



What is SQL?

 varshachoudhary[Read](#) [Discuss](#) [Courses](#) [Practice](#) [Video](#)

Structured Query Language is a computer language that we use to interact with a relational database. SQL is a tool for organizing, managing, and retrieving archived data from a computer database. The original name was given by IBM as Structured English Query Language, abbreviated by the acronym SEQUEL. When data needs to be retrieved from a database, SQL is used to make the request. The [DBMS](#) processes the [SQL](#) query retrieves the requested data and returns it to us. Rather, [SQL](#) statements describe how a collection of data should be organized or what data should be extracted or added to the database.

In common usage, [SQL](#) encompasses [DDL](#) and [DML](#) commands for [CREATE](#), [UPDATE](#), modified, or other operations on database structure.

Features of SQL

- SQL may be utilized by quite a number of users, which include people with very little programming experience.
- SQL is a non-procedural language.
- We can without difficulty create and replace databases in SQL. It isn't a time-consuming process.
- SQL is primarily based totally on ANSI standards.
- SQL does now no longer have a continuation individual.
- SQL is entered into the SQL buffer on one or extra lines.
- SQL makes use of a termination individual to execute instructions immediately. It makes use of features to carry out a few formatting.
- It uses functions to perform some formatting.

SQL Uses

1. **Data definition:** It is used to define the structure and organization of the stored data and the relationships among the stored data items.
2. **Data retrieval:** SQL can also be used for data retrieval.
3. **Data manipulation:** If the user wants to add new data, remove data, or modifying in existing data then SQL provides this facility also.
4. **Access control:** SQL can be used to restrict a user's ability to retrieve, add, and modify data, protecting stored data against unauthorized access.

5. Data sharing: SQL is used to coordinate data sharing by concurrent users, ensuring that changes made by one user do not inadvertently wipe out changes made at nearly the same time by another user.

SQL also differs from other computer languages because it describes what the user wants the computer to do rather than how the computer should do it. (In more technical terms, SQL is a declarative or descriptive language rather than a procedural one.) SQL contains no IF statement for testing conditions, and no GOTO, DO, or FOR statements for program flow control. Rather, SQL statements describe how a collection of data is to be organized, or what data is to be retrieved or added to the database. The sequence of steps to do those tasks is left for the DBMS to determine.

Rules for SQL

- A ';' is used to end SQL statements.
- Statements may be split across lines, but keywords may not.
- Identifiers, operator names, and literals are separated by one or more spaces or other delimiters.
- A comma (,) separates parameters without a clause.
- A space separates a clause.
- Reserved words cannot be used as identifiers unless enclosed with double quotes.
- Identifiers can contain up to 30 characters.
- Identifiers must start with an alphabetic character.
- Characters and date literals must be enclosed within single quotes.
- Numeric literals can be represented by simple values.
- Comments may be enclosed between /* and */ symbols and maybe multi-line.

How does SQL Function?

A server machine is used in the implementation of the structured query language (SQL), processing database queries and returning results. The following are some of the software elements that the SQL process goes through.

AD

1. Parser: The parser begins by replacing some of the words in the SQL statement with unique symbols, a process known as tokenization. The statement is then examined for the following:

2. **Correctness:** The parser checks to see if the SQL statement complies with the rules, or SQL semantics, that guarantee the query statement's accuracy. The parser, for instance, looks to see if the SQL command ends with a semicolon. The parser returns an error if the semicolon is absent.
3. **Authorization:** The parser additionally confirms that the user executing the query has the required permissions to alter the relevant data.
4. **Relational Engine:** The relational engine, also known as the query processor, develops a strategy for efficiently retrieving, writing, or updating the relevant data. For instance, it looks for queries that are similar to others, uses earlier data manipulation techniques, or develops a new one. Byte code, an intermediate-level representation of the SQL statement, is used to write the plan. To efficiently perform database searches and modifications, relational databases use byte code.
5. **Storage Engine:** The software element that interprets the byte code and executes the intended SQL statement is known as the storage engine, also known as the database engine. The data in the database files on the physical disc storage is read and stored. The storage engine delivers the outcome to the requesting application after completion.

Role of SQL

SQL plays many different roles:

- SQL is an interactive question language. Users type SQL instructions into an interactive SQL software to retrieve facts and show them on the screen, presenting a convenient, easy-to-use device for ad hoc database queries.
- SQL is a database programming language. Programmers embed SQL instructions into their utility packages to access the facts in a database. Both user-written packages and database software packages (consisting of document writers and facts access tools) use this approach for database access.
- SQL is a client/server language. Personal computer programs use SQL to communicate over a network with database servers that save shared facts. This client/server architecture is utilized by many famous enterprise-class applications.
- SQL is Internet facts access language. Internet net servers that interact with company facts and Internet utility servers all use SQL as a widespread language for getting access to company databases, frequently through embedding SQL databases get entry to inside famous scripting languages like PHP or Perl.
- SQL is a distributed database language. Distributed database control structures use SQL to assist distribute facts throughout many linked pc structures. The DBMS software program on every gadget makes use of SQL to speak with the opposite structures, sending requests for facts to get entry to.
- SQL is a database gateway language. In a pc community with a mixture of various DBMS products, SQL is frequently utilized in a gateway that lets one logo of DBMS speak with every other logo. SQL has for this reason emerged as a useful, effective device for linking people, pc packages, and pc structures to the facts saved in a relational database.

SQL Injection

A cyberattack known as SQL injection involves tricking the database with SQL queries. To retrieve, alter, or corrupt data in a SQL database, hackers use SQL injection. To execute a SQL injection attack, for instance, they might enter a SQL query in place of a person's name in a submission form.

What is SQL Server?

Microsoft's relational database management system, which uses SQL to manipulate data, is formally known as SQL Server. There are various editions of the MS SQL Server, and each is tailored for particular workloads and requirements.

Finally, SQL is not a particularly structured language, especially when compared with highly structured languages such as C, Pascal, or Java. Instead, SQL statements resemble English sentences, complete with "noise words" that don't add to the meaning of the statement but make it read more naturally. SQL has quite a few inconsistencies and also some special rules to prevent you from constructing SQL statements that look perfectly legal but that don't make sense.

Despite the inaccuracy of its name, SQL has emerged as the standard language for using relational databases. [SQL](#) is both a powerful language and one that is relatively easy to learn. So, SQL is a database management language. The database administrator is answerable for handling a minicomputer or mainframe database and makes use of SQL to outline the database shape and manipulate get entry to the saved data.

How do SQL Commands Work?

Developers use structured query language (SQL) commands, which are specific keywords or SQL statements, to work with data stored in relational databases. The following are categories for SQL commands.

Data Definition Language

SQL commands used to create the database structure are known as data definition language (DDL). Based on the needs of the business, database engineers create and modify database objects using [DDL](#). The CREATE command, for instance, is used by the database engineer to create database objects like tables, views, and indexes.

Data Query Language

Data retrieval instructions are written in the data query language ([DQL](#)), which is used to access relational databases. The SELECT command is used by software programs to filter and return particular results from a SQL table.

Data Manipulation Language

A relational database can be updated with new data using data manipulation language ([DML](#)) statements. The [INSERT](#) command, for instance, is used by an application to add a new record to the database.

Data Control language

Data control language (DCL) is a programming language used by database administrators to control or grant other users access to databases. For instance, they can allow specific applications to manipulate one or more tables by using the GRANT command.

Transaction Control Language

To automatically update databases, the relational engine uses transaction control language (TCL). For instance, the database can reverse a mistaken transaction using the ROLLBACK command.

Last Updated : 10 May, 2023

14

Similar Reads

1. Difference between Structured Query Language (SQL) and Transact-SQL (T-SQL)
2. Configure SQL Jobs in SQL Server using T-SQL
3. SQL vs NO SQL vs NEW SQL
4. SQL | Procedures in PL/SQL
5. SQL | Difference between functions and stored procedures in PL/SQL
6. SQL SERVER – Input and Output Parameter For Dynamic SQL
7. Difference between T-SQL and PL-SQL
8. Difference between SQL and T-SQL
9. SQL Server | Convert tables in T-SQL into XML
10. SQL SERVER | Bulk insert data from csv file using T-SQL command

Previous

Next

Article Contributed By :



varshachoudhary
varshachoudhary

Vote for difficulty



SQL Data Types

[Read](#)[Discuss](#)[Courses](#)[Practice](#)

The data type of a column can be defined as basically what type of data format in which each cell will store the data – it can be any type of integer, character, money, date and time, binary, etc. In this article, we'll get to learn about SQL Data Types in detail.

SQL Data Types

An SQL developer must aware of what type of data will be stored inside each column while creating a table. The data type guideline for SQL is to understand what type of data is expected inside each column and it also identifies how SQL will interact with the stored data.

[SQL](#) (Structured Query Language) is a language used to interact with relational databases. SQL data types define the type of data that can be stored in a database column or variable. Here are the most common SQL data types:

Data Type	Properties
Numeric data types	These are used to store numeric values. Examples include INT, BIGINT , DECIMAL, and FLOAT.
Character data types	These are used to store character strings. Examples include CHAR, VARCHAR, and TEXT.
Date and time data types	These are used to store date and time values. Examples include DATE , TIME, and TIMESTAMP .
Binary data types	These are used to store binary data, such as images or audio files. Examples include BLOB and BYTEA.
Boolean data type	This data type is used to store logical values. The only possible values are TRUE and FALSE.
Interval data types	These are used to store intervals of time. Examples include INTERVAL YEAR, INTERVAL MONTH, and INTERVAL DAY.

Data Type	Properties
Array data types	These are used to store arrays of values. Examples include ARRAY and JSON.
XML data type	This data type is used to store XML data.
Spatial data types	These are used to store geometric or geographic data. Examples include POINT, LINE, and POLYGON.

Different databases may have different variations of these data types, or they may have additional data types not listed here. Understanding SQL data types are important for creating tables and working with data in a database, as it affects how data is stored and processed.

AD

Like in other programming languages, SQL also has certain datatypes available. A brief idea of all the datatypes is discussed below.

Binary Datatypes

There are four subtypes of this datatype which are given below:

Data Type	Description
Binary	The maximum length of 8000 bytes(Fixed-Length binary data)
varbinary	The maximum length of 8000 bytes(Variable Length binary data)
varbinary(max)	The maximum length of 231 bytes(SQL Server 2005 only).(Variable Length binary data)
image	Maximum Length of 2,147,483,647 bytes(Variable Length binary data)

Exact Numeric Datatype

There are nine subtypes which are given below in the table. The table contains the range of data in a particular type.

Data Type	From	To
BigInt	-2^63 (-9,223,372,036,854,775,808)	2^63-1 (9,223,372,036,854,775,807)
Int	-2^31 (-2,147,483,648)	2^31-1 (2,147,483,647)
smallint	-2^15 (-32,768)	2^15-1 (32,767)
tinyint	0	2^8-1
bit	0	1
decimal	-10^38+1	10^38+1
numeric	-10^38+1	10^38+1
money	-922,337,203,685,477,5808	922,337,203,685,477,5808
smallmoney	-2,147,748	2,147,748

Approximate Numeric Datatype

The subtypes of this datatype are given in the table with the range.

Data Type	From	To
Float	-1.79E+308	1.79E+308
Real	-3.40E+38	3.40E+38

Character String Datatype

The subtypes are given in below table –

Data Type	Description
char	The maximum length of 8000 characters.(Fixed-Length non-Unicode Characters)

varchar	The maximum length of 8000 characters.(Variable-Length non-Unicode Characters)
varchar(max)	The maximum length of 231 characters(SQL Server 2005 only).(Variable Length non-Unicode data)
text	The maximum length of 2,127,483,647 characters(Variable Length non-Unicode data)

Unicode Character String Datatype

The details are given in below table –

Data Type	Description
nchar	The maximum length of 4000 characters(Fixed-Length Unicode Characters)
Nvarchar	The maximum length of 4000 characters.(Variable-Length Unicode Characters)
nvarchar(max)	The maximum length of 231 characters(SQL Server 2005 only).(Variable Length Unicode data)

Date and Time Datatype

The details are given in the below table.

Data Type	Description
DATE	A data type is used to store the data of date in a record
TIME	A data type is used to store the data of time in a record
DATETIME	A data type is used to store both the data,date, and time in the record.

XML Datatype

XML data type allows storage of XML documents and fragments in a SQL Server database

DataType	Description
XML Datatype	A Datatype used to store data in the format of XML datatype

Spatial Datatype

A datatype is used for storing planar spatial data, such as points, lines, and polygons, in a database table.

DataType	Description
Geometry	A datatype is used for storing planar spatial data, such as points, lines, and polygons, in a database table.

Array Datatype

SQL Server does not have a built-in array datatype. However, it is possible to simulate arrays using tables or XML data types.

Last Updated : 12 Apr, 2023

156

Similar Reads

1. [How to Convert Data From SQL to C Data Types?](#)
2. [Difference between Structured Query Language \(SQL\) and Transact-SQL \(T-SQL\)](#)
3. [Configure SQL Jobs in SQL Server using T-SQL](#)
4. [SQL vs NO SQL vs NEW SQL](#)
5. [SQL SERVER | Bulk insert data from csv file using T-SQL command](#)
6. [Numeric and Date-time data types in SQL Server](#)
7. [SQL | Procedures in PL/SQL](#)
8. [SQL | Difference between functions and stored procedures in PL/SQL](#)
9. [SQL SERVER – Input and Output Parameter For Dynamic SQL](#)
10. [Difference between SQL and T-SQL](#)

Previous

Next

Article Contributed By :

[Trending Now](#) [DSA](#) [Data Structures](#) [Algorithms](#) [Interview Preparation](#) [Data Science](#) [Topic-wise Practice](#) [Jobs](#)

SQL | DDL, DQL, DML, DCL and TCL Commands

[Read](#) [Discuss](#) [Courses](#) [Practice](#)

Structured Query Language(SQL) as we all know is the database language by the use of which we can perform certain operations on the existing database and also we can use this language to create a database. [SQL](#) uses certain commands like CREATE, DROP, INSERT, etc. to carry out the required tasks.

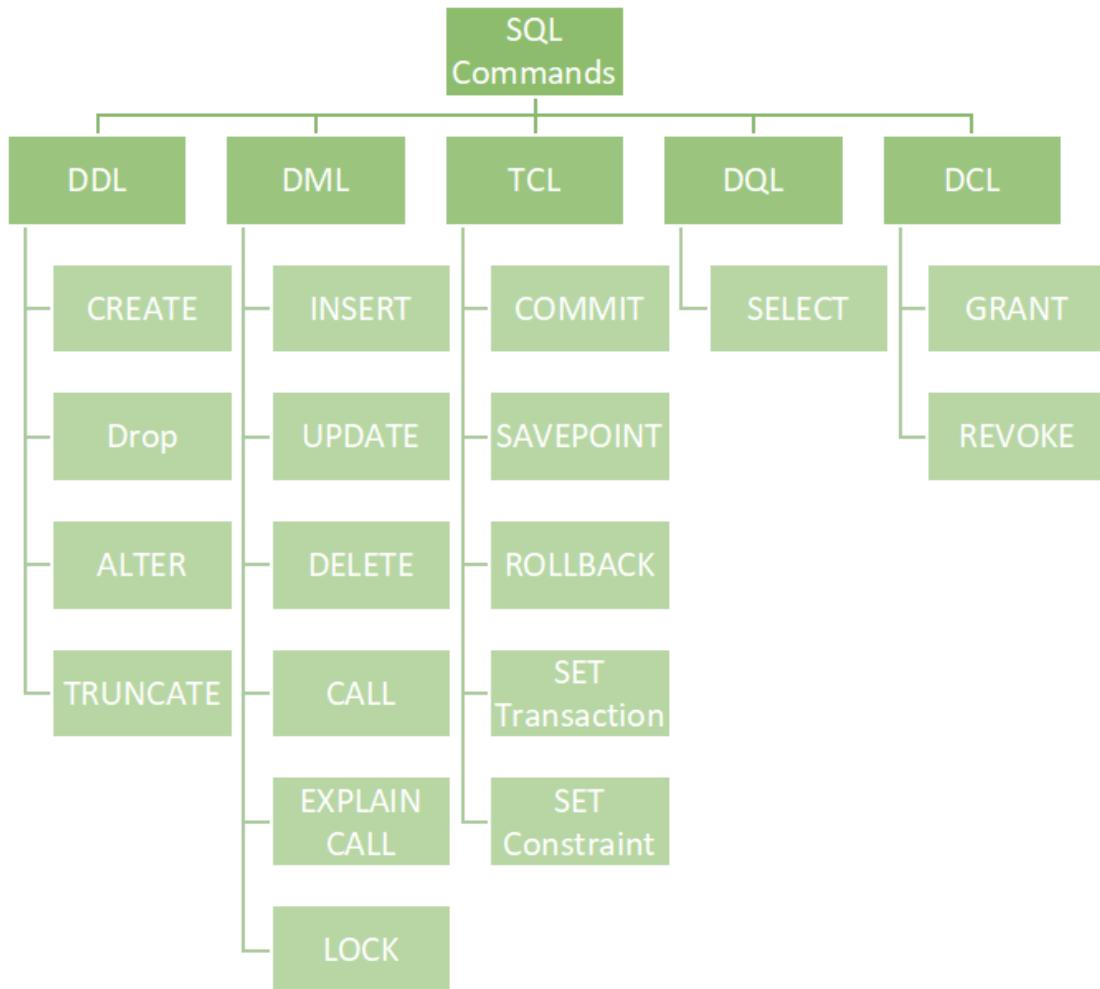
SQL commands are like instructions to a table. It is used to interact with the database with some operations. It is also used to perform specific tasks, functions, and queries of data. SQL can perform various tasks like creating a table, adding data to tables, dropping the table, modifying the table, set permission for users.

These [SQL](#) commands are mainly categorized into five categories:

1. DDL – Data Definition Language
2. DQL – Data Query Language
3. DML – Data Manipulation Language
4. DCL – Data Control Language
5. TCL – Transaction Control Language

Now, we will see all of these in detail.

AD



DDL (Data Definition Language)

DDL or Data Definition Language actually consists of the SQL commands that can be used to define the database schema. It simply deals with descriptions of the database schema and is used to create and modify the structure of database objects in the database. DDL is a set of SQL commands used to create, modify, and delete database structures but not data. These commands are normally not used by a general user, who should be accessing the database via an application.

List of DDL commands:

- **CREATE**: This command is used to create the database or its objects (like table, index, function, views, store procedure, and triggers).
- **DROP**: This command is used to delete objects from the database.
- **ALTER**: This is used to alter the structure of the database.
- **TRUNCATE**: This is used to remove all records from a table, including all spaces allocated for the records are removed.
- **COMMENT**: This is used to add comments to the data dictionary.

- **RENAME**: This is used to rename an object existing in the database.

DQL (Data Query Language)

DQL statements are used for performing queries on the data within schema objects. The purpose of the DQL Command is to get some schema relation based on the query passed to it. We can define DQL as follows it is a component of SQL statement that allows getting data from the database and imposing order upon it. It includes the SELECT statement. This command allows getting the data out of the database to perform operations with it. When a SELECT is fired against a table or tables the result is compiled into a further temporary table, which is displayed or perhaps received by the program i.e. a front-end.

List of DQL:

- **SELECT**: It is used to retrieve data from the database.

DML(Data Manipulation Language)

The SQL commands that deal with the manipulation of data present in the database belong to DML or Data Manipulation Language and this includes most of the SQL statements. It is the component of the SQL statement that controls access to data and to the database. Basically, DCL statements are grouped with DML statements.

List of DML commands:

- **INSERT**: It is used to insert data into a table.
- **UPDATE**: It is used to update existing data within a table.
- **DELETE**: It is used to delete records from a database table.
- **LOCK**: Table control concurrency.
- **CALL**: Call a PL/SQL or JAVA subprogram.
- **EXPLAIN PLAN**: It describes the access path to data.

DCL (Data Control Language)

DCL includes commands such as GRANT and REVOKE which mainly deal with the rights, permissions, and other controls of the database system.

List of DCL commands:

GRANT: This command gives users access privileges to the database.

Syntax:

GRANT SELECT, UPDATE ON MY_TABLE TO SOME_USER, ANOTHER_USER;

REVOKE: This command withdraws the user's access privileges given by using the GRANT command.

Syntax:

REVOKE SELECT, UPDATE ON MY_TABLE FROM USER1, USER2;

TCL (Transaction Control Language)

Transactions group a set of tasks into a single execution unit. Each transaction begins with a specific task and ends when all the tasks in the group successfully complete. If any of the tasks fail, the transaction fails. Therefore, a transaction has only two results: success or failure. You can explore more about transactions [here](#). Hence, the following TCL commands are used to control the execution of a transaction:

BEGIN: Opens a Transaction.

COMMIT: Commits a Transaction.

Syntax:

COMMIT;

ROLLBACK: Rollbacks a transaction in case of any error occurs.

Syntax:

ROLLBACK;

SAVEPOINT: Sets a save point within a transaction.

Syntax:

SAVEPOINT SAVEPOINT_NAME;

Last Updated : 10 May, 2023

700

Similar Reads

1. SQL | DDL, DML, TCL and DCL



SQL | Comments

[Read](#) [Discuss](#) [Courses](#) [Practice](#)

Comments are used to explain sections of SQL statements or to prevent SQL statements from being executed. As is any programming language comments matter a lot in SQL also. In this set, we will learn about writing comments in any SQL snippet.

Comments can be written in the following three formats:

1. Single-line comments
2. Multi-line comments
3. In-line comments

Single Line Comments

Comments starting and ending in a single line are considered single-line comments. A line starting with ‘–‘ is a comment and will not be executed.

Syntax:

AD

— *single line comment*

— *another comment*

*SELECT * FROM Customers;*

Let's see an example in which we have one table name Customers in which we are fetching all the data records.

Query:

```
SELECT * FROM customers;
-- This is a comment that explains
the purpose of the query.
```

Multi-Line Comments

Comments starting in one line and ending in different lines are considered as multi-line comments.

A line starting with '/*' is considered as starting point of the comment and is terminated when '*/' is encountered.

Syntax:

```
/* multi line comment
another comment */
SELECT * FROM Customers;
```

Let's consider that we want to fetch order_date with the year 2022.

Query:

```
/*
This is a multi-line comment that explains
the purpose of the query below.

The query selects all the orders from the orders
table that were placed in the year 2022.
*/
SELECT * FROM orders WHERE YEAR(order_date) = 2022;
```

In-Line Comments

In-line comments are an extension of multi-line comments, comments can be stated in between the statements and are enclosed in between '/*' and '*/'.

Syntax:

```
SELECT * FROM /* Customers; */
```

Let's Suppose we have on a table with name orders and we are fetching customer_name.

Query:

```

SELECT customer_name,
/* This column contains the name of
the customer / order_date /
This column contains the date the
order was placed */ FROM orders;

```

More Examples:

There are a few more examples of Multiline comments and inline comments.

Multi line comment ->

```

/* SELECT * FROM Students;
SELECT * FROM STUDENT_DETAILS;
SELECT * FROM Orders; */
SELECT * FROM Articles;

```

In line comment ->

```

SELECT * FROM Students;
SELECT * FROM /* STUDENT_DETAILS;
SELECT * FROM Orders;
SELECT * FROM */ Articles;

```

This article is contributed by **Pratik Agarwal**. If you like GeeksforGeeks and would like to contribute, you can also write an article using [write.geeksforgeeks.org](https://www.geeksforgeeks.org/write-for-us/) or mail your article to review-team@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks. Please write comments if you find anything incorrect, or if you want to share more information about the topic discussed above.

Last Updated : 18 Apr, 2023

62

Similar Reads

1. Difference between Structured Query Language (SQL) and Transact-SQL (T-SQL)

2. Configure SQL Jobs in SQL Server using T-SQL

3. SQL vs NO SQL vs NEW SQL

4. Django project to create a Comments System

5. SQL | Procedures in PL/SQL

6. SQL | Difference between functions and stored procedures in PL/SQL

[Trending Now](#) [DSA](#) [Data Structures](#) [Algorithms](#) [Interview Preparation](#) [Data Science](#) [Topic-wise Practice](#) [Jobs](#)

SQL | TRANSACTIONS

[Read](#)[Discuss](#)[Courses](#)[Practice](#)

What are Transactions?

Transactions group a set of tasks into a single execution unit. Each transaction begins with a specific task and ends when all the tasks in the group successfully complete. If any of the tasks fail, the transaction fails. Therefore, a transaction has only two results: **success or failure**.

Example of a transaction to transfer \$150 from account A to account B:

1. read(A)
2. A := A – 150
3. write(A)
4. read(B)
5. B := B + 150
6. write(B)

Incomplete steps result in the failure of the transaction. A database transaction, by definition, must be atomic, consistent, isolated and durable.

These are popularly known as **ACID** properties. These properties can ensure the concurrent execution of multiple transactions without conflict.

How to implement Transactions using SQL?

AD

Following commands are used to control transactions. It is important to note that these statements cannot be used while creating tables and are only used with the DML Commands such as – INSERT, UPDATE and DELETE.

1. BEGIN TRANSACTION: It indicates the start point of an explicit or local transaction.

Syntax:

```
BEGIN TRANSACTION transaction_name ;
```

2. SET TRANSACTION: Places a name on a transaction.

Syntax:

```
SET TRANSACTION [ READ WRITE | READ ONLY ];
```

3. COMMIT: If everything is in order with all statements within a single transaction, all changes are recorded together in the database is called **committed**. The COMMIT command saves all the transactions to the database since the last COMMIT or ROLLBACK command.

Syntax:

```
COMMIT;
```

Example: Sample table 1

Student				
Roll_No	Name	Address	Phone	Age
1	Ram	Delhi	9455123451	18
2	Ramesh	Gurgaon	9652431543	18
3	Sujit	Rohtak	9156253131	20
4	Suresh	Delhi	9156768971	18
3	Sujit	Rohtak	9156253131	20
2	Ramesh	Gurgaon	9652431543	18

Following is an example which would delete those records from the table which have age = 20 and then COMMIT the changes in the database.

Queries:

```
DELETE FROM Student WHERE AGE = 20;  
COMMIT;
```

Output:

Thus, two rows from the table would be deleted and the SELECT statement would look like,

Rol_No	Name	Address	Phone	Age
1	Ram	Delhi	9455123451	18
2	Ramesh	Gurgaon	9652431543	18
4	Suresh	Delhi	9156768971	18
2	Ramesh	Gurgaon	9652431543	18

4. ROLLBACK: If any error occurs with any of the SQL grouped statements, all changes need to be aborted. The process of reversing changes is called **rollback**. This command can only be used to undo transactions since the last COMMIT or ROLLBACK command was issued.

Syntax:

```
ROLLBACK;
```

Example:

From the above example **Sample table1**,

Delete those records from the table which have age = 20 and then ROLLBACK the changes in the database.

Queries:

```
DELETE FROM Student WHERE AGE = 20;
ROLLBACK;
```

Output:

Student				
Rol_No	Name	Address	Phone	Age
1	Ram	Delhi	9455123451	18
2	Ramesh	Gurgaon	9652431543	18
3	Sujit	Rohtak	9156253131	20
4	Suresh	Delhi	9156768971	18
3	Sujit	Rohtak	9156253131	20
2	Ramesh	Gurgaon	9652431543	18

5. SAVEPOINT: creates points within the groups of transactions in which to ROLLBACK.

A SAVEPOINT is a point in a transaction in which you can roll the transaction back to a certain point without rolling back the entire transaction.

Syntax for Savepoint command:

```
SAVEPOINT SAVEPOINT_NAME;
```

This command is used only in the creation of SAVEPOINT among all the transactions.

In general ROLLBACK is used to undo a group of transactions.

Syntax for rolling back to Savepoint command:

```
ROLLBACK TO SAVEPOINT_NAME;
```

you can ROLLBACK to any SAVEPOINT at any time to return the appropriate data to its original state.

Example:

From the above example **Sample table1**,

Delete those records from the table which have age = 20 and then ROLLBACK the changes in the database by keeping Savepoints.

Queries:

```
SAVEPOINT SP1;
//Savepoint created.
DELETE FROM Student WHERE AGE = 20;
//deleted
```

```
SAVEPOINT SP2;
//Savepoint created.
```

Here SP1 is first SAVEPOINT created before deletion. In this example one deletion have taken place.

After deletion again SAVEPOINT SP2 is created.

Output:

Rol_No	Name	Address	Phone	Age
1	Ram	Delhi	9455123451	18
2	Ramesh	Gurgaon	9652431543	18
4	Suresh	Delhi	9156768971	18
2	Ramesh	Gurgaon	9652431543	18

Deletion have been taken place, let us assume that you have changed your mind and decided to ROLLBACK to the SAVEPOINT that you identified as SP1 which is before deletion. deletion is undone by this statement ,

```
ROLLBACK TO SP1;
//Rollback completed.
```

Student				
Rol_No	Name	Address	Phone	Age
1	Ram	Delhi	9455123451	18
2	Ramesh	Gurgaon	9652431543	18
3	Sujit	Rohtak	9156253131	20
4	Suresh	Delhi	9156768971	18
3	Sujit	Rohtak	9156253131	20
2	Ramesh	Gurgaon	9652431543	18

6. RELEASE SAVEPOINT:- This command is used to remove a SAVEPOINT that you have created.

Syntax:

```
RELEASE SAVEPOINT SAVEPOINT_NAME
```

Once a SAVEPOINT has been released, you can no longer use the ROLLBACK command to undo transactions performed since the last SAVEPOINT.

It is used to initiate a database transaction and used to specify characteristics of the transaction that follows.

Last Updated : 25 Oct, 2022

165

Similar Reads

1. PL/SQL Transactions
2. Auto-commit commands and Compensating transactions in SQL
3. Difference between Structured Query Language (SQL) and Transact-SQL (T-SQL)
4. Configure SQL Jobs in SQL Server using T-SQL
5. SQL vs NO SQL vs NEW SQL
6. SQL | Procedures in PL/SQL
7. SQL | Difference between functions and stored procedures in PL/SQL
8. SQL SERVER – Input and Output Parameter For Dynamic SQL
9. Difference between SQL and T-SQL
10. SQL Server | Convert tables in T-SQL into XML

Previous

Next

Article Contributed By :



GeeksforGeeks

Vote for difficulty



SQL | Views

[Read](#)
[Discuss](#)
[Courses](#)
[Practice](#)
[Video](#)

Views in SQL are kind of virtual tables. A view also has rows and columns as they are in a real table in the database. We can create a view by selecting fields from one or more tables present in the database. A View can either have all the rows of a table or specific rows based on certain condition. In this article we will learn about creating , deleting and updating Views.

Sample Tables:

StudentDetails

S_ID	NAME	ADDRESS
1	Harsh	Kolkata
2	Ashish	Durgapur
3	Pratik	Delhi
4	Dhanraj	Bihar
5	Ram	Rajasthan

StudentMarks

AD

ID	NAME	MARKS	AGE
1	Harsh	90	19
2	Suresh	50	20
3	Pratik	80	19
4	Dhanraj	95	21
5	Ram	85	18

CREATING VIEWS

We can create View using **CREATE VIEW** statement. A View can be created from a single table or multiple tables. **Syntax:**

```
CREATE VIEW view_name AS
SELECT column1, column2.....
FROM table_name
WHERE condition;
```

view_name: Name for the View

table_name: Name of the table

condition: Condition to select rows

Examples:

Creating View from a single table:

- In this example we will create a View named DetailsView from the table StudentDetails.
- Query:

```
CREATE VIEW DetailsView AS
SELECT NAME, ADDRESS
FROM StudentDetails
WHERE S_ID < 5;
```

- To see the data in the View, we can query the view in the same manner as we query a table.

```
SELECT * FROM DetailsView;
```

Output:

NAME	ADDRESS
Harsh	Kolkata
Ashish	Durgapur
Pratik	Delhi
Dhanraj	Bihar

- In this example, we will create a view named StudentNames from the table StudentDetails. Query:

```
CREATE VIEW StudentNames AS
SELECT S_ID, NAME
FROM StudentDetails
ORDER BY NAME;
```

- If we now query the view as,

```
SELECT * FROM StudentNames;
```

Output:

S_ID	NAMES
2	Ashish
4	Dhanraj
1	Harsh
3	Pratik
5	Ram

- Creating View from multiple tables:** In this example we will create a View named MarksView from two tables StudentDetails and StudentMarks. To create a View from multiple tables we can simply include multiple tables in the SELECT statement. Query:

```
CREATE VIEW MarksView AS
SELECT StudentDetails.NAME, StudentDetails.ADDRESS, StudentMarks.MARKS
FROM StudentDetails, StudentMarks
WHERE StudentDetails.NAME = StudentMarks.NAME;
```

- To display data of View MarksView:

```
SELECT * FROM MarksView;
```

- Output:

NAME	ADDRESS	MARKS
Harsh	Kolkata	90
Pratik	Delhi	80
Dhanraj	Bihar	95
Ram	Rajasthan	85

LISTING ALL VIEWS IN A DATABASE

We can list View using the SHOW FULL TABLES statement or using the information_schema table. A View can be created from a single table or multiple tables.

Syntax (Using SHOW FULL TABLES):

```
use "database_name";
show full tables where table_type like "%VIEW";
```

Syntax (Using information_schema) :

```
select * from information_schema.views where table_schema = "database_name";
```

OR

```
select table_schema,table_name,view_definition from information_schema.views
where table_schema = "database_name";
```

DELETING VIEWS

We have learned about creating a View, but what if a created View is not needed any more? Obviously we will want to delete it. SQL allows us to delete an existing View. We can delete or drop a View using the DROP statement. **Syntax:**

```
DROP VIEW view_name;
```

view_name: Name of the View which we want to delete.

For example, if we want to delete the View **MarksView**, we can do this as:

```
DROP VIEW MarksView;
```

UPDATING VIEWS

There are certain conditions needed to be satisfied to update a view. If any one of these conditions is **not** met, then we will not be allowed to update the view.

1. The SELECT statement which is used to create the view should not include GROUP BY clause or ORDER BY clause.
 2. The SELECT statement should not have the DISTINCT keyword.
 3. The View should have all NOT NULL values.
 4. The view should not be created using nested queries or complex queries.
 5. The view should be created from a single table. If the view is created using multiple tables then we will not be allowed to update the view.
- We can use the **CREATE OR REPLACE VIEW** statement to add or remove fields from a view. **Syntax:**

```
CREATE OR REPLACE VIEW view_name AS
SELECT column1,column2,..
FROM table_name
WHERE condition;
```

- For example, if we want to update the view **MarksView** and add the field AGE to this View from **StudentMarks** Table, we can do this as:

```
CREATE OR REPLACE VIEW MarksView AS
SELECT StudentDetails.NAME, StudentDetails.ADDRESS, StudentMarks.MARKS,
StudentMarks.AGE
FROM StudentDetails, StudentMarks
WHERE StudentDetails.NAME = StudentMarks.NAME;
```

- If we fetch all the data from MarksView now as:

```
SELECT * FROM MarksView;
```

NAME	ADDRESS	MARKS	AGE
Harsh	Kolkata	90	19
Pratik	Delhi	80	19
Dhanraj	Bihar	95	21
Ram	Rajasthan	85	18

- Output:
- **Inserting a row in a view:** We can insert a row in a View in a same way as we do in a table. We can use the INSERT INTO statement of SQL to insert a row in a View.

Syntax:

```
INSERT INTO view_name(column1, column2 , column3,..)
VALUES(value1, value2, value3..);
```

view_name: Name of the View

Example: In the below example we will insert a new row in the View DetailsView which we have created above in the example of “creating views from a single table”.

```
INSERT INTO DetailsView(NAME, ADDRESS)
VALUES("Suresh", "Gurgaon");
```

- If we fetch all the data from DetailsView now as,

```
SELECT * FROM DetailsView;
```

Output:

NAME	ADDRESS
Harsh	Kolkata
Ashish	Durgapur
Pratik	Delhi
Dhanraj	Bihar
Suresh	Gurgaon

- **Deleting a row from a View:** Deleting rows from a view is also as simple as deleting rows from a table. We can use the DELETE statement of SQL to delete rows from a view. Also deleting a row from a view first delete the row from the actual table and the change is then reflected in the view.

