AND 'Mou';
```

Run Code »

Here, the SQL command selects **Keyboard** and **Monitor**, but not **Mouse**. It's because **Mouse** appears after **Mou**.

Recommended Reading: SQL AND, OR, and NOT Operators

Previous Tutorial: SQL IN

Next Tutorial: SQL IS NULL and NOT NULL

Related Tutorials

Programming — SQL IN Operator

Programming — SQL Operators

Programming — SQL RIGHT JOIN

Programming — SQL JOIN

SQL IS NULL and IS NOT NULL

In this tutorial, we'll learn about the SQL IS NULL and NOT NULL with the help of examples.

The `IS NULL` condition is used to select rows if the specified field is NULL. For example,

```
SELECT *
FROM Employee
WHERE email IS NULL;
```

Run Code »

Here, the SQL command selects employees who do not have email.

Table: Employee

employee_id	first_name	last_name	department	email
1	Peter	Doe	Operations	peter@gmail.com
2	Megan	Morel	Finance	NULL
3	Rose	Bailey	Operations	rose@gmail.com
4	Linda	Bailey	Finance	NULL
5	Mary	Doe	Sales	NULL

```
SELECT *
FROM Employee
WHERE email IS NULL;
```

employee_id	first_name	last_name	department	email
2	Megan	Morel	Finance	NULL
4	Linda	Bailey	Finance	NULL
5	Mary	Doe	Sales	NULL

Example: IS NULL in SQL

> **Note:** Empty values are considered `NULL`. However, **0**, `''` and spaces are not considered `NULL`.

Related Topics
SQL NOT NULL Constraint
SQL COUNT()
SQL UNIQUE Constraint
SQL SELECT DISTINCT Statement
SQL CREATE TABLE Statement
SQL INSERT INTO SELECT Statement

IS NOT NULL

In SQL, `IS NOT NULL` condition is used to select rows if the specified field is NOT NULL. For example,

```
SELECT *
FROM Employee
WHERE email IS NOT NULL;
```

Run Code »

Here, the SQL command selects employees who have emails.

Table: Employee

employee_id	first_name	last_name	department	email
1	Peter	Doe	Operations	peter@gmail.com
2	Megan	Morel	Finance	NULL
3	Rose	Bailey	Operations	rose@gmail.com
4	Linda	Bailey	Finance	NULL
5	Mary	Doe	Sales	NULL

```
SELECT *
FROM Employee
WHERE email IS NOT NULL;
```

employee_id	first_name	last_name	department	email
1	Peter	Doe	Operations	peter@gmail.com
3	Rose	Bailey	Operations	rose@gmail.com

Example: IS NOT NULL in SQL

IS NULL With COUNT()

We can use the COUNT() function with `IS NULL` to count the number of rows with an empty field. For example,

```
SELECT COUNT(*)
FROM Employee
WHERE email IS NULL;
```

Run Code »

Here, the SQL command returns the total number of employees that do not have email.

Table: Employee

employee_id	first_name	last_name	department	email
1	Peter	Doe	Operations	peter@gmail.com
2	Megan	Morel	Finance	NULL
3	Rose	Bailey	Operations	rose@gmail.com
4	Linda	Bailey	Finance	NULL
5	Mary	Doe	Sales	NULL

```
SELECT COUNT(*)
FROM Employee
WHERE email IS NULL;
```

COUNT(*)
3

Example: IS NULL with COUNT() in SQL

Similarly, we can use the `COUNT()` function with `IS NOT NULL` to count the number of non-empty fields.

Previous Tutorial: SQL BETWEEN
Next Tutorial: SQL MIN() and MAX()

Did you find this article helpful?

Related Tutorials

Programming — SQL COUNT()
Programming — SQL NOT NULL Constraint
Programming — SQL UNIQUE Constraint
Programming — SQL CREATE TABLE Statement

SQL MAX() and MIN()

In this tutorial, we'll learn about the MIN() and MAX() functions and how to use them with examples.

- The `MAX()` function returns the maximum value of a column.
- The `MIN()` function returns the minimum value of a column.

1. SQL MAX() Function

Let's take an example.

```
SELECT MAX(age)
FROM Customers;
```

Run Code

Here, the SQL command returns the largest value from the `age` column.

Example: MAX() in SQL

2. SQL MIN() Function

Let's take an example.

```
SELECT MIN(age)
FROM Customers;
```

Run Code

Here, the SQL command returns the smallest value from the `age` column.

Example: MIN() in SQL

Aliases with MAX() and MIN()

In the above examples, the field name in the result set is `MIN(age)` and `MAX(age)`.

It is also possible to give custom names to these fields using the `AS` keyword. For example,

```
SELECT MAX(age) AS max_age
FROM Customers;
```

Run Code

Here, the field name `MAX(age)` is replaced with `max_age` in the result set.

Example: MAX() in SQL with Alias

MAX() and MIN() with Strings

The `MAX()` and `MIN()` functions also work with other data types such as text, not just numbers. For example,

```
SELECT MIN(first_name) AS min_first_name
FROM Customers;
```

Run Code

Here, the SQL command selects the minimum value of `first_name` based on the dictionary order.

Example: MIN() in SQL with String

Select Row Containing Max/Min Value

If we need to select the entire row(s) containing max/min value, we can use the nested `SELECT` statement in this way:

```
-- Get entire rows containing the minimum age
SELECT *
FROM Customers
WHERE age = (
    SELECT MIN(age)
    FROM Customers
);
```

Run Code

Here, the SQL command selects all the rows having the lowest `age` value.

Example: Get Rows Containing Minimum Age

More SQL MAX() and MIN() Examples

MAX() and MIN() of Two or More Values

MAX() and MIN() with HAVING

Recommended Readings:

- SQL Alias
- SQL Subquery (Nested Select)

Previous Tutorial: SQL IS NULL and NOT NULL
Next Tutorial: SQL COUNT

Did you find this article helpful?

Related Tutorials

Programming — SQL SELECT
Programming — SQL SELECT INTO Statement
Programming — SQL IN Operator
Programming — SQL INSERT INTO SELECT Statement

SQL COUNT()

In this tutorial, we'll learn about the SQL COUNT() function with the help of various examples.

The `COUNT()` function returns the number of rows in the result set. For example,

```
SELECT COUNT(*)
FROM Customers;
```

Run Code »

Here, the SQL command count rows and returns the total number of rows of the `Customers` table.

Table: Customers

customer_id	first_name	last_name	age	country
1	John	Doe	31	USA
2	Robert	Luna	22	USA
3	David	Robinson	22	UK
4	John	Reinhardt	25	UK
5	Betty	Doe	28	UAE

```
SELECT COUNT(*)
FROM Customers;
```

COUNT(*)
5

Example: SQL COUNT() Function

Aliases with COUNT()

In the above example, the field name in the result set is `COUNT(*)` .

It is also possible to give custom names to these fields using the `AS` keyword. For example,

```
SELECT COUNT(*) AS total_customers
FROM Customers;
```

Run Code »

Here, the field name `COUNT(*)` is replaced with `total_customers` in the result set.

Table: Customers

customer_id	first_name	last_name	age	country
1	John	Doe	31	USA
2	Robert	Luna	22	USA
3	David	Robinson	22	UK
4	John	Reinhardt	25	UK
5	Betty	Doe	28	UAE

```
SELECT COUNT(*) AS total_customers
FROM Customers;
```

total_customers
5

Example: COUNT() in SQL with Alias

COUNT() with WHERE

Let's take an example,

```
SELECT COUNT(country) AS customers_in_UK
FROM Customers
WHERE country = 'UK';
```

Run Code »

Here, the SQL command returns the count of customers whose country is **UK**.

Table: Customers

customer_id	first_name	last_name	age	country
1	John	Doe	31	USA
2	Robert	Luna	22	USA
3	David	Robinson	22	UK
4	John	Reinhardt	25	UK
5	Betty	Doe	28	UAE

```
SELECT COUNT(country) AS customers_in_UK
FROM Customers
WHERE country = 'UK';
```

customers_in_UK
2

Example: SQL COUNT() Function with WHERE

COUNT() with DISTINCT

If we need to count the number of unique rows, we can use the `COUNT()` function with the `DISTINCT` clause. For example,

```
SELECT COUNT(DISTINCT country)
FROM Customers;
```

Run Code »

Here, the SQL command returns the count of unique countries.

Table: Customers

customer_id	first_name	last_name	age	country
1	John	Doe	31	USA
2	Robert	Luna	22	USA
3	David	Robinson	22	UK
4	John	Reinhardt	25	UK
5	Betty	Doe	28	UAE

```
SELECT COUNT(DISTINCT country)
FROM Customers;
```

COUNT(DISTINCT country)
3

Example: Counting unique countries

COUNT() with GROUP BY

The `COUNT()` function can be used with the GROUP BY clause to count the rows with similar values. For example,

```
SELECT country, COUNT(*) AS customers
FROM Customers
GROUP BY country;
```

Run Code »

Here, the SQL command returns the number of customers in each country.

Table: Customers

customer_id	first_name	last_name	age	country
1	John	Doe	31	USA
2	Robert	Luna	22	USA
3	David	Robinson	22	UK
4	John	Reinhardt	25	UK
5	Betty	Doe	28	UAE

```
SELECT country, COUNT(*) AS customers
FROM Customers
GROUP BY country;
```

country	customers
UAE	1
UK	2
USA	2

Example: SQL COUNT() Function with GROUP BY

COUNT() With HAVING Clause

Let's take an example,

```
SELECT COUNT(customer_id), country
FROM Customers
GROUP BY country
HAVING COUNT(customer_id) > 1;
```

Run Code »

Here, the SQL command:

1. counts the number of rows by grouping them by `country`
2. returns the result set if their **count** is greater than 1.

To learn more, visit SQL HAVING Clause.

COUNT() With NULL Values

1. `SELECT COUNT(*)` returns the **count of all records** in the result set regardless of NULL values.
2. `SELECT COUNT(attribute)` returns the **count of records containing non-NULL values** of the specified column.

Previous Tutorial: SQL MIN() and MAX()
Next Tutorial: SQL SUM() and AVERAGE()

Related Tutorials

Programming — SQL GROUP BY
Programming — SQL SELECT DISTINCT Statement
Programming — SQL HAVING Clause
Programming — SQL UNIQUE Constraint

SQL SUM() AND AVG()

In this tutorial, we'll learn about the SQL SUM() AND AVG() functions with the help of various examples.

In SQL, `SUM()` and `AVG()` functions are used to calculate total and average values in numeric columns.

SQL SUM() Function

The SQL `SUM()` function is used to calculate the sum of numeric values in a column. For example,

```
SELECT SUM(amount) AS total_sales
FROM Orders;
```

Run Code »

Here, the SQL command returns the sum of amounts of all orders.

Table: Orders

order_id	item	amount	customer_id
1	Keyboard	400	4
2	Mouse	300	4
3	Monitor	12000	3
4	Keyboard	400	1
5	Mousepad	250	2

```
SELECT SUM(amount) AS total_sales
FROM Orders;
```

total_sales
13350

Example: SQL SUM() function

Let's take a look at another example.

```
SELECT SUM(amount) AS total_of_cus4
FROM Orders
WHERE customer_id = 4;
```

Run Code »

Here, the SQL command returns the total amount to be paid by the customer having `id` 4.

Table: Orders

order_id	item	amount	customer_id
1	Keyboard	400	4
2	Mouse	300	4
3	Monitor	12000	3
4	Keyboard	400	1
5	Mousepad	250	2

```
SELECT SUM(amount) AS total_of_cus4
FROM Orders
WHERE customer_id = 4;
```

total_of_cus4
700

Example: SQL SUM() function with WHERE clause

SQL AVG() Function

The SQL `AVG()` function is used to calculate the average of numeric values in a column. For example,

```
SELECT AVG(age) AS average_age
FROM Customers;
```

Run Code »

Here, the SQL command returns the average age of all customers.

Table: Customers

customer_id	first_name	last_name	age	country
1	John	Doe	31	USA
2	Robert	Luna	22	USA
3	David	Robinson	22	UK
4	John	Reinhardt	25	UK
5	Betty	Doe	28	UAE

```
SELECT AVG(age) AS average_age
FROM Customers;
```

average_age
25.6

Example: SQL AVG() function

Let's take a look at another example

```
SELECT DISTINCT customer_id, AVG(amount) AS average_spends
FROM Orders
GROUP BY customer_id;
```

Run Code »

Here, the SQL command returns the average spending of each customer.

Table: Orders

order_id	item	amount	customer_id
1	Keyboard	400	4
2	Mouse	300	4
3	Monitor	12000	3
4	Keyboard	400	1
5	Mousepad	250	2

```
SELECT customer_id, AVG(amount) AS average_spends
FROM Orders
GROUP BY customer_id;
```

customer_id	average_spends
1	400
2	250
3	12000
4	350

Example: SQL AVG() function with GROUP BY clause

Recommended readings

- SQL Aliases
- SQL GROUP BY
- SQL DISTINCT

Previous Tutorial: SQL COUNT
Next Tutorial: SQL ORDER BY

Did you find this article helpful?

Related Tutorials

Programming — SQL GROUP BY
Programming — SQL HAVING Clause
Programming — SQL JOIN
Programming — SQL SELECT INTO Statement

SQL ORDER BY Clause

In this tutorial, we'll learn about the ORDER BY clause and how to use it with examples.

The SQL `ORDER BY` clause is used to sort the result set in either ascending or descending order. For example,

```
SELECT *
FROM Customers
ORDER BY first_name;
```

Run Code »

Here, the SQL command selects all customers and then sorts them in ascending order by `first_name`.

Table: Customers

customer_id	first_name	last_name	age	country
1	John	Doe	31	USA
2	Robert	Luna	22	USA
3	David	Robinson	22	UK
4	John	Reinhardt	25	UK
5	Betty	Doe	28	UAE