Syntax:

```
DELETE FROM view_name
WHERE condition;
```

view_name: Name of view from where we want to delete rows

condition: Condition to select rows

Example: In this example, we will delete the last row from the view DetailsView which we just added in the above example of inserting rows.

```
DELETE FROM DetailsView
WHERE NAME="Suresh";
```

- If we fetch all the data from DetailsView now as,

```
SELECT * FROM DetailsView;
```

Output:

NAME	ADDRESS
Harsh	Kolkata
Ashish	Durgapur
Pratik	Delhi
Dhanraj	Bihar

WITH CHECK OPTION

The WITH CHECK OPTION clause in SQL is a very useful clause for views. It is applicable to an updatable view. If the view is not updatable, then there is no meaning of including this clause in the CREATE VIEW statement.

- The WITH CHECK OPTION clause is used to prevent the insertion of rows in the view where the condition in the WHERE clause in CREATE VIEW statement is not satisfied.
- If we have used the WITH CHECK OPTION clause in the CREATE VIEW statement, and if the UPDATE or INSERT clause does not satisfy the conditions then they will return an error.

Example: In the below example we are creating a View SampleView from StudentDetails Table with WITH CHECK OPTION clause.

```
CREATE VIEW SampleView AS
SELECT S_ID, NAME
FROM StudentDetails
WHERE NAME IS NOT NULL
WITH CHECK OPTION;
```

In this View if we now try to insert a new row with null value in the NAME column then it will give an error because the view is created with the condition for NAME column as NOT NULL. For example, though the View is updatable but then also the below query for this View is not valid:

```
INSERT INTO SampleView(S_ID)
VALUES(6);
```

NOTE: The default value of NAME column is null.

Uses of a View: A good database should contain views due to the given reasons:

1. **Restricting data access** – Views provide an additional level of table security by restricting access to a predetermined set of rows and columns of a table.
2. **Hiding data complexity** – A view can hide the complexity that exists in multiple tables join.
3. **Simplify commands for the user** – Views allow the user to select information from multiple tables without requiring the users to actually know how to perform a join.
4. **Store complex queries** – Views can be used to store complex queries.
5. **Rename Columns** – Views can also be used to rename the columns without affecting the base tables provided the number of columns in view must match the number of columns specified in select statement. Thus, renaming helps to hide the names of the columns of the base tables.
6. **Multiple view facility** – Different views can be created on the same table for different users.

This article is contributed by [Harsh Agarwal](#). If you like GeeksforGeeks and would like to contribute, you can also write an article using contribute.geeksforgeeks.org or mail your article to contribute@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks. Please write comments if you find anything incorrect, or if you want to share more information about the topic discussed above.



SQL | Creating Roles



shreyanshi_arun

[Read](#)[Discuss](#)[Courses](#)[Practice](#)

A role is created to ease the setup and maintenance of the security model. It is a named group of related privileges that can be granted to the user. When there are many users in a [database](#) it becomes difficult to grant or revoke privileges to users. Therefore, if you define roles:

1. You can grant or revoke privileges to users, thereby automatically granting or revoking privileges.
2. You can either create Roles or use the system roles pre-defined.

Some of the privileges granted to the system roles are as given below:

System Roles	Privileges Granted to the Role
Connect	Create table, Create view, Create synonym, Create sequence, Create session, etc.
Resource	Create Procedure, Create Sequence, Create Table, Create Trigger etc. The primary usage of the Resource role is to restrict access to database objects.
DBA	All system privileges

Creating and Assigning a Role

First, the (Database Administrator)DBA must create the role. Then the DBA can assign privileges to the role and users to the role.

Syntax:

AD

CREATE ROLE manager;

Role created.

Arguments

- *role_name*: Is the name of the role to be created
- *owner_name*: AUTHORIZATION: If no user is specified, the user who executes CREATE ROLE will be the owner of the role. The role's members can be added or removed at the discretion of the role's owner or any member of an owning role.

Permissions

It requires either membership in the fixed database role db_securityadmin or the CREATE ROLE permission on the database. The following authorizations are also necessary when using the AUTHORIZATION option:

1. It takes IMPERSONATE permission from that user in order to transfer ownership of a role to another user.
2. It takes membership in the recipient role or the ALTER permission on that role to transfer ownership of one role to another.
3. An application role must have ALTER permission in order to transfer ownership of a role to it.

In the syntax: 'manager' is the name of the role to be created.

- Now that the role is created, the DBA can use the GRANT statement to assign users to the role as well as assign privileges to the role.
- It's easier to GRANT or REVOKE privileges to the users through a role rather than assigning a privilege directly to every user.
- If a role is identified by a password, then GRANT or REVOKE privileges have to be identified by the password.

Grant Privileges To a Role

```
GRANT create table, create view  
TO manager;  
Grant succeeded.
```

Grant a Role To Users

```
GRANT manager TO SAM, STARK;  
Grant succeeded.
```

Revoke Privilege from a Role

```
REVOKE create table FROM manager;
```

Drop a Role

```
DROP ROLE manager;
```

Explanation

Firstly it creates a manager role and then allows managers to create tables and views. It then grants Sam and Stark the role of managers. Now Sam and Stark can create tables and views. If users have multiple roles granted to them, they receive all of the privileges associated with all of the roles. Then create table privilege is removed from the role 'manager' using Revoke. The role is dropped from the database using drop.

Last Updated : 30 May, 2023

95

Similar Reads

1. Difference between Structured Query Language (SQL) and Transact-SQL (T-SQL)

2. Configure SQL Jobs in SQL Server using T-SQL

3. SQL vs NO SQL vs NEW SQL

4. Database Roles in CQL (Cassandra Query Language)

5. Granting Permissions to Roles in Cassandra

6. Key Roles for Data Analytics project

7. Privilege and Roles in DBMS



SQL indexes



MrinalVerma

[Read](#) [Discuss](#) [Courses](#) [Practice](#)

An index is a schema object. It is used by the server to speed up the retrieval of rows by using a pointer. It can reduce disk I/O(input/output) by using a rapid path access method to locate data quickly. An index helps to speed up select queries and where clauses, but it slows down data input, with the update and the insert statements. Indexes can be created or dropped with no effect on the data. In this article, we will see how to create, delete, and uses the INDEX in the database.

For example, if you want to reference all pages in a book that discusses a certain topic, you first refer to the index, which lists all the topics alphabetically and is then referred to one or more specific page numbers.

Creating an Index:

Syntax:

AD

```
CREATE INDEX index  
ON TABLE column;
```

where the **index** is the name given to that index and **TABLE** is the name of the table on which that index is created and **column** is the name of that column for which it is applied.

For multiple columns:

Syntax:

```
CREATE INDEX index  
ON TABLE (column1, column2,.....);
```

Unique Indexes: Unique indexes are used for the maintenance of the integrity of the data present in the table as well as for fast performance, it does not allow multiple values to enter into the table.

Syntax:

```
CREATE UNIQUE INDEX index  
ON TABLE column;
```

When should indexes be created:

- A column contains a wide range of values.
- A column does not contain a large number of null values.
- One or more columns are frequently used together in a where clause or a join condition.

When should indexes be avoided:

- The table is small
- The columns are not often used as a condition in the query
- The column is updated frequently

Removing an Index: Remove an index from the data dictionary by using the **DROP INDEX** command.

Syntax:

```
DROP INDEX index;
```

To drop an index, you must be the owner of the index or have the **DROP ANY INDEX** privilege.

Altering an Index: To modify an existing table's index by rebuilding, or reorganizing the index.

```
ALTER INDEX IndexName  
ON TableName REBUILD;
```

Confirming Indexes: You can check the different indexes present in a particular table given by the user or the server itself and their uniqueness.

Syntax:

```
select * from USER_INDEXES;
```

It will show you all the indexes present in the server, in which you can locate your own tables too.

Renaming an index: You can use the system-stored procedure `sp_rename` to rename any index in the database.

Syntax:

```
EXEC sp_rename  
    index_name,  
    new_index_name,  
    N'INDEX';
```

Lastly, let us discuss why should we use indexing?

Indexing is an important topic when considering advanced MySQL, although most people know about its definition and usage they don't understand when and where to use it to change the efficiency of our queries or stored procedures by a huge margin.

Here are some scenarios along with their explanation related to Indexing:

1. When executing a query on a table having huge data (> 100000 rows), MySQL performs a full table scan which takes much time and the server usually gets timed out. To avoid this always check the `explain` option for the query within MySQL which tells us about the state of execution. It shows which columns are being used and whether it will be a threat to huge data. On basis of the columns repeated in a similar order in conditions, we can create an index for them in the same order to maximize the speed of the query.
2. The order of the index is of huge importance as we can use the same index in many scenarios. Using only one index we can utilize it in more than one query which different conditions. like for example, in a query, we make a join with a table based on `customer_id` wards we also join another join based on `customer_id` and `order_date`. Then we can simply create a single index by the order of `customer_id, order_date` which would be used in both cases. This also saves storage.
3. We should also be careful to not make an index for each query as creating indexes also take storage and when the amount of data is huge it will create a problem. Therefore, it's important to carefully consider which columns to index based on the needs of your application. In general, it's a good practice to only create indexes on columns that are frequently used in queries and to avoid creating indexes on columns that are rarely used. It's also a good idea to periodically review the indexes in your database and remove any that are no longer needed.
4. Indexes can also improve performance when used in conjunction with sorting and grouping operations. For example, if you frequently sort or group data based on a particular column, creating an index on that column can greatly improve performance. The index allows MySQL to quickly access and sort or group the data, rather than having to perform a full table scan.
5. In some cases, MySQL may not use an index even if one exists. This can happen if the query optimizer determines that a full table scan is faster than using the index. This can occur

when the table is small, the query is highly selective, or the table has a high rate of updates.

Last Updated : 10 Mar, 2023

90

Similar Reads

1. [Multiple Indexes vs Multi-Column Indexes](#)

2. [SQL queries on clustered and non-clustered Indexes](#)

3. [Difference between Structured Query Language \(SQL\) and Transact-SQL \(T-SQL\)](#)

4. [Configure SQL Jobs in SQL Server using T-SQL](#)

5. [SQL vs NO SQL vs NEW SQL](#)

6. [Secondary Indexes on MAP Collection in Cassandra](#)

7. [Secondary Indexes on LIST Collection in Cassandra](#)

8. [Secondary Indexes on SET Collection in Cassandra](#)

9. [How to Compare Indexes of Tables From Two Different Databases in Oracle?](#)

10. [SQL | Procedures in PL/SQL](#)

[Previous](#)

[Next](#)

Article Contributed By :



MrinalVerma

MrinalVerma

Vote for difficulty

Current difficulty : [Easy](#)

Easy	Normal	Medium	Hard	Expert
------	--------	--------	------	--------

Improved By : [sunny94](#), [khushboogoyal499](#), [swchoi442](#), [varshachoudhary](#), [adichavan095](#)

Article Tags : [DBMS Indexing](#), [DBMS-SQL](#), [SQL-basics](#), [DBMS](#), [SQL](#)

Practice Tags : [DBMS](#), [SQL](#)



SQL | SEQUENCES

[Read](#)[Discuss](#)[Courses](#)[Practice](#)

SQL sequences specifies the properties of a sequence object while creating it. An object bound to a user-defined schema called a sequence produces a series of numerical values in accordance with the specification used to create it. The series of numerical values can be configured to restart (cycle) when it runs out and is generated in either ascending or descending order at a predetermined interval. Contrary to identity columns, sequences are not linked to particular tables. Applications use a sequence object to access the next value in the sequence. The application has control over how sequences and tables interact. A sequence object can be referred to by user applications, and the values can be coordinated across various rows and tables.

Let's suppose that sequence is a set of integers 1, 2, 3, ... that are generated and supported by some database systems to produce unique values on demands.

- A sequence is a user-defined schema-bound object that generates a series of numeric values.
- Sequences are frequently used in many databases because many applications require each row in a table to contain a unique value and sequences provide an easy way to generate them.
- The sequence of numeric values is generated in an ascending or descending order at defined intervals and can be configured to restart when it exceeds max_value.

Different Features of Sequences

1. A sequence is a database object that generates and produces integer values in sequential order.
2. It automatically generates the primary key and unique key values.
3. It may be in ascending or descending order.
4. It can be used for multiple tables.
5. Sequence numbers are stored and generated independently of tables.
6. It saves time by reducing application code.
7. It is used to generate unique integers.
8. It is used to create an auto number field.
9. Useful when you need to create a unique number to act as a primary key.

10. Oracle provides an object called a Sequence that can generate numeric values. The value generated can have maximum of 38 digits
11. Provide intervals between numbers.

Syntax:

CREATE SEQUENCE sequence_name

AD

START WITH initial_value

INCREMENT BY increment_value

MINVALUE minimum value

MAXVALUE maximum value

CYCLE/NOCYCLE;

Following is the sequence query creating a sequence in ascending order.

Example 1:

```
CREATE SEQUENCE sequence_1
start with 1
increment by 1
minvalue 0
maxvalue 100
cycle;
```

The above query will create a sequence named *sequence_1*. The sequence will start from 1 and will be incremented by 1 having maximum value of 100. The sequence will repeat itself from the start value after exceeding 100.

Example 2: Following is the sequence query creating a sequence in descending order.

```
CREATE SEQUENCE sequence_2
start with 100
increment by -1
```

```
min value 1
max value 100
cycle;
```

The above query will create a sequence named *sequence_2*. The sequence will start from 100 and should be less than or equal to a maximum value and will be incremented by -1 having a minimum value of 1.

Example To Use Sequence:

Create a table named students with columns as id and name.

```
CREATE TABLE students
(
ID number(10),
NAME char(20)
);
```

Now insert values into a table

```
INSERT into students VALUES
(sequence_1.nextval,'Shubham');
INSERT into students VALUES
(sequence_1.nextval,'Aman');
```

where *sequence_1.nextval* will insert id's in the id column in a sequence as defined in *sequence_1*.

Output:

ID		NAME
<hr/>		
1		Shubham
2		Aman

Cache Management

To enhance performance, [SQL Server](#) employs cache management for sequence numbers. The [CACHE](#) argument determines the number of sequence numbers pre-allocated by the server.

For instance, let's consider a new sequence with a starting value of 1 and a cache size of 15. When the first number is required, values 1 through 15 are fetched from memory and made available. The last cached value (15) is written to the system tables on disk. As all 15 numbers are utilized, the next request (for number 16) triggers the allocation of a new cache. The latest cached value (30) is then stored in the system tables.

In the event of a SQL Server shutdown after using 22 numbers, the next intended sequence number (23) in memory replaces the previously stored number in the system tables. Upon restarting, when a sequence number is needed, the starting number (23) is read from the system tables. A cache of 15 numbers (23-38) is allocated to memory, and the next number outside the cache (39) is written to the system tables.

If an abnormal shutdown occurs, such as a power failure, the sequence restarts from the number read from the system tables (39). Any sequence numbers allocated to memory but not requested by users or applications are lost. This behavior can result in gaps, but it ensures that a single sequence object will never issue the same value twice unless it is defined as CYCLE or manually restarted.

The cache is managed in memory by tracking the current value (the last issued value) and the remaining values in the cache. Hence, the cache consumes memory equivalent to two instances of the sequence object's data type.

When the cache argument is set to NO CACHE, the current sequence value is written to the system tables every time a sequence is used. This approach may slow down performance due to increased disk access, but it reduces the likelihood of unintended gaps. However, gaps can still occur if numbers are requested using the NEXT VALUE FOR or sp_sequence_get_range functions but are either unused or used within uncommitted transactions.

This article is contributed by **ARSHPREET SINGH**. If you like GeeksforGeeks and would like to contribute, you can also write an article using write.geeksforgeeks.org or mail your article to review-team@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks. Please write comments if you find anything incorrect, or if you want to share more information about the topic discussed above.

Last Updated : 05 Jun, 2023

67

Similar Reads

1. Difference between Structured Query Language (SQL) and Transact-SQL (T-SQL)

2. Configure SQL Jobs in SQL Server using T-SQL

3. SQL vs NO SQL vs NEW SQL

4. SQL | Procedures in PL/SQL

5. SQL | Difference between functions and stored procedures in PL/SQL

6. SQL SERVER – Input and Output Parameter For Dynamic SQL

7. Difference between SQL and T-SQL



SQL | Query Processing

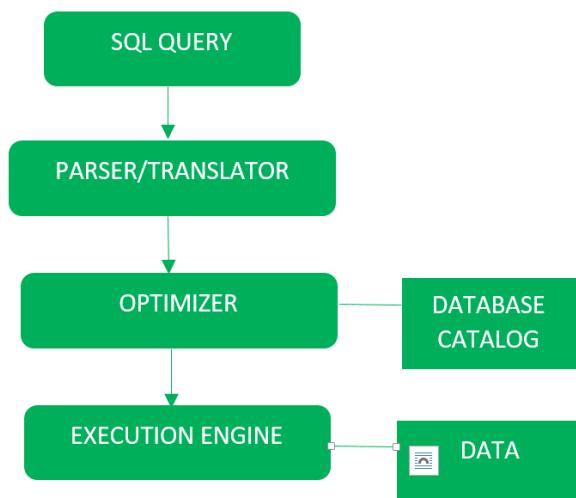


priyankagujral

[Read](#)[Discuss](#)[Courses](#)[Practice](#)

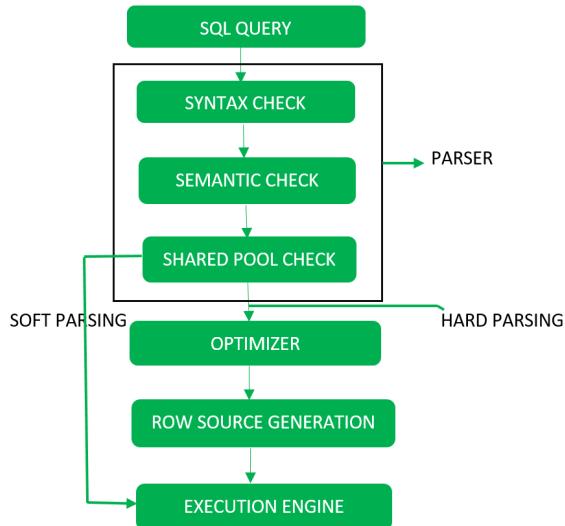
Query Processing includes translations on high level Queries into low level expressions that can be used at physical level of file system, query optimization and actual execution of query to get the actual result.

Block Diagram of Query Processing is as:



Detailed Diagram is drawn as:

AD



It is done in the following steps:

- **Step-1:**

Parser: During parse call, the database performs the following checks- Syntax check, Semantic check and Shared pool check, after converting the query into relational algebra. Parser performs the following checks as (refer detailed diagram):

1. **Syntax check** – concludes SQL syntactic validity. Example:

```
SELECT * FORM employee
```

Here error of wrong spelling of FROM is given by this check.

2. **Semantic check** – determines whether the statement is meaningful or not. Example: query contains a tablename which does not exist is checked by this check.
3. **Shared Pool check** – Every query possess a hash code during its execution. So, this check determines existence of written hash code in shared pool if code exists in shared pool then database will not take additional steps for optimization and execution.

Hard Parse and Soft Parse –

If there is a fresh query and its hash code does not exist in shared pool then that query has to pass through from the additional steps known as hard parsing otherwise if hash code exists then query does not passes through additional steps. It just passes directly to execution engine (refer detailed diagram). This is known as soft parsing.

Hard Parse includes following steps – Optimizer and Row source generation.

- **Step-2:**

Optimizer: During optimization stage, database must perform a hard parse atleast for one unique DML statement and perform optimization during this parse. This database never optimizes DDL unless it includes a DML component such as subquery that require optimization.

It is a process in which multiple query execution plan for satisfying a query are examined and most efficient query plan is satisfied for execution.

Database catalog stores the execution plans and then optimizer passes the lowest cost plan for execution.

Row Source Generation –

The Row Source Generation is a software that receives a optimal execution plan from the optimizer and produces an iterative execution plan that is usable by the rest of the database. the iterative plan is the binary program that when executes by the sql engine produces the result set.

- **Step-3:**

Execution Engine: Finally runs the query and display the required result.

Last Updated : 14 Aug, 2018

52

Similar Reads

1. Difference between Structured Query Language (SQL) and Transact-SQL (T-SQL)
2. Online Transaction Processing (OLTP) and Online Analytic Processing (OLAP)
3. SQL Query to Add Email Validation Using Only One Query
4. SQL Query to Check if Date is Greater Than Today in SQL
5. SQL Query to Add a New Column After an Existing Column in SQL
6. SQL Query to Convert Rows to Columns in SQL Server
7. Query Processing in Distributed DBMS
8. Configure SQL Jobs in SQL Server using T-SQL
9. SQL vs NO SQL vs NEW SQL
10. SQL | SELECT Query

Previous

Next

Article Contributed By :



priyankagujral
priyankagujral

Vote for difficulty

Current difficulty : Medium



Types of Keys in Relational Model (Candidate, Super, Primary, Alternate and Foreign)

[Read](#)[Discuss](#)

Pre-Requisite: [DBMS | Relational Model Introduction and Codd Rules](#)

Keys are one of the basic requirements of a [relational database model](#). It is widely used to identify the [tuples\(rows\)](#) uniquely in the table. We also use keys to set up relations amongst various columns and tables of a relational database.

Different Types of Keys in the Relational Model

1. [Candidate Key](#)
2. [Primary Key](#)
3. [Super Key](#)
4. [Alternate Key](#)

5. [Foreign Key](#)

[Engineering Mathematics](#) [Discrete Mathematics](#) [Digital Logic and Design](#) [Computer Organization and Architecture](#)

1. Candidate Key: The minimal set of attributes that can uniquely identify a tuple is known as a candidate key. For Example, STUD_NO in STUDENT relation.

- It is a minimal super key.
- It is a super key with no repeated data is called a candidate key.
- The minimal set of attributes that can uniquely identify a record.
- It must contain unique values.
- It can contain NULL values.
- Every table must have at least a single candidate key.
- A table can have multiple candidate keys but only one primary key (the primary key cannot have a NULL value, so the candidate key with a NULL value can't be the primary key).
- The value of the Candidate Key is unique and may be null for a tuple.
- There can be more than one candidate key in a relationship.

Example:



STUD_NO is the candidate key for relation STUDENT.

Table STUDENT

STUD_NO	SNAME	ADDRESS	PHONE
1	Shyam	Delhi	123456789
2	Rakesh	Kolkata	223365796
3	Suraj	Delhi	175468965

- The candidate key can be simple (having only one attribute) or composite as well.

Example:

{STUD_NO, COURSE_NO} is a composite candidate key for relation STUDENT_COURSE.

Table STUDENT_COURSE

STUD_NO	TEACHER_NO	COURSE_NO
1	001	C001
2	056	C005

Note: In SQL Server a unique constraint that has a nullable column, **allows** the value ‘null’ in that column **only once**. That’s why the STUD_PHONE attribute is a candidate here, but can not be a ‘null’ value in the primary key attribute.

2. Primary Key: There can be more than one candidate key in relation out of which one can be chosen as the primary key. For Example, STUD_NO, as well as STUD_PHONE, are candidate

keys for relation STUDENT but STUD_NO can be chosen as the primary key (only one out of many candidate keys).

- It is a unique key.
- It can identify only one tuple (a record) at a time.
- It has no duplicate values, it has unique values.
- It cannot be NULL.
- Primary keys are not necessarily to be a single column; more than one column can also be a primary key for a table.

Example:

STUDENT table -> Student(STUD_NO, SNAME, ADDRESS, PHONE) , STUD_NO is a primary key

Table STUDENT

STUD_NO	SNAME	ADDRESS	PHONE
1	Shyam	Delhi	123456789
2	Rakesh	Kolkata	223365796
3	Suraj	Delhi	175468965

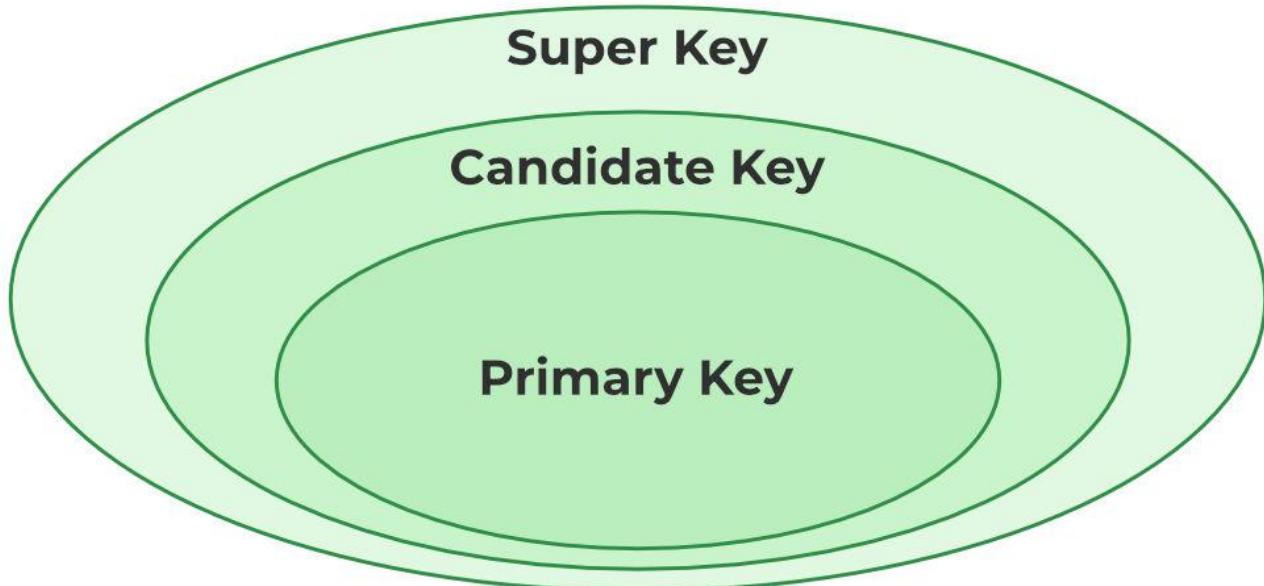
3. Super Key: The set of attributes that can uniquely identify a tuple is known as Super Key. For Example, STUD_NO, (STUD_NO, STUD_NAME), etc. A super key is a group of single or multiple keys that identifies rows in a table. It supports NULL values.

- Adding zero or more attributes to the candidate key generates the super key.
- A candidate key is a super key but vice versa is not true.

Example:

Consider the table shown above.

STUD_NO+PHONE is a super key.



Relation between Primary Key, Candidate Key, and Super Key

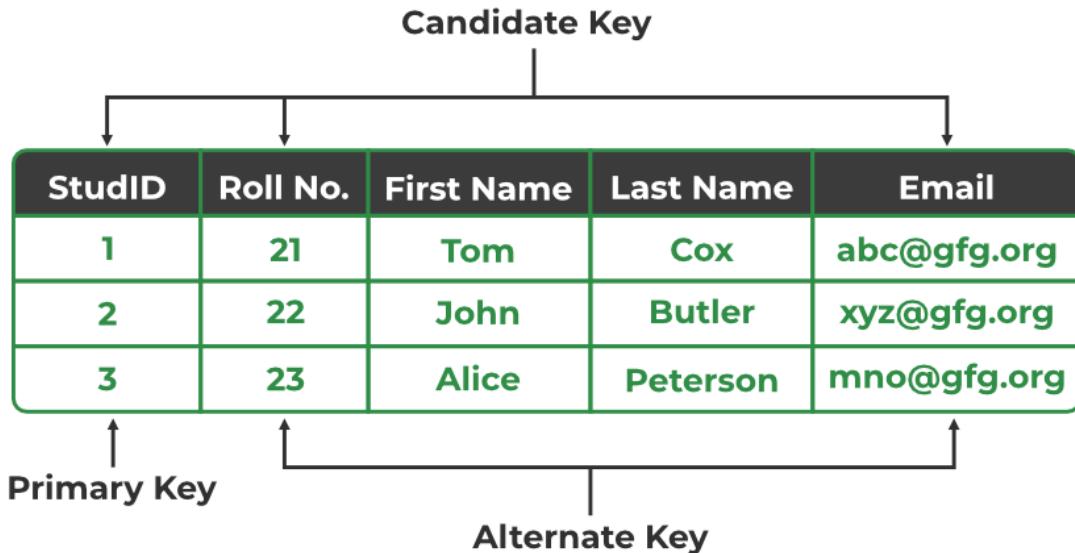
4. Alternate Key: The candidate key other than the primary key is called an alternate key.

- All the keys which are not primary keys are called alternate keys.
- It is a secondary key.
- It contains two or more fields to identify two or more records.
- These values are repeated.
- Eg:- SNAME, and ADDRESS is Alternate keys

Example:

Consider the table shown above.

STUD_NO, as well as PHONE both,
are candidate keys for relation STUDENT but
PHONE will be an alternate key
(only one out of many candidate keys).



Primary Key, Candidate Key, and Alternate Key

5. Foreign Key: If an attribute can only take the values which are present as values of some other attribute, it will be a foreign key to the attribute to which it refers. The relation which is being referenced is called referenced relation and the corresponding attribute is called referenced attribute the relation which refers to the referenced relation is called referencing relation and the corresponding attribute is called referencing attribute. The referenced attribute of the referenced relation should be the primary key to it.

- It is a key it acts as a primary key in one table and it acts as secondary key in another table.
- It combines two or more relations (tables) at a time.
- They act as a cross-reference between the tables.
- For example, DNO is a primary key in the DEPT table and a non-key in EMP

Example:

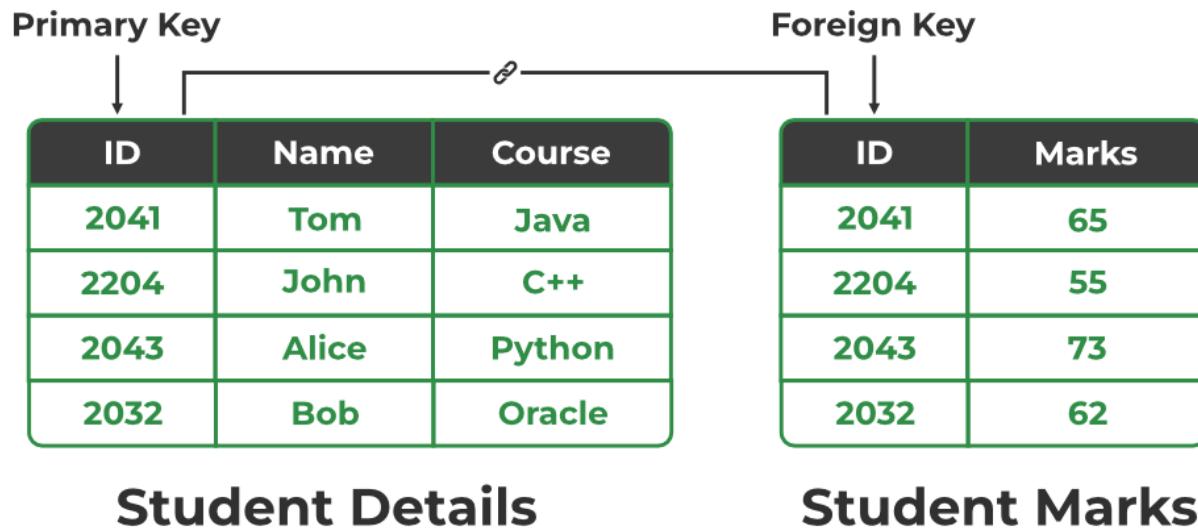
Refer Table STUDENT shown above.

STUD_NO in STUDENT_COURSE is a foreign key to STUD_NO in STUDENT relation.

Table STUDENT_COURSE

STUD_NO	TEACHER_NO	COURSE_NO
1	005	C001
2	056	C005

It may be worth noting that, unlike the Primary Key of any given relation, Foreign Key can be NULL as well as may contain duplicate tuples i.e. it need not follow uniqueness constraint. For Example, STUD_NO in the STUDENT_COURSE relation is not unique. It has been repeated for the first and third tuples. However, the STUD_NO in STUDENT relation is a primary key and it needs to be always unique, and it cannot be null.



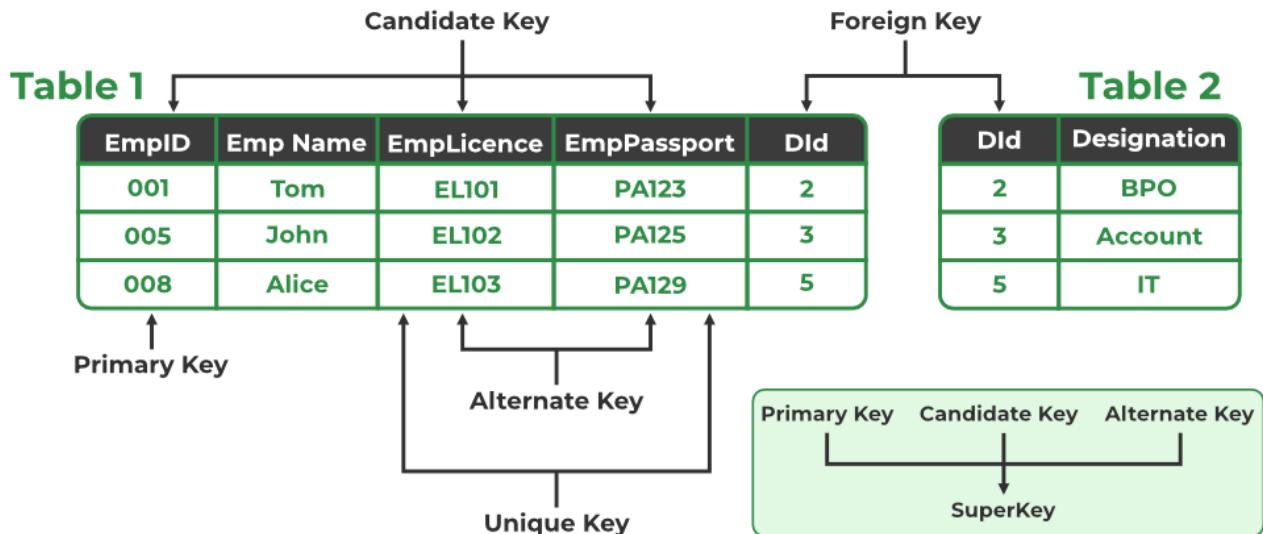
Relation between Primary Key and Foreign Key

6. Composite Key: Sometimes, a table might not have a single column/attribute that uniquely identifies all the records of a table. To uniquely identify rows of a table, a combination of two or more columns/attributes can be used. It still can give duplicate values in rare cases. So, we need to find the optimal set of attributes that can uniquely identify rows in a table.

- It acts as a primary key if there is no primary key in a table
- Two or more attributes are used together to make a composite key.
- Different combinations of attributes may give different accuracy in terms of identifying the rows uniquely.

Example:

FULLNAME + DOB can be combined together to access the details of a student.



Different Types of Keys

FAQs

Why keys are necessary for DBMS?

- Keys are one of the important aspects of DBMS. Keys help us to find the tuples(rows) uniquely in the table. It is also used in developing various relations amongst columns or tables of the database.

What is a Unique Key?

- Unique Keys are the keys that define the record uniquely in the table. It is different from Primary Keys, as Unique Key can contain one NULL value but Primary Key does not contain any NULL values.

What is Artificial Key?

- Artificial Keys are the keys that are used when no attributes contain all the properties of the Primary Key or if the Primary key is very large and complex.

Last Updated : 21 Mar, 2023

378

Similar Reads

1. Difference between Primary and Candidate Key
2. Difference between Super Key and Candidate Key
3. Why Candidate Key is Called a Minimal Super Key?
4. Difference between Primary Key and Foreign Key



SQL | Constraints

[Read](#)[Discuss](#)[Courses](#)[Practice](#)[Video](#)

Constraints are the rules that we can apply on the type of data in a table. That is, we can specify the limit on the type of data that can be stored in a particular column in a table using constraints.

The available constraints in SQL are:

- **NOT NULL:** This constraint tells that we cannot store a null value in a column. That is, if a column is specified as NOT NULL then we will not be able to store null in this particular column any more.
- **UNIQUE:** This constraint when specified with a column, tells that all the values in the column must be unique. That is, the values in any row of a column must not be repeated.
- **PRIMARY KEY:** A primary key is a field which can uniquely identify each row in a table. And this constraint is used to specify a field in a table as primary key.
- **FOREIGN KEY:** A Foreign key is a field which can uniquely identify each row in another table. And this constraint is used to specify a field as Foreign key.
- **CHECK:** This constraint helps to validate the values of a column to meet a particular condition. That is, it helps to ensure that the value stored in a column meets a specific condition.
- **DEFAULT:** This constraint specifies a default value for the column when no value is specified by the user.

How to specify constraints?

We can specify constraints at the time of creating the table using CREATE TABLE statement.

We can also specify the constraints after creating a table using ALTER TABLE statement.

Syntax:

Below is the syntax to create constraints using CREATE TABLE statement at the time of creating the table.

AD

```
CREATE TABLE sample_table
(
column1 data_type(size) constraint_name,
column2 data_type(size) constraint_name,
column3 data_type(size) constraint_name,
....;
);
```

sample_table: Name of the table to be created.

data_type: Type of data that can be stored in the field.

constraint_name: Name of the constraint. for example- NOT NULL, UNIQUE, PRIMARY KEY etc.

Let us see each of the constraint in detail.

1. NOT NULL –

If we specify a field in a table to be NOT NULL. Then the field will never accept null value. That is, you will be not allowed to insert a new row in the table without specifying any value to this field.

For example, the below query creates a table Student with the fields ID and NAME as NOT NULL. That is, we are bound to specify values for these two fields every time we wish to insert a new row.

```
CREATE TABLE Student
(
ID int(6) NOT NULL,
NAME varchar(10) NOT NULL,
ADDRESS varchar(20)
);
```

2. UNIQUE –

This constraint helps to uniquely identify each row in the table. i.e. for a particular column, all the rows should have unique values. We can have more than one UNIQUE columns in a table.

For example, the below query creates a table Student where the field ID is specified as UNIQUE. i.e, no two students can have the same ID. [Unique constraint in detail.](#)

```
CREATE TABLE Student
(
    ID int(6) NOT NULL UNIQUE,
    NAME varchar(10),
    ADDRESS varchar(20)
);
```

3. PRIMARY KEY –

Primary Key is a field which uniquely identifies each row in the table. If a field in a table as primary key, then the field will not be able to contain NULL values as well as all the rows should have unique values for this field. So, in other words we can say that this is combination of NOT NULL and UNIQUE constraints.

A table can have only one field as primary key. Below query will create a table named Student and specifies the field ID as primary key.

```
CREATE TABLE Student
(
    ID int(6) NOT NULL UNIQUE,
    NAME varchar(10),
    ADDRESS varchar(20),
    PRIMARY KEY(ID)
);
```

4. FOREIGN KEY –

Foreign Key is a field in a table which uniquely identifies each row of another table. That is, this field points to primary key of another table. This usually creates a kind of link between the tables.

Consider the two tables as shown below:

Orders

O_ID	ORDER_NOC_ID	
1	2253	3
2	3325	3
3	4521	2

4	8532	1
---	------	---

Customers

C_ID	NAME	ADDRESS
1	RAMESH	DELHI
2	SURESH	NOIDA
3	DHARMESH	GURGAON

As we can see clearly that the field C_ID in Orders table is the primary key in Customers table, i.e. it uniquely identifies each row in the Customers table. Therefore, it is a Foreign Key in Orders table.

Syntax:

```
CREATE TABLE Orders
(
O_ID int NOT NULL,
ORDER_NO int NOT NULL,
C_ID int,
PRIMARY KEY (O_ID),
FOREIGN KEY (C_ID) REFERENCES Customers(C_ID)
)
```

(i) CHECK –

Using the CHECK constraint we can specify a condition for a field, which should be satisfied at the time of entering values for this field.

For example, the below query creates a table Student and specifies the condition for the field AGE as (AGE >= 18). That is, the user will not be allowed to enter any record in the table with AGE < 18. [Check constraint in detail](#)

```
CREATE TABLE Student
(
ID int(6) NOT NULL,
NAME varchar(10) NOT NULL,
AGE int NOT NULL CHECK (AGE >= 18)
);
```

(ii) DEFAULT –

This constraint is used to provide a default value for the fields. That is, if at the time of entering new records in the table if the user does not specify any value for these fields then the default value will be assigned to them.

For example, the below query will create a table named Student and specify the default value for the field AGE as 18.

```
CREATE TABLE Student
(
    ID int(6) NOT NULL,
    NAME varchar(10) NOT NULL,
    AGE int DEFAULT 18
);
```

This article is contributed by [Harsh Agarwal](#). If you like GeeksforGeeks and would like to contribute, you can also write an article using write.geeksforgeeks.org or mail your article to review-team@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

If you like GeeksforGeeks and would like to contribute, you can also write an article and mail your article to review-team@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Last Updated : 09 Jun, 2021

124

Similar Reads

1. [Difference between Entity constraints, Referential constraints and Semantic constraints](#)
2. [SQL | Checking Existing Constraints on a Table using Data Dictionaries](#)
3. [SQL Query to Drop Unique Key Constraints Using ALTER Command](#)
4. [SQL Query to Add Foreign Key Constraints Using ALTER Command](#)
5. [SQL Query to Add Unique key Constraints Using ALTER Command](#)

[Trending Now](#) [DSA](#) [Data Structures](#) [Algorithms](#) [Interview Preparation](#) [Data Science](#) [Topic-wise Practice](#) [Jobs](#)

Primary key constraint in SQL

 dhatriganda07[Read](#) [Discuss](#) [Courses](#) [Practice](#)

A primary key constraint depicts a key comprising one or more columns that will help uniquely identify every tuple/record in a table.

Properties :

1. No duplicate values are allowed, i.e. Column assigned as primary key should have UNIQUE values only.
2. NO NULL values are present in column with Primary key. Hence there is Mandatory value in column having Primary key.
3. Only one primary key per table exist although Primary key may have multiple columns.
4. No new row can be inserted with the already existing primary key.
5. Classified as : a) Simple primary key that has a Single column 2) Composite primary key has Multiple column.
6. Defined in Create table / Alter table statement.

The primary key can be created in a table using PRIMARY KEY constraint. It can be created at two levels.

1. Column
2. Table.

SQL PRIMARY KEY at Column Level :

If Primary key contains just one column, it should be defined at column level. The following code creates the Primary key “ID” on the person table.

AD

Syntax :

```
Create Table Person
(
Id int NOT NULL PRIMARY KEY,
Name varchar2(20),
Address varchar2(50)
);
```

Here NOT NULL is added to make sure ID should have unique values. SQL will automatically set null values to the primary key if it is not specified.

Example-1 :

To verify the working of Primary key :

```
Insert into Person values(1, 'Ajay', 'Mumbai');
```

Output :

```
1 row created
```

Example-2 :

Let's see if you will insert the same values again.

```
Insert into Person values(1, 'Ajay', 'Mumbai');
```

Output :

```
Error at line 1: unique constraint violated
```

Since we are inserting duplicate values, an error will be thrown since the Primary key "Id" can contain only unique values.

Example-3 :

```
Insert into Person values('', 'Ajay', 'Mumbai');
```

Output :

```
Error at line 1: cannot insert Null into<"user"."Person"."ID">
```

Primary Key cannot contain Null Values so That too will throw an error.

SQL PRIMARY KEY at Table Level :

Whenever the primary key contains multiple columns it has to be specified at Table level.

Syntax:

```
Create Table Person
(Id int NOT NULL,
Name varchar2(20),
Address varchar2(50),
PRIMARY KEY(Id, Name)
);
```

Here, you have only one Primary Key in a table but it consists of Multiple Columns(Id, Name). However, the Following are permissible.

```
Insert into Person values(1, 'Ajay', 'Mumbai');
Insert into Person values(2, 'Ajay', 'Mumbai');
```

Since multiple columns make up Primary Key so both the rows are considered different. SQL permits either of the two values can be duplicated but the combination must be unique.

SQL PRIMARY KEY with ALTER TABLE :

Most of the time, Primary Key is defined during the creation of the table but sometimes the Primary key may not be created in the already existing table. We can however add Primary Key using Alter Statement.

Syntax :

```
Alter Table Person add Primary Key(Id);
```

To add Primary key in multiple columns using the following query.

```
Alter Table Person add Primary Key(Id, Name);
```

It is necessary that the column added as primary key MUST contain unique values or else it will be violated. An id cannot be made Primary key if it contains duplicate values. It violates the basic rule of Primary Key. Altering the table to add Id as a primary key that may contain duplicate values generates an error.

Output :

```
Error at line 1: cannot validate- primary key violated
```

Also, A column added as primary key using alter statement should not have null values. Altering table to add Id as primary key that may contain null values generates an error.

Output :

```
Error at line 1: column contains NULL values; cannot alter to NOT NULL
```

DELETE PRIMARY KEY CONSTRAINT :

To remove Primary Key constraint on table use given SQL as follows.

```
ALTER table Person DROP PRIMARY KEY;
```

Last Updated : 11 Nov, 2020

6

Similar Reads

1. Foreign Key constraint in SQL
2. SQL Query to Drop Foreign Key Constraint Using ALTER Command
3. Difference between Primary key and Unique key
4. Primary key in MS SQL Server
5. SQL Query to Check or Find the Column Name Which Is Primary Key Column
6. SQL Query to Create Table With a Primary Key
7. Change Primary Key Column in SQL Server
8. SQL Query to Remove Primary Key
9. How to Create a Composite Primary Key in SQL Server?
10. SQL | CHECK Constraint

Previous

Next

Article Contributed By :



dhatriganda07
dhatriganda07

Vote for difficulty

Current difficulty : Basic

Easy

Normal

Medium

Hard

Expert

Article Tags : DBMS-SQL, SQL



SQL NOT NULL Constraint



sarithaakshaj

[Read](#)[Discuss](#)[Courses](#)[Practice](#)

Pre-requisite: [NULL Values in SQL](#)

The SQL NOT NULL forces particular values or records should not hold a null value. It is somewhat similar to the primary key condition as the primary key can't have null values in the table although both are completely different things.

In SQL, constraints are some set of rules that are applied to the data type of the specified table, Or we can say that using constraints we can apply limits on the type of data that can be stored in the particular column of the table.

Constraints are typically placed specified along with CREATE statement. By default, a column can hold null values.

AD

Query:

```
CREATE TABLE Emp(
    EmpID INT PRIMARY KEY,
    Name VARCHAR(50),
    Country VARCHAR(50),
    Age int(2),
    Salary int(10)
);
```

Output:

Emp

EmpID	Name	Country	Age	Salary
empty				

If you don't want to have a null column or a null value you need to define constraints like NOT NULL. **NOT NULL** constraints make sure that a column does not hold null values, or in other words, NOT NULL constraints make sure that you cannot insert a new record or update a record without entering a value to the specified column(i.e., NOT NULL column).

It prevents for acceptance of NULL values. It can be applied to column-level constraints.

Syntax:

```
CREATE TABLE table_Name
(
    column1 data_type(size) NOT NULL,
    column2 data_type(size) NOT NULL,
    ....
);
```

SQL NOT NULL on CREATE a Table

In SQL, we can add NOT NULL constraints while creating a table.

For example, the “EMPID” will not accept NULL values when the EMPLOYEES table is created because NOT NULL constraints are used with these columns.

Query:

```
CREATE TABLE Emp(
    EmpID INT NOT NULL PRIMARY KEY,
    Name VARCHAR (50),
    Country VARCHAR(50),
    Age int(2),
    Salary int(10));
```

Output:

Field	Type	Null	Key	Default	Extra
EmpID	int	NO	PRI	NULL	
Name	varchar(50)		YES		NULL
Country	varchar(50)		YES		NULL
Age	int	YES		NULL	
Salary	int	YES		NULL	

SQL NOT NULL on ALTER Table

We can also add a NOT NULL constraint in the existing table using the [ALTER](#) statement. For example, if the EMPLOYEES table has already been created then add NOT NULL constraints to the “Name” column using ALTER statements in SQL as follows:

Query:

```
ALTER TABLE Emp modify Name Varchar(50) NOT NULL;
```

Last Updated : 06 Apr, 2023

3

Similar Reads

1. [Dangling, Void , Null and Wild Pointers](#)

2. [Factorial of a number in PL/SQL](#)

3. [SQL HAVING Clause with Examples](#)

4. [SQL Drop Table Statement](#)

5. [SQL USE Database Statement](#)

6. [How to Connect Python with SQL Database?](#)

7. [SET ROWCOUNT Function in SQL Server](#)

8. [Interface Python with an SQL Database](#)

9. [List Methods in Python | Set 1 \(in, not in, len\(\), min\(\), max\(\)...\)](#)

10. [Check if a number is Full Fibonacci or not](#)

Related Tutorials



SQL | UNIQUE Constraint

Read Discuss Courses Practice

[SQL Constraints](#) Unique constraint in SQL is used to check whether the sub-query has duplicate tuples in its result. It returns a boolean value indicating the presence/absence of duplicate tuples. Unique construct returns true only if the subquery has no duplicate tuples, else it returns false.

Important Points:

- Evaluates to true on an empty subquery.
- Returns true only if there are unique tuples present as the output of the sub-query (two tuples are unique if the value of any attribute of the two tuples differs).
- Returns true if the sub-query has two duplicate rows with at least one attribute as NULL.

Syntax:

```
CREATE TABLE table_name (
    column1 datatype UNIQUE,
    column2 datatype,
    ...
);
```

AD

Parameter Explanation:

1. **UNIQUE**: It means that in that particular column, the data should be unique.

We can directly calculate the unique data in the column without using unique data words in SQL but the UNIQUE keyword ease the complexity. Suppose that we have one table in which we have to calculate the unique data in the ID column.

Query:

```
SELECT table.ID
FROM table
WHERE UNIQUE (SELECT table2.ID
               FROM table2
               WHERE table.ID = table2.ID);
```

Note: During the execution, first the outer query is evaluated to obtain the table. ID. Following this, the inner subquery is processed which produces a new relation that contains the output of the inner query such as the table.ID == table2.ID. If every row in the new relation is unique then unique returns true and the corresponding table.ID is added as a tuple in the output relation produced. However, if every row in the new relationship is not unique then unique evaluates to false and the corresponding table.ID is not added to the output relation. Unique applied to a subquery return false if and only if there are two tuples t1 and t2 such that t1 = t2. It considers t1 and t2 to be two different tuples when unique is applied to a subquery that contains t1 and t2 such that t1 = t2 and at least one of the attributes of these tuples contains a NULL value. The unique predicate in this case evaluates to true. This is because, a NULL value in SQL is treated as an unknown value therefore, two NULL values are considered to be distinct.

The SQL statement without a UNIQUE clause can also be written as:

Query:

```
SELECT table.ID
FROM table
WHERE 1 <= (SELECT count(table2.ID)
              FROM table2
              WHERE table.ID = table2.ID);
```

Rules for the data in a table can be specified using SQL constraints.

The kinds of data that can be entered into a table are restricted by constraints. This guarantees the reliability and accuracy of the data in the table. The action is stopped if there is a violation between the constraint and the data action column-level or table-level constraints are both possible. Table-level constraints apply to the entire table, while column-level constraints only affect the specified column.

In SQL, the following restrictions are frequently applied:

- **NULL VALUE:** A column cannot have a NULL value by using the NOT NULL flag.

- **UNIQUE KEY:** A unique value makes sure that each value in a column is distinct.
- **PRIMARY KEY:** A NOT NULL and UNIQUE combination. Identifies each row in a table individually.
- **A FOREIGN KEY:** Guards against actions that would break links between tables.
- **CHECK:** Verifies that the values in a column meet a particular requirement.

SQL Unique Constraint on ALTER Table

We can add a unique column in a table using ALTER. Suppose that we have one table with the named instructor and we want to insert one unique column in the instructor.

Syntax:

ALTER TABLE Instructor

ADD UNIQUE(ID);

To DROP a Unique Constraint

Suppose we have to DROP that column, particularly in the table.

Syntax:

ALTER TABLE Instructor

DROP INDEX ID;

Queries:

Find all the instructors that taught at most one course in the year 2017.

Instructor relation:

EmployeeID	Name	CourseID	Year
77505	Alan	SC110	2017
77815	Will	CSE774	2017
85019	Smith	EE457	2017
92701	Sam	PYS504	2017

EmployeeID	Name	CourseID	Year
60215	Harold	HSS103	2016
77505	Alan	BIO775	2017
92701	Sam	ECO980	2017

Without Using Unique Constraint

Query:

```
SELECT I.EMPLOYEEID, I.NAME
FROM Instructor as I
WHERE UNIQUE (SELECT Inst.EMPLOYEEID
               FROM Instructor as Inst
               WHERE I.EMPLOYEEID = Inst.EMPLOYEEID
                     and Inst.YEAR = 2017);
```

By Using Unique Constraint

Query:

```
SELECT Name
FROM Instructor
WHERE Year = 2017
GROUP BY Name
HAVING COUNT(DISTINCT CourseID) = 1;
```

Output:

EmployeeID	Name
77815	Will
85019	Smith

Explanation: In the Instructor relation, only instructors Will and Smith teach a single course during the year 2017. The sub-query corresponding to these instructors contains only a single tuple and therefore the unique clause corresponding to these instructors evaluates to true thereby producing these two instructors in the output relation.

Example 2

Find all the courses in the Computer Science department that has only a single instructor allotted to that course.

Course relation:

CourseID	Name	Department	InstructorID
CSE505	Computer Network	Computer Science	11071
CSE245	Operating System	Computer Science	74505
CSE101	Programming	Computer Science	12715
HSS505	Psychology	Social Science	85017
EE475	Signals & Systems	Electrical	22150
CSE314	DBMS	Computer Science	44704
CSE505	Computer Network	Computer Science	11747
CSE314	DBMS	Computer Science	44715

Without Unique Constraint

Query:

```
SELECT C.COURSEID, C.NAME
FROM Course as C
WHERE UNIQUE (SELECT T.INSTRUCTORID
              FROM Course as T
              WHERE T.COURSEID = C.COURSEID
                and C.DEPARTMENT = 'Computer Science');
```

By using UNIQUE Constraint

Query:

```
SELECT CourseID
FROM Instructor
```

```

WHERE CourseID LIKE 'CSE%'
GROUP BY CourseID
HAVING COUNT(DISTINCT Name) = 1;

```

Output:

COURSE_ID	Name
CSE245	Operating System
CSE101	Programming

Explanation: In the course relation, the only courses in the computer science department that have a single instructor allotted are Operating Systems and Programming. The unique constraint corresponding to these two courses has only a single tuple consisting of the corresponding instructors. So, the unique clause for these two courses evaluates to true and these courses are displayed in output relation. Other courses in the Course relation either have two or more instructors or they do not belong to the computer science department and therefore, those courses aren't displayed in the output relation.

Frequently Asked Question(FAQ)

Q.1 What is a unique constraint in SQL?

Ans. A UNIQUE constraint ensures that all values in a column are unique. UNIQUE and PRIMARY KEY constraints provide uniqueness guarantees for a column or set of columns. PRIMARY KEY constraints automatically have UNIQUE constraints.

This article is contributed by **Mayank Kumar**. If you like GeeksforGeeks and would like to contribute, you can also write an article using [write.geeksforgeeks.org](https://www.geeksforgeeks.org/write-for-us/) or mail your article to review-team@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks. Please write comments if you find anything incorrect, or if you want to share more information about the topic discussed above.

Last Updated : 14 Apr, 2023

15

Similar Reads

1. Unique Constraint in MS SQL Server

 2. SQL | CHECK Constraint
-



SQL | DEFAULT Constraint



khushboogoyal499

[Read](#) [Discuss](#) [Courses](#) [Practice](#)

The **DEFAULT Constraint** is used to fill a column with a default and fixed value. The value will be added to all new records when no other value is provided.

Syntax

*CREATE TABLE tablename (Columnname **DEFAULT** 'defaultvalue');*

Using DEFAULT on CREATE TABLE

To set a DEFAULT value for the “Location” column when the “Geeks” table is created.

AD

Query:

```
CREATE TABLE Geeks (
    ID int NOT NULL,
    Name varchar(255),
    Age int,
    Location varchar(255) DEFAULT 'Noida');
INSERT INTO Geeks VALUES (4, 'Mira', 23, 'Delhi');
INSERT INTO Geeks VALUES (5, 'Hema', 27,DEFAULT);
INSERT INTO Geeks VALUES (6, 'Neha', 25, 'Delhi');
INSERT INTO Geeks VALUES (7, 'Khushi', 26,DEFAULT);
Select * from Geeks;
```

Output:

ID	Name	Age	Location
4	Mira	23	Delhi
5	Hema	27	Noida
6	Neha	25	Delhi
7	Khushi	26	Noida

DROP a DEFAULT Constraint

Syntax

ALTER TABLE tablename

ALTER COLUMN columnname

DROP DEFAULT;

Query:

```
ALTER TABLE Geeks
ALTER COLUMN Location
DROP DEFAULT;
```

Let us add 2 new rows in the Geeks table :

Query:

```
INSERT INTO Geeks VALUES (8, 'Komal', 24, 'Delhi');
INSERT INTO Geeks VALUES (9, 'Payal', 26, NULL);
```

Note – Dropping the default constraint will not affect the current data in the table, it will only apply to new rows.

```
Select * from Geeks;
```

Output:

ID	Name	Age	Location
4	Mira	23	Delhi
5	Hema	27	Noida
6	Neha	25	Delhi
7	Khushi	26	Noida
8	Komal	24	Delhi
9	Payal	26	NULL



Foreign Key constraint in SQL



dhatriganda07

[Read](#)[Discuss](#)[Courses](#)[Practice](#)

Foreign Key is a column that refers to the primary key/unique key of other table. So it demonstrates relationship between tables and act as cross reference among them. Table in which foreign key is defined is called Foreign table/Referencing table. Table that defines primary/unique key and is referenced by foreign key is called primary table/master table/Referenced Table. It is Defined in Create table/Alter table statement.

For the table that contains Foreign key, it should match the primary key in referenced table for every row. This is called Referential Integrity. Foreign key ensures referential integrity.

Properties :

- Parent that is being referenced has to be unique/Primary Key.
- Child may have duplicates and nulls.
- Parent record can be deleted if no child exists.
- Master table cannot be updated if child exists.
- Must reference PRIMARY KEY in primary table.
- Foreign key column and constraint column should have matching data types.
- Records cannot be inserted in child table if corresponding record in master table do not exist.
- Records of master table cannot be deleted if corresponding records in child table exits.

1. SQL Foreign key At column level :

Syntax –

AD

```
Create table people (no int references person,
                    Fname varchar2(20));
```

OR

```
Create table people (no int references person(id),
                     Fname varchar2(20));
```

Here Person table should have primary key with type int. If there is single columnar Primary key in table, column name in syntax can be omitted. So both the above syntax works correctly.

To check the constraint,

- If Parent table doesn't have primary key.

OUTPUT :

```
Error at line 1 : referenced table does not have a primary key.
```

- If Parent table has Primary Key of different datatype.

OUTPUT :

```
Error at line 1 : column type incompatible with referenced column type.
```

2. SQL Foreign key At table level :

Syntax –

```
create table people(no varchar2(10),
                    fname varchar2(20),
                    foreign key(no) references person);
```

OR

```
create table people(no varchar2(10),
                    fname varchar2(20),
                    foreign key(no) references person(id));
```

Column name of referenced table can be ignored.

3. Insert Operation in Foreign Key Table :

If corresponding value in foreign table doesn't exists, a record in child table cannot be inserted.

OUTPUT :

```
Error at line 1 : integrity constraint violated - parent key not found.
```

4. Delete Operation in Foreign Key Table :

When a record in master table is deleted and corresponding record in child table exists, an error message is displayed and prevents delete operation from going through.

OUTPUT :

```
Error at line 1 : integrity constraint violated - child record found.
```

5. Foreign Key with ON DELETE CASCADE :

The default behavior of foreign key can be changed using ON DELETE CASCADE. When

this option is specified in foreign key definition, if a record is deleted in master table, all corresponding record in detail table will be deleted.

Syntax –

```
create table people(no varchar2(10),
                   fname varchar2(20),
                   foreign key(no)
                   references person on delete cascade);
```

Now deleting records from person will delete all corresponding records from child table.

OUTPUT :

```
select * from person;
no rows selected
```

```
select * from people;
no rows selected
```

6. Foreign Key with ON DELETE SET NULL :

A Foreign key with SET NULL ON DELETE means if record in parent table is deleted, corresponding records in child table will have foreign key fields set to null. Records in child table will not be deleted.

Syntax –

```
create table people(no varchar2(10),
                   fname varchar2(20),
                   foreign key(no)
                   references person on delete set null);
```

OUTPUT :

```
select * from person;
no rows selected
```

```
select * from people;
NO Fname
pqr
```

Notice the field “No” in people table that was referencing Primary key of Person table. On deleting person data, it will set null in child table people. But the record will not be deleted.

Last Updated : 19 Oct, 2020

8

Similar Reads

1. SQL Query to Drop Foreign Key Constraint Using ALTER Command



SQL | CHECK Constraint

Read Discuss Courses Practice

[SQL Constraints](#) Check Constraint is used to specify a predicate that every tuple must satisfy in a given relation. It limits the values that a column can hold in a relation.

- The predicate in check constraint can hold a sub query.
- Check constraint defined on an attribute restricts the range of values for that attribute.
- If the value being added to an attribute of a tuple violates the check constraint, the check constraint evaluates to false and the corresponding update is aborted.
- Check constraint is generally specified with the CREATE TABLE command in SQL.

Syntax:

```
CREATE TABLE pets(
    ID INT NOT NULL,
    Name VARCHAR(30) NOT NULL,
    Breed VARCHAR(20) NOT NULL,
    Age INT,
    GENDER VARCHAR(9),
    PRIMARY KEY(ID),
    check(GENDER in ('Male', 'Female', 'Unknown'))
);
```

Note: The check constraint in the above SQL command restricts the GENDER to belong to only the categories specified. If a new tuple is added or an existing tuple in the relation is updated with a GENDER that doesn't belong to any of the three categories mentioned, then the corresponding database update is aborted.

Query

AD

Constraint: Only students with age ≥ 17 are can enroll themselves in a university. **Schema for student database in university:**

```
CREATE TABLE student(
    StudentID INT NOT NULL,
    Name VARCHAR(30) NOT NULL,
    Age INT NOT NULL,
    GENDER VARCHAR(9),
    PRIMARY KEY(ID),
    check(Age >= 17)
);
```

Student relation:

StudentID	Name	Age	Gender
1001	Ron	18	Male
1002	Sam	17	Male
1003	Georgia	17	Female
1004	Erik	19	Unknown
1005	Christine	17	Female

Explanation: In the above relation, the age of all students is greater than equal to 17 years, according to the constraint mentioned in the check statement in the schema of the relation. If, however following SQL statement is executed:

```
INSERT INTO student(STUDENTID, NAME, AGE, GENDER)
VALUES (1006, 'Emma', 16, 'Female');
```

There won't be any database update and as the age < 17 years. **Different options to use Check constraint:**

- **With alter:** Check constraint can also be added to an already created relation using the syntax:

```
alter table TABLE_NAME modify COLUMN_NAME check(Predicate);
```

- **Giving variable name to check constraint:** Check constraints can be given a variable name using the syntax:

```
alter table TABLE_NAME add constraint CHECK_CONST check (Predicate);
```

- **Remove check constraint:** Check constraint can be removed from the relation in the database from SQL server using the syntax:

```
alter table TABLE_NAME drop constraint CHECK_CONSTRAINT_NAME;
```

- **Drop check constraint:** Check constraint can be dropped from the relation in the database in MySQL using the syntax:

```
alter table TABLE_NAME drop check CHECK_CONSTRAINT_NAME;
```

View existing constraints on a particular table

If you want to check if a constraint or any constraint exists within the table in mysql then you can use the following command. This command will show a tabular output of all the constraint-related data for the table name you've passed in the statement, in our case we'll use the employee table.

```
SELECT *
FROM information_schema.table_constraints
WHERE table_schema = schema()
AND table_name = 'employee';
```

This article is contributed by **Mayank Kumar**. If you like GeeksforGeeks and would like to contribute, you can also write an article using [write.geeksforgeeks.org](https://www.geeksforgeeks.org/write-your-article/) or mail your article to review-team@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Last Updated : 07 Feb, 2023

18

Similar Reads

1. Check Constraint in MS SQL Server

 2. SQL | UNIQUE Constraint
-

[Trending Now](#) [DSA](#) [Data Structures](#) [Algorithms](#) [Interview Preparation](#) [Data Science](#) [Topic-wise Practice](#) [Jobs](#)

SQL Operators

 varshachoudhary[Read](#)[Discuss](#)[Courses](#)[Practice](#)

Pre-requisites: [What is SQL?](#)

Structured Query Language is a computer language that we use to interact with a relational database. In this article we will see all types of SQL operators.

In simple operator can be defined as an entity used to perform operations in a table.

Operators are the foundation of any programming language. We can define operators as symbols that help us to perform specific mathematical and logical computations on operands. In other words, we can say that an operator operates the operands. SQL operators have three different categories.

AD

Types of SQL Operators

- [Arithmetic operator](#)
- [Comparison operator](#)
- [Logical operator](#)

Arithmetic Operators

We can use various arithmetic operators on the data stored in the tables. Arithmetic Operators are:

Operator	Description
+	The addition is used to perform an addition operation on the data values.

Operator	Description
_	This operator is used for the subtraction of the data values.
/	This operator works with the 'ALL' keyword and it calculates division operations.
*	This operator is used for multiplying data values.
%	Modulus is used to get the remainder when data is divided by another.

Example Query:

```
SELECT * FROM employee WHERE emp_city NOT LIKE 'A%';
```

Output:

Results		Messages		
	emp_id	emp_name	emp_city	emp_country
1	101	Utkarsh Tripathi	Varanasi	India
2	102	Abhinav Singh	Varanasi	India
3	103	Utkarsh Raghuvanshi	Varanasi	India
4	106	Ashutosh Kumar	Patna	India

Comparison Operators

Another important operator in SQL is a [comparison operator](#), which is used to compare one expression's value to other expressions. SQL supports different types of comparison operator, which is described below:

Operator	Description
=	Equal to.
>	Greater than.
<	Less than.
>=	Greater than equal to.
<=	Less than equal to.

Operator	Description
<>	Not equal to.

Example Query:

```
SELECT * FROM MATHS WHERE MARKS=50;
```

Output:

The screenshot shows a SQL query window titled "SQLQuery1.sql - D...P-S3KL81P\HP (60)*". The query "SELECT * FROM MATHS WHERE MARKS=50;" is entered. Below the query, the results pane displays a single row of data from the MATHS table:

	ROLL_NUMBER	S_NAME	MARKS
1	5	MOHAN	50

Logical Operators

The Logical operators are those that are true or false. They return true or false values to combine one or more true or false values.

Operator	Description
<u>AND</u>	Logical AND compares two Booleans as expressions and returns true when both expressions are true.
<u>OR</u>	Logical OR compares two Booleans as expressions and returns true when one of the expressions is true.
<u>NOT</u>	Not takes a single Boolean as an argument and change its value from false to true or from true to false.

Example Query:

```
SELECT * FROM employee WHERE emp_city =
'Allahabad' AND emp_country = 'India';
```

Output:

Results		Messages		
	emp_id	emp_name	emp_city	emp_country
1	104	Utkarsh Singh	Allahabad	India
2	105	Sudhanshu Yadav	Allahabad	India

Special Operators

Operators	Description
<u>ALL</u>	ALL is used to select all records of a SELECT STATEMENT. It compares a value to every value in a list of results from a query. The ALL must be preceded by the comparison operators and evaluated to TRUE if the query returns no rows.
<u>ANY</u>	ANY compares a value to each value in a list of results from a query and evaluates to true if the result of an inner query contains at least one row.
<u>BETWEEN</u>	The SQL BETWEEN operator tests an expression against a range. The range consists of a beginning, followed by an AND keyword and an end expression.
<u>IN</u>	The IN operator checks a value within a set of values separated by commas and retrieves the rows from the table that match.
<u>EXISTS</u>	The EXISTS checks the existence of a result of a subquery. The EXISTS subquery tests whether a subquery fetches at least one row. When no data is returned then this operator returns 'FALSE'.
<u>SOME</u>	SOME operator evaluates the condition between the outer and inner tables and evaluates to true if the final result returns any one row. If not, then it evaluates to false.
UNIQUE	The UNIQUE operator searches every unique row of a specified table.

Example Query:

```
SELECT * FROM employee WHERE emp_id BETWEEN 101 AND 104;
```

Output:

Results		Messages		
	emp_id	emp_name	emp_city	emp_country
1	101	Utkarsh Tripathi	Varanasi	India
2	102	Abhinav Singh	Varanasi	India
3	103	Utkarsh Raghuvanshi	Varanasi	India
4	104	Utkarsh Singh	Allahabad	India

Last Updated : 06 Apr, 2023

5

Similar Reads

1. Difference between Structured Query Language (SQL) and Transact-SQL (T-SQL)

2. Configure SQL Jobs in SQL Server using T-SQL

3. SQL AND and OR operators

4. SQL | Wildcard operators

5. SQL | Arithmetic Operators

6. SQL - Logical Operators

7. Comparison Operators in SQL

8. SQL | Procedures in PL/SQL

9. SQL | Difference between functions and stored procedures in PL/SQL

10. SQL SERVER – Input and Output Parameter For Dynamic SQL

[Previous](#)[Next](#)**Article Contributed By :**

varshachoudhary
varshachoudhary

Vote for difficultyCurrent difficulty : [Basic](#)



SQL | Arithmetic Operators



classyallrounder

[Read](#)[Discuss](#)[Courses](#)[Practice](#)

Prerequisite: [Basic Select statement](#), [Insert into clause](#), [Sql Create Clause](#), [SQL Aliases](#)

We can use various Arithmetic Operators on the data stored in the tables.

Arithmetic Operators are:

+	[Addition]
-	[Subtraction]
/	[Division]
*	[Multiplication]
%	[Modulus]

Addition (+) :

It is used to perform **addition operation** on the data items, items include either single column or multiple columns.

AD

Implementation:

```
SELECT employee_id, employee_name, salary, salary + 100
AS "salary + 100" FROM addition;
```

Output:

employee_id	employee_name	salary	salary+100
1	alex	25000	25100

2	rr	55000	55100
3	jpm	52000	52100
4	ggshmr	12312	12412

Here we have done addition of 100 to each Employee's salary i.e, addition operation on single column.

Let's perform **addition of 2 columns**:

```
SELECT employee_id, employee_name, salary, salary + employee_id
AS "salary + employee_id" FROM addition;
```

Output:

employee_id	employee_name	salary	salary+employee_id
1	alex	25000	25001
2	rr	55000	55002
3	jpm	52000	52003
4	ggshmr	12312	12316

Here we have done addition of 2 columns with each other i.e, each employee's employee_id is added with its salary.

Subtraction (-) :

It is use to perform **subtraction operation** on the data items, items include either single column or multiple columns.

Implementation:

```
SELECT employee_id, employee_name, salary, salary - 100
AS "salary - 100" FROM subtraction;
```

Output:

employee_id	employee_name	salary	salary-100

12	Finch	15000	14900
22	Peter	25000	24900
32	Warner	5600	5500
42	Watson	90000	89900

Here we have done subtraction of 100 to each Employee's salary i.e, subtraction operation on single column.

Let's perform **subtraction of 2 columns**:

```
SELECT employee_id, employee_name, salary, salary - employee_id
AS "salary - employee_id" FROM subtraction;
```

Output:

employee_id	employee_name	salary	salary - employee_id
12	Finch	15000	14988
22	Peter	25000	24978
32	Warner	5600	5568
42	Watson	90000	89958

Here we have done subtraction of 2 columns with each other i.e, each employee's employee_id is subtracted from its salary.

Division (/) : For **Division** refer this link- [Division in SQL](#)

Multiplication (*) :

It is use to perform **multiplication** of data items.

Implementation:

```
SELECT employee_id, employee_name, salary, salary * 100
AS "salary * 100" FROM addition;
```

Output:

employee_id	employee_name	salary	salary * 100
1	Finch	25000	2500000
2	Peter	55000	5500000
3	Warner	52000	5200000
4	Watson	12312	1231200

Here we have done multiplication of 100 to each Employee's salary i.e, multiplication operation on single column.

Let's perform **multiplication of 2 columns**:

```
SELECT employee_id, employee_name, salary, salary * employee_id
      AS "salary * employee_id" FROM addition;
```

Output:

employee_id	employee_name	salary	salary * employee_id
1	Finch	25000	25000
2	Peter	55000	110000
3	Warner	52000	156000
4	Watson	12312	49248

Here we have done multiplication of 2 columns with each other i.e, each employee's employee_id is multiplied with its salary.