```
SELECT *
FROM Customers
ORDER BY first_name;
```

customer_id	first_name	last_name	age	country
5	Betty	Doe	28	UAE
3	David	Robinson	22	UK
1	John	Doe	31	USA
4	John	Reinhardt	25	UK
2	Robert	Luna	22	USA

Example: ORDER BY in SQL

ORDER BY ASC (Ascending Order)

We can use the `ASC` keyword explicitly to sort selected records in **ascending order**. For example,

```
SELECT *
FROM Customers
ORDER BY age ASC;
```

Run Code »

Here, the SQL command selects all the customers and then sorts them in ascending order by `age`.

Table: Customers

customer_id	first_name	last_name	age	country
1	John	Doe	31	USA
2	Robert	Luna	22	USA
3	David	Robinson	22	UK
4	John	Reinhardt	25	UK
5	Betty	Doe	28	UAE

```
SELECT *
FROM Customers
ORDER BY age ASC;
```

customer_id	first_name	last_name	age	country
2	Robert	Luna	22	USA
3	David	Robinson	22	UK
4	John	Reinhardt	25	UK
5	Betty	Doe	28	UAE
1	John	Doe	31	USA

Example: ORDER BY ASC in SQL

Note: The `ORDER BY` clause sorts result set in ascending by default; it's not necessary to use `ASC` explicitly.

ORDER BY DESC (Descending Order)

We use the `DESC` keyword to sort the selected records in **descending order**. For example,

```
SELECT *
FROM Customers
ORDER BY age DESC;
```

Run Code »

Here, the SQL command selects all the customers and then sorts them in descending order by `age`.

Table: Customers

customer_id	first_name	last_name	age	country
1	John	Doe	31	USA
2	Robert	Luna	22	USA
3	David	Robinson	22	UK
4	John	Reinhardt	25	UK
5	Betty	Doe	28	UAE

```
SELECT *
FROM Customers
ORDER BY age DESC;
```

customer_id	first_name	last_name	age	country
1	John	Doe	31	USA
5	Betty	Doe	28	UAE
4	John	Reinhardt	25	UK
2	Robert	Luna	22	USA
3	David	Robinson	22	UK

Example: ORDER BY DESC in SQL

ORDER BY With Multiple Columns

We can also use `ORDER BY` with multiple columns. For example,

```
SELECT *
FROM Customers
ORDER BY first_name, age;
```

Run Code »

Here, the SQL command selects all the records and then sorts them by `first_name`. If the `first_name` repeats more than once, it sorts those records by `age`.

Table: Customers

customer_id	first_name	last_name	age	country
1	John	Doe	31	USA
2	Robert	Luna	22	USA
3	David	Robinson	22	UK
4	John	Reinhardt	25	UK
5	Betty	Doe	28	UAE

```
SELECT *
FROM Customers
ORDER BY first_name, age;
```

customer_id	first_name	last_name	age	country
5	Betty	Doe	28	UAE
3	David	Robinson	22	UK
4	John	Reinhardt	25	UK
1	John	Doe	31	USA
2	Robert	Luna	22	USA

Example: SQL ORDER BY with multiple columns

ORDER BY With WHERE

We can also use ORDER BY with the SELECT WHERE clause. For example,

```
SELECT last_name, age
FROM Customers
WHERE NOT country = 'UK'
ORDER BY last_name DESC;
```

Run Code »

Here,

- The SQL command first selects `last_name` and `age` fields from the `Customers` table if their `country` is not **UK**.
- Then, the selected records are sorted in descending order by their `last_name`.

Table: Customers

customer_id	first_name	last_name	age	country
1	John	Doe	31	USA
2	Robert	Luna	22	USA
3	David	Robinson	22	UK
4	John	Reinhardt	25	UK
5	Betty	Doe	28	UAE

```
SELECT last_name, age
FROM Customers
WHERE NOT country = 'UK'
ORDER BY last_name DESC;
```

last_name	age
Luna	22
Doe	31
Doe	28

Example: ORDER BY with WHERE in SQL

Note: The `WHERE` clause must appear before the `ORDER BY` clause, while using the `WHERE` clause with `ORDER BY`.

Previous Tutorial: SQL SUM() and AVERAGE()
Next Tutorial: SQL GROUP BY

Did you find this article helpful?

Related Tutorials

Programming — SQL LIMIT, TOP and FETCH FIRST
Programming — SQL SELECT
Programming — SQL SELECT DISTINCT Statement
Programming — SQL Subquery

SQL GROUP BY

In this tutorial, we'll learn about GROUP BY in SQL with the help of examples.

In SQL, the `GROUP BY` clause is used to group rows by one or more columns. For example,

```
SELECT country, COUNT(*) AS number
FROM Customers
GROUP BY country;
```

Run Code »

Here, the SQL command groups the rows by the `country` column, and counts the number of each country (because of the `COUNT()` function).

> **Note:** The `GROUP BY` clause is used in conjunction with aggregate functions such as MIN(), MAX(), SUM(), AVG() and COUNT(), etc.

Table: Customers

customer_id	first_name	last_name	age	country
1	John	Doe	31	USA
2	Robert	Luna	22	USA
3	David	Robinson	22	UK
4	John	Reinhardt	25	UK
5	Betty	Doe	28	UAE

```
SELECT country, COUNT(*) AS number
FROM Customers
GROUP BY country;
```

country	number
UAE	1
UK	2
USA	2

Example: SQL GROUP BY

Here, the column name of the `COUNT()` function in the result-set is `number`, because of the `AS` alias. To learn more, visit SQL AS Alias.

Example Two: GROUP BY in SQL

Let's try to find the total amount for each customer who has placed an order.

```
SELECT customer_id, SUM(amount) AS total
FROM Orders
GROUP BY customer_id;
```

Run Code »

Here, the SQL command sums the `amount` after grouping rows with `customer_id`.

Table: Orders

order_id	item	amount	customer_id
1	Keyboard	400	4
2	Mouse	300	4
3	Monitor	12000	3
4	Keyboard	400	1
5	Mousepad	250	2

```
SELECT customer_id, SUM(amount) AS total
FROM Orders
GROUP BY customer_id;
```

customer_id	total
1	400
2	250
3	12000
4	700

Example: SQL GROUP BY

GROUP BY Clause With JOIN in SQL

We can also use the `GROUP BY` clause with the `JOIN` clause. For example,

```
SELECT Customers.customer_id, Customers.first_name, Count(Orders.order_id) AS order_count
FROM Customers
LEFT JOIN Orders
ON Customers.customer_id = Orders.customer_id
GROUP BY Customers.customer_id;
```

Run Code »

Here, the SQL command joins `Customers` and `Orders` tables, and groups the result set by `customer_id` (a customer). This gives us the number of orders that are placed by each customer.

Visit SQL JOIN to read more about the `JOIN` clause in SQL.

GROUP BY With Multiple Columns

`GROUP BY` can also be used to group rows based on multiple columns. For example,

```
SELECT country, state, MIN(age) as min_age
FROM Persons
GROUP BY country, state;
```

Run Code »

Here, the SQL command groups all persons with similar `country` and `state`, and gives the minimum `age` of each group.

GROUP BY With HAVING Clause

We can use the `GROUP BY` clause with the `HAVING` clause to filter the result set based on aggregate functions. For example,

```
SELECT COUNT(customer_id), country
FROM Customers
GROUP BY country
HAVING COUNT(customer_id) > 1;
```

Run Code »

Here, the SQL command:

1. counts the number of rows by grouping them by `country`
2. returns the result set if their count is greater than 1

To learn more, visit SQL HAVING Clause.

Previous Tutorial: SQL ORDER BY

Next Tutorial: SQL LIKE

Related Tutorials

Programming — SQL HAVING Clause
Programming — SQL COUNT()
Programming — SQL SELECT INTO Statement
Programming — SQL SUM() AND AVG()

SQL LIKE and NOT LIKE Operators

In this tutorial, we'll learn about the LIKE operator in SQL and how to use them with examples.

SQL LIKE Operator

The `LIKE` operator in SQL is used with the WHERE clause to get a result set that matches the given string pattern. For example,

```
SELECT *
FROM Customers
WHERE country LIKE 'UK';
```

Run Code »

Here, the SQL command selects customers whose `country` is **UK**.

Table: Customers

customer_id	first_name	last_name	age	country
1	John	Doe	31	USA
2	Robert	Luna	22	USA
3	David	Robinson	22	UK
4	John	Reinhardt	25	UK
5	Betty	Doe	28	UAE

```
SELECT *
FROM Customers
WHERE country LIKE 'UK';
```

customer_id	first_name	last_name	age	country
3	David	Robinson	22	UK
4	John	Reinhardt	25	UK

Example: SQL LIKE

Note: Although the `LIKE` operator behaves similar to the `=` operator in this example, they are not the same. The `=` operator is used to check equality whereas `LIKE` operator is used to match string patterns only.

SQL LIKE With Wildcards

The `LIKE` operator in SQL is often used with wildcards to match a pattern of string. For example,

```
SELECT *
FROM Customers
WHERE last_name LIKE 'R%';
```

Run Code »

Here, `%` (means zero or more characters) is a wildcard character. Hence, the SQL command selects customers whose `last_name` starts with **R** followed by zero or more characters after it.

Table: Customers

customer_id	first_name	last_name	age	country
1	John	Doe	31	USA
2	Robert	Luna	22	USA
3	David	Robinson	22	UK
4	John	Reinhardt	25	UK
5	Betty	Doe	28	UAE

```
SELECT *
FROM Customers
WHERE last_name LIKE 'R%';
```

customer_id	first_name	last_name	age	country
3	David	Robinson	22	UK
4	John	Reinhardt	25	UK

Example: SQL Wildcards

Example Two: SQL LIKE With Wildcards

There are more wildcard characters we can use. Lets see another example using `_` wildcard character with `LIKE` in SQL.

```
SELECT *
FROM Customers
WHERE country LIKE 'U_';
```

Run Code »

Here, the SQL command selects customers whose `country` name starts with **U** and is followed by only one character.

SQL NOT LIKE Operator

We can also invert the working of `LIKE` operator and ignore the result set matching with the given string pattern by using the `NOT` operator. For example,

```
SELECT *
FROM Customers
WHERE country NOT LIKE 'USA';
```

Run Code »

Here, the SQL command selects all customers except those, whose `country` is **USA**.

SQL LIKE With Multiple Values

We can use the `LIKE` operator with multiple string patterns to select rows by using with the OR operator. For example,

```
SELECT *
FROM Customers
WHERE last_name LIKE 'R%s' OR last_name LIKE '%e';
```

Run Code »

Here, the SQL command selects customers whose `last_name` starts with **R** and ends with **t**, or customers whose `last_name` ends with **e**.

More SQL LIKE and NOT LIKE Examples

- Find customers whose last name starts with **R**.
- Find customers whose country is exactly one character after **U**.
- Find customers whose last name starts with **R** and ends with **t**.

Previous Tutorial: SQL GROUP BY

Next Tutorial: SQL Wildcards

Did you find this article helpful?

Related Topics

- SQL SELECT
- SQL Wildcards
- SQL AND, OR, and NOT Operators
- SQL IN Operator
- SQL SELECT INTO Statement
- SQL LIMIT, TOP and FETCH FIRST

Related Tutorials

Programming — SQL SELECT

Programming — SQL AND, OR, and NOT Operators

Programming — SQL Wildcards

Programming — SQL IN Operator

SQL Wildcards

In this tutorial, we'll learn about the Wildcards in SQL and how to use them with examples.

A wildcard character in SQL is used with the `LIKE` clause to replace a single or set of characters in any string. For example,

```
SELECT *
FROM Customers
WHERE last_name LIKE 'R%';
```

Run Code »

Here, `%` (means zero or more characters) is a wildcard character. Hence, the SQL command selects customers whose `last_name` starts with **R** followed by zero or more characters after it.

Table: Customers

customer_id	first_name	last_name	age	country
1	John	Doe	31	USA
2	Robert	Luna	22	USA
3	David	Robinson	22	UK
4	John	Reinhardt	25	UK
5	Betty	Doe	28	UAE

```
SELECT *
FROM Customers
WHERE last_name LIKE 'R%';
```

customer_id	first_name	last_name	age	country
3	David	Robinson	22	UK
4	John	Reinhardt	25	UK

Example: SQL Wildcards

Recommended Reading: SQL LIKE Operator

% Wildcard in SQL

The `%` wildcard in SQL is used to represent zero or more characters. For example,

```
SELECT *
FROM Customers
WHERE last_name LIKE 'R%';
```

Run Code »

Here, the SQL command selects customers whose last name starts with R followed by zero or more characters.

Expression	String	Matched?
`R%`	`R`	match
	`Run`	match
	`Mere`	no match
	`Summer`	no match

_ Wildcard in SQL

The `_` wildcard in SQL is used to represent exactly one character in a string. For example,

```
SELECT *
FROM Customers
WHERE country LIKE 'U_';
```

Run Code »

Here, the SQL command selects customers whose `country` name starts with **U** and is followed by only one character.

Expression	String	Matched?
`U_`	`U`	no match
	`UK`	match
	`USA`	no match

[] Wildcard in SQL

The `[]` wildcard in SQL is used to represent any one character inside brackets. For example,

```
SELECT *
FROM Customers
WHERE country LIKE 'U[KA]%';
```

Here, the SQL command selects customers whose `country` name starts with **U** and is followed by either **K** or **A**. Any number of characters are allowed afterwards.

Expression	String	Matched?
`U[KA]%`	`U`	no match
	`UK`	match
	`UAE`	match
	`USA`	no match

! Wildcard in SQL

The `!` wildcard in SQL is used to exclude characters from a string. For example,