Modulus (%) :

It is use to get **remainder** when one data is divided by another.

Implementation:

```
SELECT employee_id, employee_name, salary, salary % 25000
      AS "salary % 25000" FROM addition;
```

Output:

employee_id	employee_name	salary	salary %
1	Finch	25000	0
2	Peter	55000	5000
3	Warner	52000	2000
4	Watson	12312	12312

Here we have done modulus of 100 to each Employee's salary i.e, modulus operation on single column.

Let's perform **modulus operation between 2 columns**:

```
SELECT employee_id, employee_name, salary, salary % employee_id
      AS "salary % employee_id" FROM addition;
```

Output:

employee_id	employee_name	salary	salary % employee_id
1	Finch	25000	0
2	Peter	55000	0
3	Warner	52000	1
4	Watson	12312	0

Here we have done modulus of 2 columns with each other i.e, each employee's salary is divided with its id and corresponding remainder is shown.

Basically, **modulus** is use to check whether a number is **Even** or **Odd**. Suppose a given number if divided by 2 and gives 1 as remainder, then it is an *odd number* or if on dividing by 2 and gives 0 as remainder, then it is an *even number*.

Concept of NULL :

If we perform any arithmetic operation on **NULL**, then answer is *always null*.

Implementation:

```
SELECT employee_id, employee_name, salary, type, type + 100
AS "type+100" FROM addition;
```

Output:

employee_id	employee_name	salary	type	type + 100
1	Finch	25000	NULL	NULL
2	Peter	55000	NULL	NULL
3	Warner	52000	NULL	NULL
4	Watson	12312	NULL	NULL

Here output always came null, since performing any operation on null will always result in a *null value*.

Note: Make sure that NULL is **unavailable, unassigned, unknown**. Null is **not** same as *blank space or zero*.

To get in depth understanding of NULL, refer [THIS link](#).

References: [Oracle Docs](#)

Last Updated : 21 Mar, 2018

30

Similar Reads

1. Difference between Structured Query Language (SQL) and Transact-SQL (T-SQL)

2. Configure SQL Jobs in SQL Server using T-SQL

3. SQL vs NO SQL vs NEW SQL

4. Spatial Operators, Dynamic Spatial Operators and Spatial Queries in DBMS

5. SQL AND and OR operators

6. SQL | Wildcard operators

[Trending Now](#) [DSA](#) [Data Structures](#) [Algorithms](#) [Interview Preparation](#) [Data Science](#) [Topic-wise Practice](#) [Jobs](#)

Comparison Operators in SQL



abhisri459

[Read](#) [Discuss](#) [Courses](#) [Practice](#)

In SQL, there are six comparison operators available which help us run queries to perform various operations. We will use the [WHERE](#) command along with the [conditional operator](#) to achieve this in SQL. For this article, we will be using the Microsoft SQL Server as our database.

Syntax:

```
SELECT * FROM TABLE_NAME WHERE  
ATTRIBUTE CONDITION_OPERATOR GIVEN_VALUE;
```

Step 1: Create a Database. For this use the below command to create a database named GeeksForGeeks.

Query:

AD

```
CREATE DATABASE GeeksForGeeks
```

Output:

The screenshot shows a SQL query window titled 'SQLQuery1.sql - D...P-S3KL81P\HP (62)*'. The command entered is 'CREATE DATABASE GeeksForGeeks'. Below the command, the status bar indicates 'Commands completed successfully.' and 'Completion time: 2021-10-18T19:45:02.2453362+05:30'.

Step 2: Use the GeeksForGeeks database. For this use the below command.

Query:

```
USE GeeksForGeeks
```

Output:

The screenshot shows a SQL query window titled 'SQLQuery1.sql - D...P-S3KL81P\HP (62)*'. The command entered is 'USE GeeksForGeeks'. Below the command, the status bar indicates 'Commands completed successfully.' and 'Completion time: 2021-10-18T19:45:33.4253364+05:30'.

Step 3: Create a table MATHS inside the database GeeksForGeeks. This table has 3 columns namely ROLL_NUMBER, S_NAME and MARKS containing roll number, student name, and marks obtained in math's subject by various students.

Query:

```
CREATE TABLE MATHS(  
ROLL_NUMBER INT,  
S_NAME VARCHAR(10),  
MARKS INT);
```

Output:

```
CREATE TABLE MATHS(
    ROLL_NUMBER INT,
    S_NAME VARCHAR(10),
    MARKS INT);
```

100 %

Messages

Commands completed successfully.

Completion time: 2021-10-27T13:09:26.0163002+05:30

Step 4: Display the structure of the MATHS table.

Query:

```
EXEC SP_COLUMNS 'MATHS';
```

Output:

	TABLE_QUALIFIER	TABLE_OWNER	TABLE_NAME	COLUMN_NAME	DATA_TYPE	TYPE_NAME	PRECISION	LENGTH	SCALE	RADIX	NULLABLE	REMARKS	COLUMN_DEF
1	master	dbo	MATHS	ROLL_NUMBER	4	int	10	4	0	10	1	NULL	NULL
2	master	dbo	MATHS	S_NAME	12	varchar	10	10	NULL	NULL	1	NULL	NULL
3	master	dbo	MATHS	MARKS	4	int	10	4	0	10	1	NULL	NULL

Step 5: Insert 10 rows into the MATHS table.

Query:

```
INSERT INTO MATHS VALUES(1, 'ABHI', 70);
INSERT INTO MATHS VALUES(2, 'RAVI', 80);
INSERT INTO MATHS VALUES(3, 'ARJUN', 90);
INSERT INTO MATHS VALUES(4, 'SAM', 100);
INSERT INTO MATHS VALUES(5, 'MOHAN', 50);
INSERT INTO MATHS VALUES(6, 'ROHAN', 10);
INSERT INTO MATHS VALUES(7, 'ROCKY', 20);
INSERT INTO MATHS VALUES(8, 'AYUSH', 40);
```

```
INSERT INTO MATHS VALUES(9, 'NEHA', 30);
INSERT INTO MATHS VALUES(10, 'KRITI', 60);
```

Output:

```
SQLQuery1.sql - D...P-S3KL81P\HP (60)*  ↗ X
[...] INSERT INTO MATHS VALUES(1, 'ABHI', 70);
[...] INSERT INTO MATHS VALUES(2, 'RAVI', 80);
[...] INSERT INTO MATHS VALUES(3, 'ARJUN', 90);
[...] INSERT INTO MATHS VALUES(4, 'SAM', 100);
[...] INSERT INTO MATHS VALUES(5, 'MOHAN', 50);
[...] INSERT INTO MATHS VALUES(6, 'ROHAN', 10);
[...] INSERT INTO MATHS VALUES(7, 'ROCKY', 20);
[...] INSERT INTO MATHS VALUES(8, 'AYUSH', 40);
[...] INSERT INTO MATHS VALUES(9, 'NEHA', 30);
[...] INSERT INTO MATHS VALUES(10, 'KRITI', 60);

100 %  ↴
Messages

(1 row affected)
(1 row affected)
(1 row affected)
|
(1 row affected)

Completion time: 2021-10-27T13:33:16.0801864+05:30
```

Step 6: Display all the rows of the MATHS table.

Query:

```
SELECT * FROM MATHS;
```

Output:

```
SQLQuery1.sql - D...P-S3KL81P\HP (60)*  ↗ X
[...] SELECT * FROM MATHS;

100 %  ↴
Results  ↪ Messages

ROLL_NUMBER S_NAME MARKS
1 1 ABHI 70
2 2 RAVI 80
3 3 ARJUN 90
4 4 SAM 100
5 5 MOHAN 50
6 6 ROHAN 10
7 7 ROCKY 20
8 8 AYUSH 40
9 9 NEHA 30
10 10 KRITI 60
```

Demonstration of various Comparison Operators in SQL:

- **Equal to (=) Operator:** It returns the rows/tuples which have the value of the attribute equal to the given value.

Query:

```
SELECT * FROM MATHS WHERE MARKS=50;
```

Output:

The screenshot shows a SQL query window titled "SQLQuery1.sql - D...P-S3KL81P\HP (60)*". The query is: "SELECT * FROM MATHS WHERE MARKS=50;". Below the query results, there is a table with three columns: ROLL_NUMBER, S_NAME, and MARKS. A single row is displayed: ROLL_NUMBER 1, S_NAME MOHAN, and MARKS 50.

ROLL_NUMBER	S_NAME	MARKS
1	MOHAN	50

- **Greater than (>) Operator:** It returns the rows/tuples which have the value of the attribute greater than the given value.

Query:

```
SELECT * FROM MATHS WHERE MARKS>60;
```

Output:

The screenshot shows a SQL query window titled "SQLQuery1.sql - D...P-S3KL81P\HP (60)*". The query is: "SELECT * FROM MATHS WHERE MARKS>60;". Below the query results, there is a table with three columns: ROLL_NUMBER, S_NAME, and MARKS. Four rows are displayed: ROLL_NUMBER 1, S_NAME ABHI, MARKS 70; ROLL_NUMBER 2, S_NAME RAVI, MARKS 80; ROLL_NUMBER 3, S_NAME ARJUN, MARKS 90; and ROLL_NUMBER 4, S_NAME SAM, MARKS 100.

ROLL_NUMBER	S_NAME	MARKS
1	ABHI	70
2	RAVI	80
3	ARJUN	90
4	SAM	100

- **Less than (<) Operator:** It returns the rows/tuples which have the value of the attribute lesser than the given value.

Query:

```
SELECT * FROM MATHS WHERE MARKS<40;
```

Output:

```
SQLQuery1.sql - D...P-S3KL81P\HP (60)*
SELECT * FROM MATHS WHERE MARKS<40;

100 %
Results Messages
ROLL_NUMBER S_NAME MARKS
1 6 ROHAN 10
2 7 ROCKY 20
3 9 NEHA 30
```

- **Greater than or equal to (\geq) Operator:** It returns the rows/tuples which have the value of the attribute greater or equal to the given value.

Query:

```
SELECT * FROM MATHS WHERE MARKS>=80;
```

Output:

```
SQLQuery1.sql - D...P-S3KL81P\HP (60)*
SELECT * FROM MATHS WHERE MARKS>=80;

100 %
Results Messages
ROLL_NUMBER S_NAME MARKS
1 2 RAVI 80
2 3 ARJUN 90
3 4 SAM 100
```

- **Less than or equal to (\leq) Operator:** It returns the rows/tuples which have the value of the attribute lesser or equal to the given value.

Query:

```
SELECT * FROM MATHS WHERE MARKS<=30;
```

Output:

```
SQLQuery1.sql - D...P-S3KL81P\HP (60)*
SELECT * FROM MATHS WHERE MARKS<=30;

100 %
Results Messages
ROLL_NUMBER S_NAME MARKS
1 6 ROHAN 10
2 7 ROCKY 20
3 9 NEHA 30
```

- Not equal to (<>) Operator:** It returns the rows/tuples which have the value of the attribute not equal to the given value.

Query:

```
SELECT * FROM MATHS WHERE MARKS<>70;
```

Output:

```
SQLQuery1.sql - D...P-S3KL81P\HP (60)*
SELECT * FROM MATHS WHERE MARKS<>70;

100 %
Results Messages
ROLL_NUMBER S_NAME MARKS
1 2 RAVI 80
2 3 ARIJUN 90
3 4 SAM 100
4 5 MOHAN 50
5 6 ROHAN 10
6 7 ROCKY 20
7 8 AYUSH 40
8 9 NEHA 30
9 10 KRITI 60
```

Last Updated : 14 Nov, 2021

2

Similar Reads

1. Difference between Structured Query Language (SQL) and Transact-SQL (T-SQL)

2. Configure SQL Jobs in SQL Server using T-SQL

3. SQL AND and OR operators

4. SQL | Wildcard operators



SQL AND and OR operators

Read Discuss Courses Practice

In SQL, the AND & OR operators are used for filtering the data and getting precise results based on conditions. The SQL AND & OR operators are also used to combine multiple conditions. These two operators can be combined to test for multiple conditions in a SELECT, INSERT, UPDATE, or DELETE statement.

When combining these conditions, it is important to use parentheses so that the database knows what order to evaluate each condition.

- The AND and OR operators are used with the WHERE clause.
- These two operators are called conjunctive operators.

AND Operator:

This operator displays only those records where both the conditions condition1 and condition2 evaluates to True.

Syntax:

AD

```
SELECT * FROM table_name WHERE condition1 AND condition2 and ...conditionN;
```

table_name: name of the table

condition1,2,...N : first condition, second condition and so on

OR Operator:

This operator displays the records where either one of the conditions condition1 and condition2 evaluates to True. That is, either condition1 is True or condition2 is True.

Syntax:

```
SELECT * FROM table_name WHERE condition1 OR condition2 OR... conditionN;
```

table_name: name of the table

condition1,2,...N : first condition, second condition and so on

Now, we consider a table database to demonstrate AND & OR operators with multiple cases:

Student				
ROLL_NO	NAME	ADDRESS	PHONE	Age
1	Ram	Delhi	XXXXXXXXXX	18
2	RAMESH	GURGAON	XXXXXXXXXX	18
3	SUJIT	ROHTAK	XXXXXXXXXX	20
4	SURESH	Delhi	XXXXXXXXXX	18
3	SUJIT	ROHTAK	XXXXXXXXXX	20
2	RAMESH	GURGAON	XXXXXXXXXX	18

If suppose we want to fetch all the records from the Student table where Age is 18 and ADDRESS is Delhi. then the query will be:

Query:

```
SELECT * FROM Student WHERE Age = 18 AND ADDRESS = 'Delhi';
```

Output:

ROLL_NO	NAME	ADDRESS	PHONE	Age
1	Ram	Delhi	XXXXXXXXXX	18
4	SURESH	Delhi	XXXXXXXXXX	18

Take another example, to fetch all the records from the Student table where NAME is Ram and Age is 18.

Query:

```
SELECT * FROM Student WHERE Age = 18 AND NAME = 'Ram';
```

Output:

ROLL_NO	NAME	ADDRESS	PHONE	Age
1	Ram	Delhi	XXXXXXXXXX	18

To fetch all the records from the Student table where NAME is Ram or NAME is SUJIT.

Query:

```
SELECT * FROM Student WHERE NAME = 'Ram' OR NAME = 'SUJIT';
```

Output:

ROLL_NO	NAME	ADDRESS	PHONE	Age
1	Ram	Delhi	XXXXXXXXXX	18
3	SUJIT	ROHTAK	XXXXXXXXXX	20
3	SUJIT	ROHTAK	XXXXXXXXXX	20

To fetch all the records from the Student table where NAME is Ram or Age is 20.

Query:

```
SELECT * FROM Student WHERE NAME = 'Ram' OR Age = 20;
```

Output:

ROLL_NO	NAME	ADDRESS	PHONE	Age
1	Ram	Delhi	XXXXXXXXXX	18

ROLL_NO	NAME	ADDRESS	PHONE	Age
3	SUJIT	ROHTAK	XXXXXXXXXX	20
3	SUJIT	ROHTAK	XXXXXXXXXX	20

Combining AND and OR:

We can combine AND and OR operators in the below manner to write complex queries.

Syntax:

```
SELECT * FROM table_name WHERE condition1 AND (condition2 OR condition3);
```

Take an example to fetch all the records from the Student table where Age is 18 NAME is Ram or RAMESH.

Query:

```
SELECT * FROM Student WHERE Age = 18 AND (NAME = 'Ram' OR NAME = 'RAMESH');
```

Output:

ROLL_NO	NAME	ADDRESS	PHONE	Age
1	Ram	Delhi	XXXXXXXXXX	18
2	RAMESH	GURGAON	XXXXXXXXXX	18

Similar Reads

1. Difference between Structured Query Language (SQL) and Transact-SQL (T-SQL)

2. Configure SQL Jobs in SQL Server using T-SQL

3. SQL vs NO SQL vs NEW SQL



SQL | ALL and ANY

[Read](#) [Discuss](#)

ALL & ANY are logical operators in SQL. They return boolean value as a result.

ALL

ALL operator is used to select all tuples of SELECT STATEMENT. It is also used to compare a value to every value in another value set or result from a subquery.

- The ALL operator returns TRUE if all of the subqueries values meet the condition. The ALL must be preceded by comparison operators and evaluates true if all of the subqueries values meet the condition.
- ALL is used with SELECT, WHERE, HAVING statement.

ALL with SELECT Statement:

AD

Syntax:

```
SELECT ALL field_name  
FROM table_name  
WHERE condition(s);
```

ALL with WHERE or HAVING Statement:

Syntax:

```
SELECT column_name(s)  
FROM table_name  
WHERE column_name comparison_operator ALL  
(SELECT column_name
```

```
FROM table_name
WHERE condition(s);
```

Example: Consider the following Products Table and OrderDetails Table,

ProductID	ProductName	SupplierID	CategoryID	Price
1	Chais	1	1	18
2	Chang	1	1	19
3	Aniseed Syrup	1	2	10
4	Chef Anton's Cajun Seasoning	2	2	22
5	Chef Anton's Gumbo Mix	2	2	21
6	Boysenberry Spread	3	2	25
7	Organic Dried Pears	3	7	30
8	Northwoods Cranberry Sauce	3	2	40
9	Mishi Kobe Niku	4	6	97

OrderDetails Table

OrderDetailsID	OrderID	ProductID	Quantity
1	10248	1	12
2	10248	2	10
3	10248	3	15
4	10249	1	8
5	10249	4	4
6	10249	5	6
7	10250	3	5
8	10250	4	18
9	10251	5	2
10	10251	6	8
11	10252	7	9
12	10252	8	9
13	10250	9	20
14	10249	9	4

Queries

- Find the name of all the products.

```
SELECT ALL ProductName
FROM Products
WHERE TRUE;
```

ProductName
Chais
Chang
Aniseed Syrup
Chef Anton's
Cajun Seasoning
Chef Anton's
Gumbo Mix
Boysenberry
Spread
Organic Dried
Pears
Northwoods
Cranberry Sauce
Mishi Kobe Niku

- Output:
- Find the name of the product if all the records in the OrderDetails has Quantity either equal to 6 or 2.

```
SELECT ProductName
FROM Products
WHERE ProductID = ALL (SELECT ProductId
                        FROM OrderDetails
                        WHERE Quantity = 6 OR Quantity = 2);
```

ProductName
Chef Anton's
Gumbo Mix

- Output:
- Find the OrderID whose maximum Quantity among all product of that OrderID is greater than average quantity of all OrderID.

```
SELECT OrderID
FROM OrderDetails
GROUP BY OrderID
HAVING max(Quantity) > ALL (SELECT avg(Quantity)
                             FROM OrderDetails
                             GROUP BY OrderID);
```

OrderID
10248
10250

- Output:

ANY

ANY compares a value to each value in a list or results from a query and evaluates to true if the result of an inner query contains at least one row.

- ANY return true if any of the subqueries values meet the condition.
- ANY must be preceded by comparison operators. **Syntax:**

```

SELECT column_name(s)
FROM table_name
WHERE column_name comparison_operator ANY
(SELECT column_name
FROM table_name
WHERE condition(s));

```

Queries

- Find the Distinct CategoryID of the products which have any record in OrderDetails Table.

```

SELECT DISTINCT CategoryID
FROM Products
WHERE ProductID = ANY (SELECT ProductID
                        FROM OrderDetails);

```

CategoryID
1
2
7
6

- Output:
- Finds any records in the OrderDetails table that Quantity = 9.

```

SELECT ProductName
FROM Products
WHERE ProductID = ANY (SELECT ProductID
                        FROM OrderDetails
                        WHERE Quantity = 9);

```

ProductName
Organic Dried Pears
Northwoods Cranberry Sauce

This article is contributed by [Anuj Chauhan](#). If you like GeeksforGeeks and would like to contribute, you can also write an article using [write.geeksforgeeks.org](#) or mail your article to review-team@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Last Updated : 28 Nov, 2022

44

Similar Reads



SQL | Wildcard operators

[Read](#) [Discuss](#) [Courses](#) [Practice](#)

Prerequisite: [SQL | WHERE Clause](#)

In the above-mentioned article WHERE Clause is discussed in which the [LIKE operator](#) is also explained, where you must have encountered the word wildcards now let's get deeper into Wildcards. Wildcard operators are used with the LIKE operator, there are four basic operators:

Operator Table

Operator	Description
%	It is used in substitute of zero or more characters.
_	It is used as a substitute for one character.
-	It is used to substitute a range of characters.
[range_of_characters]	It is used to fetch a matching set or range of characters specified inside the brackets.

Syntax:

SELECT column1,column2 FROM table_name

AD

WHERE column LIKE wildcard_operator;

column1,column2: fields in the table

table_name: name of table

column: name of field used for filtering data

CREATE Table:

```
CREATE TABLE Customer(
    CustomerID INT PRIMARY KEY,
    CustomerName VARCHAR(50),
    LastName VARCHAR(50),
    Country VARCHAR(50),
    Age int(2),
    Phone int(10)
);
-- Insert some sample data into the Customers table
INSERT INTO Customer (CustomerID, CustomerName, LastName, Country, Age, Phone)
VALUES (1, 'Shubham', 'Thakur', 'India', '23', 'xxxxxxxxxx'),
       (2, 'Aman ', 'Chopra', 'Australia', '21', 'xxxxxxxxxx'),
       (3, 'Naveen', 'Tulasi', 'Sri lanka', '24', 'xxxxxxxxxx'),
       (4, 'Aditya', 'Arpan', 'Austria', '21', 'xxxxxxxxxx'),
       (5, 'Nishant. Salchichas S.A.', 'Jain', 'Spain', '22', 'xxxxxxxxxx');
Select * from Customer;
```

Output:

Customer

CustomerID	CustomerName	LastName	Country	Age	Phone
1	Shubham	Thakur	India	23	xxxxxxxxxx
2	Aman	Chopra	Australia	21	xxxxxxxxxx
3	Naveen	Tulasi	Sri lanka	24	xxxxxxxxxx
4	Aditya	Arpan	Austria	21	xxxxxxxxxx
5	Nishant. Salchichas S.A.	Jain	Spain	22	xxxxxxxxxx

1. To fetch records from the Customer table with NAME starting with the letter 'A'.

Query:

```
SELECT * FROM Customer WHERE CustomerName LIKE 'A%';
```

Output:

CustomerID	CustomerName	LastName	Country	Age	Phone
2	Aman	Chopra	Australia	21	xxxxxxxxxx
4	Aditya	Arpan	Austria	21	xxxxxxxxxx

2. To fetch records from the Customer table with NAME ending with the letter 'A'.

Query:

```
SELECT * FROM Customer WHERE CustomerName LIKE '%A';
```

Output:

CustomerID	CustomerName	LastName	Country	Age	Phone
4	Aditya	Arpan	Austria	21	xxxxxxxxxx

3. To fetch records from the Customer table with NAME with the letter 'A' at any position.

Query:

```
SELECT * FROM Customer WHERE CustomerName LIKE '%A%';
```

Output:

CustomerID	CustomerName	LastName	Country	Age	Phone
1	Shubham	Thakur	India	23	xxxxxxxxxx
2	Aman	Chopra	Australia	21	xxxxxxxxxx
3	Naveen	Tulasi	Sri Lanka	24	xxxxxxxxxx
4	Aditya	Arpan	Austria	21	xxxxxxxxxx
5	Nishant. Salchichas S.A.	Jain	Spain	22	xxxxxxxxxx

4. To fetch records from the Student table with NAME ending any letter but starting from 'Nav'.

Query:

```
SELECT * FROM Customer WHERE CustomerName LIKE 'Nav____';
```

Output:

CustomerID	CustomerName	LastName	Country	Age	Phone
3	Naveen	Tulasi	Sri Lanka	24	234565663

5. To fetch records from the Student table with LastName containing letters 'a', 'b', or 'c'.

Query:

```
SELECT * FROM Student WHERE LastName REGEXP '[A-C]';
```

Output:

CustomerID	CustomerName	Lastname	Country	Age	Phone
1	Shubham Thakur	India	23	862357843	
2	Aman Chopra	Australia	21	436577545	
3	Naveen Tulasi	Sri lanka	24	234565663	
4	Aditya Arpan	Austria	21	234565676	
5	Nishant. Salchichas S.A.		Jain	Spain	22 234567346

6. To fetch records from the Student table with LastName not containing letters 'y', or 'z'.

Query:

```
SELECT * FROM Students WHERE LastName NOT LIKE '%[y-z]%' ;
```

Output:

CustomerID	CustomerName	Lastname	Country	Age	Phone
1	Shubham Thakur	India	23	862357843	
2	Aman Chopra	Australia	21	436577545	
3	Naveen Tulasi	Sri lanka	24	234565663	
4	Aditya Arpan	Austria	21	234565676	
5	Nishant. Salchichas S.A.		Jain	Spain	22 234567346

7. To fetch records from the Student table with the PHONE field having an '8' in the 1st position and a '3' in the 3rd position.

Query:

```
SELECT * FROM Student WHERE PHONE LIKE '8__3%' ;
```

Output:

CustomerID	CustomerName	Lastname	Country	Age	Phone
1	Shubham Thakur	India	23	862357843	

8. To fetch records from the Student table with Country containing a total of 7 characters.

Query:

```
SELECT * FROM Students WHERE Country LIKE '_____';
```

Output:

CustomerID	CustomerName	LastName	Country	Age	Phone
4	Aditya	Arpan	Austria	21	234565676

9. To fetch records from the Student table with the LastName containing 'ra' at any position, and the result set should not contain duplicate data.

Query:

```
SELECT DISTINCT * FROM Students WHERE Country LIKE '%ra%';
```

Output:

CustomerID	CustomerName	LastName	Country	Age	Phone
2	Aman	Chopra	Australia	21	436577545

Frequently Asked Question**Question: What is a wildcard operator in SQL?**

Ans: The LIKE operator makes use of wildcard characters. The LIKE operator is used in a WHERE clause to look for a specific pattern in a column.

This article is contributed by **Pratik Agarwal**. If you like GeeksforGeeks and would like to contribute, you can also write an article using [write.geeksforgeeks.org](https://www.geeksforgeeks.org/write-for-us/) or mail your article to review-team@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks. Please comment if you find anything incorrect or if you want to share more information about the topic discussed above.

Last Updated : 06 Apr, 2023

42

Similar Reads

1. Difference between Structured Query Language (SQL) and Transact-SQL (T-SQL)

2. Configure SQL Jobs in SQL Server using T-SQL

3. SQL vs NO SQL vs NEW SQL



SQL | Concatenation Operator

[Read](#) [Discuss](#) [Courses](#) [Practice](#)

Prerequisite: [Basic Select statement](#), [Insert into clause](#), [SQL Create Clause](#), [SQL Aliases](#)

|| or concatenation operator is used to **link columns or character strings**. We can also use a **literal**. A literal is a **character, number or date** that is included in the SELECT statement.

Let's demonstrate it through an example:

Syntax:

AD

```
SELECT id, first_name, last_name, first_name || last_name,
       salary, first_name || salary FROM myTable
```

Output (Third and Fifth Columns show values concatenated by operator ||)

	id	first_name	last_name	first_name last_name	salary	first_name salary
1	Rajat	Rawat		RajatRawat	10000	Rajat10000
2	Geeks	ForGeeks		GeeksForGeeks	20000	Geeks20000
3	Shane	Watson		ShaneWatson	50000	Shane50000
4	Kedar	Jadhav		KedarJadhav	90000	Kedar90000

Note: Here above we have used **||** which is known as **Concatenation operator** which is used to link 2 or as **many** columns as you want in your select query and it is **independent of the datatype** of column. Here above we have linked 2 columns i.e, **first_name+last_name** as well as **first_name+salary**.

We can also use **literals** in the **concatenation** operator. Let's see:

Example 1: Using character literal

Syntax:

```
SELECT id, first_name, last_name, salary,
       first_name||' has salary'||salary as "new" FROM myTable
```

Output : (Concatenating three values and giving a name 'new')

	id	first_name	last_name	salary	new
1	Rajat	Rawat	10000	Rajat	has salary 10000
2	Geeks	ForGeeks	20000	Geeks	has salary 20000
3	Shane	Watson	50000	Shane	has salary 50000
4	Kedar	Jadhav	90000	Kedar	has salary 90000

Note: Here above we have used **has salary** as a character literal in our select statement. Similarly we can use number literal or date literal according to our requirement.

Example 2: Using character as well as number literal

Syntax:

```
SELECT id, first_name, last_name, salary, first_name||100||'
       has id'||id AS "new" FROM myTable
```

Output (Making readable output by concatenating a string with values)

	id	first_name	last_name	salary	new
1	Rajat	Rawat	10000	Rajat	100 has id 1
2	Geeks	ForGeeks	20000	Geeks	100 has id 2
3	Shane	Watson	50000	Shane	100 has id 3
4	Kedar	Jadhav	90000	Kedar	100 has id 4

Here above we have used **has salary** as a character literal as well as **100** as number literal in our select statement.

References:

- 1) About Concatenation operator: [Oracle Docs](#)
- 2) Performing SQL Queries Online: [Oracle Live SQL](#)

Note: For performing SQL Queries online you must have account on Oracle, if you don't have then you can make by opening above link.

Last Updated : 21 Mar, 2018

17

Similar Reads

1. Difference between Structured Query Language (SQL) and Transact-SQL (T-SQL)



SQL | MINUS Operator

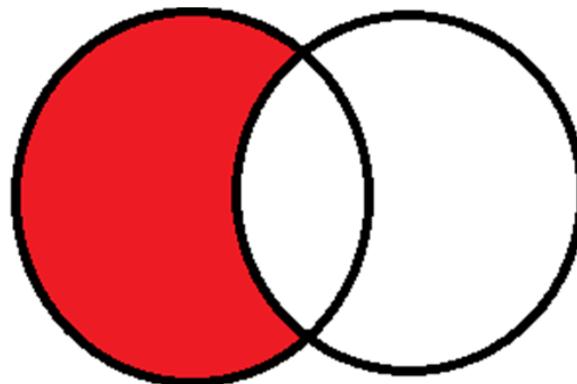
[Read](#)
[Discuss](#)
[Courses](#)
[Practice](#)
[Video](#)

The Minus Operator in SQL is used with two SELECT statements. The MINUS operator is used to subtract the result set obtained by first SELECT query from the result set obtained by second SELECT query. In simple words, we can say that MINUS operator will return only those rows which are unique in only first SELECT query and not those rows which are common to both first and second SELECT queries.

Pictorial Representation:

Table 1

Table 2



As you can see in the above diagram, the MINUS operator will return only those rows which are present in the result set from Table1 and not present in the result set of Table2.

Basic Syntax:

```

SELECT column1 , column2 , ... columnN
FROM table_name
WHERE condition
MINUS
SELECT column1 , column2 , ... columnN
FROM table_name
WHERE condition;
    
```

columnN: column1, column2.. are the name of columns of the table.

Important Points:

- The WHERE clause is optional in the above query.
- The number of columns in both SELECT statements must be same.
- The data type of corresponding columns of both SELECT statement must be same.

Sample Tables:

AD

Table1

Table 1

Name	Address	Age	Grade
Harsh	Delhi	20	A
Gaurav	Jaipur	21	B
Pratik	Mumbai	21	A
Dhanraj	Kolkata	22	B

Table 2

Name	Age	Phone	Grade
Akash	20	XXXXXXXXXX	A
Dhiraj	21	XXXXXXXXXX	B
Vaibhav	21	XXXXXXXXXX	A
Dhanraj	22	XXXXXXXXXX	B

Queries:

```
SELECT NAME, AGE , GRADE
FROM Table1
MINUS
SELECT NAME, AGE, GRADE
FROM Table2
```

Output:

The above query will return only those rows which are unique in 'Table1'. We can clearly see that values in the fields NAME, AGE and GRADE for the last row in both tables are same. Therefore, the output will be the first three rows from Table1. The obtained output is shown below:

Name	Age	Grade
Harsh	20	A
Gaurav	21	B
Pratik	21	A

Note: The MINUS operator is not supported with all databases. It is supported by Oracle database but not SQL server or PostgreSQL.

This article is contributed by [Harsh Agarwal](#). If you like GeeksforGeeks and would like to contribute, you can also write an article using write.geeksforgeeks.org or mail your article to review-team@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Last Updated : 15 Jul, 2022

20

Similar Reads

1. Difference between Structured Query Language (SQL) and Transact-SQL (T-SQL)

2. Configure SQL Jobs in SQL Server using T-SQL

3. SQL | BETWEEN & IN Operator

4. SQL | NOT Operator

5. SQL | Concatenation Operator

6. SQL | Alternative Quote Operator

7. Difference between = and IN operator in SQL

8. SQL | UNION Operator

9. Inequality Operator in SQL



SQL | DIVISION

[Read](#)
[Discuss](#)
[Courses](#)
[Practice](#)

Division is typically required when you want to find out entities that are interacting with **all entities** of a set of different type entities.

The division operator is used when we have to evaluate queries which contain the keyword ‘all’.

Some instances where division operator is used are:

- Which person has account in all the banks of a particular city?
- Which students have taken all the courses required to graduate?

In all these queries, the description after the keyword ‘all’ defines a set which contains some elements and the final result contains those units who satisfy these requirements.

Important: Division is not supported by SQL implementations. However, it can be represented using other operations.(like cross join, Except, In)

AD

SQL Implementation of Division

Given two relations(tables): R(x,y) , S(y).

R and S : tables

x and y : column of R

y : column of S

R(x,y) div S(y) means gives all distinct values of x from R that are associated with all values of y in S.

Computation of Division : R(x,y) div S(y)

Steps:

- Find out all possible combinations of S(y) with R(x) by computing R(x) x(cross join) S(y), say r1

- Subtract actual R(x,y) from r1, say r2
- x in r2 are those that are not associated with every value in S(y); therefore R(x)-r2(x) gives us x that are associated with all values in S

Queries

1. Implementation 1:

```
SELECT * FROM R
WHERE x not in ( SELECT x FROM (
(SELECT x , y FROM (select y from S ) as p cross join
(select distinct x from R) as sp)
EXCEPT
(SELECT x , y FROM R) ) AS r );
```

2. Implementation 2 : Using correlated subquery

```
SELECT * FROM R as sx
WHERE NOT EXISTS (
(SELECT p.y FROM S as p )
EXCEPT
(SELECT sp.y FROM R as sp WHERE sp.x = sx.x ) );
```

Relational algebra

Using steps which is mention above:

All possible combinations

$r1 \leftarrow \pi x(R) \times S$

x values with “incomplete combinations”,

$r2x \leftarrow \pi x(r1-R)$

and

$result \leftarrow \pi x(R)-r2x$

$R \text{ div } S = \pi x(R) - \pi x((\pi x(R) \times S) - R)$

Examples

Supply Schema

sid (integer)	pid (integer)
101	1
102	1
101	3
103	2
102	2
102	3
102	4
102	5

pid (integer)
1
2
3
4
5

Here **sid** means **supplierID** and **pid** means **partsID**.

Tables: suppliers(sid,pid) , parts(pid)

1. Find suppliers that supply all parts.

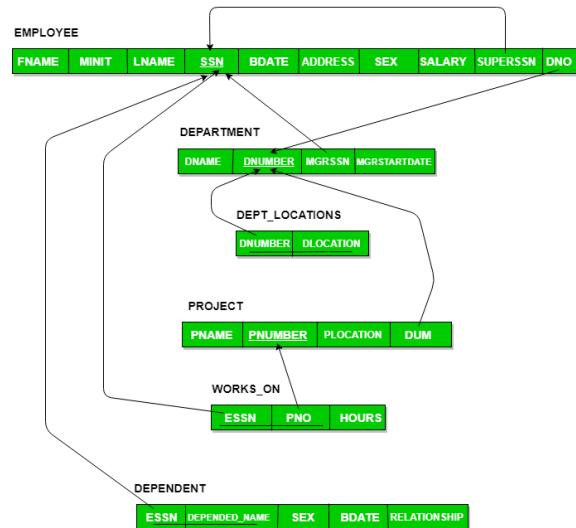
Ans 1 : Using implementation 1

```
SELECT * FROM suppliers
WHERE sid not in ( SELECT sid FROM ( (SELECT sid, pid FROM (select pid from
parts) as p
cross join
(select distinct sid from supplies) as sp)
EXCEPT
(SELECT sid, pid FROM supplies)) AS r );
```

Ans 2: Using implementation 2

```
SELECT * FROM suppliers as s
WHERE NOT EXISTS (( SELECT p.pid FROM parts as p )
EXCEPT
(SELECT sp.pid FROM supplies sp WHERE sp.sid = s.sid ) );
```

Company schema



2. List employees who work on all projects controlled by dno=4.

Ans 1. Using implementation 1

```
SELECT * FROM employee AS e
WHERE ssn NOT IN (
  SELECT essn FROM (
    (SELECT essn, pno FROM (select pno from project where dno=4)
    as p cross join (select distinct essn from works_on) as w)
  EXCEPT (SELECT essn, pno FROM works_on)) AS r );
```

Ans 2. Using implementation 2

```
SELECT * FROM employee AS e
WHERE NOT EXISTS (
  (SELECT pno FROM project WHERE dno = 4)
  EXCEPT
  (SELECT pno FROM works_on WHERE essn = e.ssn) );
```

Important : For division correlated query seems simpler to write but may expensive to execute.

Some more Examples.

1. List supplier who supply all 'Red' Parts.(supply schema)
2. Retrieve the names of employees, who work on all the projects that 'John Smith' works (company schema)

This article is contributed by [Kadam Patel](#). If you like GeeksforGeeks and would like to contribute, you can also write an article using contribute.geeksforgeeks.org or mail your article to contribute@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.



SQL | NOT Operator

[Read](#)
[Discuss](#)
[Courses](#)
[Practice](#)

NOT Syntax `SELECT column1, column2, ... FROM table_name WHERE NOT condition;` [Demo](#)

Database Below is a selection from the “Customers” table in the Northwind sample database:

Customer ID	Customer Name	City	PostalCode	Country
1	John Wick	New York	1248	USA
2	Around the Horn	London	WA1 1DP	UK
3	Rohan	New Delhi	100084	India

NOT Example The following SQL statement selects all fields from “Customers” where country is not “UK” `SELECT * FROM Customers WHERE NOT Country='UK';`

Customer ID	Customer Name	City	PostalCode	Country
1	John Wick	New York	1248	USA
3	Rohan	New Delhi	100084	India

Combining AND, OR and NOT You can also combine the AND, OR, and NOT operators.

Example: 1.) `SELECT * FROM Customers WHERE NOT Country='USA' AND NOT Country='UK';`

Customer ID	Customer Name	City	PostalCode	Country

3	Rohan	New Delhi	100084	India
---	-------	-----------	--------	-------

Alternatively you can use <> (Not Operator) to get the desired result :-

AD

```
SELECT * FROM Customer WHERE Country <>'USA';
```

Output :-

	CUSTOMER_ID	CUSTOMER_NAME	CITY	POSTALCODE	COUNTRY
1	2	Around the Horn	London	WA1 1DP	UK
2	3	Rohan	New Delhi	100084	India

Last Updated : 24 Mar, 2023

36

Similar Reads

1. Difference between Structured Query Language (SQL) and Transact-SQL (T-SQL)

2. Configure SQL Jobs in SQL Server using T-SQL

3. SQL | BETWEEN & IN Operator

4. SQL | MINUS Operator

5. SQL | Concatenation Operator

6. SQL | Alternative Quote Operator

7. Difference between = and IN operator in SQL

8. SQL | UNION Operator

9. Inequality Operator in SQL



SQL | BETWEEN & IN Operator

[Read](#)[Discuss](#)[Courses](#)[Practice](#)[Video](#)

Pre-requisites: [SQL Operators](#)

Operators are the foundation of any programming language. We can define operators as symbols that help us to perform specific mathematical and logical computations on operands. In other words, we can say that an operator operates the operands.

In this article, we will see BETWEEN & IN Operator of SQL.

Between Operator

The SQL BETWEEN condition allows you to easily test if an expression is within a range of values (inclusive). The values can be text, date, or numbers. It can be used in a [SELECT](#), [INSERT](#), [UPDATE](#), or [DELETE](#) statement. The SQL BETWEEN Condition will return the records where the expression is within the range of value1 and value2.

AD

Syntax:

SELECT column_name(s)

FROM table_name

WHERE column_name BETWEEN value1 AND value2;

Let's create a database to understand BETWEEN & IN Operator in SQL.

Query:

```

CREATE TABLE Emp(
    EmpID INT PRIMARY KEY,
    Name VARCHAR(50),
    Country VARCHAR(50),
    Age int(2),
    Salary int(10)
);
-- Insert some sample data into the Customers table
INSERT INTO Emp (EmpID, Name, Country, Age, Salary)
VALUES (1, 'Shubham', 'India', '23', '30000'),
       (2, 'Aman ', 'Australia', '21', '45000'),
       (3, 'Naveen', 'Sri lanka', '24', '40000'),
       (4, 'Aditya', 'Austria', '21', '35000'),
       (5, 'Nishant', 'Spain', '22', '25000');
Select * from Emp;

```

Output:

Emp

EmpID	Name	Country	Age	Salary
1	Shubham	India	23	30000
2	Aman	Australia	21	45000
3	Naveen	Sri lanka	24	40000
4	Aditya	Austria	21	35000
5	Nishant	Spain	22	25000

Using BETWEEN with Numeric Values

List all the Employee's Names who is having salary between 30000 and 45000.

Query:

```

SELECT Name
FROM Emp
WHERE Salary
BETWEEN 30000 AND 45000;

```

Output:

Name
Shubham
Aman
Naveen
Aditya

Using BETWEEN with Date Values

Find all the Employees an Age Between 22 to 24.

Query:

```
SELECT Name  
FROM Emp  
where Age  
BETWEEN '22' AND '24';
```

Output:

Name
Shubham
Naveen
Nishant

Using the NOT Operator with BETWEEN

Find all the Employee names whose salary is not in the range of 30000 and 45000.

Query:

```
SELECT Name  
FROM Emp  
WHERE Salary  
NOT BETWEEN 30000 AND 45000;
```

Output:

Name
Nishant

IN Operator

IN operator allows you to easily test if the expression matches any value in the list of values. It is used to remove the need for multiple OR conditions in SELECT, INSERT, UPDATE, or DELETE. You can also use NOT IN to exclude the rows in your list. We should note that any kind of duplicate entry will be retained.

Syntax:

```
SELECT column_name(s)
FROM table_name
WHERE column_name IN (list_of_values);
```

Find the Fname, and Lname of the Employees who have a Salary equal to 30000, 40000, or 25000.

Query:

```
SELECT Name
FROM Emp
WHERE Salary IN (30000, 40000, 25000);
```

Output:

Name
Shubham
Naveen
Nishant

Find the Fname and Lname of all the Employees who has a Salary not equal to 25000 or 30000.

Query:

```
SELECT Name  
FROM Emp  
WHERE Salary NOT IN (25000, 30000);
```

Output:

Name
Aman
Naveen
Aditya

This article is contributed by **Anuj Chauhan**. If you like GeeksforGeeks and would like to contribute, you can also write an article using [write.geeksforgeeks.org](https://www.geeksforgeeks.org/write-for-us/) or mail your article to review-team@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.

Please write comments if you find anything incorrect, or if you want to share more information about the topic discussed above.

Last Updated : 05 Apr, 2023

31

Similar Reads

1. Difference between Structured Query Language (SQL) and Transact-SQL (T-SQL)
2. Configure SQL Jobs in SQL Server using T-SQL
3. SQL vs NO SQL vs NEW SQL
4. Difference between = and IN operator in SQL
5. SQL | Difference between functions and stored procedures in PL/SQL
6. Difference between T-SQL and PL-SQL
7. Difference between SQL and T-SQL
8. SQL | MINUS Operator
9. SQL | NOT Operator



SQL LIKE

[Read](#)
[Discuss](#)
[Courses](#)
[Practice](#)

Sometimes we may require tuples from the database which match certain patterns. For example, we want to retrieve all columns where the tuples start with the letter 'y', or start with 'b' and end with 'l', or even more complicated and restrictive string patterns. This is where the **SQL LIKE Clause** comes to the rescue, often coupled with the WHERE Clause in [SQL](#).

In SQL, the LIKE operator is mainly used in the WHERE clause to search for a enumerate pattern in a column.

Two barriers are often used in conjunction with the LIKE :

1. %: Used to match zero or more characters. (Variable Length)
2. _: Used to match exactly one character. (Fixed Length)

SQL Like Syntax:

```
SELECT column1, column2, ...
FROM table_name
WHERE columnN LIKE pattern;
```

The following are the rules for pattern matching with the LIKE Clause :

AD

Pattern	Meaning
'a%	Match strings that start with 'a'
%a'	Match strings with end with 'a'

Pattern	Meaning
'a%t'	Match strings that contain the start with 'a' and end with 't'.
'%wow%'	Match strings that contain the substring 'wow' in them at any position.
'_wow%'	Match strings that contain the substring 'wow' in them at the second position.
'_a%'	Match strings that contain 'a' at the second position.
'a_ _%'	Match strings that start with 'a' and contain at least 2 more characters.

Example: Say we have a relation, Supplier. We want to test various patterns using the LIKE clause:

Supplier Table

SupplierID	Name	Address
S1	Paragon Suppliers	21-3, Okhla, Delhi
S2	Mango Nation	21, Faridabad, Haryana
S3	Canadian Biz	6/7, Okhla Phase II, Delhi
S4	Caravan Traders	2-A, Pitampura, Delhi
S5	Harish and Sons	Gurgaon, NCR
S6	Om Suppliers	2/1, Faridabad, Haryana

SQL LIKE – Sample Queries and Outputs

Query 1:

```
SELECT SupplierID, Name, Address
FROM Suppliers
WHERE Name LIKE 'Ca%';
```

Output:

S3	Canadian Biz	6/7, Okhla Phase II, Delhi
S4	Caravan Traders	2-A, Pitampura, Delhi

Query 2:

```
SELECT *
FROM Suppliers
WHERE Address LIKE '%Okhla%';
```

Output:

S1	Paragon Suppliers	21-3, Okhla, Delhi
S3	Canadian Biz	6/7, Okhla Phase II, Delhi

Query 3:

```
SELECT SupplierID, Name, Address
FROM Suppliers
WHERE Name LIKE '_ango%';
```

Output:

S2	Mango Nation	21, Faridabad, Haryana
----	--------------	------------------------

SQL LIKE Application

The LIKE operator is extremely resourceful in situations such as address filtering wherein we know only a segment or a portion of the entire address (such as locality or city) and would like to retrieve results based on that. The wildcards can be resourcefully exploited to yield even better and more filtered tuples based on the requirement.

Important to know about SQL LIKE

One important thing to note about the LIKE operator is that it is **case-insensitive** by default in most database systems. This means that if you search for “apple” using the LIKE operator, it will return results that include “Apple”, “APPLE”, “aPpLe”, and so on.

For making the LIKE operator case-sensitive, you can use the “**BINARY**” keyword in MySQL or the “**COLLATE**” keyword in other database systems.

For example:

XML

```
SELECT * FROM products WHERE name LIKE BINARY 'apple%'
```

This following query will only return products whose name starts with “apple” and is spelled exactly like that, without capital letters.

This article is contributed by **Anannya Uberoi**. If you like GeeksforGeeks and would like to contribute, you can also write an article using [write.geeksforgeeks.org](https://www.geeksforgeeks.org/write-for-us/) or mail your article to review-team@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks. Please write comments if you find anything incorrect, or if you want to share more information about the topic discussed above.

Last Updated : 04 May, 2023

29

Similar Reads

1. [Difference between Structured Query Language \(SQL\) and Transact-SQL \(T-SQL\)](#)
2. [Configure SQL Jobs in SQL Server using T-SQL](#)
3. [How to Escape Square Brackets in a LIKE Clause in SQL Server?](#)
4. [SQL | Procedures in PL/SQL](#)
5. [SQL | Difference between functions and stored procedures in PL/SQL](#)
6. [SQL SERVER – Input and Output Parameter For Dynamic SQL](#)
7. [Difference between SQL and T-SQL](#)
8. [SQL Server | Convert tables in T-SQL into XML](#)
9. [SQL SERVER | Bulk insert data from csv file using T-SQL command](#)
10. [SQL - SELECT from Multiple Tables with MS SQL Server](#)

Previous

Next



SQL | SOME

[Read](#)
[Discuss](#)
[Courses](#)
[Practice](#)

SQL | ALL and ANY

SOME operator evaluates the condition between the outer and inner tables and evaluates to true if the final result returns **any one** row. If not, then it evaluates to false.

- The SOME and ANY comparison conditions are similar to each other and are completely interchangeable.
- SOME must match at least one row in the subquery and must be preceded by comparison operators.

Syntax:

```
SELECT column_name(s)
FROM table_name
WHERE expression comparison_operator SOME (subquery)
```

Instructor Table:

Name	Department	Salary
Chandra	Computational Biology	1
Visweswaran	Electronics	1.5
Abraham	Computer Science	1.3
John	Electronics	1.2
Samantha	Computer Science	2
Jyoti	Electronics	1.2
Debarka	Computer Science	2

Ganesh	Computational Biology	0.9
--------	-----------------------	-----

Sample Queries and Outputs:

AD

```
select name
from instructor
where Salary > some(select Salary
from instructor
where dept='Computer Science');
```

Output:

Visweswaran
Samantha
Debarka

Explanation

The instructors with salary > (salary of some instructor in the 'Computer Science' department) get returned. The salaries in the 'Computer Science' department are 1.3, 2 and 2. This implies any instructor with a salary greater than 1.3 can be included in the final result.

Exercise: Try to write same query using ANY clause.

This article is contributed by **Anannya Uberoi**. If you like GeeksforGeeks and would like to contribute, you can also write an article using contribute.geeksforgeeks.org or mail your article to contribute@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Last Updated : 21 Mar, 2018

16

Similar Reads



SQL | EXISTS

[Read](#)[Discuss](#)

The EXISTS condition in SQL is used to check whether the result of a correlated nested query is empty (contains no tuples) or not. The result of EXISTS is a boolean value True or False. It can be used in a SELECT, UPDATE, INSERT or DELETE statement.

Syntax:

```
SELECT column_name(s)
FROM table_name
WHERE EXISTS
  (SELECT column_name(s)
   FROM table_name
   WHERE condition);
```

Examples:

Consider the following two relation “Customers” and “Orders”.

Customers

customer_id	lname	fname	website
401	Singh	Dolly	abc.com
402	Chauhan	Anuj	def.com
403	Kumar	Niteesh	ghi.com
404	Gupta	Shubham	JKL.com
405	Walecha	Divya	abc.com
406	Jain	Sandeep	JKL.com
407	Mehta	Rajiv	abc.com
408	Mehra	Anand	abc.com

Orders

order_id	c_id	order_date
1	407	2017-03-03
2	405	2017-03-05
3	408	2017-01-18
4	404	2017-02-05

Queries

AD

1. Using EXISTS condition with SELECT statement

To fetch the first and last name of the customers who placed atleast one order.

```
SELECT fname, lname
FROM Customers
WHERE EXISTS (SELECT *
               FROM Orders
              WHERE Customers.customer_id = Orders.c_id);
```

Output:

fname	lname
Shubham	Gupta
Divya	Walecha
Rajiv	Mehta
Anand	Mehra

2. Using NOT with EXISTS

Fetch last and first name of the customers who has not placed any order.

```
SELECT lname, fname
FROM Customer
WHERE NOT EXISTS (SELECT *
                   FROM Orders
                   WHERE Customers.customer_id = Orders.c_id);
```

Output:

lname	fname
Singh	Dolly
Chauhan	Anuj
Kumar	Niteesh
Jain	Sandeep

3. Using EXISTS condition with DELETE statement

Delete the record of all the customer from Order Table whose last name is 'Mehra'.

```
DELETE
FROM Orders
WHERE EXISTS (SELECT *
               FROM customers
               WHERE Customers.customer_id = Orders.cid
               AND Customers.lname = 'Mehra');
```

```
SELECT * FROM Orders;
```

Output:

order_id	c_id	order_date
1	407	2017-03-03
2	405	2017-03-05
4	404	2017-02-05

4. Using EXISTS condition with UPDATE statement

Update the lname as 'Kumari' of customer in Customer Table whose customer_id is 401.

```
UPDATE Customers
SET lname = 'Kumari'
WHERE EXISTS (SELECT *
               FROM Customers
              WHERE customer_id = 401);
```

```
SELECT * FROM Customers;
```

Output:

customer_id	lname	fname	website
401	Kumari	Dolly	abc.com
402	Chauhan	Anuj	def.com
403	Kumar	Niteesh	ghi.com
404	Gupta	Shubham	jk.com
405	Walecha	Divya	abc.com
406	Jain	Sandeep	jk.com
407	Mehta	Rajiv	abc.com
408	Mehra	Anand	abc.com

This article is contributed by [Anuj Chauhan](#). If you like GeeksforGeeks and would like to contribute, you can also write an article using [contribute.geeksforgeeks.org](#) or mail your article to contribute@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Last Updated : 27 Apr, 2017

48

Similar Reads



SQL | Aliases

[Read](#) [Discuss](#) [Courses](#) [Practice](#)

Pre-Requisites:[SQL table](#)

Aliases are the temporary names given to tables or columns for the purpose of a particular [SQL](#) query. It is used when the name of a column or table is used other than its original name, but the modified name is only temporary.

- Aliases are created to make table or column names more readable.
- The renaming is just a temporary change and the table name does not change in the original database.
- Aliases are useful when table or column names are big or not very readable.
- These are preferred when there is more than one table involved in a query.

Syntax for Column Alias

SELECT column as alias_name FROM table_name;

column: fields in the table

AD

alias_name: temporary alias name to be used in

replacement of original column name

table_name: name of table

Parameter Explanation

The following table explains the arguments in detail:

- Column_Name: The column name can be defined as the column on which we are going to create an alias name.
- Alias_Name: It can be defined as a temporary name that we are going to assign for the column or table.
- AS: It is optional. If you have not specified it, there is no effect on the query execution.

Syntax for Table Alias

```
SELECT column FROM table_name as alias_name;
column: fields in the table
table_name: name of table
alias_name: temporary alias name to be used in replacement
of original table name
```

Lets see examples for SQL Aliases.

```
CREATE TABLE Customer(
    CustomerID INT PRIMARY KEY,
    CustomerName VARCHAR(50),
    LastName VARCHAR(50),
    Country VARCHAR(50),
    Age int(2),
    Phone int(10)
);
-- Insert some sample data into the Customers table
INSERT INTO Customer (CustomerID, CustomerName, LastName, Country, Age, Phone)
VALUES (1, 'Shubham', 'Thakur', 'India', '23', 'xxxxxxxxxx'),
       (2, 'Aman ', 'Chopra', 'Australia', '21', 'xxxxxxxxxx'),
       (3, 'Naveen', 'Tulasi', 'Sri lanka', '24', 'xxxxxxxxxx'),
       (4, 'Aditya', 'Arpan', 'Austria', '21', 'xxxxxxxxxx'),
       (5, 'Nishant. Salchichas S.A.', 'Jain', 'Spain', '22', 'xxxxxxxxxx');
Select * from Customer;
```

Output:

CustomerID	CustomerName	LastName	Country	Age	Phone
1	Shubham	Thakur	India	23	xxxxxxxxxx
2	Aman	Chopra	Australia	21	xxxxxxxxxx
3	Naveen	Tulasi	Sri Lanka	24	xxxxxxxxxx
4	Aditya	Arpan	Austria	21	xxxxxxxxxx
5	Nishant. Salchichas S.A.	Jain	Spain	22	xxxxxxxxxx

Example 1: Column Alias

To fetch SSN from the customer table using CustomerID as an alias name.

Query:

```
SELECT CustomerID AS SSN FROM Customer;
```

Output:

SSN
1
2
3
4
5

Example 2: Table Alias

Generally, table aliases are used to fetch the data from more than just a single table and connect them through field relations.

To fetch the CustomerName and Country of the customer with Age = 21.

Query:

```
SELECT s.CustomerName, d.Country
FROM Customer AS s, Customer
AS d WHERE s.Age=21 AND
s.CustomerID=d.CustomerID;
```

Output:

CustomerName	Country
Aman	Australia
Aditya	Austria

Advantages of SQL Alias

1. It is useful when you use the function in the query.
2. It can also allow us to combine two or more columns.
3. It is also useful when the column names are big or not readable.
4. It is used to combine two or more columns.

This article is contributed by **Pratik Agarwal**. If you like GeeksforGeeks and would like to contribute, you can also write an article using write.geeksforgeeks.org or mail your article to review-team@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks. Please comment if you find anything incorrect or want to share more information about the topic discussed above.

Last Updated : 05 Apr, 2023

24

Similar Reads

1. Difference between Structured Query Language (SQL) and Transact-SQL (T-SQL)

2. Configure SQL Jobs in SQL Server using T-SQL

3. SQL vs NO SQL vs NEW SQL

4. SQL | Procedures in PL/SQL

5. SQL | Difference between functions and stored procedures in PL/SQL

6. SQL SERVER – Input and Output Parameter For Dynamic SQL

7. Difference between T-SQL and PL-SQL

8. Difference between SQL and T-SQL

9. SQL Server | Convert tables in T-SQL into XML

10. SQL SERVER | Bulk insert data from csv file using T-SQL command



Trending Now DSA Data Structures Algorithms Interview Preparation Data Science Topic-wise Practice J.

SQL | Alternative Quote Operator



classyallrounder

[Read](#)

[Discuss](#)

[Courses](#)

[Practice](#)

This post is a continuation of the [SQL Concatenation Operator](#).

Now, suppose we want to use **apostrophe** in our literal value but we can't use it directly.

See **Incorrect** code:

```
SELECT id, first_name, last_name, salary,
first_name||' has salary's '||salary
AS "new" FROM one
```

So above we are getting error, because Oracle server thinking that the **first apostrophe** is for the **starting literal** and the **second apostrophe** is for the **ending literal**, so what about the **third apostrophe**???. That's why we get error.

AD

Alternative Quote Operator(*q*)

To **overcome** the above problem Oracle introduce an operator known as **Alternative Quote Operator(*q*)**.

Let's see through an example:

Query that uses Alternative Quote Operator(*q*)

```
SELECT id, first_name, last_name, salary,
first_name||q'{ has salary's }'||salary
AS "new" FROM myTable
```

Output:

See, we are able to use apostrophe in the new column as a literal value of myTable

ID	FIRST_NAME	LAST_NAME	SALARY	new
3	Shane	Watson	50000	Shane has salary's 50000
1	Rajat	Rawat	10000	Rajat has salary's 10000
2	Geeks	ForGeeks	20000	Geeks has salary's 20000
3	MS	Dhoni	90000	MS has salary's 90000

Here above see, q'{ indicates starting of our literal value and then we use }' which indicates end of our literal value. So see here we have used apostrophe in our literal value easily(means we easily use 's in salary) without any error that's why we get output as Rajat has salary's 50000.

So to use apostrophe in literal we first need to use **q** which is known as **alternative quote operator** after that we need to use an apostrophe ' and after that we need to use a **delimiter** and after delimiter we write our literal value, when we finished writing our literal value then again we need to **close the delimiter** which we have **opened before** and after that we need to put an **apostrophe again** and hence in this way we can use apostrophe in our literal value. This concept is known as **Alternative Quote Operator(q)**.

We can use **any character** such as {, <, (, [, ! or any character as **delimiter**. These characters are known as **delimiters**.

1 another example

:

Without using Quote Operator:

Here we get **Error** since we are using **apostrophe** in our literal value directly.

Error code below:

```
SELECT id, name, dept, name||' work's in '||dept||'
      department' AS "work" FROM myTable2
```

Using Quote Operator:

```
SELECT id, name, dept, name||q'[ work's in ']||dept||'
      department' AS "work" FROM myTable2
```

Output:

See, we are able to use apostrophe in the work column as a literal value of myTable2

ID	NAME	DEPT	work
1	RR	Executive	RR work's in 'Executive department
2	GFG	HR	GFG work's in 'HR department
3	Steve	Sales	Steve work's in 'Sales department
4	Bhuvi	CSE	Bhuvi work's in 'CSE department

Here above see, **q'['** indicates starting of our literal value and the we use **']'** which indicate end of our literal value. So see here we have used **apostrophe** in our literal value easily(means we easily use 's in work) without any error that's why we get output as RR **work's** in Executive Department.]

Here above we use **[** as delimiter so it is **not** a limitation in using delimiter means we can use any character as delimiter.

References:

[About Alternative Quote Operator,](#)

[Performing SQL Queries Online](#)

Last Updated : 21 Mar, 2018

6

Similar Reads

1. What is Alternative to _N_ in PROC SQL?

2. QUOTE () function in MySQL

3. Difference between Structured Query Language (SQL) and Transact-SQL (T-SQL)

4. Configure SQL Jobs in SQL Server using T-SQL

5. SQL | BETWEEN & IN Operator

6. SQL | MINUS Operator

7. SQL | NOT Operator

8. SQL | Concatenation Operator

9. Difference between = and IN operator in SQL

10. SQL | UNION Operator

Previous

Next



SQL | WHERE Clause

[Read](#) [Discuss](#) [Courses](#) [Practice](#)

WHERE keyword is used for fetching **filtered data** in a result set. It is used to fetch data according to particular criteria. **WHERE** keyword can also be used to filter data by matching patterns.

Syntax:

SELECT column1,column2 FROM table_name WHERE column_name operator value;

Parameter Explanation:

AD

1. **column1,column2**: fields in the table
2. **table_name**: name of table
3. **column_name**: name of field used for filtering the data
4. **operator**: operation to be considered for filtering
5. **value**: exact value or pattern to get related data in result

List of Operators that Can be Used with WHERE Clause

Operator	Description
>	Greater Than
>=	Greater than or Equal to

Operator	Description
<	Less Than
<=	Less than or Equal to
=	Equal to
<>	Not Equal to
BETWEEN	In an inclusive Range
LIKE	Search for a pattern
IN	To specify multiple possible values for a column

Query:

```

CREATE TABLE Emp1(
    EmpID INT PRIMARY KEY,
    Name VARCHAR(50),
    Country VARCHAR(50),
    Age int(2),
    mob int(10)
);
-- Insert some sample data into the Customers table
INSERT INTO Emp1 (EmpID, Name,Country, Age, mob)
VALUES (1, 'Shubham', 'India','23','738479734'),
       (2, 'Aman ', 'Australia','21','436789555'),
       (3, 'Naveen', 'Sri lanka','24','34873847'),
       (4, 'Aditya', 'Austria','21','328440934'),
       (5, 'Nishant', 'Spain','22','73248679');
Select * from Emp1;

```

Where Clause with Logical Operators

To fetch records of Employee with ages equal to 24.

Query:

```
SELECT * FROM Emp1 WHERE Age=24;
```

Output:

EmpID	Name	Country	Age	mob
3	Naveen	Sri lanka	24	34873847

To fetch the EmpID, Name and Country of Employees with Age greater than 21.

Query:

```
SELECT EmpID, Name, Country FROM Emp1 WHERE Age > 21;
```

Output:

EmpID	Name	Country
1	Shubham	India
3	Naveen	Sri lanka
5	Nishant	Spain

Where Clause with BETWEEN Operator

It is used to fetch filtered data in a given range inclusive of two values.

Syntax:

```
SELECT column1,column2 FROM table_name
```

```
WHERE column_name BETWEEN value1 AND value2;
```

Parameter Explanation:

1. **BETWEEN**: operator name
2. **value1 AND value2**: exact value from value1 to value2 to get related data in result set.

To fetch records of Employees where Age is between 22 and 24 (inclusive).

Query:

```
SELECT * FROM Emp1 WHERE Age BETWEEN 22 AND 24;
```

Output:

EmpID	Name	Country	Age	mob
1	Shubham	India	23	738479734
3	Naveen	Sri lanka	24	34873847
5	Nishant	Spain	22	73248679

Where Clause with LIKE Operator

It is used to fetch filtered data by searching for a particular pattern in the where clause.

Syntax:

```
SELECT column1,column2 FROM
table_name WHERE column_name LIKE pattern;
```

Parameters Explanation:

1. **LIKE**: operator name
2. **pattern**: exact value extracted from the pattern to get related data in the result set.

Note: The character(s) in the pattern is case sensitive.

To fetch records of Employees where Name starts with the letter S.

Query:

```
SELECT * FROM Emp1 WHERE Name LIKE 'S%';
```

The '%' wildcard signifies the later characters here which can be of any length and value.

Output:

EmpID	Name	Country	Age	mob
1	Shubham	India	23	738479734

To fetch records of Employees where Name contains the pattern 'M'.

Query:

```
SELECT * FROM Emp1 WHERE Name LIKE '%M%';
```

Output:

EmpID	Name	Country	Age	mob
1	Shubham	India	23	738479734
2	Aman	Australia	21	436789555

Where Clause with IN Operator

It is used to fetch the filtered data same as fetched by '=' operator just the difference is that here we can specify multiple values for which we can get the result set.

Syntax:

```
SELECT column1,column2 FROM table_name WHERE column_name IN  
(value1,value2,...);
```

Parameters Explanation:

1. IN: operator name
2. value1,value2,: exact value matching the values given and get related data in the result set.

To fetch the Names of Employees where Age is 21 or 23.

Query:

```
SELECT Name FROM Emp1 WHERE Age IN (21,23);
```

Output:

Name
Shubham
Aman
Aditya

This article is contributed by **Harsh Agarwal**. If you like GeeksforGeeks and would like to contribute, you can also write an article using [write.geeksforgeeks.org](https://www.geeksforgeeks.org/write-for-us/) or mail your article to review-team@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks. Please write comments if you find anything incorrect, or if you want to share more information about the topic discussed above.



SQL | USING Clause



akanshgupta

[Read](#)[Discuss](#)[Courses](#)[Practice](#)

If several columns have the same names but the datatypes do not match, the [NATURAL JOIN](#) clause can be modified with the **USING** clause to specify the columns that should be used for an [EQUIJOIN](#).

- USING Clause is used to match only one column when more than one column matches.
- NATURAL JOIN and USING Clause are mutually exclusive.
- It should not have a qualifier(table name or Alias) in the referenced columns.
- NATURAL JOIN uses all the columns with matching names and datatypes to join the tables. The USING Clause can be used to specify only those columns that should be used for an EQUIJOIN.

EXAMPLES:

We will apply the below mentioned commands on the following base tables:

AD

Home > SQL > SQL Commands

Autocommit Display 10

```
SELECT * FROM Employees;
```

Results Explain Describe Saved SQL History

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COMMISSION_PCT	MANAGER_ID	DEPARTMENT_ID
100	Steven	King	SKING	515.123.4567	17-JUN-87	AD_PRES	24000	-	-	90
101	Neena	Kochhar	NKOCHHAR	515.123.4568	21-SEP-89	AD_VP	17000	-	100	90
102	Lex	De Haan	LDEHAAN	515.123.4569	13-JAN-93	AD_VP	17000	-	100	90
103	Alexander	Hunold	AHUNOLD	590.423.4567	03-JAN-90	IT_PROG	9000	-	102	60
104	Bruce	Ernst	BERNST	590.423.4568	21-MAY-91	IT_PROG	6000	-	103	60
105	David	Austin	DAUSTIN	590.423.4569	25-JUN-97	IT_PROG	4800	-	103	60
106	Valli	Pataballa	VPATABAL	590.423.4560	05-FEB-98	IT_PROG	4800	-	103	60
107	Diana	Lorentz	DLORENTZ	590.423.5567	07-FEB-99	IT_PROG	4200	-	103	60
108	Nancy	Greenberg	NGREENBE	515.124.4569	17-AUG-94	FL_MGR	12000	-	101	100
109	Daniel	Faviet	DFAVIET	515.124.4169	16-AUG-94	FI_ACCOUNT	9000	-	108	100

More than 10 rows available. Increase rows selector to view more rows.

Employee Table

DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
10	Administration	200	1700
20	Marketing	201	1600
30	Purchasing	114	1700
40	Human Resources	203	2400
50	Shipping	121	1500
60	IT	103	1400
70	Public Relations	204	2700
80	Sales	145	2500
90	Executive	100	1700
100	Finance	106	1700

More than 10 rows available. Increase rows selector to view more rows.
10 rows returned in 0.03 seconds [CSV Export](#)

Department Table

QUERY 1: Write SQL query to find the working location of the employees. Also give their respective employee_id and last_name?

Input : SELECT e.EMPLOYEE_ID, e.LAST_NAME, d.LOCATION_ID
FROM Employees e JOIN Departments d
USING(DEPARTMENT_ID);
Output :

EMPLOYEE_ID	LAST_NAME	LOCATION_ID
100	King	1700
101	Kochhar	1700
102	De Haan	1700
103	Hunold	1400
104	Ernst	1400
105	Austin	1400
106	Pataballa	1400
107	Lorentz	1400
108	Greenberg	1700
109	Falet	1700

More than 10 rows available. Increase rows selector to view more rows.
10 rows returned in 0.05 seconds [CSV Export](#)

Explanation: The example shown joins the DEPARTMENT_ID column in the EMPLOYEES and DEPARTMENTS

tables, and thus shows the location where an employee works.

We will apply the below mentioned commands on the following base tables:

The screenshot shows the Oracle Database Express Edition interface. The SQL command entered is:

```
select * from countries;
```

The results are displayed in a table:

COUNTRY_ID	COUNTRY_NAME	REGION_ID
AR	Argentina	2
AU	Australia	3
BE	Belgium	1
BR	Brazil	2
CA	Canada	2
CH	Switzerland	1
CN	China	3
DE	Germany	1
DK	Denmark	1
EG	Egypt	4

More than 10 rows available. Increase rows selector to view more rows.

10 rows returned in 0.00 seconds [CSV Export](#)

Country Table

The screenshot shows the Oracle Database Express Edition interface. The SQL command entered is:

```
select * from locations;
```

The results are displayed in a table:

LOCATION_ID	STREET_ADDRESS	POSTAL_CODE	CITY	STATE_PROVINCE	COUNTRY_ID
1000	1237 Via Cola di Rie	00889	Roma	-	IT
1100	93091 Calle della Testa	10934	Venice	-	IT
1200	2017 Shinjuku-ku	1689	Tokyo	Tokyo Prefecture	JP
1300	9450 Kamiya-cho	8823	Hiroshima	-	JP
1400	2014 Jabberwocky Rd	26192	Southlake	Texas	US
1500	2011 Interior Blvd	98236	South San Francisco	California	US
1600	2007 Zagora St	50090	South Brunswick	New Jersey	US
1700	2004 Charade Rd	98199	Seattle	Washington	US
1800	147 Spadina Ave	M5V 2L7	Toronto	Ontario	CA
1900	6092 Boxwood St	Y5W 8T2	Whitehorse	Yukon	CA

More than 10 rows available. Increase rows selector to view more rows.

10 rows returned in 0.00 seconds [CSV Export](#)

Location Table

QUERY 2: Write SQL query to find the location_id, street_address, postal_code and their respective country name?

Input : `SELECT l.location_id, l.street_address, l.postal_code, c.country_name
FROM locations l JOIN countries c
USING(country_id);`

Output :

The screenshot shows the Oracle Database Express Edition interface. The SQL command entered is:

```
SELECT l.location_id, l.street_address, l.postal_code, c.country_name
FROM locations l JOIN countries c
USING(country_id);
```

The results table displays 10 rows of data:

LOCATION_ID	STREET_ADDRESS	POSTAL_CODE	COUNTRY_NAME
2200	12-98 Victoria Street	2901	Australia
2800	Rua Frei Caneca 1360	01307-002	Brazil
1800	147 Spadina Ave	MSV 2L7	Canada
1900	6092 Boxwood St	YSW 9T2	Canada
2900	29 Rue des Corps-Saints	1730	Switzerland
3000	Murtenthalstrasse 921	3095	Switzerland
2000	40-5-12 Lüggen gen	190518	China
2700	Schwanthalerstr. 7031	80925	Germany
2100	1296 Vileparle (E)	490231	India
1000	1297 Via Cola di Rie	00989	Italy

More than 10 rows available. Increase rows selector to view more rows.

CSV Export

Explanation: The example shown joins the COUNTRY_ID column in the LOCATIONS and COUNTRIES

tables, and thus shows the required details.