```
SELECT *
FROM Customers
WHERE last_name LIKE '[!DR]%';
```

Here, the SQL command selects customers whose `last_name` does not start with **D** or **R**.

Expression	String	Matched?
`[!DR]%`	`Doe`	no match
	`Reinhardt`	no match
	`Luna`	match
	`D`	no match
	`O`	match
	`R`	no match

Wildcard Characters in Different Databases

Different databases have different sets of wildcard characters. Some of them are listed here.

Database Systems

- SQL Server
- PostgreSQL and MySQL
- Oracle

Previous Tutorial: SQL LIKE

Next Tutorial: SQL UNION

Did you find this article helpful?

Related Tutorials

Programming — SQL LIKE and NOT LIKE Operators

Programming — SQL SELECT

Programming — SQL IN Operator

Programming — SQL AND, OR, and NOT Operators

SQL UNION

In this tutorial, we'll learn to use the UNION operator in SQL with the help of examples.

In SQL, the `UNION` operator selects rows from two or more tables.

If rows of tables are the same, those rows are only included once in the result set. For example,

```
SELECT age
FROM Teachers
UNION
SELECT age
FROM Students;
```

Run Code

Here, the SQL command returns the `age` column from the `Teachers` table and the `Students` table, ignoring the duplicate rows.

Example: SQL UNION

Things to Note While Using UNION

To use `UNION` in SQL, we must always remember,

- Column count in all tables must be the same. For example, `Teachers` and `Students` both tables have three columns.
- The data types of columns must be the same. For example, the `age` column in `Teachers` is integer, so is the `age` in `Students` table.
- The columns must be in the same order in each table. For example, the order of columns is **id-name-age** in `Teachers`, so in the `Students` table.

SQL UNION ALL Operator

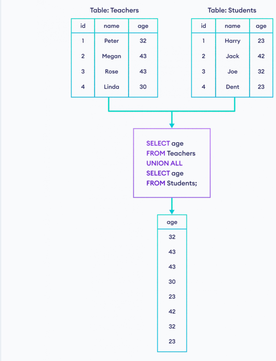
The `UNION ALL` operator selects rows from two or more tables similar to `UNION`. However, unlike `UNION`, `UNION ALL` doesn't ignore duplicate rows.

Let's try the previous SQL command again using `UNION ALL` instead of `UNION`.

```
SELECT age
FROM Teachers
UNION ALL
SELECT age
FROM Students;
```

Run Code

Here, the SQL command selects rows from both tables including duplicate rows.



Example: SQL UNION ALL

SQL UNION Vs UNION ALL

SQL UNION	SQL UNION ALL
It only returns distinct rows from the result set of two queries.	It returns the duplicate values from the result set of two queries.
Slower in comparison to the `UNION ALL` operator.	Executes fast as there is no need to filter the result-sets by removing duplicate values.

It is recommended to use `UNION ALL` when we know the result set will have unique values as it improves the performance.

SQL UNION Vs SQL JOIN

SQL JOIN	SQL UNION
It is used to combine data into new columns from different tables.	It is used to combine data into new rows from the result of different queries.
It uses the common column in both of the tables to fetch the data.	It selects data from two tables and combines the output.
Any number of columns can be present in tables.	Column counts must be the same in both of the tables.
Data type of columns can be different.	Data type of columns must be the same.

To learn more, visit SQL JOIN.

Previous Tutorial: SQL Wildcards
Next Tutorial: SQL Subquery

Did you find this article helpful?

Related Tutorials

Programming
SQL IN Operator

Programming
SQL INSERT INTO SELECT Statement

Programming
SQL UNIQUE Constraint

Programming
SQL JOIN

SQL Subquery

In this tutorial, we'll learn about subqueries in SQL with the help of examples.

In SQL, it's possible to place a SQL query inside another query known as subquery. For example,

```
SELECT *
FROM Customers
WHERE age = (
  SELECT MIN(age)
  FROM Customers
);
```

Run Code »

In a subquery, the outer query's result is dependent on the result-set of the inner subquery. That's why subqueries are also called nested queries.

Here, the SQL command

1. executes the subquery first; selects minimum `age` from the `Customers` table.
2. executes the outer query; selects rows where `age` is **equal to the result of subquery.**

Example: SQL Subqueries

Example 2: SQL Subquery

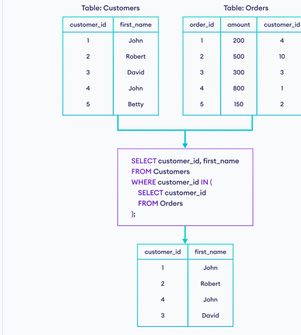
Suppose we want details of **customers** who have placed an **order**. Here's how we can do that using a subquery:

```
SELECT customer_id, first_name
FROM Customers
WHERE customer_id IN (
  SELECT customer_id
  FROM Orders
);
```

Run Code »

Here, the SQL command

1. selects `customer_id` from `Orders` table
2. select rows from `Customers` table where `customer_id` is in the result set of subquery



Example: SQL Subquery

SQL Subquery and JOIN

In some scenarios, we can get the same result set using a subquery and the JOIN clause. For example,

The result set of this query

```
SELECT DISTINCT Customers.customer_id, Customers.first_name
FROM Customers
INNER JOIN Orders
ON Customers.customer_id = Orders.customer_id
ORDER BY Customers.customer_id;
```

Run Code »

will be the same as

```
SELECT customer_id, first_name
FROM Customers
WHERE customer_id IN (
  SELECT customer_id
  FROM Orders
);
```

Run Code »

> **Note:** It's preferred to use the `JOIN` clause instead of a subquery whenever possible. It's because the execution speed of `JOIN` is faster and more optimized than a subquery.

Previous Tutorial: SQL UNION
Next Tutorial: SQL ANY and ALL

Did you find this article helpful?

Related Tutorials
Programming SQL IN Operator
Programming SQL EXISTS
Programming SQL JOIN
Programming SQL FULL OUTER JOIN

SQL ANY and ALL

In this tutorial, we'll learn about SQL ANY and ALL operators with the help of examples.

SQL ANY

SQL `ANY` compares a value of the first table with all values of the second table and returns the row if there is a match with any value.

For example, if we want to find teachers whose age is similar to any of the student's age, we can use

```
SELECT *
FROM Teachers
WHERE age = ANY (
  SELECT age
  FROM Students
);
```

Here, the sub query

```
SELECT age
FROM Students
```

returns all the ages from the `Students` table. And, the condition

```
WHERE age = ANY (...)
```

compares the student ages (returned by subquery) with the teacher's age. If there is any match, the corresponding row of the `Teachers` table is selected.

Example: ANY in SQL

SQL ALL

SQL `ALL` compares a value of the first table with all values of the second table and returns the row if there is a match with all values.

For example, if we want to find teachers whose age is greater than all students, we can use

```
SELECT *
FROM Teachers
WHERE age >  ALL (
  SELECT age
  FROM Students
);
```

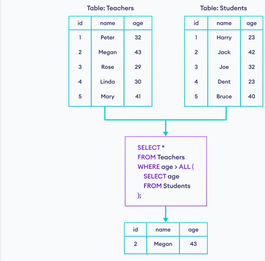
Here, the sub query

```
SELECT age
FROM Students
```

returns all the ages from the `Students` table. And, the condition

```
WHERE age > ALL (...)
```

compares the student ages (returned by subquery) with the teacher's age. If the teacher's age is greater than all student's ages, the corresponding row of the `Teachers` table is selected.



Example: ALL in SQL

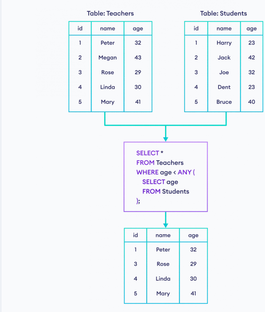
ANY and ALL with Comparison Operators

We can use any comparison operators like `=`, `>`, `<`, etc. with the `ANY` and `ALL` keywords.

Let's see an example where we want teachers whose age is less than any student.

```
SELECT *
FROM Teachers
WHERE age < ANY (
  SELECT age
  FROM Students
);
```

Here, the SQL command selects rows if `age` in the **outer query** is less than any `age` in a **subquery**.



Example: ANY in SQL

Related Topics
SQL UNION
SQL Subquery
SQL Comments
SQL SELECT
SQL SELECT INTO Statement
SQL IN Operator

Previous Tutorial: SQL Subquery
Next Tutorial: SQL CASE

Related Tutorials

Programming
SQL UNION

Programming
SQL Comments

Programming
SQL Subquery

Programming
SQL IN Operator

SQL CASE

In this tutorial, we'll learn about the CASE statement in SQL and how to use them with examples.

The `CASE` statement in SQL is used to check conditions and perform tasks on each row while selecting data. For example,

```
SELECT customer_id, first_name,
CASE
  WHEN age >= 18 THEN 'Allowed'
END AS can_vote
FROM Customers;
```

Run Code »

Here, the SQL command checks each row with the given case.

If `age` is greater than or equal to **18**, the result set contains

- columns with `customer_id` and `first_name` with their values
- **Allowed** is returned as a `can_vote` column.

Table: Customers

customer_id	first_name	last_name	age	country
1	John	Doe	31	USA
2	Robert	Luna	22	USA
3	David	Robinson	22	UK
4	John	Reinhardt	25	UK
5	Betty	Doe	28	UAE

```
SELECT customer_id, first_name
CASE
  WHEN age >= 18 THEN 'Allowed'
END AS can_vote
FROM Customers;
```

customer_id	first_name	can_vote
1	John	Allowed
2	Robert	Allowed
3	David	Allowed
4	John	Allowed
5	Betty	Allowed

Example: CASE in SQL

Note: The syntax of `CASE` always starts with the `CASE` keyword and ends with the `END` keyword followed by column name alias.

Example Two: SQL CASE Statement

Let's take a look at another example where we want to provide a **10% discount** on each order for a Christmas sale **if the amount is more than 400**.

```
SELECT order_id, customer_id,
CASE
    WHEN amount >= 400 THEN (amount - amount * 10/100)
END AS offer_price
FROM Orders;
```

Run Code »

Here, the SQL command checks if the `amount` is greater than or equal to **400**. If this condition is satisfied, a new column `offer_price` will contain the values that's equal to `amount - amount*10/100`.

Multiple Cases

It is also possible to stack multiple conditions inside the `CASE` clause.

```
SELECT customer_id, first_name,
CASE
    WHEN country = 'USA' THEN 'United States of America'
    WHEN country = 'UK' THEN 'United Kingdom'
END AS country_name
FROM Customers;
```

Run Code »

Here, the result set will contain a column named `country_name` along with `customer_id` and `first_name` columns.

The value of `country_name` will be **United States of America** if the `country` is equal to **USA**.

Similarly, the value of `country_name` will be **United Kingdom** if the `country` is equal to **UK**.

CASE With ELSE in SQL

A `CASE` statement can have an optional `ELSE` clause. The `ELSE` clause is executed if none of the conditions in the `CASE` statement is matched. For example,

```
SELECT customer_id, first_name,
CASE
    WHEN country = 'USA' THEN 'United States of America'
    WHEN country = 'UK' THEN 'United Kingdom'
    ELSE 'Unknown Country'
END AS country_name
FROM Customers;
```

Run Code »

Here, the result set will contain a field named `country_name` along with `customer_id` and `first_name`.

The value of the `country_name` will be:

- **United States of America** if the `country` is **USA**
- **United Kingdom** if the `country` is **UK**
- **Unknown Country** if the `country` is neither **USA** nor **UK** (because of the `ELSE` clause).

Table: Customers

customer_id	first_name	last_name	age	country
1	John	Doe	31	USA
2	Robert	Luna	22	USA
3	David	Robinson	22	UK
4	John	Reinhardt	25	UK
5	Betty	Doe	28	UAE

```
SELECT customer_id, first_name
CASE
  WHEN country = 'USA' THEN 'United States of America'
  WHEN country = 'UK' THEN 'United Kingdom'
  ELSE 'Unknown Country'
END AS country_name
FROM Customers;
```

customer_id	first_name	country_name
1	John	United States of America
2	Robert	United States of America
3	David	United Kingdom
4	John	United Kingdom
5	Betty	Unknown Country

Example: CASE With ELSE in SQL

Related Topics
SQL SELECT INTO Statement
SQL GROUP BY
SQL Stored Procedures
SQL INSERT INTO SELECT Statement
SQL HAVING Clause
SQL IN Operator

Previous Tutorial: SQL ANY and ALL
Next Tutorial: SQL HAVING

Related Tutorials

Programming SQL GROUP BY
Programming SQL SELECT INTO Statement
Programming SQL Stored Procedures
Programming SQL INSERT INTO SELECT Statement

SQL HAVING Clause

In this tutorial, we'll learn about the HAVING clause in SQL and how to use them with examples.

The `HAVING` clause in SQL is used if we need to filter the result set based on aggregate functions such as MIN() and MAX(), SUM() and AVG() and COUNT().