NOTE: When we use the **USING** clause in a join statement, the join column is not qualified with table Alias. Do not Alias it even if the same column is used elsewhere in the SQL statement:

Example:

Input: `SELECT l.location_id, l.street_address, l.postal_code, c.country_name
FROM locations l JOIN countries c
USING(country_id)
WHERE c.country_id='IT';`

Output:

The screenshot shows the Oracle Database Express Edition interface. The SQL command entered is:

```
SELECT l.location_id, l.street_address, l.postal_code, c.country_name
FROM locations l JOIN countries c
USING(c.country_id)
WHERE c.country_id='IT';
```

An error message is displayed at the bottom:

ORA-25154: column part of USING clause cannot have qualifier

Explanation: Since the column in USING Clause is used again in WHERE Clause, thus it throws an error to the user.

Last Updated : 21 Mar, 2018

14

Similar Reads

1. Difference between Having clause and Group by clause

2. Combining aggregate and non-aggregate values in SQL using Joins and Over clause

3. SQL Full Outer Join Using Left and Right Outer Join and Union Clause

4. SQL query using COUNT and HAVING clause

5. SQL Full Outer Join Using Union Clause

6. SQL Full Outer Join Using Where Clause

7. Using CASE in ORDER BY Clause to Sort Records By Lowest Value of 2 Columns in SQL

8. Configure SQL Jobs in SQL Server using T-SQL

9. SQL | Distinct Clause

10. SQL | WHERE Clause

Previous

Next

Article Contributed By :



akanshgupta
akanshgupta

Vote for difficulty

Current difficulty : [Easy](#)

Easy	Normal	Medium	Hard	Expert
------	--------	--------	------	--------

Article Tags : [SQL-Clauses-Operators](#), [SQL](#)

Practice Tags : [SQL](#)



SQL | MERGE Statement

[Read](#)[Discuss](#)[Courses](#)[Practice](#)

Prerequisite – [INSERT](#), [UPDATE](#), [DELETE](#)

The **MERGE** command in SQL is actually a combination of three SQL statements: **INSERT**, **UPDATE** and **DELETE**. In simple words, the MERGE statement in SQL provides a convenient way to perform all these three operations together which can be very helpful when it comes to handle the large running databases. But unlike INSERT, UPDATE and DELETE statements MERGE statement requires a source table to perform these operations on the required table which is called as target table.

Now we know that the MERGE in SQL requires two tables : one the target table on which we want to perform INSERT, UPDATE and DELETE operations, and the other one is source table which contains the new modified and correct data for target table and is actually compared with the actual target table in order to modify it.

In other words, the MERGE statement in SQL basically merges data from a source result set to a target table based on a condition that is specified. The syntax of MERGE statement can be complex to understand at first but its very easy once you know what it means. So, not to get confused first let's discuss some basics. Suppose you have two tables: source and target, now think if you want to make changes in the required target table with the help of provided source table which consists of latest details.

AD

- When will you need to insert the data in the target table?
Obviously when there is data in source table and not in target table *i.e* when data not matched with target table.
- When will you need to update the data?
When the data in source table is matched with target table but any entry other than the primary key is not matched.

- When will you need to delete the data?

When there is data in target table and not in source table *i.e* when data not matched with source table.

Now, we know when to use INSERT, UPDATE and DELETE statements in case we want to use MERGE statement so there should be no problem for you understanding the syntax given below :

```
//.....syntax of MERGE statement....//  
  
//you can use any other name in place of target  
MERGE target_table_name AS TARGET  
  
//you can use any other name in place of source  
USING source_table_name AS SOURCE  
ON condition (for matching source and target table)  
WHEN MATCHED (another condition for updation)  
  
//now use update statement syntax accordingly  
THEN UPDATE  
WHEN NOT MATCHED BY TARGET  
  
//now use insert statement syntax accordingly  
THEN INSERT  
WHEN NOT MATCHED BY SOURCE  
THEN DELETE;
```

That's all about the MERGE statement and its syntax.

References –

[MERGE – docs.microsoft](#)

[MERGE – docs.oracle](#)

This article is contributed by [Dimpy Varshni](#) If you like GeeksforGeeks and would like to contribute, you can also write an article using [contribute.geeksforgeeks.org](#) or mail your article to contribute@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Last Updated : 31 Jan, 2019

20

Similar Reads



MERGE Statement in SQL Explained

[Read](#) [Discuss](#) [Courses](#) [Practice](#)

Prerequisite – MERGE Statement

As MERGE statement in SQL, as discussed before in the [previous post](#), is the combination of three INSERT, DELETE and UPDATE statements. So if there is a **Source table** and a **Target table** that are to be merged, then with the help of MERGE statement, all the three operations (INSERT, UPDATE, DELETE) can be performed at once.

A simple example will clarify the use of MERGE Statement.

Example:

Suppose there are two tables:

- **PRODUCT_LIST** which is the table that contains the current details about the products available with fields P_ID, P_NAME, and P_PRICE corresponding to the ID, name and price of each product.
- **UPDATED_LIST** which is the table that contains the new details about the products available with fields P_ID, P_NAME, and P_PRICE corresponding to the ID, name and price of each product.

PRODUCT_LIST

P_ID	P_NAME	P_PRICE
101	TEA	10.00
102	COFFEE	15.00
103	BISCUIT	20.00

UPDATED_LIST

P_ID	P_NAME	P_PRICE
101	TEA	10.00
102	COFFEE	25.00
104	CHIPS	22.00

AD

The task is to update the details of the products in the PRODUCT_LIST as per the UPDATED_LIST.

Solution

Now in order to explain this example better, let's split the example into steps.

Step 1: Recognise the TARGET and the SOURCE table

So in this example, since it is asked to update the products in the PRODUCT_LIST as per the UPDATED_LIST, hence the PRODUCT_LIST will act as the TARGET and UPDATED_LIST will act as the SOURCE table.



TARGET			SOURCE		
<u>PRODUCT_LIST</u>			<u>UPDATED_LIST</u>		
P_ID	P_NAME	P_PRICE	P_ID	P_NAME	P_PRICE
101	TEA	10.00	101	TEA	10.00
102	COFFEE	15.00	102	COFFEE	25.00
103	BISCUIT	20.00	104	CHIPS	22.00

Step 2: Recognise the operations to be performed.

Now as it can be seen that there are three mismatches between the TARGET and the SOURCE table, which are:

1. The cost of COFFEE in TARGET is 15.00 while in SOURCE it is 25.00

PRODUCT_LIST

102 COFFEE 15.00

UPDATED_LIST

102 COFFEE 25.00

2. There is no BISCUIT product in SOURCE but it is in TARGET

```
PRODUCT_LIST
103      BISCUIT    20.00
```

3. There is no CHIPS product in TARGET but it is in SOURCE

```
UPDATED_LIST
104      CHIPS     22.00
```

Therefore, three operations need to be done in the TARGET according to the above discrepancies. They are:

1. UPDATE operation

```
102      COFFEE    25.00
```

2. DELETE operation

```
103      BISCUIT   20.00
```

3. INSERT operation

```
104      CHIPS     22.00
```

Step 3: Write the SQL Query.

Note: Refer [this post](#) for the syntax of MERGE statement.

The **SQL query** to perform the above-mentioned operations with the help of **MERGE statement** is:

SQL

```
/* Selecting the Target and the Source */
MERGE PRODUCT_LIST AS TARGET
  USING UPDATE_LIST AS SOURCE

  /* 1. Performing the UPDATE operation */

  /* If the P_ID is same,
     check for change in P_NAME or P_PRICE */
  ON (TARGET.P_ID = SOURCE.P_ID)
  WHEN MATCHED
    AND TARGET.P_NAME <> SOURCE.P_NAME
    OR TARGET.P_PRICE <> SOURCE.P_PRICE

  /* Update the records in TARGET */
  THEN UPDATE
```

```

SET TARGET.P_NAME = SOURCE.P_NAME,
TARGET.P_PRICE = SOURCE.P_PRICE

/* 2. Performing the INSERT operation */

/* When no records are matched with TARGET table
Then insert the records in the target table */
WHEN NOT MATCHED BY TARGET
THEN INSERT (P_ID, P_NAME, P_PRICE)
VALUES (SOURCE.P_ID, SOURCE.P_NAME, SOURCE.P_PRICE)

/* 3. Performing the DELETE operation */

/* When no records are matched with SOURCE table
Then delete the records from the target table */
WHEN NOT MATCHED BY SOURCE
THEN DELETE

/* END OF MERGE */

```

Output:

PRODUCT_LIST		
P_ID	P_NAME	P_PRICE
101	TEA	10.00
102	COFFEE	25.00
104	CHIPS	22.00

So, in this way all we can perform all these three main statements in SQL together with the help of MERGE statement.

Note: Any name other than target and source can be used in the MERGE syntax. They are used only to give you a better explanation.

Last Updated : 09 Sep, 2021

21

Similar Reads

1. SQL | MERGE Statement

2. Difference between Structured Query Language (SQL) and Transact-SQL (T-SQL)

3. Configure SQL Jobs in SQL Server using T-SQL

4. SQL vs NO SQL vs NEW SQL

5. SQL INSERT INTO Statement

6. SQL | DELETE Statement



SQL | Intersect & Except clause

Trending Now DSA Data Structures Algorithms Interview Preparation Data Science Topic-wise Practice J:

[Read](#)

[Discuss](#)

[Courses](#)

[Practice](#)

1. INTERSECT clause : As the name suggests, the intersect clause is used to provide the result of the intersection of two select statements. This implies the result contains all the rows which are common to both the SELECT statements. **Syntax :**

```
SELECT column-1, column-2 .....
FROM table 1
WHERE.....
```

INTERSECT

```
SELECT column-1, column-2 .....
FROM table 2
WHERE.....
```

Example : Table 1 containing Employee Details

ID	Name	Age	City
1	Suresh	24	Delhi
2	Ramesh	23	pune
3	Kashish	34	Agra

Table

2 containing details of employees who are provided bonus

Bonus_ID	Employee_ID	Bonus (in RS.)
43	1	20,000
45	3	30,000

Query :

```
SELECT ID, Name, Bonus
FROM
table1
LEFT JOIN
table2
ON table1.ID = table2.Employee_ID
```

INTERSECT

```
SELECT ID, Name, Bonus
FROM
table1
RIGHT JOIN
table2
ON table1.ID = table2.Employee_ID;
```

Result :

ID	Name	Bonus
1	Suresh	20,000
3	Kashish	30,000

EXCEPT clause : contains all the rows that are returned by the first SELECT operation, and not returned by the second SELECT operation. **Syntax :**

```
SELECT column-1, column-2 ....  
FROM table 1  
WHERE.... .
```

EXCEPT

```
SELECT column-1, column-2 ....  
FROM table 2  
WHERE.... .
```

Example : Table 1 containing Employee Details

ID	Name	Age	City
1	Suresh	24	Delhi
2	Ramesh	23	pune
3	Kashish	34	Agra

Table

2 containing details of employees who are provided bonus

Bonus_ID	Employee_ID	Bonus (in RS.)
43	1	20,000
45	3	30,000

Query :

AD

```
SELECT ID, Name, Bonus
FROM
table1
LEFT JOIN
table2
ON table1.ID = table2.Employee_ID

EXCEPT

SELECT ID, Name, Bonus
FROM
table1
RIGHT JOIN
table2
ON table1.ID = table2.Employee_ID;
```

Result :

ID	Name	Bonus
2	Ramesh	Null

Similar Reads

1. SQL | Except Clause



SQL | Distinct Clause

Read Discuss Courses Practice

The distinct keyword is used in conjunction with the select keyword. It is helpful when there is a need to avoid duplicate values present in any specific columns/table. When we use distinct keywords only the **unique values** are fetched.

Syntax :

SELECT DISTINCT column1, column2

FROM table_name

AD

1. **column1, column2:** Names of the fields of the table.
2. **Table_name:** Table from where we want to fetch the records.

This query will return all the unique combinations of rows in the table with fields column1, and column2.

NOTE: If a distinct keyword is used with multiple columns, the distinct combination is displayed in the result set.

Distinct Operations

Query:

```
CREATE TABLE students (
    ROLL_NO INT,
    NAME VARCHAR(50),
    ADDRESS VARCHAR(100),
```

```
PHONE VARCHAR(20),
AGE INT
);
```

Inserting some random data to perform distinct operations.

```
INSERT INTO students (ROLL_NO, NAME, ADDRESS, PHONE, AGE)
VALUES
(1, 'Shubham Kumar', '123 Main Street, Bangalore', '9876543210', 23),
(2, 'Shreya Gupta', '456 Park Road, Mumbai', '9876543211', 23),
(3, 'Naveen Singh', '789 Market Lane, Delhi', '9876543212', 26),
(4, 'Aman Chopra', '246 Forest Avenue, Kolkata', '9876543213', 22),
(5, 'Aditya Patel', '7898 Ocean Drive, Chennai', '9876543214', 27),
(6, 'Avdeep Desai', '34 River View, Hyderabad', '9876543215', 24);
```

Output:

ROLL_NO	NAME	ADDRESS	PHONE	AGE
1	Shubham Kumar	123 Main Street, Bangalore	9876543210	23
2	Shreya Gupta	456 Park Road, Mumbai	9876543211	23
3	Naveen Singh	789 Market Lane, Delhi	9876543212	26
4	Aman Chopra	246 Forest Avenue, Kolkata	9876543213	22
5	Aditya Patel	7898 Ocean Drive, Chennai	9876543214	27
6	Avdeep Desai	34 River View, Hyderabad	9876543215	24

Now, to fetch unique names from the NAME field.

Query:

```
SELECT DISTINCT NAME FROM Student;
```

Output :

NAME
Shubham Kumar
Shreya Gupta
Naveen Singh
Aman Chopra
Aditya Patel
Avdeep Desai

Now, to fetch a unique combination of rows from the whole table.

Syntax:

*SELECT DISTINCT * FROM Table_name;*

Query:

```
SELECT DISTINCT * FROM students;
```

Output :

ROLL_NO	NAME	ADDRESS	PHONE	AGE
1	Shubham Kumar	123 Main Street, Bangalore	9876543210	23
2	Shreya Gupta	456 Park Road, Mumbai	9876543211	23
3	Naveen Singh	789 Market Lane, Delhi	9876543212	26
4	Aman Chopra	246 Forest Avenue, Kolkata	9876543213	22
5	Aditya Patel	7898 Ocean Drive, Chennai	9876543214	27
6	Avdeep Desai	34 River View, Hyderabad	9876543215	24

Using Distinct Clause with Order By

Here, we will check the order by clause with a Distinct clause which will filter out the data on the basis of the order by clause.

Query:

```
SELECT DISTINCT ROLL_NO FROM Students ORDER BY AGE;
```

Output:

ROLL_NO
4
1
2
6
3
5

How the DISTINCT Clause Handles NULL Values?

Finally, does the DISTINCT clause consider a NULL to be a unique value in SQL? The answer is yes.

CREATE TABLE:

```
CREATE TABLE students (
    ROLL_NO INT,
```

```

        NAME VARCHAR(50),
        ADDRESS VARCHAR(100),
        PHONE VARCHAR(20),
        AGE INT
    );
INSERT INTO students (ROLL_NO, NAME, ADDRESS, PHONE, AGE)
VALUES
    (1, 'Shubham Kumar', '123 Main Street, Bangalore', '9876543210', 23),
    (2, 'Shreya Gupta', '456 Park Road, Mumbai', '9876543211', 23),
    (3, 'Naveen Singh', '789 Market Lane, Delhi', '9876543212', 26),
    (4, 'Aman Chopra', '246 Forest Avenue, Kolkata', '9876543213', 22),
    (5, 'Aditya Patel', '7898 Ocean Drive, Chennai', '9876543214', 27),
    (6, 'Avdeep Desai', '34 River View, Hyderabad', '9876543215', NULL);

```

Output:

ROLL_NO	NAME	ADDRESS	PHONE	AGE
1	Shubham Kumar	123 Main Street, Bangalore	9876543210	23
2	Shreya Gupta	456 Park Road, Mumbai	9876543211	23
3	Naveen Singh	789 Market Lane, Delhi	9876543212	26
4	Aman Chopra	246 Forest Avenue, Kolkata	9876543213	22
5	Aditya Patel	7898 Ocean Drive, Chennai	9876543214	27
6	Avdeep Desai	34 River View, Hyderabad	9876543215	

Query:

```

SELECT DISTINCT AGE
FROM students;

```

AGE
22
23
26
27

Note: Without the keyword distinct in both the above examples 6 records would have been fetched instead of 4, since in the original table there are 6 records with the duplicate values.

This article is contributed by **Harsh Agarwal**. If you like GeeksforGeeks and would like to contribute, you can also write an article using write.geeksforgeeks.org or mail your article to

review-team@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.

Please write comments if you find anything incorrect, or if you want to share more information about the topic discussed above.

Last Updated : 10 May, 2023

33

Similar Reads

1. Difference between Having clause and Group by clause

2. Distinct clause in MS SQL Server

3. SQL | WHERE Clause

4. Top Clause in Microsoft SQL Server

5. SQL | Union Clause

6. SQL | WITH clause

7. SQL | Except Clause

8. SQL | OFFSET-FETCH Clause

9. Having vs Where Clause in SQL

10. SQL | LIMIT Clause

Previous

Next

Article Contributed By :



GeeksforGeeks

Vote for difficulty

Current difficulty : Basic

Easy	Normal	Medium	Hard	Expert
------	--------	--------	------	--------

Improved By : Anshika Goyal, shubhamthakur05



SQL | LIMIT Clause

[Read](#)[Discuss](#)[Courses](#)[Practice](#)

SQL limit clause is very useful in some scenarios where we really need the data in some sorted manner suppose if there are a large number of tuples satisfying the query conditions, it might be resourceful to view only a handful of them at a time.

Important points to remember:

- The LIMIT clause is used to set an upper limit on the number of tuples returned by SQL.
- It is important to note that this clause is not supported by all SQL versions.
- The LIMIT clause can also be specified using the SQL 2008 OFFSET/FETCH FIRST clauses.
- The limit/offset expressions must be a non-negative integer.

Example:

Let's assume that we have one sample table name Student and to understand better we will see some query about limit clause. Say we have a relationship, Student.

AD

Student Table:

```
CREATE TABLE student (
    id INT PRIMARY KEY,
    name VARCHAR(50),
    age INT
);

INSERT INTO student (id, name, age)
VALUES (1, 'Shubham Thakur', 18),
       (2, 'Aman Chopra', 19),
```

```
(3, 'Bhavika uppala', 20),
(4, 'Anshi Shrivastava', 22);
```

Output:

id	name	age
1	Shubham Thakur	18
2	Aman Chopra	19
3	Bhavika uppala	20
4	Anshi Shrivastava	22

Queries:

```
SELECT *
FROM student
LIMIT 3;
```

Output:

id	name	age
1	Shubham Thakur	18
2	Aman Chopra	19
3	Bhavika uppala	20

Other Query to check with ORDER BY Clause:

```
SELECT *
FROM Student
ORDER BY Grade DESC
LIMIT 3;
```

Output:

id	name	age
4	Anshi Shrivastava	22
3	Bhavika uppala	20
2	Aman Chopra	19

The LIMIT operator can be used in situations such as the above, where we need to find the top 3 students in a class and do not want to use any conditional statements.

Using LIMIT along with OFFSET

LIMIT x OFFSET y simply means skip the first y entries and then return the next x entries. OFFSET can only be used with the ORDER BY clause. It cannot be used on its own. OFFSET value must be greater than or equal to zero. It cannot be negative, else returns an error.

Queries:

```
SELECT *
FROM Student
ORDER BY ROLLNO LIMIT 5 OFFSET 2;
```

or

```
SELECT *
FROM Student
ORDER BY ROLLNO LIMIT 2,5; # it skips the
first 2 values and then return the next 5 entries
```

The first query and second query return the same results. In the second query, limit is followed by two values. LIMIT X, Y The first value X is the offset value (skips X number of entries) and the second value Y is the limit (it returns the next Y number of entries).

Output:

id	name	age
3	Bhavika uppala	20
4	Anshi Shrivastava	22

SQL LIMIT to Get the nth Highest or Lowest Value

Now we will look for LIMIT use in finding highest or lowest value we need to retrieve the rows with the nth highest or lowest value. In that situation, we can use the subsequent MySQL LIMIT clause to obtain the desired outcome.

Syntax:

```
SELECT column_list
FROM table_name
ORDER BY expression
LIMIT n-1, 1;
```

Query:

```
SELECT age FROM Student  
ORDER BY age LIMIT 2, 1;
```

Output:

age
20

The Limit in MySQL with Where

The WHERE clause can also be used with MySQL Limit. It produces the rows that matched the condition after checking the specified condition in the table.

Query:

```
SELECT age  
FROM Student  
WHERE id<4  
ORDER BY age  
LIMIT 2, 1;
```

Output:

age
20

Restrictions on the LIMIT clause

The LIMIT clause's limitations. The following situations do not allow the LIMIT clause to be used:

- With regard to defining a view
- The use of nested SELECT statements
- Except for subqueries with table expressions specified in the FROM clause.
- Embedded SELECT statements are used as expressions in a singleton SELECT (where max = 1) within an SPL routine where embedded SELECT statements are used as expressions.

This article is contributed by **Anannya Uberoi**. If you like GeeksforGeeks and would like to contribute, you can also write an article using [write.geeksforgeeks.org](https://www.geeksforgeeks.org/write-for-us/) or mail your article to review-team@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.

Please write comments if you find anything incorrect, or if you want to share more information about the topic discussed above.



SQL | Except Clause

Read Discuss Courses Practice

In SQL, EXCEPT returns those tuples that are returned by the first SELECT operation, and not returned by the second SELECT operation.

This is the same as using a subtract operator in relational algebra.

Example:

Say we have two relations, Students and TA (Teaching Assistant). We want to return all those students who are not teaching assistants. The query can be formulated as:

Students Table:

AD

StudentID	Name	Course
1	Rohan	DBMS
2	Kevin	OS
3	Mansi	DBMS
4	Mansi	ADA
5	Rekha	ADA
6	Megha	OS

TA Table:

StudentID	Name	Course
1	Kevin	TOC
2	Sita	IP
3	Manik	AP
4	Rekha	SNS

```
SELECT Name
      FROM Students
EXCEPT
SELECT NAME
      FROM TA;
```

Output:

Rohan
Mansi
Megha

To retain duplicates, we must explicitly write **EXCEPTALL** instead of EXCEPT.

```
SELECT Name
      FROM Students
EXCEPTALL
SELECT Name
      FROM TA;
```

Output:

Rohan
Mansi
Mansi
Megha

Difference between EXCEPT and NOT IN Clause

EXCEPT automatically removes all duplicates in the final result, whereas NOT IN retains duplicate tuples. It is also important to note that EXCEPT is not supported by MySQL.

This article is contributed by **Anannya Uberoi**. If you like GeeksforGeeks and would like to contribute, you can also write an article using contribute.geeksforgeeks.org or mail your article



SQL | WITH clause

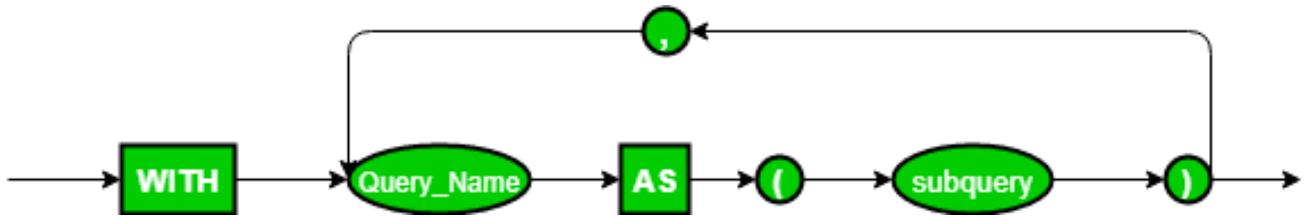
[Read](#)[Discuss](#)[Courses](#)[Practice](#)[Video](#)

The SQL WITH clause was introduced by Oracle in the Oracle 9i release 2 database. The SQL WITH clause allows you to give a sub-query block a name (a process also called sub-query refactoring), which can be referenced in several places within the main SQL query.

- The clause is used for defining a temporary relation such that the output of this temporary relation is available and is used by the query that is associated with the WITH clause.
- Queries that have an associated WITH clause can also be written using nested sub-queries but doing so add more complexity to read/debug the SQL query.
- WITH clause is not supported by all database system.
- The name assigned to the sub-query is treated as though it was an inline view or table
- The SQL WITH clause was introduced by Oracle in the Oracle 9i release 2 database.

Syntax:

```
WITH temporaryTable (averageValue) as
    (SELECT avg(Attr1)
     FROM Table)
    SELECT Attr1
     FROM Table, temporaryTable
    WHERE Table.Attr1 > temporaryTable.averageValue;
```



In this query, WITH clause is used to define a temporary relation temporaryTable that has only 1 attribute `averageValue`. `averageValue` holds the average value of column Attr1 described in relation Table. The SELECT statement that follows the WITH clause will produce only those tuples where the value of Attr1 in relation Table is greater than the average value obtained from the WITH clause statement.

AD

Note: When a query with a WITH clause is executed, first the query mentioned within the clause is evaluated and the output of this evaluation is stored in a temporary relation. Following this, the main query associated with the WITH clause is finally executed that would use the temporary relation produced.

Queries

Example 1: Find all the employee whose salary is more than the average salary of all employees.

Name of the relation: **Employee**

EmployeeID	Name	Salary
100011	Smith	50000
100022	Bill	94000
100027	Sam	70550

100845	Walden	80000
115585	Erik	60000
1100070	Kate	69000

SQL Query:

```
WITH temporaryTable(averageValue) as
  (SELECT avg(Salary)
   from Employee)
  SELECT EmployeeID, Name, Salary
  FROM Employee, temporaryTable
  WHERE Employee.Salary > temporaryTable.averageValue;
```

Output:

EmployeeID	Name	Salary
100022	Bill	94000
100845	Walden	80000

Explanation: The average salary of all employees is 70591. Therefore, all employees whose salary is more than the obtained average lies in the output relation.

Example 2: Find all the airlines where the total salary of all pilots in that airline is more than the average of total salary of all pilots in the database.

Name of the relation: **Pilot**

EmployeeID	Airline	Name	Salary
70007	Airbus 380	Kim	60000
70002	Boeing	Laura	20000
10027	Airbus 380	Will	80050
10778	Airbus 380	Warren	80780
115585	Boeing	Smith	25000
114070	Airbus 380	Katy	78000

SQL Query:

```
WITH totalSalary(Airline, total) as
    (SELECT Airline, sum(Salary)
     FROM Pilot
     GROUP BY Airline),
airlineAverage(avgSalary) as
    (SELECT avg(Salary)
     FROM Pilot )
SELECT Airline
FROM totalSalary, airlineAverage
WHERE totalSalary.total > airlineAverage.avgSalary;
```

Output:

Airline
Airbus 380

Explanation: The total salary of all pilots of Airbus 380 = 298,830 and that of Boeing = 45000. Average salary of all pilots in the table Pilot = 57305. Since only the total salary of all pilots of Airbus 380 is greater than the average salary obtained, so Airbus 380 lies in the output relation.

Important Points:

- The SQL WITH clause is good when used with complex SQL statements rather than simple ones
- It also allows you to break down complex SQL queries into smaller ones which make it easy for debugging and processing the complex queries.
- The SQL WITH clause is basically a drop-in replacement to the normal sub-query.

This article is contributed by **Mayank Kumar**. If you like GeeksforGeeks and would like to contribute, you can also write an article using write.geeksforgeeks.org or mail your article to review-team@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Last Updated : 13 Aug, 2021

111

Similar Reads

1. Difference between Having clause and Group by clause



SQL | With Ties Clause



classyallrounder

[Read](#)[Discuss](#)[Courses](#)[Practice](#)

This post is a continuation of [SQL Offset-Fetch Clause](#)

Now, we understand that how to use the Fetch Clause in Oracle Database, along with the Specified Offset and we also understand that Fetch clause is the newly added clause in the Oracle Database 12c or it is the new feature added in the Oracle database 12c.

Now consider the below example:

Suppose we have a table named **myTable** with below data:

AD

ID	NAME	SALARY
<hr/>		
1	Geeks	10000
4	Finch	10000
2	RR	6000
3	Dhoni	16000
5	Karthik	7000
6	Watson	10000

Now, suppose we want the first three rows to be Ordered by Salary in descending order, then the below query must be executed:

Query:

```
SELECT * from myTable
order by salary desc
fetch first 3 rows only;
```

Output:

We got only first 3 rows order by Salary in Descending Order

ID	NAME	SALARY

3	Dhoni	16000
1	Geeks	10000
4	Finch	10000

Note: In the above result we got first 3 rows, ordered by Salary in Descending Order, but we have one more row with same salary i.e, the row with name **Watson** and Salary **10000**, but it didn't came up, because we restricted our output to first three rows only. But this is not optimal, because most of the time in live applications we will be required to display the tied rows also.

Real Life Example – Suppose we have 10 Racers running, and we have only 3 prizes i.e, first, second, third, but suppose, Racers 3 and 4 finished the race together in same time, so in this case we have a tie between 3 and 4 and that's why both are holder of Position 3.

With Ties

So, to overcome the above problem, Oracle introduces a clause known as **With Ties** clause. Now, let's see our previous example using With Ties clause.

Query:

```
SELECT * from myTable
order by salary desc
fetch first 3 rows With Ties;
```

Output:

See we get only first 3 rows order by Salary in Descending Order along with **Tied Row** also

ID	NAME	SALARY

3	Dhoni	16000
1	Geeks	10000
6	Watson	10000 // We get Tied Row also
4	Finch	10000

Now, see we got the **tied row** also, which we were not getting previously.

Note: We **get** the tied row in our output, only when we use the **order by** clause in our Select statement. Suppose, if we won't use order by clause, and still we are using **with ties** clause,

then we won't get the tied row in our output and the query behaves same as, if we are using **ONLY** clause instead of With Ties clause.

Example – Suppose we execute the below query(without using order by clause) :

Query:

```
SELECT * from myTable
fetch first 3 rows With Ties;
```

Output:

See we won't get the tied row because we didn't use order by clause

ID	NAME	SALARY
1	Geeks	10000
4	Finch	10000
2	RR	6000

In the above result we won't get the tied row and we get only first 3 rows. So **With Ties** is tied with **order by** clause, i.e, we get the tied row in output if and only if we use With Ties along with Order by clause.

Note: Please make sure that, you run these queries in Oracle Database 12c, because Fetch clause is the newly added feature in Oracle 12c, also With Ties, runs only in Oracle Database 12c, these queries **won't** run in below versions of 12c like 10g or 11g.

References: [About Fetch Clause as well as With Ties Clause](#), [Performing SQL Queries Online](#)
Last Updated : 21 Mar, 2018

33

Similar Reads

1. Difference between Having clause and Group by clause
-

2. SQL | Distinct Clause
-

3. SQL | WHERE Clause
-

4. Top Clause in Microsoft SQL Server
-

5. SQL | Union Clause
-

6. SQL | WITH clause
-

7. SQL | Except Clause
-

[Trending Now](#) [DSA](#) [Data Structures](#) [Algorithms](#) [Interview Preparation](#) [Data Science](#) [Topic-wise Practice](#) [Jobs](#)

SQL | OFFSET-FETCH Clause

[Read](#)[Discuss](#)[Courses](#)[Practice](#)

OFFSET and FETCH Clause are used in conjunction with SELECT and ORDER BY clause to provide a means to retrieve a range of records.

OFFSET

The OFFSET argument is used to identify the starting point to return rows from a result set. Basically, it exclude the first set of records.

Note:

- OFFSET can only be used with ORDER BY clause. It cannot be used on its own.
- OFFSET value must be greater than or equal to zero. It cannot be negative, else return error.

Syntax:

AD

```
SELECT column_name(s)
FROM table_name
WHERE condition
ORDER BY column_name
OFFSET rows_to_skip ROWS;
```

Examples:

Consider the following Employee table,

Fname	Lname	SSN	Salary	Super_ssn
John	Smith	123456789	30000	33344555
Franklin	Wong	333445555	40000	888665555
Joyce	English	453453453	80000	333445555
Ramesh	Narayan	666884444	38000	333445555
James	Borg	888665555	55000	NULL
Jennifer	Wallace	987654321	43000	88866555
Ahmad	Jabbar	987987987	25000	987654321
Alicia	Zeala	999887777	25000	987654321

- Print Fname, Lname of all the Employee except the employee having lowest salary.

```
SELECT Fname, Lname
FROM Employee
ORDER BY Salary
OFFSET 1 ROWS;
```

Output:

Fname	Lname
Alicia	Zeala
John	Smith
Ramesh	Narayan
Franklin	Wong
Jennifer	Wallace
James	Borg
Joyce	English

FETCH

The FETCH argument is used to return a set of number of rows. FETCH can't be used itself, it is used in conjunction with OFFSET.

Syntax:

```
SELECT column_name(s)
FROM table_name
ORDER BY column_name
OFFSET rows_to_skip
FETCH NEXT number_of_rows ROWS ONLY;
```

Example:

- Print the Fname, Lname from 3rd to 6th tuple of Employee table when sorted according to the Salary.

```
SELECT Fname, Lname
FROM Employee
ORDER BY Salary
OFFSET 2 ROWS
FETCH NEXT 4 ROWS ONLY;
```

Output:

Fname	Lname
John	Smith
Ramesh	Narayan
Franklin	Wong
Jennifer	Wallace

- Print the bottom 2 tuples of Employee table when sorted by Salary.

```
SELECT Fname, Lname
FROM Employee
ORDER BY Salary
OFFSET (SELECT COUNT(*) FROM EMPLOYEE) - 2 ROWS
FETCH NEXT 2 ROWS;
```

Output:

Fname	Lname
James	Borg
Joyce	English

Important Points:

1. OFFSET clause is mandatory with FETCH. You can never use, ORDER BY ... FETCH.
2. TOP cannot be combined with OFFSET and FETCH.
3. The OFFSET/FETCH row count expression can be only be any arithmetic, constant, or parameter expression which will return an integer value.
4. ORDER BY is mandatory to be used with OFFSET and FETCH clause.
5. OFFSET value must be greater than or equal to zero. It cannot be negative, else return error.

This article is contributed by **Anuj Chauhan**. If you like GeeksforGeeks and would like to contribute, you can also write an article using contribute.geeksforgeeks.org or mail your article to contribute@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Last Updated : 27 Dec, 2021

32

Similar Reads

1. [Offset-Fetch in MS SQL Server](#)
2. [SQL - TOP, LIMIT, FETCH FIRST Clause](#)
3. [Difference between Having clause and Group by clause](#)
4. [SQL Query to find the Nth Largest Value in a Column using Limit and Offset](#)
5. [FETCH in SQL](#)
6. [How to execute an SQL query and fetch results using PHP ?](#)
7. [SQL | Distinct Clause](#)
8. [SQL | WHERE Clause](#)
9. [Top Clause in Microsoft SQL Server](#)
10. [SQL | Union Clause](#)

Previous

Next

Article Contributed By :



SQL | Union Clause

[Read](#) [Discuss](#) [Courses](#) [Practice](#)

The Union Clause is used to combine two separate select statements and produce the result set as a union of both select statements.

NOTE:

1. The fields to be used in both the select statements must be in the same order, same number, and same data type.
2. The Union clause produces distinct values in the result set, to fetch the duplicate values too UNION ALL must be used instead of just UNION.

Syntax for UNION:

SELECT column_name(s) FROM table1

AD

UNION

SELECT column_name(s) FROM table2;

Syntax for UNION ALL:

The resultant set consists of distinct values.

SELECT column_name(s) FROM table1

UNION ALL

```
SELECT column_name(s) FROM table2;
```

The resultant set consists of duplicate values too.

Consider we have two tables name **Student** and **Student_Details**. Suppose we want to do a union operation to find the common roll no from both tables. Let's first create a table with the name student and insert some random data similarly, we will create another table with the name Student_details keeping in mind that there will be at least one column common here we will take roll_no as a common column.

CREATE :

```
CREATE TABLE students (
    roll_no INT,
    address VARCHAR(255),
    name VARCHAR(255),
    phone VARCHAR(20),
    age INT
);
INSERT INTO students (roll_no, address, name, phone, age)
VALUES
    (1, '123 Main St, Anytown USA', 'John Doe', '555-1234', 20),
    (2, '456 Oak St, Anytown USA', 'Jane Smith', '555-5678', 22),
    (3, '789 Maple St, Anytown USA', 'Bob Johnson', '555-9012', 19),
    (4, '234 Elm St, Anytown USA', 'Sarah Lee', '555-3456', 21),
    (5, '567 Pine St, Anytown USA', 'David Kim', '555-7890', 18);
```

Output:

roll_no	address	name	phone	age
1	123 Main St, Anytown USA	John Doe	555-1234	20
2	456 Oak St, Anytown USA	Jane Smith	555-5678	22
3	789 Maple St, Anytown USA	Bob Johnson	555-9012	19
4	234 Elm St, Anytown USA	Sarah Lee	555-3456	21
5	567 Pine St, Anytown USA	David Kim	555-7890	18

Let's create a second table with the name Student details here it will contain three columns roll_no, branch, and grade.

CREATE :

```

CREATE TABLE student_details (
    roll_no INT,
    branch VARCHAR(50),
    grade VARCHAR(2)
);
INSERT INTO student_details (roll_no, branch, grade)
VALUES
    (1, 'Computer Science', 'A'),
    (2, 'Electrical Engineering', 'B'),
    (3, 'Mechanical Engineering', 'C');

```

Output:

roll_no	branch	grade
1	Computer Science	A
2	Electrical Engineering	B
3	Mechanical Engineering	C

UNION Clause

To fetch distinct ROLL_NO from Student and Student_Details table.

Query:

```

SELECT ROLL_NO FROM Students UNION
SELECT ROLL_NO FROM Student_Details;

```

Output:

roll_no
1
2
3
4
5

The UNION ALL Clause

To fetch ROLL_NO from Student and Student_Details table including duplicate values.

Query:

```

SELECT ROLL_NO FROM Students UNION ALL

```

```
SELECT ROLL_NO FROM Student_Details;
```

Output:

roll_no
1
2
3
4
5
1
2
3

The UNION ALL Clause with Where Condition

To fetch ROLL_NO, NAME from Student table WHERE ROLL_NO is greater than 3 and ROLL_NO, Branch from Student_Details table WHERE ROLL_NO is less than 3, including duplicate values and finally sorting the data by ROLL_NO.

Query:

```
SELECT ROLL_NO,NAME FROM Students WHERE ROLL_NO>3
UNION ALL
SELECT ROLL_NO,Branch FROM Student_Details WHERE ROLL_NO<3
ORDER BY 1;
```

Output:

roll_no	name
1	Computer Science
2	Electrical Engineering
4	Sarah Lee
5	David Kim

Note: The column names in both the select statements can be different but the data type must be same. And in the result set the name of column used in the first select statement will appear.

This article is contributed by **Pratik Agarwal**. If you like GeeksforGeeks and would like to contribute, you can also write an article using [write.geeksforgeeks.org](https://www.geeksforgeeks.org/write-for-us/) or mail your article to review-team@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks. Please write comments if you find anything incorrect, or if you want to share more information about the topic discussed above.



Engineering Mathematics Discrete Mathematics Digital Logic and Design Computer Organization and Architecture

Top Clause in Microsoft SQL Server

Read Discuss Courses Practice

THE SELECT TOP clause is used to fetch a limited number of rows from a database. This clause is very useful while dealing with large databases. The top Clause will be useful for fetching the data records in larger datasets as it will drastically reduce the complexity.

Syntax

SELECT TOP value column1,column2 FROM table_name;

value: number of rows to return from top

AD

column1 , column2 fields in the table

table_name: name of table

Syntax Using Percent

SELECT TOP value PERCENT column1,column2 FROM table_name;

value: percentage of number of rows to return from top

column1 , column2: fields in the table

table_name: name of table

Parameter Explanation

1. **TOP:** Clause is used for fetching the top records from a huge dataset.

Lets us see examples for Top Clause in Microsoft SQL Server, for this we create a database.

Query:

```
CREATE TABLE Customer(
    CustomerID INT PRIMARY KEY,
    CustomerName VARCHAR(50),
    LastName VARCHAR(50),
    Country VARCHAR(50),
    Age int(2),
    Phone int(10)
);

-- Insert some sample data into the Customers table
INSERT INTO Customer (CustomerID, CustomerName, LastName, Country, Age, Phone)
VALUES (1, 'Shubham', 'Thakur', 'India', '23', 'xxxxxxxxxx'),
       (2, 'Aman ', 'Chopra', 'Australia', '21', 'xxxxxxxxxx'),
       (3, 'Naveen', 'Tulasi', 'Sri lanka', '24', 'xxxxxxxxxx'),
       (4, 'Aditya', 'Arpan', 'Austria', '21', 'xxxxxxxxxx'),
       (5, 'Nishant. Salchichas S.A.', 'Jain', 'Spain', '22', 'xxxxxxxxxx');
```

Output:

CustomerID	CustomerName	LastName	Country
Age	Phone		
1	Shubham	Thakur	India
2	Aman	Chopra	Australia
3	Naveen	Tulasi	Sri lanka
4	Aditya	Arpan	Austria
5	Nishant. Salchichas S.A.		Jain
22	98763		Spain

Query:

To fetch the first two data sets from the Customer table.

```
SELECT TOP 2 * FROM Customer;
```

Output

CustomerID	CustomerName	LastName	Country
Age	Phone		
1	Shubham Thakur	India	23 45267
2	Aman Chopra	Australia	21 43627

Add WHERE Clause in SQL Server

We can fetch data records by using a where clause with some condition was well.

Query:

```
SELECT TOP 1 * FROM Customers
WHERE Country='Spain';
```

Output:

CustomerID	CustomerName	LastName	Country
Age	Phone		
5	Nishant. Salchichas S.A.	Jain	Spain
22	98763		

Note:

To get the same functionality on MySQL and Oracle databases there is a bit of difference in the basic syntax;

- **For MySQL databases:**

```
SELECT column1,column2 FROM table_name LIMIT value;
column1 , column2: fields int the table
table_name: name of table
value: number of rows to return from top
```

- **For Oracle databases:**

```
SELECT column1,column2 FROM table_name WHERE ROWNUM <= value;
column1 , column2: fields int the table
table_name: name of table
value: number of rows to return from top
```

This article is contributed by **Pratik Agarwal**. If you like GeeksforGeeks and would like to contribute, you can also write an article using [write.geeksforgeeks.org](https://www.geeksforgeeks.org/write-for-us/) or mail your article to review-team@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page



SQL | CREATE DOMAIN

[Read](#)[Discuss](#)[Courses](#)[Practice](#)

Used in : Postgre sql

CREATE DOMAIN creates a new domain. A domain is essentially a data type with optional constraints (restrictions on the allowed set of values). The user who defines a domain becomes its owner.

Domains are useful for abstracting common constraints on fields into a single location for maintenance. For example, several tables might contain email address columns, all requiring the same CHECK constraint to verify the address syntax. Define a domain rather than setting up each table's constraint individually.

Examples:

```
CREATE DOMAIN CPI_DATA AS REAL CHECK  
(value >= 0 AND value <= 10);
```

Now CPI_DATA domain is created so, we can use this domain anywhere in any table of database as below :

AD

```
CREATE TABLE student(  
    sid char(9) PRIMARY KEY,  
    name varchar(30),  
    cpi CPI_DATA  
);
```

Every time cpi_data will check the constraint, when you add data in student table.

Example 1 :

```
Insert into student values (201501408,Raj,7.5);
```

This will not violate the property of cpi.

Example 2 :

```
Insert into student values (201501188,Dhaval,12);
```

ERROR. This will violate the property of cpi.

This article is contributed by **Dhavalkumar Prajapati**. If you like GeeksforGeeks and would like to contribute, you can also write an article using [contribute.geeksforgeeks.org](#) or mail your article to contribute@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Last Updated : 07 Sep, 2018

16

Similar Reads

1. [Extract domain of Email from table in SQL Server](#)
2. [Difference between Structured Query Language \(SQL\) and Transact-SQL \(T-SQL\)](#)
3. [Configure SQL Jobs in SQL Server using T-SQL](#)
4. [SQL | Procedures in PL/SQL](#)
5. [SQL | Difference between functions and stored procedures in PL/SQL](#)
6. [SQL SERVER – Input and Output Parameter For Dynamic SQL](#)
7. [Difference between SQL and T-SQL](#)
8. [SQL Server | Convert tables in T-SQL into XML](#)
9. [SQL SERVER | Bulk insert data from csv file using T-SQL command](#)
10. [SQL - SELECT from Multiple Tables with MS SQL Server](#)

Previous

Next

Article Contributed By :



SQL CREATE TABLE

[Read](#) [Discuss](#) [Courses](#) [Practice](#)

In the [SQL](#) database for creating a table, we use a command called **CREATE TABLE**.

SQL CREATE TABLE Statement

A **Table** is a combination of rows and columns. For creating a table we have to define the structure of a table by adding names to columns and providing data type and size of data to be stored in columns.

Syntax:

CREATE table table_name

AD

(

Column1 datatype (size),

column2 datatype (size),

.

columnN datatype(size)

);

Here **table_name** is name of the table, **column** is the name of column

SQL CREATE TABLE Example

Let us create a table to store data of Customers, so the table name is Customer, Columns are Name, Country, age, phone, and so on.

```
CREATE TABLE Customer(
    CustomerID INT PRIMARY KEY,
    CustomerName VARCHAR(50),
    LastName VARCHAR(50),
    Country VARCHAR(50),
    Age int(2),
    Phone int(10)
);
```

Output:

Customer					
CustomerID	CustomerName	LastName	Country	Age	Phone
empty					

Insert Data into Table

To add data to the table, we use INSERT INTO, the syntax is as shown below:

Syntax:

//Below query adds data in specific column, (like Column1=Value1)//

Insert into Table_name(Column1, Column2, Column3)

Values (Value1, value2, value3);

//Below query adds data in table in sequence of column name(Value1 will be added in Column1 and so on)//

Insert into Table_name

Values (Value1, value2, value3);

//Adding multiple data in the table in one go//

Insert into Table_name

Values (Value01, value02, value03),

(Value11, value12, value13),

(Value21, value22, value23),

(ValueN1, valueN2, valueN3)

Example Query

This query will add data in the table named Subject

```
-- Insert some sample data into the Customers table
INSERT INTO Customer (CustomerID, CustomerName, LastName, Country, Age, Phone)
VALUES (1, 'Shubham', 'Thakur', 'India', '23', 'xxxxxxxxxx'),
       (2, 'Aman ', 'Chopra', 'Australia', '21', 'xxxxxxxxxx'),
       (3, 'Naveen', 'Tulasi', 'Sri lanka', '24', 'xxxxxxxxxx'),
       (4, 'Aditya', 'Arpan', 'Austria', '21', 'xxxxxxxxxx'),
       (5, 'Nishant. Salchichas S.A.', 'Jain', 'Spain', '22', 'xxxxxxxxxx');
```

Output:

Customer

CustomerID	CustomerName	LastName	Country	Age	Phone
1	Shubham	Thakur	India	23	xxxxxxxxxx
2	Aman	Chopra	Australia	21	xxxxxxxxxx
3	Naveen	Tulasi	Sri lanka	24	xxxxxxxxxx
4	Aditya	Arpan	Austria	21	xxxxxxxxxx
5	Nishant. Salchichas S.A.	Jain	Spain	22	xxxxxxxxxx

Create a Table Using Another Table

We can also use CREATE TABLE to create a copy of an existing table. In the new table, it gets the exact column definition all columns or specific columns can be selected.

If an existing table was used to create a new table, by default the new table would be populated with the existing values from the old table.

Syntax:

CREATE TABLE new_table_name AS

SELECT column1, column2,...

FROM existing_table_name

WHERE ...;

Query:

```
CREATE TABLE SubTable AS  
SELECT CustomerID, CustomerName  
FROM customer;
```

Output:**SubTable**

CustomerID	CustomerName
1	Shubham
2	Aman
3	Naveen
4	Aditya

This article is contributed by **Saylli Walve**. If you like GeeksforGeeks and would like to contribute, you can also write an article using [write.geeksforgeeks.org](https://www.geeksforgeeks.org/write-for-us/) or mail your article to review-team@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks. Please write comments if you find anything incorrect, or if you want to share more information about the topic discussed above.

Last Updated : 05 Apr, 2023

178

Similar Reads

1. [How to Select All Records from One Table That Do Not Exist in Another Table in SQL?](#)
2. [SQL Query to Filter a Table using Another Table](#)
3. [Difference between Structured Query Language \(SQL\) and Transact-SQL \(T-SQL\)](#)
4. [Configure SQL Jobs in SQL Server using T-SQL](#)
5. [SQL vs NO SQL vs NEW SQL](#)
6. [SQL | Create Table Extension](#)
7. [How to Create a Table With a Foreign Key in SQL?](#)
8. [SQL Query to Create Table With a Primary Key](#)
9. [How to Create a Table With Multiple Foreign Keys in SQL?](#)

[Trending Now](#) [DSA](#) [Data Structures](#) [Algorithms](#) [Interview Preparation](#) [Data Science](#) [Topic-wise Practice](#) [Jobs](#)

SQL | DESCRIBE Statement

[Read](#) [Discuss](#) [Courses](#) [Practice](#)

Prerequisite: [SQL Create Clause](#)

As the name suggests, DESCRIBE is used to describe something. Since in a database, we have tables, that's why do we use **DESCRIBE** or **DESC**(both are the same) commands to describe the **structure** of a table.

Syntax:

DESCRIBE one;

AD

OR

DESC one;

Note: We can use either **DESCRIBE** or **DESC**(both are **Case Insensitive**). Suppose our table whose name is **one** has 4 columns named id, name, email, and age and all are of can **contain** null values.

Query:

```
CREATE TABLE users (
    id INT PRIMARY KEY,
    name VARCHAR(50),
    email VARCHAR(100),
    age INT
);
```

```
DESC users;
```

Output:

Field	Type	Null	Key	Default	Extra
id	int(11)	NO	PRI	NULL	
name	varchar(50)	YES		NULL	
email	varchar(100)	YES		NULL	
age	int(11)	YES		NULL	

Here, above on using **DESC** or either **DESCRIBE** we are able to see the **structure** of a table but **not** on the console tab, the structure of the table is shown in the **describe** tab of the Database System Software.

So **desc** or **describe** command shows the **structure** of the table which include the **name** of the column, the **data type** of the column and the **nullability** which means, that column can contain null values or not.

All of these features of the table are described at the time of **Creation** of the table.

Creating a Table or Defining the Structure of a Table

Query:

```
create table one
(
    id int not null,
    name char(25)
)
```

Here, we created a table whose name is **one** and its columns are **ID**, **NAME** and the **id** is of **not null** type i.e., we **can't** put null values in the **ID** column but we **can** put null values in the **NAME** column.

Demonstrate DESC

Step 1: Defining the structure of the table.

Creating a table:

```
create table one
(
    id int not null,
    name char(25),
```

```
city varchar2(25)
)
```

Step 2: Displaying the structure of the table:

Table:

DESC one

OR

DESCRIBE one

Output:

Field	Type	Null	Key	Default	Extra
id	int(11)	NO		NULL	
name	char(25)	YES		NULL	
city	varchar2	YES		NULL	

Note: Here above **ID** column is of **not null** type and rest 2 columns can contain null values.

Note: You have to execute the DESC command on your system software only, because this command won't run on any editor. Make sure to run this command on your own installed Database **only References:** [Oracle.com](#)

This article is contributed by **Rajat Rawat 4**. If you like GeeksforGeeks and would like to contribute, you can also write an article using [write.geeksforgeeks.org](#) or mail your article to review-team@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks. Please write comments if you find anything incorrect, or if you want to share more information about the topic discussed above.

Last Updated : 10 May, 2023

20

Similar Reads

1. Difference between Structured Query Language (SQL) and Transact-SQL (T-SQL)

2. Configure SQL Jobs in SQL Server using T-SQL

3. SQL INSERT INTO Statement

4. SQL | DELETE Statement

5. SQL | UPDATE Statement



SQL | ALTER (RENAME)

[Read](#)[Discuss](#)[Courses](#)[Practice](#)

Sometimes we may want to rename our table to give it a more relevant name. For this purpose, we can use **ALTER TABLE** to rename the name of the table. SQL ALTER TABLE is a command used to modify the structure of an existing table in a database.

Note: Syntax may vary in different databases.

Syntax(Oracle,MySQL,MariaDB):

ALTER TABLE table_name

RENAME TO new_table_name;

AD

Columns can also be given a new name with the use of **ALTER TABLE**.

Syntax(MySQL, Oracle):

ALTER TABLE table_name

RENAME COLUMN old_name TO new_name;

Syntax(MariaDB):

ALTER TABLE table_name

CHANGE COLUMN old_name TO new_name;

Query:

```
CREATE TABLE Student (
    id INT PRIMARY KEY,
    name VARCHAR(50),
    age INT,
    email VARCHAR(50),
    phone VARCHAR(20)
);
```

Let's insert some data and then perform ALTER operation to understand better bout alter command.

INSERT:

```
INSERT INTO Student (id, name, age, email, phone)
VALUES
(1, 'Amit', 20, 'amit@gmail.com', '9999999999'),
(2, 'Rahul', 22, 'rahul@yahoo.com', '8888888888'),
(3, 'Priya', 21, 'priya@hotmail.com', '7777777777'),
(4, 'Sonia', 23, 'sonia@gmail.com', '6666666666'),
(5, 'Kiran', 19, 'kiran@yahoo.com', '5555555555');
```

Output:

id	name	age	email	phone
1	Shubham	23	shubham@gmail.com	9999999999
2	Bhavika	21	bhavika@yahoo.com	8888888888
3	Aman	21	aman@hotmail.com	7777777777
4	Sonia	23	sonia@gmail.com	6666666666
5	Kiran	19	kiran@yahoo.com	5555555555

Example 1:

Change the name of column name to FIRST_NAME in table Student.

Syntax:

ALTER TABLE Student RENAME COLUMN NAME TO FIRST_NAME;

Query:

```
ALTER TABLE Student RENAME name TO FIRST_NAME;
```

Output:

id	FIRST_NAME	age	email	phone
1	Shubham	23	shubham@gmail.com	9999999999
2	Bhavika	21	bhavika@yahoo.com	8888888888
3	Aman	21	aman@hotmail.com	7777777777
4	Sonia	23	sonia@gmail.com	6666666666
5	Kiran	19	kiran@yahoo.com	5555555555

Change the name of the table Student to Student_Details.

Query:

```
ALTER TABLE Student RENAME TO Student_Details;
```

Output:**Student_Details**

Student_Details				
id	FIRST_NAME	age	email	phone
1	Shubham	23	shubham@gmail.com	9999999999
2	Bhavika	21	bhavika@yahoo.com	8888888888
3	Aman	21	aman@hotmail.com	7777777777
4	Sonia	23	sonia@gmail.com	6666666666
5	Kiran	19	kiran@yahoo.com	5555555555

To Add a New Column with ALTER TABLE

To add a new column to the existing table, we first need to select the table with ALTER TABLE command table_name, and then we will write the name of the new column and its datatype with ADD column_name datatype. Let's have a look below to understand better.

Syntax:

```
ALTER TABLE table_name
```

```
ADD column_name datatype;
```

Query:

```
ALTER TABLE Student ADD marks INT;
```

Output:

id	FIRST_NAME	age	email	phone	marks
1	Shubham	23	shubham@gmail.com	9999999999	
2	Bhavika	21	bhavika@yahoo.com	8888888888	
3	Aman	21	aman@hotmail.com	7777777777	
4	Sonia	23	sonia@gmail.com	6666666666	
5	Kiran	19	kiran@yahoo.com	5555555555	

This article is contributed by **Shubham Chaudhary**. If you like GeeksforGeeks and would like to contribute, you can also write an article using [write.geeksforgeeks.org](https://www.geeksforgeeks.org/write-for-us/) or mail your article to review-team@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.

Please write comments if you find anything incorrect, or if you want to share more information about the topic discussed above.

Last Updated : 04 May, 2023

70

Similar Reads

1. [SQL ALTER TABLE - ADD, DROP, MODIFY](#)

2. [Difference between ALTER and UPDATE Command in SQL](#)

3. [Create, Alter and Drop schema in MS SQL Server](#)

4. [Alter login in SQL Server](#)

5. [ALTER SCHEMA in SQL Server](#)

6. [SQL Query to Drop Foreign Key Constraint Using ALTER Command](#)

7. [SQL Query to Drop Unique Key Constraints Using ALTER Command](#)

8. [SQL Query to Add Foreign Key Constraints Using ALTER Command](#)

9. [SQL Query to Add Unique key Constraints Using ALTER Command](#)



SQL ALTER TABLE – ADD, DROP, MODIFY

[Read](#)[Discuss](#)[Courses](#)[Practice](#)

The **ALTER TABLE statement in SQL** is used to add, remove, or modify columns in an existing table. The ALTER TABLE statement is also used to add and remove various constraints on existing tables.

ALTER TABLE ADD Column Statement in SQL

ADD is used to add columns to the existing table. Sometimes we may require to add additional information, in that case, we do not require to create the whole database again, **ADD** comes to our rescue.

ALTER TABLE ADD Column Statement Syntax:

```
ALTER TABLE table_name ADD (Columnname_1 datatype,  
Columnname_2 datatype, ...Columnname_n datatype);
```

AD

The following SQL adds an “Email” column to the “Students” table:

ALTER TABLE ADD Column Statement Example:

```
ALTER TABLE Students  
ADD Email varchar(255);
```

ALTER TABLE DROP Column Statement

DROP COLUMN is used to drop columns in a table. Deleting the unwanted columns from the table.

ALTER TABLE DROP Column Statement Syntax:

ALTER TABLE table_name

DROP COLUMN column_name;

The following SQL drop an “Email” column to the “Students” table:

ALTER TABLE DROP Column Statement Example:

```
ALTER TABLE Students
```

```
DROP COLUMN Email;
```

ALTER TABLE MODIFY Column Statement in SQL

It is used to modify the existing columns in a table. Multiple columns can also be modified at once. *Syntax may vary slightly in different databases.

ALTER TABLE MODIFY Column Statement Syntax:

ALTER TABLE table_name

MODIFY column_name column_type;

ALTER TABLE MODIFY Column Statement Syntax(SQL Server):

ALTER TABLE table_name

ALTER COLUMN column_name column_type;

ALTER TABLE MODIFY Column Statement Example:

```
ALTER TABLE table_name
```

```
MODIFY COLUMN column_name datatype;
```

SQL ALTER TABLE Queries

Suppose there is a student database:

ROLL_NO	NAME
1	Ram
2	Abhi
3	Rahul
4	Tanu

To ADD 2 columns AGE and COURSE to table Student.

Query:

```
ALTER TABLE Student ADD
(AGE number(3), COURSE varchar(40));
```

Output:

ROLL_NO	NAME	AGE	COURSE
1	Ram		
2	Abhi		
3	Rahul		
4	Tanu		

MODIFY column COURSE in table Student.

Query:

```
ALTER TABLE Student
MODIFY COURSE varchar(20);
```

After running the above query the maximum size of the Course Column is reduced to 20 from 40.

DROP column COURSE in table Student.

Query:

```
ALTER TABLE Student
DROP COLUMN COURSE;
```

Output:

ROLL_NO	NAME	AGE
1	Ram	
2	Abhi	
3	Rahul	
4	Tanu	

This article is contributed by **Shubham Chaudhary**. If you like GeeksforGeeks and would like to contribute, you can also write an article using [write.geeksforgeeks.org](https://www.geeksforgeeks.org/write-for-us/) or mail your article to review-team@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks. Please write comments if you find anything incorrect, or if you want to share more information about the topic discussed above.

Last Updated : 05 Apr, 2023

77

Similar Reads

1. Create, Alter and Drop schema in MS SQL Server

2. SQL Query to Drop Foreign Key Constraint Using ALTER Command

3. SQL Query to Drop Unique Key Constraints Using ALTER Command

4. SQL Query to Add Foreign Key Constraints Using ALTER Command

5. SQL Query to Add Unique key Constraints Using ALTER Command

6. SQL | ALTER (RENAME)



SQL | UPDATE Statement

[Read](#) [Discuss](#) [Courses](#) [Practice](#)

The UPDATE statement in [SQL](#) is used to update the data of an existing table in the database. We can update single columns as well as multiple columns using the UPDATE statement as per our requirement.

In a very simple way, we can say that SQL commands(UPDATE and DELETE) are used to change the data that is already in the database. The SQL DELETE command uses a WHERE clause.

Syntax

UPDATE table_name SET column1 = value1, column2 = value2,...

AD

WHERE condition;

table_name: name of the table

column1: name of first , second, third column....

value1: new value for first, second, third column....

condition: condition to select the rows for which the values of columns needs to be updated.

Parameter Explanation

- 1. UPDATE:** Command is used to update the column value in the table.
- 2. WHERE:** Specifies the condition which we want to implement on the table.

Note: In the above query the **SET** statement is used to set new values to the particular column and the **WHERE** clause is used to select the rows for which the columns are needed to be updated. If we have not used the WHERE clause then the columns in all the rows will be updated. So the WHERE clause is used to choose the particular rows.

Let's see the SQL update statement with examples.

Query:

```
CREATE TABLE Customer(
    CustomerID INT PRIMARY KEY,
    CustomerName VARCHAR(50),
    LastName VARCHAR(50),
    Country VARCHAR(50),
    Age int(2),
    Phone int(10)
);

-- Insert some sample data into the Customers table
INSERT INTO Customer (CustomerID, CustomerName, LastName, Country, Age, Phone)
VALUES (1, 'Shubham', 'Thakur', 'India', '23', 'xxxxxxxxxx'),
       (2, 'Aman ', 'Chopra', 'Australia', '21', 'xxxxxxxxxx'),
       (3, 'Naveen', 'Tulasi', 'Sri lanka', '24', 'xxxxxxxxxx'),
       (4, 'Aditya', 'Arpan', 'Austria', '21', 'xxxxxxxxxx'),
       (5, 'Nishant. Salchichas S.A.', 'Jain', 'Spain', '22', 'xxxxxxxxxx');

Select * from Customer;
```

Output:

Customer

CustomerID	CustomerName	LastName	Country	Age	Phone
1	Shubham	Thakur	India	23	xxxxxxxxxx
2	Aman	Chopra	Australia	21	xxxxxxxxxx
3	Naveen	Tulasi	Sri lanka	24	xxxxxxxxxx
4	Aditya	Arpan	Austria	21	xxxxxxxxxx
5	Nishant. Salchichas S.A.	Jain	Spain	22	xxxxxxxxxx

Update Single Column

Update the column NAME and set the value to 'Nitin' in the rows where the Age is 22.

```
UPDATE Customer SET CustomerName
= 'Nitin' WHERE Age = 22;
```

Output:

Customer

CustomerID	CustomerName	LastName	Country	Age	Phone
1	Shubham	Thakur	India	23	xxxxxxxxxx
2	Aman	Chopra	Australia	21	xxxxxxxxxx
3	Naveen	Tulasi	Sri lanka	24	xxxxxxxxxx
4	Aditya	Arpan	Austria	21	xxxxxxxxxx
5	Nitin	Jain	Spain	22	xxxxxxxxxx

Updating Multiple Columns

Update the columns NAME to 'Satyam' and Country to 'USA' where CustomerID is 1.

```
UPDATE Customer SET CustomerName = 'Satyam',
Country = 'USA' WHERE CustomerID = 1;
```

Output:

Customer

CustomerID	CustomerName	LastName	Country	Age	Phone
1	Satyam	Thakur	USA	23	xxxxxxxxxx
2	Aman	Chopra	Australia	21	xxxxxxxxxx
3	Naveen	Tulasi	Sri lanka	24	xxxxxxxxxx
4	Aditya	Arpan	Austria	21	xxxxxxxxxx
5	Nitin	Jain	Spain	22	xxxxxxxxxx

Note: For updating multiple columns we have used comma(,) to separate the names and values of two columns.

Omitting WHERE Clause

If we omit the WHERE clause from the update query then all of the rows will get updated.

```
UPDATE Customer SET CustomerName = 'Shubham';
```

Output:

The table Customer will now look like this,

Customer

CustomerID	CustomerName	LastName	Country	Age	Phone
1	Shubham	Thakur	USA	23	xxxxxxxxxx
2	Shubham	Chopra	Australia	21	xxxxxxxxxx
3	Shubham	Tulasi	Sri Lanka	24	xxxxxxxxxx
4	Shubham	Arpan	Austria	21	xxxxxxxxxx
5	Shubham	Jain	Spain	22	xxxxxxxxxx

This article is contributed by **Harsh Agarwal**. If you like GeeksforGeeks and would like to contribute, you can also write an article using [write.geeksforgeeks.org](https://www.geeksforgeeks.org/write-for-us/) or mail your article to review-team@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks. Please write comments if you find anything incorrect, or if you want to share more information about the topic discussed above.

Last Updated : 05 Apr, 2023

63

Similar Reads

1. [How to Update Multiple Columns in Single Update Statement in SQL?](#)

2. [How to Update Two Tables in One Statement in SQL Server?](#)

3. [Difference between Deferred update and Immediate update](#)

4. [update conflicts with concurrent update](#)

5. [Difference between Structured Query Language \(SQL\) and Transact-SQL \(T-SQL\)](#)

6. [Configure SQL Jobs in SQL Server using T-SQL](#)

7. [SQL vs NO SQL vs NEW SQL](#)

8. [SQL INSERT INTO Statement](#)

9. [SQL | DELETE Statement](#)

10. [SQL | INSERT IGNORE Statement](#)

Previous

Next



SQL | DELETE Statement

[Read](#) [Discuss](#) [Courses](#) [Practice](#)

Pre-requisites: [SQL Commands](#)

Existing records in a table can be deleted using the SQL DELETE Statement. We can delete a single record or multiple records depending on the condition we specify in the WHERE clause.

Syntax:

DELETE FROM table_name WHERE some_condition;

AD

table_name: name of the table

Parameter Explanation

- **Some_condition:** condition to choose a particular record.
- **DELETE FROM table_name**(means we have to delete from table).

Note: We can delete single as well as multiple records depending on the condition we provide in the WHERE clause. If we omit the WHERE clause then all of the records will be deleted and the table will be empty.

The sample table is as follows:

GFG_Employees

id	name	email	department
1	Jessie	jessie23@gmail.com	Development
2	Praveen	praveen_dagger@yahoo.com	HR
3	Bisa	dragonBall@gmail.com	Sales
4	Rithvik	msvv@hotmail.com	IT
5	Suraj	srjsunny@gmail.com	Quality Assurance
6	Om	OmShukla@yahoo.com	IT
7	Naruto	uzumaki@konoha.com	Development

Deleting Single Record

Delete the rows where NAME = 'Rithvik'. This will delete only the fourth row.

Query:

```
DELETE FROM GFG_EMPL0yees WHERE NAME = 'Rithvik';
```

Output:

GFG_Employees

id	name	email	department
1	Jessie	jessie23@gmail.com	Development
2	Praveen	praveen_dagger@yahoo.com	HR
3	Bisa	dragonBall@gmail.com	Sales
5	Suraj	srjsunny@gmail.com	Quality Assurance
6	Om	OmShukla@yahoo.com	IT
7	Naruto	uzumaki@konoha.com	Development

Deleting Multiple Records

Delete the rows from the table GFG_EMPL0yees where the department is "Development". This will delete 2 rows(the first row and the seventh row).

Query:

```
DELETE FROM GFG_EMPLOyees
WHERE department = 'Development';
```

Output:

GFG_Employees

id	name	email	department
2	Praveen	praveen_dagger@yahoo.com	HR
3	Bisa	dragonBall@gmail.com	Sales
5	Suraj	srjsunny@gmail.com	Quality Assurance
6	Om	OmShukla@yahoo.com	IT

Delete All of the Records

There are two queries to do this as shown below,

Query:

```
DELETE FROM GFG_EMPLOyees;
```

Or

```
DELETE * FROM GFG_EMPLOyees;
```

Output:

All of the records in the table will be deleted, there are no records left to display. The table GFG_EMPLOyees will become empty!

GFG_Employees

id	name	email	department
empty			

Important Note: DELETE is a [DML](#) (Data Manipulation Language) command hence operation performed by DELETE can be rolled back or undone.

[SQL Quiz](#) This article is contributed by **Harsh Agarwal**. If you like GeeksforGeeks and would like to contribute, you can also write an article using [write.geeksforgeeks.org](https://www.geeksforgeeks.org/write-for-us/) or mail your article to review-team@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.



SQL INSERT INTO Statement

Read Discuss Courses Practice

The INSERT INTO statement of SQL is used to insert a new row/record in a table. There are two ways of using the SQL INSERT INTO statement for inserting rows.

SQL INSERT Query

1. Only Values

The first method is to specify only the value of data to be inserted without the column names.

INSERT INTO Syntax:

INSERT INTO table_name VALUES (value1, value2, value3);

table_name: name of the table. value1, value2

AD

value of first column, second column,... for the new record

Column Names And Values Both

In the second method we will specify both the columns which we want to fill and their corresponding values as shown below:

Insert Data in Specified Columns – Syntax:

INSERT INTO table_name (column1, column2, column3)

VALUES (value1, value2, value3); table_name:

name of the table.

column1: name of first column, second column .

value1, value2, value3 value of first column, second column,... for the new record

Suppose there is a Student database and we want to add values.

ROLL_NO	NAME	ADDRESS	PHONE	AGE
1	Ram	Delhi	xxxxxxxxxxxxxx	18
2	RAMESH	GURGAON	xxxxxxxxxxxxxx	18
3	SUJIT	ROHTAK	xxxxxxxxxxxxxx	20
4	SURESH	ROHTAK	xxxxxxxxxxxxxx	18
3	SUJIT	ROHTAK	xxxxxxxxxxxxxx	20
2	RAMESH	GURGAON	xxxxxxxxxxxxxx	18

Method 1 (Inserting only values) – SQL INSERT Query

If we want to insert only values then we use the following query:

Query:

```
INSERT INTO Student VALUES
('5','HARSH','WEST BENGAL',
'XXXXXXXXXX','19');
```

Output:

The table **Student** will now look like this:

ROLL_NO	NAME	ADDRESS	PHONE	Age
1	Ram	Delhi	XXXXXXXXXX	18
2	RAMESH	GURGAON	XXXXXXXXXX	18

ROLL_NO	NAME	ADDRESS	PHONE	Age
3	SUJIT	ROHTAK	XXXXXXXXXX	20
4	SURESH	Delhi	XXXXXXXXXX	18
3	SUJIT	ROHTAK	XXXXXXXXXX	20
2	RAMESH	GURGAON	XXXXXXXXXX	18
5	HARSH	WEST BENGAL	XXXXXXXXXX	19

Method 2 (Inserting values in only specified columns) – SQL INSERT INTO Statement

If we want to insert values in the specified columns then we use the following query:

Query:

```
INSERT INTO Student (ROLL_NO,
NAME, Age) VALUES ('5','PRATIK','19');
```

Output:

The table **Student** will now look like this:

ROLL_NO	NAME	ADDRESS	PHONE	Age
1	Ram	Delhi	XXXXXXXXXX	18
2	RAMESH	GURGAON	XXXXXXXXXX	18
3	SUJIT	ROHTAK	XXXXXXXXXX	20
4	SURESH	Delhi	XXXXXXXXXX	18
3	SUJIT	ROHTAK	XXXXXXXXXX	20
2	RAMESH	GURGAON	XXXXXXXXXX	18
5	PRATIK	null	null	19

Notice that the columns for which the values are not provided are filled by null. Which are the default values for those columns?

2. Using SELECT in INSERT INTO Statement

We can use the SELECT statement with INSERT INTO statement to copy rows from one table and insert them into another table. The use of this statement is similar to that of the INSERT INTO statement. The difference is that the SELECT statement is used here to select data from a different table. The different ways of using INSERT INTO SELECT statement are shown below:

Inserting all columns of a table – INSERT INTO SELECT Statement

We can copy all the data of a table and insert it into a different table.

Syntax:

*INSERT INTO first_table SELECT * FROM second_table;*

first_table: name of first table.

second_table: name of second table.

We have used the SELECT statement to copy the data from one table and the INSERT INTO statement to insert from a different table.

Inserting specific columns of a table – INSERT INTO SELECT Statement

We can copy only those columns of a table that we want to insert into a different table.

Syntax:

INSERT INTO first_table(names_of_columns1)

SELECT names_of_columns2 FROM second_table;

first_table: name of first table. second_table: name of second table.

names of columns1: name of columns separated by comma(,) for table 1.

names of columns2: name of columns separated by comma(,) for table 2.

We have used the SELECT statement to copy the data of the selected columns only from the second table and the INSERT INTO statement to insert in the first table.