```
SELECT COUNT(customer_id), country
FROM Customers
GROUP BY country
HAVING COUNT(customer_id) > 1;
```

Run Code »

Here, the SQL command:

1. counts the number of rows by grouping them by `country`
2. returns the result set if their **count** is greater than **1**.

Table: Customers

customer_id	first_name	last_name	age	country
1	John	Doe	31	USA
2	Robert	Luna	22	USA
3	David	Robinson	22	UK
4	John	Reinhardt	25	UK
5	Betty	Doe	28	UAE

```
SELECT COUNT(customer_id), country
FROM Customers
GROUP BY country
HAVING COUNT(customer_id) > 1;
```

COUNT(customer_id)	country
2	UK
2	USA

Example: HAVING in SQL

Note: The `HAVING` clause was introduced because the `WHERE` clause does not support aggregate functions. Also, `GROUP BY` must be used before the `HAVING` clause. To learn more, visit SQL GROUP BY.

SQL HAVING Vs WHERE

HAVING Clause	WHERE Clause
The `HAVING` clause checks the condition on **a group of rows**.	The `WHERE` clause checks the condition **on each individual row**.
The `HAVING` is used with aggregate functions.	The `WHERE` clause cannot be used with aggregate functions.
The `HAVING` clause is executed after the `GROUP BY` clause.	The `WHERE` clause is executed before the `GROUP BY` clause.

Let's take a look at an example,

If we want to select rows where the value of the `amount` column in the `Orders` table is less than **500**, we can write,

```
SELECT customer_id, amount
FROM Orders
WHERE amount < 500;
```

Run Code »

Now, if we want to select rows and calculate sum off each amount, we can write,

```
SELECT customer_id, SUM(amount) AS total
FROM Orders
GROUP BY customer_id;
```

Run Code »

Thats it.

But if we need to select rows if the sum of amounts is less than 500 for any customer, we need to write,

```
SELECT customer_id, SUM(amount) AS total
FROM Orders
GROUP BY customer_id
HAVING SUM(amount) < 500;
```

Run Code »

Related Topics
SQL GROUP BY
SQL SELECT INTO Statement
SQL SUM() AND AVG()
SQL COUNT()
SQL INSERT INTO SELECT Statement
SQL CASE

Previous Tutorial: SQL CASE
Next Tutorial: SQL EXISTS

Related Tutorials

Programming — SQL GROUP BY
Programming — SQL SUM() AND AVG()
Programming — SQL SELECT INTO Statement
Programming — SQL COUNT()

SQL EXISTS

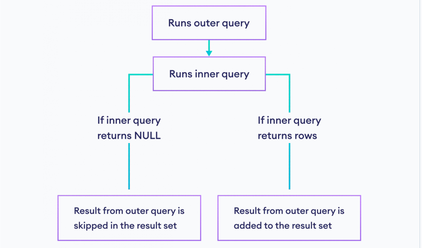
In this tutorial, we'll learn about the EXISTS operator in SQL and how to use them with examples.

The SQL `EXISTS` operator executes the outer SQL query if the subquery is not `NULL` (empty result-set). For example,

```
SELECT customer_id, first_name
FROM Customers
WHERE EXISTS (
  SELECT order_id
  FROM Orders
  WHERE Orders.customer_id = Customers.customer_id
);
```

Run Code »

Here is how the SQL command works:



Working: EXISTS in SQL

This process is repeated for each row of the outer query.

Table: Customers

customer_id	first_name	last_name	age	country
1	John	Doe	31	USA
2	Robert	Luna	22	USA
3	David	Robinson	22	UK
4	John	Reinhardt	25	UK
5	Betty	Doe	28	UAE

```
SELECT customer_id, first_name
FROM Customers
WHERE EXISTS (
  SELECT order_id
  FROM Orders
  WHERE Orders.customer_id = Customers.customer_id
);
```

customer_id	first_name
1	John
2	Robert
3	David
4	John

Example: EXISTS in SQL

SQL NOT EXISTS

We can also use the NOT operator to inverse the working of the `EXISTS` clause. The SQL command executes if the subquery returns an empty result-set. For example,

```
SELECT customer_id, first_name
FROM Customers
WHERE NOT EXISTS (
  SELECT order_id
  FROM Orders
  WHERE Orders.customer_id = Customers.customer_id
);
```

Run Code »

Here, the SQL command returns a row from the `Customers` table if the related row is not in the `Orders` table.

SQL EXISTS Examples

Previous Tutorial: SQL HAVING
Next Tutorial: SQL JOIN

Did you find this article helpful?

Related Tutorials

Programming — SQL Subquery
Programming — SQL INSERT INTO SELECT Statement
Programming — SQL IN Operator
Programming — SQL SELECT INTO Statement

SQL JOIN

In this tutorial, we'll learn about the JOIN clause in SQL with the help of examples.

The SQL `JOIN` joins two tables based on a common column, and selects records that have matching values in these columns.

Example

```
SELECT Customers.customer_id, Customers.first_name, Orders.amount
FROM Customers
JOIN Orders
ON Customers.customer_id = Orders.customer;
```

Run Code »

Here's how this code works:

Example: SQL JOIN

Here, the SQL command selects **customer_id** and **first_name** columns (from the **Customers** table) and the **amount** column (from the **Orders** table).

And, the result set will contain those rows where there is a match between **customer_id** (of the **Customers** table) and **customer** (of the **Orders** table).

Types of SQL JOINs

The `JOIN` command we performed earlier is `INNER JOIN`. There are mainly four types of joins.

- INNER JOIN (Same as `JOIN`)
- LEFT JOIN
- RIGHT JOIN
- FULL OUTER JOIN

SQL JOIN and Aliases

We can use AS aliases with table names to make our snippet short and clean. For example,

```
SELECT C.customer_id, C.first_name, O.amount
FROM Customers AS C
JOIN Orders AS O
ON C.customer_id = O.customer;
```

Run Code »

Also, we can change the column names temporarily using AS aliases. For example,

```
SELECT C.customer_id AS cid, C.first_name AS name, O.amount
FROM Customers AS C
JOIN Orders AS O
ON C.customer_id = O.customer;
```

Run Code »

Here, these snippets would work exactly the same as earlier.

Previous Tutorial: SQL EXISTS

Next Tutorial: SQL INNER JOIN

Did you find this article helpful?

Related Tutorials

Programming — SQL RIGHT JOIN

Programming — SQL FULL OUTER JOIN

Programming — SQL INNER JOIN

Programming — SQL LEFT JOIN

SQL INNER JOIN

In this tutorial, we'll learn about SQL INNER JOIN with the help of examples.

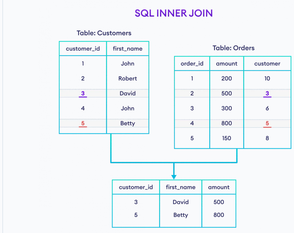
The SQL `INNER JOIN` joins two tables based on a common column, and selects records that have matching values in these columns.

Example

```
SELECT Customers.customer_id, Customers.first_name, Orders.amount
FROM Customers
INNER JOIN Orders
ON Customers.customer_id = Orders.customer;
```

Run Code »

Here's how this code works:



Example: SQL INNER JOIN

Here, the SQL command selects **customer_id** and **first_name** columns (from the **Customers** table) and the **amount** column (from the **Orders** table).

And, the result set will contain those rows where there is a match between **customer_id** (of the **Customers** table) and **customer** (of the **Orders** table).

Syntax of INNER JOIN

The syntax of `INNER JOIN` is:

```
SELECT columns
FROM table1
INNER JOIN table2
ON table1.column_name = table2.column_name;
```

INNER JOIN With WHERE Clause

Here's an example of the `INNER JOIN` with the WHERE clause:

```
SELECT Customers.customer_id, Customers.first_name, Orders.amount
FROM Customers
INNER JOIN Orders
ON Customers.customer_id = Orders.customer
WHERE Orders.amount >= 500;
```

Run Code »

Here, the SQL command joins two tables and selects rows where the `amount` is **greater than or equal to 500**.

SQL INNER JOIN With AS Alias

We can use AS aliases inside `INNER JOIN` to make our snippet short and clean. For example,

```
SELECT C.cat_name, P.prod_title
FROM Categories AS C
INNER JOIN Products AS P
ON C.cat_id= P.cat_id;
```

Run Code »

Here, the SQL command selects common rows between `Category` and `Products` table.

SQL INNER JOIN With Three Tables

We can also join more than two tables using the `INNER JOIN`. For example,

```
SELECT C.customer_id, C.first_name, O.amount, S.status
FROM Customers AS C
INNER JOIN Orders AS O
ON C.customer_id = O.customer
INNER JOIN Shippings AS S
ON C.customer_id = S.customer;
```

Run Code »

Here, the SQL command

- joins **Customers** and **Orders** table based on `customer_id`
- and joins **Customers** and **Status** table based on `customer_id`

The command returns those rows where there is a match between column values in both join conditions.

> **Note:** For this command to run, there must be the `customer_id` column in each individual table.

Inner Join Vs Other Joins

- INNER JOIN Vs JOIN
- INNER JOIN Vs LEFT JOIN
- INNER JOIN Vs RIGHT JOIN
- INNER JOIN Vs FULL OUTER JOIN

Recommended Readings

- SQL JOIN
- SQL LEFT JOIN
- SQL RIGHT JOIN
- SQL FULL OUTER JOIN

Previous Tutorial: SQL JOIN
Next Tutorial: SQL LEFT JOIN

Did you find this article helpful?

Related Tutorials

Programming — SQL JOIN
Programming — SQL FULL OUTER JOIN
Programming — SQL RIGHT JOIN
Programming — SQL LEFT JOIN

SQL LEFT JOIN

In this tutorial, we'll learn about SQL LEFT JOIN with the help of examples.

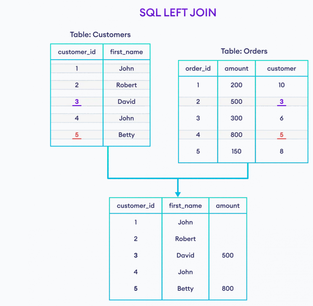
The SQL `LEFT JOIN` joins two tables based on a common column, and selects records that have matching values in these columns and remaining rows from the left table.

Example

```
SELECT Customers.customer_id, Customers.first_name, Orders.amount
FROM Customers
LEFT JOIN Orders
ON Customers.customer_id = Orders.customer;
```

Run Code »

Here's how this code works:



Example: SQL LEFT JOIN

Here, the SQL command selects **customer_id** and **first_name** columns (from the **Customers** table) and the **amount** column (from the **Orders** table).

And, the result set will contain those rows where there is a match between **customer_id** (of the **Customers** table) and **customer** (of the **Orders** table) along with all the remaining rows from the **Customers** table.

Syntax of LEFT JOIN

The syntax of `LEFT JOIN` is:

```
SELECT columns
FROM table1
LEFT JOIN table2
ON table1.column_name = table2.column_name;
```

LEFT JOIN With WHERE Clause

The SQL command can have an optional WHERE clause with the `LEFT JOIN` statement. For example,

```
SELECT Customers.customer_id, Customers.first_name, Orders.amount
FROM Customers
LEFT JOIN Orders
ON Customers.customer_id = Orders.customer
WHERE Orders.amount >= 500;
```

Run Code »

Here, the SQL command joins two tables and selects rows where the `amount` is **greater than or equal to 500**.

SQL LEFT JOIN With AS Alias

We can use AS aliases inside `LEFT JOIN` to make our snippet short and clean. For example,

```
SELECT C.cat_name, P.prod_title
FROM Categories1 AS C
LEFT JOIN Products AS P
ON C.cat_id= P.cat_id;
```

Run Code »

Here, the SQL command selects common rows between `Category` and `Products` table.

Left Join Vs Other Joins

- LEFT JOIN Vs LEFT OUTER JOIN
- LEFT JOIN Vs INNER JOIN
- LEFT JOIN Vs RIGHT JOIN
- LEFT JOIN Vs FULL OUTER JOIN

Recommended Readings

- SQL JOIN
- SQL INNER JOIN
- SQL RIGHT JOIN
- SQL FULL OUTER JOIN

Previous Tutorial: SQL INNER JOIN

Next Tutorial: SQL RIGHT JOIN

Did you find this article helpful?

Related Tutorials

Programming — SQL JOIN

Programming — SQL FULL OUTER JOIN

Programming — SQL RIGHT JOIN

Programming — SQL INNER JOIN

SQL RIGHT JOIN

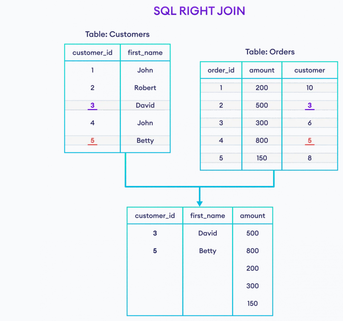
In this tutorial, we'll learn about SQL RIGHT JOIN with the help of examples.

The SQL `RIGHT JOIN` joins two tables based on a common column, and selects records that have matching values in these columns and remaining rows from the right table.

Example

```
SELECT Customers.customer_id, Customers.first_name, Orders.amount
FROM Customers
RIGHT JOIN Orders
ON Customers.customer_id = Orders.customer;
```

Here's how this code works:



Example: SQL RIGHT JOIN

Here, the SQL command selects **customer_id** and **first_name** columns (from the **Customers** table) and the **amount** column (from the **Orders** table).

And, the result set will contain those rows where there is a match between **customer_id** (of the **Customers** table) and **customer** (of the **Orders** table) along with all the remaining rows from the **Orders** table.

Syntax of RIGHT JOIN

The syntax of `RIGHT JOIN` is:

```
SELECT columns
FROM table1
RIGHT JOIN table2
ON table1.column_name = table2.column_name;
```

RIGHT JOIN With WHERE Clause

The SQL command can have an optional WHERE clause with the `RIGHT JOIN` statement. For example,

```
SELECT Customers.customer_id, Customers.first_name, Orders.amount
FROM Customers
INNER JOIN Orders
ON Customers.customer_id = Orders.customer
WHERE Orders.amount >= 500;
```

Here, the SQL command joins two tables and selects rows where the `amount` is **greater than or equal to 500**.

SQL RIGHT JOIN With AS Alias

We can use AS aliases inside `RIGHT JOIN` to make our snippet short and clean. For example,

```
SELECT C.cat_name, P.prod_title
FROM Category AS C
RIGHT JOIN Products AS P
ON C.cat_id= P.cat_id;
```

Here, the SQL command selects common rows between `Category` and `Products` table.

Right Join Vs Other Joins

- RIGHT JOIN Vs RIGHT OUTER JOIN
- RIGHT JOIN Vs INNER JOIN
- RIGHT JOIN Vs LEFT JOIN
- RIGHT JOIN Vs FULL OUTER JOIN

Recommended Readings

- SQL JOIN
- SQL INNER JOIN
- SQL LEFT JOIN
- SQL FULL OUTER JOIN

Previous Tutorial: SQL LEFT JOIN

Next Tutorial: SQL FULL OUTER JOIN

Did you find this article helpful?

Related Tutorials

Programming — SQL JOIN

Programming — SQL FULL OUTER JOIN

Programming — SQL LEFT JOIN

Programming — SQL INNER JOIN

SQL FULL OUTER JOIN

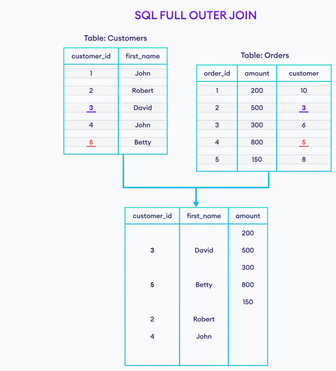
In this tutorial, we'll learn about SQL FULL OUTER JOIN with the help of examples.

The SQL `FULL OUTER JOIN` joins two tables based on a common column, and selects records that have matching values in these columns and remaining rows from both of the tables.

Example

```
SELECT Customers.customer_id, Customers.first_name, Orders.amount
FROM Customers
FULL OUTER JOIN Orders
ON Customers.customer_id = Orders.customer;
```

Here's how this code works:



Example: SQL FULL OUTER JOIN

Here, the SQL command selects **customer_id** and **first_name** columns (from the **Customers** table) and the **amount** column (from the **Orders** table).

And, the result set will contain those rows where there is a match between **customer_id** (of the **Customers** table) and **customer** (of the **Orders** table) along with all the remaining rows from **both of the tables.**

Syntax of FULL OUTER JOIN

The syntax of `FULL OUTER JOIN` is:

```
SELECT columns
FROM table1
FULL OUTER JOIN table2
ON table1.column_name = table2.column_name;
```

FULL OUTER JOIN With WHERE Clause

The SQL command can have an optional WHERE clause with the `FULL OUTER JOIN` statement. For example,

```
SELECT Customers.customer_id, Customers.first_name, Orders.amount
FROM Customers
FULL OUTER JOIN Orders
ON Customers.customer_id = Orders.customer
WHERE Orders.amount >= 500;
```

Here, the SQL command joins two tables and selects rows where the `amount` is **greater than or equal to 500**.

SQL FULL OUTER JOIN With AS Alias

We can use AS aliases inside `FULL OUTER JOIN` to make our snippet short and clean. For example,

```
SELECT C.cat_name, P.prod_title
FROM Category AS C
FULL OUTER JOIN Products AS P
ON C.cat_id= P.cat_id;
```

Here, the SQL command selects common rows between `Category` and `Products` table.

Full Outer Join Vs Other Joins

- FULL OUTER JOIN Vs FULL JOIN
- FULL OUTER JOIN Vs INNER JOIN
- FULL OUTER JOIN Vs LEFT JOIN
- FULL OUTER JOIN Vs RIGHT JOIN

Recommended Readings

- SQL JOIN
- SQL INNER JOIN
- SQL LEFT JOIN
- SQL RIGHT JOIN

Previous Tutorial: SQL RIGHT JOIN
Next Tutorial: SQL Create Database

Did you find this article helpful?

Related Tutorials

Programming — SQL JOIN
Programming — SQL RIGHT JOIN
Programming — SQL LEFT JOIN
Programming — SQL INNER JOIN

SQL CREATE DATABASE Statement

In this tutorial, we'll learn about creating databases in SQL with examples.

Before we can work with database tables, we must create a database first.

The `CREATE DATABASE` statement is used to create database tables. For example,

```
CREATE DATABASE my_db;
```

Here, the SQL command creates a database named `my_db`.

CREATE DATABASE IF NOT EXISTS

If there is already a database with the same name, SQL will throw an error while creating a database.

In such situations, we can use the `CREATE DATABASE IF NOT EXISTS` statement to create a database only if there is no existing database with the same name. For example,

```
CREATE DATABASE IF NOT EXISTS my_db;
```

Here, the SQL command creates a database named `my_db` only if there is no existing database with the same name.

List all Databases

There could be multiple databases in a database management system. To show the list of databases, we can run the following statement.

```
SHOW DATABASES;
```

Here, the SQL command lists all the available databases in the DBMS.

Switch Databases

We have to work across multiple databases from time to time. To switch between available databases, we can run the following statement.

```
USE my_db;
```

This code selects the `my_db` database and all SQL operations will be performed inside this database.

Recommended Reading

- SQL Create Table

Previous Tutorial: SQL FULL OUTER JOIN

Next Tutorial: SQL Create Table

Did you find this article helpful?

Related Tutorials

Programming — SQL DROP DATABASE Statement

Programming — SQL BACKUP DATABASE Statement

Programming — Introduction to Databases and SQL

Programming — SQL CREATE TABLE Statement

SQL CREATE TABLE Statement

In this tutorial, we'll learn about creating tables in SQL with examples.

A database table is used to store records (data). To create a database table, we use the SQL `CREATE TABLE` statement. For example,

```
CREATE TABLE Companies (
  id int,
  name varchar(50),
  address text,
  email varchar(50),
  phone varchar(10)
);
```

Run Code »

Here, the SQL command creates a database named `companies`. The table contains column (field) `id`, `name`, `address`, `email` and `phone`.

The `int`, `varchar(50)` and `text` are data types that tell what data could be stored in that field. Some commonly used data types are as follows.

Data Type	Description	Example
`int`	can store numbers	`400`, `-300`
`varchar(x)`	can store variable characters with maximum length of x	`John Doe`, `United States of America`
`text`	can store texts up to 65535 characters	`This is a really long paragraph that can go over lines.`

> **Note:** We must provide data types for each column while creating a table. Learn more about SQL Data Types.

CREATE TABLE IF NOT EXISTS

While creating a table that already exists, throws an error. To fix this issue, we can add the optional `IF NOT EXISTS` command while creating a table. For example,

```
CREATE TABLE IF NOT EXISTS Companies (
  id int,
  name varchar(50),
  address text,
  email varchar(50),
  phone varchar(10)
);
```

Run Code »

Here, the SQL command will only create a table if there is not one with a similar name.

CREATE TABLE AS

We can also create a table using records from any other existing table using the `CREATE TABLE AS` command. For example,

```
CREATE TABLE USACustomers
AS (
  SELECT *
  FROM Customers
  WHERE country = 'USA'
);
```

Here, the SQL command creates a table named `USACustomers` and copies the records of the nested query into the new table.

Related Topics: CREATE TABLE

- How to create a table with a Primary Key?
- How to define constraints while creating a table?

Recommended Readings

- SQL Views
- SQL CREATE DATABASE

Previous Tutorial: SQL Create Database

Next Tutorial: SQL Drop Database

Did you find this article helpful?

Related Tutorials

Programming — SQL Constraints

Programming — SQL ALTER TABLE Statement

Programming — SQL FOREIGN KEY

Programming — SQL PRIMARY KEY

SQL DROP DATABASE Statement

In this tutorial, we'll learn about deleting databases in SQL with examples.

In SQL, `DROP DATABASE` is used to delete the database in our Database Management System. For example,

```
DROP DATABASE my_db;
```

Here, the SQL command will delete a database named `my_db`.

Also make sure you have **admin** or **DROP** permission to run this command.

> **Note:** When we delete a database, all tables and records within a database are also deleted.

List all Databases

To verify the drop database, we can run the following command to list the available databases.

```
SHOW DATABASES;
```

Here, the SQL command lists all the available databases in the DBMS.

Recommended Reading

- SQL Create table

Previous Tutorial: SQL Create Table

Next Tutorial: SQL Drop Table

Related Tutorials

Programming
SQL CREATE DATABASE Statement

Programming
SQL DROP TABLE Statement

Programming
SQL BACKUP DATABASE Statement

Programming
Introduction to Databases and SQL

SQL DROP TABLE Statement

In this tutorial, we'll learn about deleting tables in SQL with examples.

In SQL, `DROP TABLE` is used to delete the tables in our database. For example,

```
DROP TABLE my_table;
```

Here, the SQL command will delete a table named `my_table`.

Also make sure you have **admin** or **DROP** permission to run this command.

> **Note:** When we delete a database table, all records within a table are also deleted.

DROP TABLE IF EXISTS

While dropping a table that does not exist, throws an error. To fix this issue, we can add an optional `IF EXISTS` command while dropping a table. For example,

```
DROP TABLE IF EXISTS my_table;
```

Here, the SQL command will only drop a table if there is one with a same name.

Recommended Reading

- SQL Delete and Truncate Rows
- SQL CREATE TABLE
- SQL ALTER TABLE

Previous Tutorial: SQL Drop Database

Next Tutorial: SQL Alter Table

Did you find this article helpful?

Related Topics

SQL DROP DATABASE Statement
SQL ALTER TABLE Statement
SQL Delete and Truncate Rows
SQL CREATE INDEX
SQL CREATE TABLE Statement
SQL EXISTS

Related Tutorials

Programming
SQL DROP DATABASE Statement

Programming
SQL ALTER TABLE Statement

Programming
SQL Delete and Truncate Rows

Programming
SQL CREATE INDEX

SQL ALTER TABLE Statement

In this tutorial, we'll learn to change table structure with the help of examples.

We can change the structure of a table using the `ALTER TABLE` command. We can

- Add a column
- Rename a column
- Modify a column
- Delete a column
- Rename a table

Add Column in a Table

We can add columns in a table using the `ALTER TABLE` command with the `ADD` clause. For example,

```
ALTER TABLE Customers
ADD phone varchar(10);
```

Run Code »

Here, the SQL command adds a column named `phone` in the `Customers` table.

Add Multiple Columns in a Table

We can also add multiple columns at once in a table. For example,

```
ALTER TABLE Customers
ADD phone varchar(10), age int;
```

Here, the SQL command adds the `phone` and `age` column in the `Customers` table.

Rename Column in a Table

We can rename columns in a table using the `ALTER TABLE` command with the `RENAME COLUMN` clause. For example,

```
ALTER TABLE Customers
RENAME COLUMN customer_id TO c_id;
```

Run Code »

Here, the SQL command changes the column name of `customer_id` to `c_id` in the `Customers` table.

Modify Column in a Table

We can also change the column's data type using the `ALTER TABLE` command with `MODIFY` or `ALTER COLUMN` clause. For example,

SQL Server

```
ALTER TABLE Customers
ALTER COLUMN age VARCHAR(2);
```

MySQL

```
ALTER TABLE Customers
MODIFY COLUMN age VARCHAR(2);
```

Oracle

```
ALTER TABLE Customers
MODIFY age VARCHAR(2);
```

PostgreSQL

```
ALTER TABLE Customers
ALTER COLUMN age TYPE VARCHAR(2);
```

Here, the SQL command changes the data type of the `age` column to `VARCHAR` in the `Customers` table.

Drop Column in a Table

We can also drop (remove) columns in a table using the `ALTER TABLE` command with the `DROP` clause. For example,

```
ALTER TABLE Customers
DROP COLUMN age;
```

Run Code »

Here, the SQL command removes the `phone` column from the `Customers` table.

Rename a Table

We can change the name of a table using the `ALTER TABLE` command with the `RENAME` clause. For example,

```
ALTER TABLE Customers
RENAME TO newCustomers;
```

Run Code »

Here, the SQL command renames the `Customers` table to `newCustomers`.

Previous Tutorial: SQL Drop Table
Next Tutorial: SQL Backup Database

Related Tutorials

Programming — SQL NOT NULL Constraint

Programming — SQL DEFAULT Constraint

Programming — SQL UNIQUE Constraint

Programming — SQL FOREIGN KEY

SQL BACKUP DATABASE Statement

In this tutorial, we'll learn about backing up databases with the help of examples.

It is important to create database backups regularly so that our data won't get lost if the database gets corrupted.

In SQL, we can create database backups using the `BACKUP DATABASE` statement. For example,

```
BACKUP DATABASE my_db
TO DISK = 'C:\my_db_backup.bak';
```

Here, the SQL command creates a backup file of the `my_db` database inside **C** drive, named `my_db_backup.bak`.

> **Note:** It's a common convention to use the **.bak** file extension for database backup files, however, it's not mandatory.

Backup Only New Changes in SQL

In SQL, we can also backup only the new changes compared with previous backup by using the `WITH DIFFERENTIAL` command. For example,

```
BACKUP DATABASE my_db
TO DISK = 'C:\my_db_backup.bak'
WITH DIFFERENTIAL;
```

Here, the SQL command appends only new changes to the previous backup file. Hence, this command may work faster.

Restore Database From Backup

To restore a backup file to the database management system, we can use the `RESTORE DATABASE` statement. For example,

```
RESTORE DATABASE my_db
FROM DISK = 'C:\my_db_backup.bak';
```

Here, the SQL command restores the `my_db_backup.bak` file in the database named `my_db`.

Previous Tutorial: SQL Alter Table

Next Tutorial: SQL Insert Into

Related Tutorials

Programming — SQL CREATE DATABASE Statement

Programming — SQL DROP DATABASE Statement

Programming — SQL SELECT INTO Statement

Programming — Introduction to Databases and SQL

SQL INSERT INTO Statement

In this tutorial, we'll learn to insert a row in the table with the help of examples.

In SQL, the `INSERT INTO` statement is used to insert new row(s) in a database table. For example,

```
INSERT INTO Customers(customer_id, first_name, last_name, age, country)
VALUES
(5, 'Harry', 'Potter', 31, 'USA');
```

Run Code »

Here, the SQL command inserts a new row in the `Customers` table with the given values.

Table: Customers

customer_id	first_name	last_name	age	country
1	John	Doe	31	USA
2	Robert	Luna	22	USA
3	David	Robinson	22	UK
4	John	Reinhardt	25	UK

```
INSERT INTO Customers(customer_id, first_name,
last_name, age, country)
VALUES (5, 'Harry', 'Potter', 31, 'USA');
```

customer_id	first_name	last_name	age	country
1	John	Doe	31	USA
2	Robert	Luna	22	USA
3	David	Robinson	22	UK
4	John	Reinhardt	25	UK
5	Harry	Potter	31	USA

Example: SQL Insert Into

Note: If we need to insert data from any other existing table , we can use the SQL INSERT INTO SELECT statement.

Insert Row Providing Value Explicitly

It's possible to provide default values to a column (for example, auto incrementing a column). In a database table, the **ID** field is usually unique auto incremented.

In such cases, we can omit the value for that column during row insertion. For example,

```
INSERT INTO Customers(first_name, last_name, age, country)
VALUES
('James', 'Bond', 48, 'USA');
```

Run Code »

Here, the SQL command automatically sets the new `customer_id` for the new row and inserts it in a table.

Table: Customers

customer_id	first_name	last_name	age	country
1	John	Doe	31	USA
2	Robert	Luna	22	USA
3	David	Robinson	22	UK
4	John	Reinhardt	25	UK

```
INSERT INTO Customers(first_name, last_name,
age, country)
VALUES ('James', 'Bond', 48, 'USA');
```

customer_id	first_name	last_name	age	country
1	John	Doe	31	USA
2	Robert	Luna	22	USA
3	David	Robinson	22	UK
4	John	Reinhardt	25	UK
5	James	Bond	48	USA

Example: SQL INSERT INTO

Insert Multiple Rows at Once in SQL

It's also possible to insert multiple rows to a database table at once. For example,

```
INSERT INTO Customers(first_name, last_name, age, country)
VALUES
('Harry', 'Potter', 31, 'USA'),
('Chris', 'Hemsworth', 43, 'USA'),
('Tom', 'Holland', 26, 'UK');
```

Run Code »

Here, the SQL command inserts three rows to the `Customers` table.

SQL INSERT Topics

- Insert rows Without Specifying Column Names.
- Not Including All Columns During Insertion.

Previous Tutorial: SQL Backup Database

Next Tutorial: SQL Update

Did you find this article helpful?

Related Tutorials

Programming — SQL INSERT INTO SELECT Statement

Programming — SQL UPDATE Statement

Programming — SQL NOT NULL Constraint

Programming — SQL SELECT INTO Statement

SQL UPDATE Statement

In this tutorial, we'll learn to update rows in SQL with the help of examples.

The SQL `UPDATE` statement is used to edit existing rows in a database table. For example,

```
UPDATE Customers
SET first_name = 'Johnny'
WHERE customer_id = 1;
```

Run Code

Here, the SQL command changes the value of the `first_name` column will be **Johnny** if `customer_id` is equal to **1**.

Table: Customers

customer_id	first_name	last_name	age	country
1	John	Doe	31	USA
2	Robert	Luna	22	USA
3	David	Robinson	22	UK
4	John	Reinhardt	25	UK
5	Betty	Doe	28	UAE

```
UPDATE Customers
SET first_name = 'Johnny'
WHERE customer_id = 1;
```

customer_id	first_name	last_name	age	country
1	Johnny	Doe	31	USA
2	Robert	Luna	22	USA
3	David	Robinson	22	UK
4	John	Reinhardt	25	UK
5	Harry	Potter	31	USA

Example: SQL UPDATE Statement

> **Note:** If we need to insert a new row instead of updating an existing row, we can use the SQL INSERT INTO statement.

Update Multiple Values in a Row

We can also update multiple values in a row at once. For example,

```
UPDATE Customers
SET first_name = 'Johnny', last_name = 'Depp'
WHERE customer_id = 1;
```

Run Code

Here, the SQL command changes the value of the `first_name` column to **Johnny** and `last_name` to **Depp** if `customer_id` is equal to **1**.

Update Multiple Rows

The `UPDATE` statement can update multiple rows at once. For example,

```
UPDATE Customers
SET country = 'NP'
WHERE age = 22;
```

Run Code

Here, the SQL command changes the value of the country column to **NP** if `age` is **22**. If there are more than one rows with `age` equals to **22**, all the matching rows will be edited.

Update all Rows

We can update all the rows in a table at once by omitting the `WHERE` clause. For example,

```
UPDATE Customers
SET country = 'NP';
```

Run Code

Here, the SQL command changes the value of the `country` column to **NP** for all rows.

> **Note:** We should be cautious while using the `UPDATE` statement. If we omit the `WHERE` clause, all the rows will be changed and this change is irreversible.

SQL Update With JOIN

We can also use the `UPDATE` statement with the JOIN clause in SQL.

Take a look at the Stackoverflow thread on, How to use the UPDATE statement with JOIN?

Previous Tutorial: SQL Insert Into

Next Tutorial: SQL Select Into

Did you find this article helpful?

Related Tutorials

Programming — SQL INSERT INTO SELECT Statement

Programming — SQL SELECT INTO Statement

Programming — SQL Delete and Truncate Rows

Programming — SQL INSERT INTO Statement

SQL SELECT INTO Statement

In this tutorial, we'll learn to copy data from one table to another in SQL with the help of examples.

In SQL, we can copy data from one database table to a new table using the `SELECT INTO` command. For example,

```
SELECT *
INTO CustomersCopy
FROM Customers;
```

Here, the SQL command copies all data from the `Customers` table to the new `CustomersCopy` table.

> **Note:** The `SELECT INTO` statement creates a new table. If the database already has a table with the same name, `SELECT INTO` gives an error.
>
> If we want to copy data to an existing table (rather than creating a new table), we should use the INSERT INTO SELECT statement.

Copy Selected Columns Only

We can also copy only selected columns from the old table to a new table. For example,

```
SELECT customer_id, country
INTO CustomersCountry
FROM Customers;
```

Here, the SQL command only copies `customer_id` and `country` columns to the `CustomersCopy` table.

Copy Records Matching a Condition

We can use the `WHERE` clause with `SELECT INTO` to copy those rows that match the specified condition. For example,

```
SELECT customer_id, age
INTO USACustomersAge
FROM Customers
WHERE country = 'USA';
```

Here, the SQL command

1. creates the `USACustomersAge` table with `customer_id` and `age` column
2. copies the rows to the new table if the value of the `country` column is **USA**.

Copy to Another Database

By default, `SELECT INTO` creates a new table in the current database. If we want to copy data to a table in a different database, we can do that by using the `IN` clause. For example,

```
SELECT *
INTO CustomersCopy
IN another_db.mdb
FROM Customers;
```

Here, the SQL command copies the `Customers` table to the `CustomersCopy` table in the `another_db.mdb` database.

> **Note:** The user must have **WRITE** privilege to copy data to a table in a different database.

Copy From Two Tables to One

We can also copy records from two different tables to a new table using the `JOIN` clause with `SELECT INTO`. For example,

```
SELECT Customers.customer_id, Customers.first_name, Orders.amount
INTO CustomerOrders
FROM Customers
JOIN Orders
ON Customers.customer_id = Orders.customer_id;
```

Here, the SQL command copies `customer_id` and `first_name` from the `Customers` table and the `amount` from the `Orders` table to a new table `CustomerOrders`. To learn more, visit SQL JOIN.

Copy Table Schema Only

We can also use the `SELECT INTO` statement to create a new table with the given schema (without coping the data). For that, we use the `WHERE` clause with a condition that returns false.

```
SELECT *
INTO NewCustomers
FROM Customers
WHERE false;
```

Here, the SQL command creates an empty table named `NewCustomers` with the same structure as the `Customers` table.

Recommended Readings

- SQL CREATE TABLE AS Statement
- SQL BACKUP DATABASE Statement

Previous Tutorial: SQL Update

Next Tutorial: SQL Select Into Insert

Related Tutorials

Programming — SQL INSERT INTO SELECT Statement

Programming — SQL SELECT

Programming — SQL JOIN

Programming — SQL Subquery

SQL INSERT INTO SELECT Statement

In this tutorial, we'll learn to copy records from one table to another with the help of examples.

The `INSERT INTO SELECT` statement is used to copy records from one table to another existing table. For example,

```
INSERT INTO OldCustomers
SELECT *
FROM Customers;
```

Run Code »

Here, the SQL command copies all records from the `Customers` table to the `OldCustomers` table.

> **Note:** To run this command,
>
> Old
> - the database must already have a table named `OldCustomers`
> - the column names of the `OldCustomers` table and the `Customers` table must match
>
> If we want to copy data to a new table (rather than copying in an existing table), we should use the SELECT INTO statement.

Copy Selected Columns Only

We can also copy only the selected columns from one table to another. For example,

```
INSERT INTO OldCustomers(customer_id, age)
SELECT customer_id, age
FROM Customers;
```

Run Code »

Here, the SQL command only copies records from the `customer_id` column and `country` column to the `OldCustomers` table.

> **Note:** If there are columns other than `customer_id` and `age` in the `OldCustomers` table, the value of those columns will be `NULL`.

Copy Records Matching a Condition

We can use the `WHERE` clause with `INSERT INTO` to copy those rows that match the specified condition. For example,

```
INSERT INTO OldCustomers
SELECT *
FROM Customers
WHERE country = 'USA';
```

Run Code »

Here, the SQL command copies rows that are the value of the `country` column as **USA**.

Copy From two Tables to One

We can also copy records from two different tables using the `JOIN` clause with `INSERT INTO SELECT`. For example,

```
INSERT INTO OldCustomerOrders
SELECT Customers.customer_id, Customers.first_name, Orders.amount
FROM Customers
JOIN Orders
ON Customers.customer_id = Orders.customer_id;
```

Run Code »

Here, the SQL command copies `customer_id` and `first_name` from the `Customers` table and the `amount` from the `Orders` table in an existing table `OldCustomerOrders`. To learn more, visit SQL JOIN.

> **Note:** If records are already present in an existing table, new rows will be appended. Columns in the existing table may throw errors such as `NOT NULL Constraint Failed`, `UNIQUE Constraint Failed` while copying data.

Avoid Duplicates in INSERT INTO SELECT

If there is already a row with a similar value, SQL may throw an error while using the `INSERT INTO SELECT` command.

However, we can skip copying duplicate rows using the `NOT EXISTS` clause. For example,

```
INSERT INTO OldCustomers(customer_id, age)
SELECT customer_id, age
FROM Customers
WHERE NOT EXISTS(
  SELECT customer_id
  FROM OldCustomers
  WHERE OldCustomers.customer_id = Customers.customer_id
);
```

Run Code »

Here, the SQL command will only copy row to a new table if the `customer_id` does not have the same value.

Recommended Readings

- SQL INSERT INTO Statement

Previous Tutorial: SQL Select Into
Next Tutorial: SQL Delete and Truncate Rows

Did you find this article helpful?

Related Tutorials

Programming — SQL SELECT INTO Statement
Programming — SQL JOIN
Programming — SQL Subquery
Programming — SQL Views

SQL Delete and Truncate Rows

In this tutorial, we'll learn to delete rows from a table in SQL with the help of examples.

In SQL, we use the `DELETE` statement to delete row(s) from a database table. For example,

```
DELETE FROM Customers
WHERE customer_id = 5;
```

Run Code »

Here, the SQL command will delete a row from the `Customers` table where `customer_id` is **5**.

Table: Customers

customer_id	first_name	last_name	age	country
1	John	Doe	31	USA
2	Robert	Luna	22	USA
3	David	Robinson	22	UK
4	John	Reinhardt	25	UK
5	Betty	Doe	28	UAE

DELETE FROM Customers
WHERE customer_id = 5;

customer_id	first_name	last_name	age	country
1	John	Doe	31	USA
2	Robert	Luna	22	USA
3	David	Robinson	22	UK
4	John	Reinhardt	25	UK

Example: SQL DELETE command

Delete all Rows in a Table

The `WHERE` clause determines which rows to delete. However, we can delete all rows at once if we omit the `WHERE` clause. For example,

```
DELETE FROM Customers;
```

Run Code »

Here, the SQL command deletes all rows from a table.

> **Note:** Be careful when you use `DELETE`. Records may lose permanently if the database is not backed up.

Truncate Table in SQL

The `TRUNCATE TABLE` clause is another way to delete all rows from a table at once. For example,

```
TRUNCATE TABLE Customers;
```

Here, the SQL command does exactly the same thing the above command does.

> **Note**: The `TRUNCATE` clause doesn't support the `WHERE` clause.

Delete Vs Truncate

The main difference between both statements is that `DELETE FROM` statement supports `WHERE` clause whereas `TRUNCATE` does not.

That means, we can delete single or multiple rows using the `DELETE FROM` statement while the `TRUNCATE` statement deletes all records from the table at once.

We can mimic the `TRUNCATE` statement with `DELETE FROM` statement by omitting the `WHERE` clause. For example,

```
DELETE FROM Customers;
```

Run Code »

is similar to,

```
TRUNCATE TABLE Customers;
```

Previous Tutorial: SQL Select Into Insert

Next Tutorial: SQL Constraints

Did you find this article helpful?

Related Tutorials

Programming — SQL DROP TABLE Statement
Programming — SQL SELECT INTO Statement
Programming — SQL ALTER TABLE Statement
Programming — SQL INSERT INTO SELECT Statement

SQL Constraints

In this tutorial, we'll learn about constraints in SQL with the help of examples.

In a database table, we can add rules to a column known as **constraints**. These rules control the data that can be stored in a column.

For example, if a column has `NOT NULL` constraint, it means the column cannot store `NULL` values.

The constraints used in SQL are:

Constraint	Description
`NOT NULL`	values cannot be null
`UNIQUE`	values cannot match any older value
`PRIMARY KEY`	used to uniquely identify a row
`FOREIGN KEY`	references a row in another table
`CHECK`	validates condition for new value
`DEFAULT`	set default value if not passed
`CREATE INDEX`	used to speedup the read process

> **Note:** These constraints are also called integrity constraints.

NOT NULL Constraint

The `NOT NULL` constraint in a column means that the column cannot store `NULL` values. For example,

```
CREATE TABLE Colleges (
  college_id INT NOT NULL,
  college_code VARCHAR(20) NOT NULL,
  college_name VARCHAR(50)
);
```

Run Code »

Here, the `college_id` and the `college_code` columns of the `Colleges` table won't allow `NULL` values.

To learn more, visit SQL NOT NULL Constraint.

UNIQUE Constraint

The `UNIQUE` constraint in a column means that the column must have unique value. For example,

```
CREATE TABLE Colleges (
  college_id INT NOT NULL UNIQUE,
  college_code VARCHAR(20) UNIQUE,
  college_name VARCHAR(50)
);
```

Run Code »

Here, the value of the `college_code` column must be unique. Similarly, the value of `college_id` must be unique as well as it cannot store `NULL` values.

To learn more, visit SQL UNIQUE Constraint.

PRIMARY KEY Constraint

The `PRIMARY KEY` constraint is simply a combination of `NOT NULL` and `UNIQUE` constraints. It means that the column value is used to uniquely identify the row. For example,

```
CREATE TABLE Colleges (
  college_id INT PRIMARY KEY,
  college_code VARCHAR(20) NOT NULL,
  college_name VARCHAR(50)
);
```

Run Code »

Here, the value of the `college_id` column is a unique identifier for a row. Similarly, it cannot store `NULL` value and must be `UNIQUE`.

To learn more, visit SQL PRIMARY KEY.

FOREIGN KEY Constraint

The `FOREIGN KEY ( REFERENCES` in some databases) constraint in a column is used to reference a record that exists in another table. For example,

```
CREATE TABLE Orders (
  order_id INT PRIMARY KEY,
  customer_id int REFERENCES Customers(id)
);
```

Run Code »

Here, the value of the `college_code` column references the row in another table named `Customers`.

It means that the value of `customer_id` in the `Orders` table must be a value from the `id` column of the `Customers` table.

To learn more, visit SQL FOREIGN KEY.

CHECK Constraint

The `CHECK` constraint checks the condition before allowing values in a table. For example,

```
CREATE TABLE Orders (
  order_id INT PRIMARY KEY,
  amount int CHECK (amount >= 100)
);
```

Run Code »

Here, the value of the `amount` column must be **greater than or equal to 100**. If not, the SQL statement results in an error.

To learn more, visit SQL CHECK Constraint.

DEFAULT Constraint

The `DEFAULT` constraint is used to set the default value if we try to store `NULL` in a column. For example,

```
CREATE TABLE College (
  college_id INT PRIMARY KEY,
  college_code VARCHAR(20),
  college_country VARCHAR(20) DEFAULT 'US'
);
```

Run Code »

Here, the default value of the `college_country` column is **US**.

If we try to store the `NULL` value in the `college_country` column, its value will be **US**.

To learn more, visit SQL DEFAULT Constraint.

CREATE INDEX Constraint

If a column has `CREATE INDEX` constraint, it's faster to retrieve data if we use that column for data retrieval. For example,

```
-- create table
CREATE TABLE Colleges (
  college_id INT PRIMARY KEY,
  college_code VARCHAR(20) NOT NULL,
  college_name VARCHAR(50)
);

-- create index
CREATE INDEX college_index
ON Colleges(college_code);
```

Run Code »

Here, the SQL command creates an index named `customers_index` on the `Customers` table using `customer_id` column.

> **Note:** We cannot see the speed difference with less records in a table. However, we can easily notice the speed difference between using indexes and not using indexes.

To learn more, visit SQL CREATE INDEX.

Previous Tutorial: SQL Delete and Truncate Rows

Next Tutorial: SQL Not Null Constraint

Related Tutorials

Programming — SQL UNIQUE Constraint
Programming — SQL NOT NULL Constraint
Programming — SQL PRIMARY KEY
Programming — SQL DEFAULT Constraint

SQL NOT NULL Constraint

In this tutorial, we'll learn to use the NOT NULL constraint with the help of examples.

The `NOT NULL` constraint in a column means that the column cannot store `NULL` values. For example,

```
CREATE TABLE Colleges (
  college_id INT NOT NULL,
  college_code VARCHAR(20),
  college_name VARCHAR(50)
);
```

Run Code »

Here, the `college_id` and the `college_code` columns of the `Colleges` table won't allow `NULL` values.

> **Note:** The `NOT NULL` constraint is used to add a constraint table whereas IS NULL and NOT NULL are used with the WHERE clause to select rows from the table.

Remove NOT NULL Constraint

We can also remove the NOT NULL constraint if that is no longer needed. For example,

SQL Server

```
ALTER TABLE Customers
ALTER COLUMN age INT;
```

Oracle

```
ALTER TABLE Customers
MODIFY (age NULL);
```

MySQL

```
ALTER TABLE Customers
MODIFY age INT;
```

PostgreSQL

```
ALTER TABLE Customer
ALTER COLUMN age DROP NOT NULL;
```

NOT NULL Constraint With Alter Table

We can also add the `NOT NULL` constraint to a column in an existing table using the ALTER TABLE command. For example,

SQL Server

```
ALTER TABLE Customers
ALTER COLUMN age INT NOT NULL;
```

Oracle

```
ALTER TABLE Customers
MODIFY age INT NOT NULL;
```

MySQL

```
ALTER TABLE Customers
MODIFY COLUMN age INT NOT NULL;
```

PostgreSQL

```
ALTER TABLE Customers
ALTER COLUMN age SET NOT NULL;
```

Here, the SQL command adds the `NOT NULL` constraint to the column `college_id` in an existing table.

Now when we try to insert records in a Colleges table without value for `college_id` , SQL will give us an error. For example,

```
INSERT INTO Colleges(college_code, college_name)
VALUES ('NYC', "US");
```

Run Code »

Here, the SQL command gives us an error because we cannot skip the `college_id` field in a table because of the `NOT NULL` constraint.

Recommended Readings

- SQL Constraints
- SQL INSERT INTO
- SQL PRIMARY KEY
- SQL FOREIGN KEY

Previous Tutorial: SQL Constraints

Next Tutorial: SQL Unique Constraints

Did you find this article helpful?

Related Tutorials

Programming — SQL DEFAULT Constraint

Programming — SQL UNIQUE Constraint

Programming — SQL PRIMARY KEY

Programming — SQL ALTER TABLE Statement

SQL UNIQUE Constraint

In this tutorial, we'll learn to use the UNIQUE constraint with the help of examples.

In SQL, the `UNIQUE` constraint in a column means that the column must have unique values. For example,

```
CREATE TABLE Colleges (
  college_id INT NOT NULL UNIQUE,
  college_code VARCHAR(20) UNIQUE,
  college_name VARCHAR(50)
);
```

Run Code »

Here, the values of the `college_code` column must be unique. Similarly, the values of `college_id` must be unique as well as it cannot store `NULL` values.

UNIQUE Vs DISTINCT

The `UNIQUE` constraint is used to make column's value unique. However, to select unique rows from the table, we have to use SQL SELECT DISTINCT. For example,

```
SELECT DISTINCT country
FROM Customers;
```

Run Code »

Here, the SQL command selects unique countries from the Customers table.

Count UNIQUE Rows

If we need to count the number of unique rows, we can use the COUNT() function with the SELECT DISTINCT clause. For example,

```
SELECT COUNT(DISTINCT country)
FROM Customers;
```

Run Code »

Here, the SQL command returns the count of unique countries.

UNIQUE Constraint With Alter Table

We can also add the `UNIQUE` constraint to an existing column using the ALTER TABLE command. For example,

For single column

```
ALTER TABLE Colleges
ADD UNIQUE (college_id);
```

For multiple column

```
ALTER TABLE Colleges
ADD UNIQUE UniqueCollege (college_id, college_code);
```

Here, the SQL command adds the `UNIQUE` constraint to the specified column(s) in an existing table.

Error When Inserting Duplicate Values

If we try to insert duplicate values in a column with the `UNIQUE` constraint, we will get an error.

```
CREATE TABLE Colleges (
  college_id INT NOT NULL UNIQUE,
  college_code VARCHAR(20) UNIQUE,
  college_name VARCHAR(50)
);

-- Inserting values to the Colleges table
INSERT INTO Colleges(college_id, college_code, college_name)
VALUES (1, "ARD12", "Star Public School"), (2, "ARD12", "Galaxy School");
```

Run Code »

Here, we are trying to insert **ARD12** in the `college_code` column in two different rows. Hence, the INSERT INTO command results in an error.

CREATE UNIQUE INDEX for Unique Values

If we want to create indexes for unique values in a column, we use the `CREATE UNIQUE INDEX` constraint. For example,

```
-- create unique index
CREATE UNIQUE INDEX college_index
ON Colleges(college_code);
```

Run Code »

Here, the SQL command creates a unique index named `college_index` on the `Colleges` table using the `college_code` column.

> **Note:** Although the index is created for only unique values, the original data in the table remains unaltered.

Recommended Readings

- SQL CREATE INDEX
- SQL Constraints
- SQL NOT NULL Constraint
- SQL PRIMARY KEY

Previous Tutorial: SQL Not Null Constraint

Next Tutorial: SQL Primary Key

Did you find this article helpful?

Related Tutorials

Programming — SQL CREATE INDEX

Programming — SQL PRIMARY KEY

Programming — SQL Constraints

Programming — SQL NOT NULL Constraint

SQL PRIMARY KEY

In this tutorial, we'll learn about the PRIMARY KEY in SQL and how to use them with the help of examples.

In SQL, the `PRIMARY KEY` constraint is used to uniquely identify rows.

The `PRIMARY KEY` constraint is simply a combination of `NOT NULL` and `UNIQUE` constraints. Meaning, the column cannot contain duplicate as well as `NULL` values.

Primary Key Syntax

```
CREATE TABLE Colleges (
  college_id INT,
  college_code VARCHAR(20) NOT NULL,
  college_name VARCHAR(50),
  CONSTRAINT CollegePK PRIMARY KEY (college_id)
);
```

Run Code »

Here, the `college_id` column is the `PRIMARY KEY`. This means, the values of this column must be unique as well as it cannot contain `NULL` values.

> **Note:** The above code works in all major database systems. However, depending on a database, there may be alternative syntaxes to create the primary key.

Primary Key Error

If we try to insert null or duplicate values in the `college_id` column--in the above table--we will get an error. For example,

```
-- Insertion Success
INSERT INTO Colleges(college_id, college_code, college_name)
VALUES (1, "ARD12", "Star Public School");

-- UNIQUE Constraint Error
-- The value of college_id is not unique
INSERT INTO Colleges(college_id, college_code, college_name)
VALUES (1, "ARD12", "Star Public School");
```

Run Code »

Here, the SQL command gives us an error because we cannot insert same value for the `college_id` field in a table because of the `UNIQUE` constraint.

> **Note:** In a table, there can be only one primary key.

Primary Key With Multiple Columns

A primary key may also be made up of multiple columns. For example,

```
CREATE TABLE Colleges (
  college_id INT,
  college_code VARCHAR(20),
  college_name VARCHAR(50),
  CONSTRAINT CollegePK PRIMARY KEY (college_id, college_code)
);
```

Run Code »

Here, the `PRIMARY KEY` constraint named `CollegePK` is made up of `college_id` and `college_code` columns.

This means, the combination of `college_id` and `college_code` must be unique as well as these two columns cannot contain `NULL` values.

Now let's try to insert records in the `Colleges` table,

```
-- Insertion Success
INSERT INTO Colleges(college_id, college_code, college_name)
VALUES (1, "ARD12", "Star Public School");

-- Insertion Success
INSERT INTO Colleges(college_id, college_code, college_name)
VALUES (1, "ARD13", "Star Public School");

-- Insertion Success
INSERT INTO Colleges(college_id, college_code, college_name)
VALUES (2, "ARD12", "Star Public School");

-- UNIQUE Constraint Error
-- A row already has 1 as college_id and "ARD12" as college_code
INSERT INTO Colleges(college_id, college_code, college_name)
VALUES (1, "ARD12", "Star Public School");
```

Run Code »

Primary Key Constraint With Alter Table

We can also add the `PRIMARY KEY` constraint to a column in an existing table using the `ALTER TABLE` command. For example,

For single column

```
ALTER TABLE Colleges
ADD PRIMARY KEY (college_id);
```

For multiple column

```
ALTER TABLE Colleges
ADD CONSTRAINT CollegePK PRIMARY KEY (college_id, college_code);
```

Here, the SQL command adds the `PRIMARY KEY` constraint to the specified column(s) in the existing table.

Auto Increment Primary Key

It is a common practice to automatically increase the value of the primary key when a new row is inserted. For example,

SQL Server

```
-- using IDENTITY(x, y) to auto increment the value
-- x -> start value, y -> steps to increase
CREATE TABLE Colleges (
  college_id INT IDENTITY(1,1),
  college_code VARCHAR(20) NOT NULL,
  college_name VARCHAR(50),
  CONSTRAINT CollegePK PRIMARY KEY (college_id)
);

-- inserting record without college_id
INSERT INTO Colleges(college_code, college_name)
VALUES ("ARD13", "Star Public School");
```

Oracle

```
-- creating sequence of numbers
CREATE SEQUENCE auto_inc
MINVALUE 1
START WITH 1
INCREMENT BY 1
CACHE 10;

CREATE TABLE Colleges (
  college_id INT,
  college_code VARCHAR(20) NOT NULL,
  college_name VARCHAR(50),
  CONSTRAINT CollegePK PRIMARY KEY (college_id)
);

-- creating trigger before insert to
-- add auto incremented value
CREATE TRIGGER auto_inc_trigger
BEFORE INSERT ON Colleges
FOR EACH ROW
BEGIN
SELECT auto_inc.nextval INTO :new.college_id FROM dual
END;

-- inserting record without college_id
INSERT INTO Colleges(college_code, college_name)
VALUES ("ARD13", "Star Public School");
```

MySQL

```
-- AUTO_INCREMENT keyword auto increments the value
CREATE TABLE Colleges (
  college_id INT AUTO_INCREMENT,
  college_code VARCHAR(20) NOT NULL,
  college_name VARCHAR(50),
  CONSTRAINT CollegePK PRIMARY KEY (college_id)
);

-- inserting record without college_id
INSERT INTO Colleges(college_code, college_name)
VALUES ("ARD13", "Star Public School");
```

PostgreSQL

```
-- SERIAL keyword auto increments the value
CREATE TABLE Colleges (
  college_id INT SERIAL,
  college_code VARCHAR(20) NOT NULL,
  college_name VARCHAR(50),
  CONSTRAINT CollegePK PRIMARY KEY (college_id)
);

-- inserting record without college_id
INSERT INTO Colleges(college_code, college_name)
VALUES ("ARD13", "Star Public School");
```

Remove Primary Key Constraint

We can remove the `PRIMARY KEY` constraint in a table using the `DROP` clause. For example,

SQL Server, Oracle

```
ALTER TABLE Colleges
DROP CONSTRAINT CollegePK;
```

MySQL

```
ALTER TABLE Colleges
DROP PRIMARY KEY;
```

Here, the SQL command removes the `PRIMARY KEY` constraint from the `Colleges` table.

Recommended Reading:

- SQL FORIEGN KEY
- SQL Constraints

Previous Tutorial: SQL Unique Constraints
Next Tutorial: SQL Foreign Key

Did you find this article helpful?

Related Tutorials

Programming — SQL UNIQUE Constraint
Programming — SQL NOT NULL Constraint
Programming — SQL Constraints
Programming — SQL DEFAULT Constraint

SQL FOREIGN KEY

In this tutorial, we'll learn about the FOREIGN KEY in SQL and how to use them with the help of examples.

In SQL, we can create a relationship between two tables using the `FOREIGN KEY` constraint.

Table: Orders

Foreign Key

order_id	product	total	customer_id
1	Paper	500	5
2	Pen	10	2
3	Marker	120	3
4	Books	1000	1
5	Erasers	20	4

Table: Customers

id	first_name	last_name	age	country
1	John	Doe	31	USA
2	Robert	Luna	22	USA
3	David	Robinson	22	UK
4	John	Reinhardt	25	UK
5	Betty	Doe	28	UAE

Example: Foreign Key in SQL

Here, the `customer_id` field in the `Orders` table is `FOREIGN KEY` which references the `id` field in the `Customers` table.

This means that the value of the `customer_id` (of the `Orders` table) must be a value from the `id` column (of the `Customers` table).

> **Note:** The Foreign key can be referenced to any column in the parent table. However, it is general practice to reference the foreign key to the primary key of the parent table.

Related Topics
SQL Constraints
SQL NOT NULL Constraint
SQL PRIMARY KEY
SQL CREATE TABLE Statement
SQL ALTER TABLE Statement
SQL CHECK Constraint

Creating FOREIGN Key

Now, let's see how we can create foreign key constraints in a database.

```
-- This table doesn't have a foreign key
CREATE TABLE Customers (
  id INT,
  first_name VARCHAR(40),
  last_name VARCHAR(40),
  age INT,
  country VARCHAR(10),
  CONSTRAINT CustomersPK PRIMARY KEY (id)
);

-- Adding foreign key to the customer_id field
-- The foreign key references to the id field of the Customers table
CREATE TABLE Orders (
  order_id INT,
  item VARCHAR(40),
  amount INT,
  customer_id INT REFERENCES Customers(id),
  CONSTRAINT OrdersPK PRIMARY KEY (order_id)
);
```

Run Code »

Here, the value of the `customer_id` column in the `Orders` table references the row in another table named `Customers` with its `id` column.

> **Note:** The above code works in all major database systems. However, there may be the alternate syntax to create foreign keys depending on the database. Refer to their respective database documentation for more information.

Inserting Records in Table with Foreign Key

Lets try to insert records in a table with foreign keys.

```
-- Inserting record in table with no foreign key first
INSERT INTO Customers
VALUES
(1, 'John', 'Doe', 31, 'USA'),
(2, 'Robert', 'Luna', 22, 'USA');

-- Insertion Success
INSERT INTO Orders
VALUES
(1, 'Keyboard', 400, 2),
(2, 'Mouse', 300, 2),
(3, 'Monitor', 12000, 1);

-- Insertion Error because customer with id of 7 does not exist
INSERT INTO Orders
VALUES (4, 'Keyboard', 400, 7);
```

Run Code »

Why use Foreign Key?

To normalize data

The `FOREIGN KEY` helps us to normalize the data in multiple tables and reduce the redundancy. This means, a database can have multiple tables that are related to each other.

Prevent Wrong Data From Insertion

If two database tables are related through a field (attribute), using `FOREIGN KEY` makes sure that wrong data is not inserted in that field. This helps to eliminate bugs in the database level.

Recommended Reading: SQL JOIN

Foreign Key with Alter Table

It is possible to add the `FOREIGN KEY` constraint to an existing table using the `ALTER TABLE` command. For example,

```
CREATE TABLE Customers (
  id INT,
  first_name VARCHAR(40),
  last_name VARCHAR(40),
  age INT,
  country VARCHAR(10),
  CONSTRAINT CustomersPK PRIMARY KEY (id)
);

CREATE TABLE Orders (
  order_id INT,
  item VARCHAR(40),
  amount INT,
  customer_id INT,
  CONSTRAINT OrdersPK PRIMARY KEY (order_id)
);

-- Adding foreign key to the customer_id field
-- The foreign key references to the id field of the Customers table
ALTER TABLE Orders
ADD FOREIGN KEY (customer_id) REFERENCES Customers(id);
```

Multiple Foreign Key in a Table

A database table can also have multiple foreign keys.

Lets say, we have to record all transactions where each user is a buyer and a seller.

```
-- This table doesn't have a foreign key
CREATE TABLE Users (
  id INT,
  first_name VARCHAR(40),
  last_name VARCHAR(40),
  age INT,
  country VARCHAR(10),
  CONSTRAINT CustomersPK PRIMARY KEY (id)
);

-- Adding foreign key to the buyer and seller field
-- The foreign key references to the id field of the Users table
CREATE TABLE Transactions (
  transaction_id INT,
  amount INT,
  seller INT REFERENCES Users(id),
  buyer INT REFERENCES Users(id),
  CONSTRAINT TransactionsPK PRIMARY KEY (transaction_id)
);
```

Run Code »

Here, the SQL command creates two foreign keys (`buyer` and `seller`) in the `Transactions` table.

Recommended Reading: SQL Constraints

Previous Tutorial: SQL Primary Key
Next Tutorial: SQL Check

Did you find this article helpful?

Related Tutorials

Programming — SQL Constraints
Programming — SQL NOT NULL Constraint
Programming — SQL PRIMARY KEY
Programming — SQL CREATE TABLE Statement

SQL CHECK Constraint

In this tutorial, we'll learn about the CHECK constraint in SQL and how to use them with examples.

In SQL, the `CHECK` constraint is used to specify the condition that must be validated in order to insert data to a table. For example,

```
CREATE TABLE Orders (
  order_id INT PRIMARY KEY,
  amount INT CHECK (amount > 0)
);
```

Run Code »

Here, the amount column has a check condition: **greater than 0**. Now, let's try to insert records to the Orders table.

Example 1

```
-- amount equal to 100
-- record is inserted
INSERT INTO Orders(amount) VALUES(100);
```

Run Code »

Example 2

```
-- amount equal to -5
-- results in an error
INSERT INTO Orders(amount) VALUES(-5);
```

Run Code »

> **Note:** The `CHECK` constraint is used to validate data while insertion only. To check if the row exists or not, visit SQL EXISTS.

Create Named CHECK Constraint

It's a good practice to create **named constraints** so that it is easier to alter and drop constraints.

Here's an example to create named `CHECK` constraint:

```
-- creates a named constraint named amount CK
-- the constraint makes sure that amount is greater than 0
CREATE TABLE Orders (
  order_id INT PRIMARY KEY,
  amount INT,
  CONSTRAINT amountCK CHECK (amount > 0)
);
```

Run Code »

CHECK Constraint in Existing Table

We can add the `CHECK` constraint to an existing table by using the `ALTER TABLE` clause. For example,

```
-- Adding CHECK constraint without name

ALTER TABLE Orders
ADD CHECK (amount > 0);
```

Here's how we can add a named `CHECK` constraint. For example,

```
-- Adding CHECK constraint named amountCK

ALTER TABLE Orders
ADD CONSTRAINT amountCK CHECK (amount > 0);
```

> **Note:** If we try to add the `CHECK` constraint `amount > 0` to a column that already has value less than 0, we will get an error.

Remove CHECK Constraint

We can remove the `CHECK` constraint using the `DROP` clause. For example,

SQL Server, PostgreSQL, Oracle

```
-- removing CHECK constraint named amountCK
ALTER TABLE Orders
DROP CONSTRAINT amountCK;
```

MySQL

```
-- removing CHECK constraint named amountCK
ALTER TABLE Orders
DROP CHECK amountCK;
```

Recommended Readings: SQL Constraints

Previous Tutorial: SQL Foreign Key
Next Tutorial: SQL Default

Related Tutorials

Programming — SQL Constraints
Programming — SQL NOT NULL Constraint
Programming — SQL DEFAULT Constraint
Programming — SQL FOREIGN KEY

SQL DEFAULT Constraint

In this tutorial, we'll learn about the DEFAULT constraint in SQL and how to use them with examples.

In SQL, the `DEFAULT` constraint is used to set a default value if we try to insert an empty value in a column. For example,

```
CREATE TABLE Colleges (
  college_id INT PRIMARY KEY,
  college_code VARCHAR(20),
  college_country VARCHAR(20) DEFAULT 'US'
);
```

Run Code »

Here, the default value of the `college_country` column is **US**.

If we try to store the `NULL` value in the `college_country` column, its value will be **US** by default. For example,

```
-- Inserts 'US' to the college_country column
INSERT INTO Colleges (college_id, college_code)
VALUES (1, 'ARP76');

-- Inserts 'UAE' to the college_country column
INSERT INTO Colleges (college_id, college_code, college_country)
VALUES (2, 'JWS89', 'UAE');
```

Run Code »

DEFAULT Constraint With Alter Table

We can also add the `DEFAULT` constraint to an existing column using the ALTER TABLE command. For example,

SQL Server

```
ALTER TABLE Colleges
ADD CONSTRAINT country_default
DEFAULT 'US' FOR college_country;
```

PostgreSQL

```
ALTER TABLE Colleges
ALTER COLUMN college_code SET DEFAULT 'US';
```

MySQL

```
ALTER TABLE Colleges
ALTER college_country SET DEFAULT 'US';
```

Oracle

```
ALTER TABLE Colleges
MODIFY college_country DEFAULT 'US';
```

Here, the default value of `college_country` column is set to **US** if `NULL` is passed during insertion.

Remove Default Constraint

We can remove the default constraint in a column using the `DROP` clause. For example,

SQL Server, PostgreSQL, Oracle

```
ALTER TABLE Colleges
ALTER COLUMN college_country DROP DEFAULT;
```

MySQL

```
ALTER TABLE Colleges
ALTER college_country DROP DEFAULT;
```

Here, the SQL command removes the `DEFAULT` constraint from the `college_country` column.

Related Topics
- SQL NOT NULL Constraint
- SQL Constraints
- SQL PRIMARY KEY
- SQL UNIQUE Constraint
- SQL CREATE INDEX
- SQL ALTER TABLE Statement

Previous Tutorial: SQL Check

Next Tutorial: SQL Create Index

Did you find this article helpful?

Related Tutorials

Programming — SQL NOT NULL Constraint

Programming — SQL Constraints

Programming — SQL PRIMARY KEY

Programming — SQL CREATE INDEX

SQL CREATE INDEX

In this tutorial, we'll learn about indexes in SQL and how to use them with examples.

In SQL, if a column has `CREATE INDEX` constraint, it's faster to retrieve data if we use that column for data retrieval. For example,

```
-- create table
CREATE TABLE Colleges (
  college_id INT PRIMARY KEY,
  college_code VARCHAR(20) NOT NULL,
  college_name VARCHAR(50)
);

-- create index
CREATE INDEX college_index
ON Colleges(college_code);
```

Run Code »

Here, the SQL command creates an index named `college_index` on the `Colleges` table using the `college_code` column.

> **Note:** Since database systems are very fast by default, the difference in speed is noticeable only when we are working with a table that has a large number of records..

CREATE UNIQUE INDEX for Unique Values

If we want to create indexes for unique values in a column, we use the `CREATE UNIQUE INDEX` constraint. For example,

```
-- create unique index
CREATE UNIQUE INDEX college_index
ON Colleges(college_code);
```

Here, the SQL command creates a unique index named `college_index` on the `Colleges` table using the `college_code` column.

> **Note:** Although the index is created for only unique values, the original data in table remains unaltered.

Remove Index From Tables

To remove **index** from a table, we can use the `DROP INDEX` command. For example,

SQL Server

```
DROP INDEX Colleges.college_index;
```

PostgreSQL, Oracle

```
DROP INDEX college_index;
```

MySQL

```
ALTER TABLE Colleges
DROP INDEX college_index;
```

Here, the SQL command removes the `college_index` constraint from the `Colleges` table.

> **Note:** Deleting an index means the index is only deleted. The data in the original table remains unaltered.

Previous Tutorial: SQL Default

Next Tutorial: SQL Data Types

Did you find this article helpful?

Related Tutorials

Programming — SQL UNIQUE Constraint

Programming — SQL Constraints

Programming — SQL PRIMARY KEY

Programming — SQL DEFAULT Constraint