Copying specific rows from a table – INSERT INTO SELECT Statement

We can copy specific rows from a table to insert into another table by using the WHERE clause with the SELECT statement. We have to provide appropriate conditions in the WHERE clause to select specific rows.

*INSERT INTO table1 SELECT * FROM table2 WHERE condition;*

first_table: name of first table.

second_table: name of second table.

condition: condition to select specific rows.

Suppose there is a LateralStudent database.

ROLL_NO	NAME	ADDRESS	PHONE	Age
7	SOUVIK	HYDERABAD	XXXXXXXXXX	18
8	NIRAJ	NOIDA	XXXXXXXXXX	19
9	SOMESH	ROHTAK	XXXXXXXXXX	20

Method 1 – (Inserting all rows and columns)

If we want to insert only values then we use the following query:

SQL INSERT INTO SELECT Query:

```
INSERT INTO Student
SELECT * FROM LateralStudent;
```

Output:

This query will insert all the data of the table LateralStudent in the table Student. The table Student will now look like this,

ROLL_NO	NAME	ADDRESS	PHONE	Age
1	Ram	Delhi	XXXXXXXXXX	18

ROLL_NO	NAME	ADDRESS	PHONE	Age
2	RAMESH	GURGAON	XXXXXXXXXX	18
3	SUJIT	ROHTAK	XXXXXXXXXX	20
4	SURESH	Delhi	XXXXXXXXXX	18
3	SUJIT	ROHTAK	XXXXXXXXXX	20
2	RAMESH	GURGAON	XXXXXXXXXX	18
7	SOUVIK	DUMDUM	XXXXXXXXXX	18
8	NIRAJ	NOIDA	XXXXXXXXXX	19
9	SOMESH	ROHTAK	XXXXXXXXXX	20

Method 2(Inserting specific columns)

If we want to insert values in the specified columns then we use the following query:

SQL INSERT INTO SELECT Query:

```
INSERT INTO Student(ROLL_NO,NAME,Age)
SELECT ROLL_NO, NAME, Age FROM LateralStudent;
```

Output:

This query will insert the data in the columns ROLL_NO, NAME, and Age of the table LateralStudent in the table Student and the remaining columns in the Student table will be filled by *null* which is the default value of the remaining columns. The table Student will now look like this,

ROLL_NO	NAME	ADDRESS	PHONE	Age
1	Ram	Delhi	XXXXXXXXXX	18
2	RAMESH	GURGAON	XXXXXXXXXX	18
3	SUJIT	ROHTAK	XXXXXXXXXX	20

ROLL_NO	NAME	ADDRESS	PHONE	Age
4	SURESH	Delhi	XXXXXXXXXX	18
3	SUJIT	ROHTAK	XXXXXXXXXX	20
2	RAMESH	GURGAON	XXXXXXXXXX	18
7	SOUVIK	null	null	18
8	NIRAJ	null	null	19
9	SOMESH	null	null	20

Select specific rows to insert:

```
INSERT INTO Student SELECT *
FROM LateralStudent WHERE Age = 18;
```

Output:

This query will select only the first row from table LateralStudent to insert into the table Student. The table Student will now look like this,

ROLL_NO	NAME	ADDRESS	PHONE	Age
1	Ram	Delhi	XXXXXXXXXX	18
2	RAMESH	GURGAON	XXXXXXXXXX	18
3	SUJIT	ROHTAK	XXXXXXXXXX	20
4	SURESH	Delhi	XXXXXXXXXX	18
3	SUJIT	ROHTAK	XXXXXXXXXX	20
2	RAMESH	GURGAON	XXXXXXXXXX	18
7	SOUVIK	DUMDUM	XXXXXXXXXX	18

To insert multiple rows in a table using Single SQL Statement:

Syntax:

```
INSERT INTO table_name(Column1,Column2,Column3,.....)  

VALUES (Value1, Value2,Value3,.....),  

(Value1, Value2,Value3,.....),  

(Value1, Value2,Value3,.....),  

.....;
```

Where,

- **table_name:** name of the table.
- **Column 1:** name of the first column, second column.
- **Values:** Value1, Value2, Value3: the value of the first column, second column.
- For each new row inserted, you need To provide Multiple lists of values where each list is separated by “,”. Every list of values corresponds to values to be inserted in each new row of the table. Values in the next list tell values to be inserted in the next Row of the table.

Example:

The following SQL statement inserts multiple rows in Student Table.

Query:

```
INSERT INTO STUDENT(ID, NAME,AGE,GRADE,CITY)
VALUES(1,"AMIT KUMAR",15,10,"DELHI"),
(2,"GAURI RAO",18,12,"BANGALORE"),
(3,"MANAV BHATT",17,11,"NEW DELHI"),
(4,"RIYA KAPOOR",10,5,"UDAIPUR");
```

Output:

Thus STUDENT Table will look like this:

ID	NAME	AGE	GRADE	CITY
1	AMIT KUMAR	15	10	DELHI
2	GAURI RAO	18	12	BANGALORE

ID	NAME	AGE	GRADE	CITY
3	MANAV BHATT	17	11	NEW DELHI
4	RIYA KAPOOR	10	5	UDAIPUR

This article is contributed by **Harsh Agarwal**. If you like GeeksforGeeks and would like to contribute, you can also write an article using [write.geeksforgeeks.org](https://www.geeksforgeeks.org/write-for-us/) or mail your article to review-team@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks. Please write comments if you find anything incorrect, or if you want to share more information about the topic discussed above.

Last Updated : 12 Apr, 2023

64

Similar Reads

1. [Insert multiple values into multiple tables using a single statement in SQL Server](#)

2. [Insert Into Select statement in MS SQL Server](#)

3. [SQL | INSERT IGNORE Statement](#)

4. [Insert statement in MS SQL Server](#)

5. [SQL SERVER | Bulk insert data from csv file using T-SQL command](#)

6. [SELECT INTO Statement in SQL](#)

7. [Difference between Structured Query Language \(SQL\) and Transact-SQL \(T-SQL\)](#)

8. [Configure SQL Jobs in SQL Server using T-SQL](#)

9. [SQL vs NO SQL vs NEW SQL](#)

10. [SQL Server | Convert tables in T-SQL into XML](#)

Previous

Next

Article Contributed By :



SQL | SELECT Query

Read Discuss Courses Practice

The SELECT Statement in SQL is used to retrieve or fetch data from a [database](#).

We can fetch either the entire table or according to some specified rules. The data returned is stored in a result table. This result table is also called the result set. With the SELECT clause of a SELECT command statement, we specify the columns that we want to be displayed in the query result and, optionally, which column headings we prefer to see above the result table.

The select clause is the first clause and is one of the last clauses of the select statement that the database server evaluates. The reason for this is that before we can determine what to include in the final result set, we need to know all of the possible columns that could be included in the final result set.

CREATE TABLE:

AD

```
CREATE TABLE Customer(
    CustomerID INT PRIMARY KEY,
    CustomerName VARCHAR(50),
    LastName VARCHAR(50),
    Country VARCHAR(50),
    Age int(2),
    Phone int(10)
);
-- Insert some sample data into the Customers table
INSERT INTO Customer (CustomerID, CustomerName, LastName, Country, Age, Phone)
VALUES (1, 'Shubham', 'Thakur', 'India', '23', 'xxxxxxxxxx'),
       (2, 'Aman ', 'Chopra', 'Australia', '21', 'xxxxxxxxxx'),
       (3, 'Naveen', 'Tulasi', 'Sri lanka', '24', 'xxxxxxxxxx'),
```

```
(4, 'Aditya', 'Arpan', 'Austria', '21', 'xxxxxxxxxx'),
(5, 'Nishant. Salchichas S.A.', 'Jain', 'Spain', '22', 'xxxxxxxxxx');
```

Output:

CustomerID	CustomerName	LastName	Country	Age	Phone
1	Shubham	Thakur	India	23	xxxxxxxxxx
2	Aman	Chopra	Australia	21	xxxxxxxxxx
3	Naveen	Tulasi	Sri lanka	24	xxxxxxxxxx
4	Aditya	Arpan	Austria	21	xxxxxxxxxx
5	Nishant. Salchichas S.A.	Jain	Spain	22	xxxxxxxxxx

To fetch any column in the table.

Syntax:

```
SELECT column1,column2 FROM table_name
column1 , column2: names of the fields of the table
table_name: from where we want to apply query
```

This query will return all the rows in the table with fields column1 and column2.

SELECT Statement in SQL

To fetch the entire table or all the fields in the table:

Syntax:

```
SELECT * FROM table_name;
— asterisks represent all attributes of the table
```

Query to fetch the fields CustomerName, LastName from the table Customer:

```
SELECT CustomerName, LastName FROM Customer;
```

Output:

CustomerName	LastName
Shubham	Thakur
Aman	Chopra
Naveen	Tulasi
Aditya	Arpan
Nishant. Salchichas S.A.	Jain

To fetch all the fields from the table Customer:

```
SELECT * FROM Customer;
```

Output:

CustomerID	CustomerName	LastName	Country	Age	Phone
1	Shubham	Thakur	India	23	xxxxxxxxxx
2	Aman	Chopra	Australia	21	xxxxxxxxxx
3	Naveen	Tulasi	Sri lanka	24	xxxxxxxxxx
4	Aditya	Arpan	Austria	21	xxxxxxxxxx
5	Nishant. Salchichas S.A.	Jain	Spain	22	xxxxxxxxxx

SELECT Statement with WHERE Clause

Suppose we want to see table values with specific conditions then Where Clause is used with select statement.

Query:

```
SELECT CustomerName FROM Customer where Age = '21';
```

Output:

CustomerName
Aman
Aditya

SQL SELECT Statement with GROUP BY Clause

Query:

```
SELECT COUNT (item), Customer_id FROM Orders GROUP BY order_id;
```

Output:

COUNT (item)	customer_id
1	4
1	4
1	3
1	1
1	2

SELECT Statement with HAVING Clause

Consider the following database for Having Clause:

Results		Messages				
	EmployeeId	Name	Gender	Salary	Department	Experience
1	1	Rachit	M	50000	Engineering	6 year
2	2	Shobit	M	37000	HR	3 year
3	3	Isha	F	56000	Sales	7 year
4	4	Devi	F	43000	Management	4 year
5	5	Akhil	M	90000	Engineering	15 year

Query:

```
SELECT Department, sum(Salary) as Salary
FROM employee
GROUP BY department
HAVING SUM(Salary) >= 50000;
```

Output:

Results Messages

	Department ▾	Salary ▾
1	Engineering	140000
2	Sales	56000

SELECT Statement with ORDER BY clause in SQL

Query:

```
SELECT * FROM Customer ORDER BY Age DESC;
```

Output:

CustomerID	CustomerName	LastName	Country	Age	Phone
3	Naveen	Tulasi	Sri lanka	24	xxxxxxxxxx
1	Shubham	Thakur	India	23	xxxxxxxxxx
5	Nishant. Salchichas S.A.	Jain	Spain	22	xxxxxxxxxx
2	Aman	Chopra	Australia	21	xxxxxxxxxx
4	Aditya	Arpan	Austria	21	xxxxxxxxxx

This article is contributed by **Harsh Agarwal**. If you like GeeksforGeeks and would like to contribute, you can also write an article using [write.geeksforgeeks.org](https://www.geeksforgeeks.org/write-for-us/) or mail your article to review-team@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.

Please write comments if you find anything incorrect, or if you want to share more information about the topic discussed above.

Last Updated : 07 May, 2023

81

Similar Reads



SQL | DROP, TRUNCATE

[Read](#) [Discuss](#) [Courses](#) [Practice](#)

SQL commands are broadly classified into two types [DDL](#), [DML](#) here we will be learning about DDL commands, and in DDL we will be learning about DROP and TRUNCATE in this article.

DROP

DROP is used to delete a whole [database](#) or just a table.

In this article, we will be learning about the DROP statement which destroys objects like an existing database, table, index, or view. A DROP statement in SQL removes a component from a relational database management system (RDBMS).

Syntax:

AD

DROP object object_name

Examples 1:

To Drop a table

```
DROP TABLE table_name;
```

table_name: Name of the table to be deleted.

Examples 2:

To Drop a database

```
DROP DATABASE database_name;
```

database_name: Name of the database to be deleted.

TRUNCATE

The major difference between TRUNCATE and DROP is that truncate is used to delete the data inside the table not the whole table.

TRUNCATE statement is a Data Definition Language (DDL) operation that is used to mark the extent of a table for deallocation (empty for reuse). The result of this operation quickly removes all data from a table, typically bypassing several integrity-enforcing mechanisms. It was officially introduced in the SQL:2008 standard. The TRUNCATE TABLE mytable statement is logically (though not physically) equivalent to the DELETE FROM mytable statement (without a WHERE clause).

Syntax:

```
TRUNCATE TABLE table_name;
```

table_name: Name of the table to be truncated.

DATABASE name – student_data

DROP vs TRUNCATE

- Truncate is normally ultra-fast and it's ideal for deleting data from a temporary table.
- Truncate preserves the structure of the table for future use, unlike drop table where the table is deleted with its full structure.
- Table or Database deletion using a DROP statement **cannot** be rolled back, so it must be used wisely.

Difference between DROP and TRUNCATE

DROP	TRUNCATE
In the drop table data and its definition is deleted with their full structure.	It preserves the structure of the table for further use exist but deletes all the data.
Drop is used to eliminate existing complications and fewer complications in the whole database from the table.	Truncate is used to eliminate the tuples from the table.

DROP	TRUNCATE
Integrity constraints get removed in the DROP command.	Integrity constraint doesn't get removed in the Truncate command.
Since the structure does not exist, the View of the table does not exist in the Drop command.	Since the structure exists, the View of the table exists in the Truncate command.
Drop query frees the table space complications from memory.	This query does not free the table space from memory.
It is slow as there are so many complications compared to the TRUNCATE command.	It is fast as compared to the DROP command as there are fewer complications.

let's consider the given database:

Student				
ROLL_NO	NAME	ADDRESS	PHONE	Age
1	Ram	Delhi	xxxxxxxxxx	18
2	RAMESH	GURGAON	xxxxxxxxxx	18
3	SUJIT	ROHTAK	xxxxxxxxxx	20
4	SURESH	Delhi	xxxxxxxxxx	18
3	SUJIT	ROHTAK	xxxxxxxxxx	20
2	RAMESH	GURGAON	xxxxxxxxxx	18

To delete the whole database

Query:

```
DROP DATABASE student_data;
```

After running the above query whole database will be deleted.

To truncate the Student_details table from the student_data database.

Query:

```
TRUNCATE TABLE Student_details;
```

After running the above query Student_details table will be truncated, i.e, the data will be deleted but the structure will remain in the memory for further operations.

This article is contributed by **Pratik Agarwal**. If you like GeeksforGeeks and would like to contribute, you can also write an article using write.geeksforgeeks.org or mail your article to review-team@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks. Please write comments if you find anything incorrect, or if you want to share more information about the topic discussed above.

Last Updated : 12 Jun, 2023

78

Similar Reads

1. [Difference between DROP and TRUNCATE in SQL](#)
2. [Difference between DELETE, DROP and TRUNCATE](#)
3. [Difference between Structured Query Language \(SQL\) and Transact-SQL \(T-SQL\)](#)
4. [Configure SQL Jobs in SQL Server using T-SQL](#)
5. [SQL vs NO SQL vs NEW SQL](#)
6. [SQL ALTER TABLE - ADD, DROP, MODIFY](#)
7. [Difference between DELETE and DROP in SQL](#)
8. [Create, Alter and Drop schema in MS SQL Server](#)
9. [Drop login in SQL Server](#)
10. [CREATE and DROP INDEX Statement in SQL](#)

Previous

Next

Article Contributed By :



GeeksforGeeks

Vote for difficulty

Current difficulty : [Easy](#)



SQL | INSERT IGNORE Statement

[Read](#)
[Discuss](#)
[Courses](#)
[Practice](#)

We know that a primary key of a table cannot be duplicated. For instance, the roll number of a student in the student table must always be distinct. Similarly, the EmployeeID is expected to be unique in an employee table. When we try to insert a tuple into a table where the primary key is repeated, it results in an error. However, with the `INSERT IGNORE` statement, we can prevent such errors from popping up, especially when inserting entries in bulk and such errors can interrupt the flow of insertion. Instead, only a warning is generated.

Cases where `INSERT IGNORE` avoids error

- Upon insertion of a duplicate key where the column must contain a PRIMARY KEY or UNIQUE constraint
- Upon insertion of NULL value where the column has a NOT NULL constraint.
- Upon insertion of a row to a partitioned table where the inserted values go against the partition format.

Example:

Say we have a relation, Employee.

AD

Employee Table:

EmployeeID	Name	City
15001	Aakash	Delhi
15003	Sahil	Bangalore

15010	John	Hyderabad
15008	Shelley	Delhi
15002	Ananya	Mumbai
15004	Sia	Pune

As we can notice, the entries are not sorted on the basis of their primary key, i.e. EmployeeID.

Sample Query:

```
INSERT IGNORE INTO Employee (EmployeeID, Name, City)
VALUES (15002, 'Ram', 'Mumbai');
```

Output:

No entry inserted.

Sample Query:

Inserting multiple records

When inserting multiple records at once, any that cannot be inserted will not be, but any that can will be:

```
INSERT IGNORE INTO Employee (EmployeeID, Name, City)
VALUES (15007, 'Shikha', 'Delhi'), (15002, 'Ram', 'Mumbai'), (15009, 'Sam',
'Ahmedabad');
```

Output:

The first and the last entries get inserted; the middle entry is simply ignored. No error is flashed.

Disadvantage

Most users do not prefer INSERT IGNORE over INSERT since some errors may slip unnoticed. This may cause inconsistencies in the table, thereby causing some tuples to not get inserted without the user having a chance to correct them. Hence, INSERT IGNORE must be used in very specific conditions.

This article is contributed by **Anannya Uberoi**. If you like GeeksforGeeks and would like to contribute, you can also write an article using contribute.geeksforgeeks.org or mail your article to contribute@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.



SQL | Case Statement

[Read](#)[Discuss](#)[Courses](#)[Practice](#)

Control statements form the heart of most languages since they control the execution of other sets of statements. These are also found in [SQL](#) and should be exploited for uses such as query filtering and query optimization by carefully selecting tuples that match our requirements.

In this article, we explore the Case-Switch statement in SQL. The CASE statement is SQL's way of handling if/then logic.

There can be two valid ways of going about the case-switch statements.

The first takes a variable called `case_value` and matches it with some `statement_list`.

AD

Syntax:

CASE case_value

WHEN when_value THEN statement_list

[WHEN when_value THEN statement_list] ...

[ELSE statement_list]

END CASE

The second considers a `search_condition` instead of variable equality and executes the `statement_list` accordingly.

Syntax:

CASE

WHEN search_condition THEN statement_list

[WHEN search_condition THEN statement_list] ...

[ELSE statement_list]

END CASE

Example:

CREATE TABLE:

Below is a selection from the “Customer” table in the sample database:

```
CREATE TABLE Customer(
    CustomerID INT PRIMARY KEY,
    CustomerName VARCHAR(50),
    LastName VARCHAR(50),
    Country VARCHAR(50),
    Age int(2),
    Phone int(10)
);
-- Insert some sample data into the Customers table
INSERT INTO Customer (CustomerID, CustomerName, LastName, Country, Age, Phone)
VALUES (1, 'Shubham', 'Thakur', 'India', '23', 'xxxxxxxxxx'),
       (2, 'Aman ', 'Chopra', 'Australia', '21', 'xxxxxxxxxx'),
       (3, 'Naveen', 'Tulasi', 'Sri lanka', '24', 'xxxxxxxxxx'),
       (4, 'Aditya', 'Arpan', 'Austria', '21', 'xxxxxxxxxx'),
       (5, 'Nishant. Salchichas S.A.', 'Jain', 'Spain', '22', 'xxxxxxxxxx');
```

Output:

CustomerID	CustomerName	LastName	Country	Age	Phone
1	Shubham	Thakur	India	23	xxxxxxxxxx
2	Aman	Chopra	Australia	21	xxxxxxxxxx
3	Naveen	Tulasi	Sri lanka	24	xxxxxxxxxx
4	Aditya	Arpan	Austria	21	xxxxxxxxxx
5	Nishant. Salchichas S.A.	Jain	Spain	22	xxxxxxxxxx

Adding Multiple Conditions to a CASE statement

Query:

By adding multiple conditions in SQL

```
SELECT CustomerName, Age,
CASE
    WHEN Age > 22 THEN 'The Age is greater than 20'
    WHEN Age = 21 THEN 'The Age is 21'
    ELSE 'The Age is over 30'
END AS AgeText
FROM Customer ;
```

Output:

CustomerName	Age	QuantityText
Shubham	23	The Age is greater than 20
Aman	21	The Age is 21
Naveen	24	The Age is greater than 20
Aditya	21	The Age is 21
Nishant. Salchichas S.A.	22	The Age is over 30

CASE Statement With ORDER BY Clause**Query:**

By using Order by Clause in SQL

```
SELECT CustomerName, Country
FROM Customer
ORDER BY
(CASE
    WHEN Country IS 'India' THEN Country
    ELSE Age
END);
```

Output:

CustomerName	Country
Aman	Australia
Aditya	Austria
Nishant. Salchichas S.A.	Spain
Naveen	Sri Lanka
Shubham	India

Some important points about CASE statements:

1. There should always be a SELECT in the case statement.
2. END ELSE is an optional component but WHEN THEN these cases must be included in the CASE statement.
3. We can make any conditional statement using any conditional operator (like WHERE) between WHEN and THEN. This includes stringing together multiple conditional statements using AND and OR.
4. We can include multiple WHEN statements and an ELSE statement to counter with unaddressed conditions.

Last Updated : 13 Apr, 2023

29

Similar Reads

1. Difference between Structured Query Language (SQL) and Transact-SQL (T-SQL)
2. Configure SQL Jobs in SQL Server using T-SQL
3. SQL INSERT INTO Statement
4. SQL | DELETE Statement
5. SQL | UPDATE Statement
6. SQL | INSERT IGNORE Statement
7. SQL | DESCRIBE Statement
8. SQL | MERGE Statement
9. MERGE Statement in SQL Explained
10. Delete statement in MS SQL Server

Previous

Next

Article Contributed By :



GeeksforGeeks



SQL | GROUP BY

[Read](#)[Discuss](#)[Courses](#)[Practice](#)[Video](#)

The GROUP BY Statement in [SQL](#) is used to arrange identical data into groups with the help of some functions. i.e. if a particular column has the same values in different rows then it will arrange these rows in a group.

Features

- GROUP BY clause is used with the [SELECT](#) statement.
- In the query, the GROUP BY clause is placed after the [WHERE](#) clause.
- In the query, the GROUP BY clause is placed before the [ORDER](#) BY clause if used.
- In the query, the Group BY clause is placed before the Having clause.
- Place condition in the [having clause](#).

Syntax:

SELECT column1, function_name(column2)

FROM table_name

AD

WHERE condition

GROUP BY column1, column2

ORDER BY column1, column2;

Explanation:

1. **function_name**: Name of the function used for example, SUM() , AVG().

2. **table_name**: Name of the table.

3. **condition**: Condition used.

Let's assume that we have two tables Employee and Student Sample Table is as follows after adding two tables we will do some specific operations to learn about GROUP BY.

Employee Table:

```
CREATE TABLE emp (
    emp_no INT PRIMARY KEY,
    name VARCHAR(50),
    sal DECIMAL(10,2),
    age INT
);
```

Insert some random data into a table and then we will perform some operations in GROUP BY.

Query:

```
INSERT INTO emp (emp_no, name, sal, age) VALUES
(1, 'Aarav', 50000.00, 25),
(2, 'Aditi', 60000.50, 30),
(3, 'Amit', 75000.75, 35),
(4, 'Anjali', 45000.25, 28),
(5, 'Chetan', 80000.00, 32),
(6, 'Divya', 65000.00, 27),
(7, 'Gaurav', 55000.50, 29),
(8, 'Isha', 72000.75, 31),
(9, 'Kavita', 48000.25, 26),
(10, 'Mohan', 83000.00, 33);
```

Output:

emp_no	name	sal	age
1	Aman	50000	25
2	Shubham	60000.5	30
3	Aditya	75000.75	35
4	Navin	45000.25	28
5	Chetan	80000	32
6	Divya	65000	27
7	Gaurav	55000.5	29
8	Isha	72000.75	31
9	Kavita	48000.25	26
10	Mohan	83000	33

Student Table:**Query:**

```
CREATE TABLE student (
    name VARCHAR(50),
    year INT,
    subject VARCHAR(50)
);
INSERT INTO student (name, year, subject) VALUES
('Alice', 1, 'Mathematics'),
('Bob', 2, 'English'),
('Charlie', 3, 'Science'),
('David', 1, 'History'),
('Emily', 2, 'Art'),
('Frank', 3, 'Computer Science');
```

Output:

name	year	subject
Alice	1	Mathematics
Bob	2	English
Charlie	3	Science
David	1	History
Emily	2	Art
Frank	3	Computer Science

Group By single column

Group By single column means, placing all the rows with the same value of only that particular column in one group. Consider the query as shown below:

Query:

```
SELECT NAME, SUM(SALARY) FROM emp
GROUP BY NAME;
```

The above query will produce the below output:

name	SUM(sal)
Aditya	75000.75
Aman	50000
Chetan	80000
Divya	65000
Gaurav	55000.5
Isha	72000.75
Kavita	48000.25
Mohan	83000
Navin	45000.25

As you can see in the above output, the rows with duplicate NAMES are grouped under the same NAME and their corresponding SALARY is the sum of the SALARY of duplicate rows. The SUM() function of SQL is used here to calculate the sum.

Group By Multiple Columns

Group by multiple columns is say, for example, **GROUP BY column1, column2**. This means placing all the rows with the same values of columns **column 1** and **column 2** in one group. Consider the below query:

Query:

```
SELECT SUBJECT, YEAR, Count(*)
FROM Student
GROUP BY SUBJECT, YEAR;
```

Output:

subject	year	Count(*)
Art	2	1
Computer Science	3	1
English	2	1
History	1	1
Mathematics	1	1
Science	3	1

Output: As you can see in the above output the students with both the same SUBJECT and YEAR are placed in the same group. And those whose only SUBJECT is the same but not YEAR belong to different groups. So here we have grouped the table according to two columns or more than one column.

HAVING Clause in GROUP BY Clause

We know that the WHERE clause is used to place conditions on columns but what if we want to place conditions on groups? This is where the HAVING clause comes into use. We can use the HAVING clause to place conditions to decide which group will be part of the final result set.

Also, we can not use aggregate functions like SUM(), COUNT(), etc. with the WHERE clause. So we have to use the HAVING clause if we want to use any of these functions in the conditions.

Syntax:

SELECT column1, function_name(column2)

FROM table_name

WHERE condition

GROUP BY column1, column2

HAVING condition

ORDER BY column1, column2;

Explanation:

1. **function_name**: Name of the function used for example, SUM() , AVG().
2. **table_name**: Name of the table.
3. **condition**: Condition used.

Example:

```
SELECT NAME, SUM(sal) FROM Emp
GROUP BY name
HAVING SUM(sal)>3000;
```

Output:

name	SUM(sal)
Aditya	75000.75
Aman	50000
Chetan	80000
Divya	65000
Gaurav	55000.5
Isha	72000.75
Kavita	48000.25
Mohan	83000
Navin	45000.25
...

As you can see in the above output only one group out of the three groups appears in the result set as it is the only group where sum of SALARY is greater than 3000. So we have used the HAVING clause here to place this condition as the condition is required to be placed on groups not columns.

This article is contributed by **Harsh Agarwal**. If you like GeeksforGeeks and would like to contribute, you can also write an article using [write.geeksforgeeks.org](https://www.geeksforgeeks.org/write-for-us/) or mail your article to review-team@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.

Please write comments if you find anything incorrect, or if you want to share more information about the topic discussed above.

Last Updated : 06 May, 2023

166

Similar Reads

1. [Difference between Structured Query Language \(SQL\) and Transact-SQL \(T-SQL\)](#)
2. [Configure SQL Jobs in SQL Server using T-SQL](#)
3. [SQL vs NO SQL vs NEW SQL](#)
4. [Difference between order by and group by clause in SQL](#)
5. [Group by clause in MS SQL Server](#)
6. [How to Group and Aggregate Data Using SQL?](#)
7. [SQL - Using GROUP BY to COUNT the Number of Rows For Each Unique Entry in a Column](#)
8. [SQL - count\(\) with Group By clause](#)
9. [How to Select Group of Rows that Match All Items on a List in SQL Server?](#)
10. [How to Select the First Row of Each GROUP BY in SQL?](#)

Previous

Next

Article Contributed By :



Vote for difficulty

Current difficulty : [Easy](#)



SQL – ORDER BY

Read Discuss Courses Practice

The ORDER BY statement in [SQL](#) is used to sort the fetched data in either ascending or descending according to one or more columns.

- By default ORDER BY sorts the data in ascending order.
- We can use the keyword DESC to sort the data in descending order and the keyword ASC to sort in ascending order.

Sort According To One Column

To sort in ascending or descending order we can use the keywords ASC or DESC respectively.

Syntax:

*SELECT * FROM table_name ORDER BY column_name ASC / DESC*

AD

//Where

table_name: name of the table.

column_name: name of the column according to which the data
is needed to be arranged.

ASC: to sort the data in ascending order.

DESC: to sort the data in descending order.

// : use either ASC or DESC to sort in ascending or descending order//

Sort According To Multiple Columns

To sort in ascending or descending order we can use the keywords ASC or DESC respectively.

To sort according to multiple columns, separate the names of columns by the (,) operator.

Syntax:

```
SELECT * FROM table_name ORDER BY column1 ASC/DESC , column2 ASC/DESC
```

Query:

```
CREATE TABLE students (
    roll_no INT NOT NULL,
    age INT NOT NULL,
    name VARCHAR(50) NOT NULL,
    address VARCHAR(100) NOT NULL,
    phone VARCHAR(20) NOT NULL,
    PRIMARY KEY (roll_no)
);
INSERT INTO students (roll_no, age, name, address, phone)
VALUES
    (1, 18, 'Shubham Thakur', '123 Main St, Mumbai', '9876543210'),
    (2, 19, 'Aman Chopra', '456 Park Ave, Delhi', '9876543211'),
    (3, 20, 'Naveen Tulasi', '789 Broadway, Ahmedabad', '9876543212'),
    (4, 21, 'Aditya arpan', '246 5th Ave, Kolkata', '9876543213'),
    (5, 22, 'Nishant Jain', '369 3rd St, Bengaluru', '9876543214');
```

Output:

roll_no	age	name	address	phone
1	18	Shubham Thakur	123 Main St, Mumbai	9876543210
2	19	Aman Chopra	456 Park Ave, Delhi	9876543211
3	20	Naveen Tulasi	789 Broadway, Ahmedabad	9876543212
4	21	Aditya arpan	246 5th Ave, Kolkata	9876543213
5	22	Nishant Jain	369 3rd St, Bengaluru	9876543214

Now consider the above database table and find the results of different queries.

Sort According To a Single Column

In this example, we will fetch all data from the table Student and sort the result in descending order according to the column ROLL_NO.

Query:

```
SELECT * FROM students ORDER BY ROLL_NO DESC;
```

Output:

roll_no	age	name	address	phone
5	22	Nishant Jain	369 3rd St, Bengaluru	9876543214
4	21	Aditya arpan	246 5th Ave, Kolkata	9876543213
3	20	Naveen Tulasi	789 Broadway, Ahmedabad	9876543212
2	19	Aman Chopra	456 Park Ave, Delhi	9876543211
1	18	Shubham Thakur	123 Main St, Mumbai	9876543210

In the above example, if we want to sort in ascending order we have to use ASC in place of DESC.

Sort According To Multiple Columns

In this example, we will fetch all data from the table Student and then sort the result in ascending order first according to the column Age. and then in descending order according to the column ROLL_NO.

Query:

```
SELECT * FROM students ORDER BY Age ASC , ROLL_NO DESC;
```

Output:

roll_no	age	name	address	phone
1	18	Shubham Thakur	123 Main St, Mumbai	9876543210
2	19	Aman Chopra	456 Park Ave, Delhi	9876543211
3	20	Naveen Tulasi	789 Broadway, Ahmedabad	9876543212
4	21	Aditya arpan	246 5th Ave, Kolkata	9876543213
5	22	Nishant Jain	369 3rd St, Bengaluru	9876543214

In the above output, we can see that first the result is sorted in ascending order according to Age. There are multiple rows of having the same Age. Now, sorting further this result-set according to ROLL_NO will sort the rows with the same Age according to ROLL_NO in descending order.

Note:

ASC is the default value for the ORDER BY clause. So, if we don't specify anything after the column name in the ORDER BY clause, the output will be sorted in ascending order by default.

Take another example of the following query that will give similar output as the above:

Query:

```
SELECT * FROM students ORDER BY Age , ROLL_NO DESC;
```

Output:

roll_no	age	name	address	phone
1	18	Shubham Thakur	123 Main St, Mumbai	9876543210
2	19	Aman Chopra	456 Park Ave, Delhi	9876543211
3	20	Naveen Tulasi	789 Broadway, Ahmedabad	9876543212
4	21	Aditya arpan	246 5th Ave, Kolkata	9876543213
5	22	Nishant Jain	369 3rd St, Bengaluru	9876543214

Sorting By Column Number (instead of name)

An integer that identifies the number of the column in the SelectItems in the underlying query of the [SELECT statement](#). Column number must be greater than 0 and not greater than the number of columns in the result table. In other words, if we want to order by a column, that column must be specified in the SELECT list.

The rule checks for ORDER BY clauses that reference select list columns using the column number instead of the column name. The column numbers in the ORDER BY clause impair the readability of the SQL statement. Further, changing the order of columns in the SELECT list has no impact on the ORDER BY when the columns are referred to by names instead of numbers.

Syntax:

Order by Column_Number asc/desc

Here we take an example to sort a database table according to column 1 i.e Roll_Number. For this a query will be:

Query:

```
CREATE TABLE studentinfo
( Roll_no INT,
NAME VARCHAR(25),
Address VARCHAR(20),
CONTACTNO BIGINT NOT NULL,
Age INT );

INSERT INTO studentinfo
VALUES (7,'ROHIT','GHAZIABAD',9193458625,18),
(4,'DEEP','RAMNAGAR',9193458546,18),
(1,'HARSH','DELHI',9193342625,18),
```

```
(8, 'NIRAJ', 'ALIPUR', 9193678625, 19),
(5, 'SAPTARHI', 'KOLKATA', 9193789625, 19),
(2, 'PRATIK', 'BIHAR', 9193457825, 19),
(6, 'DHANRAJ', 'BARABAJAR', 9193358625, 20),
(3, 'RIYANKA', 'SILIGURI', 9193218625, 20);
```

```
SELECT Name, Address
FROM studentinfo
ORDER BY 1
```

Output:

NAME	Address
DEEP	RAMNAGAR
DHANRAJ	BARABAJAR
HARSH	DELHI
NIRAJ	ALIPUR
PRATIK	BIHAR
RIYANKA	SILIGURI
ROHIT	GHAZIABAD
SAPTARHI	KOLKATA

Last Updated : 03 May, 2023

47

Similar Reads

1. Difference between Structured Query Language (SQL) and Transact-SQL (T-SQL)

2. Configure SQL Jobs in SQL Server using T-SQL

3. SQL vs NO SQL vs NEW SQL

4. SQL | Procedures in PL/SQL

5. SQL | Difference between functions and stored procedures in PL/SQL

6. SQL SERVER – Input and Output Parameter For Dynamic SQL

7. Difference between T-SQL and PL-SQL

8. Difference between SQL and T-SQL

9. SQL Server | Convert tables in T-SQL into XML



SQL HAVING Clause with Examples



xishaapatel

[Read](#) [Discuss](#) [Courses](#) [Practice](#) [Video](#)

The HAVING clause was introduced in SQL to allow the filtering of query results based on aggregate functions and groupings, which cannot be achieved using the [WHERE](#) clause that is used to filter individual rows.

In simpler terms MSSQL, the HAVING clause is used to apply a filter on the result of [GROUP BY](#) based on the specified condition. The conditions are Boolean type i.e. *use of logical operators (AND, OR)*. This clause was included in SQL as the [WHERE](#) keyword failed when we use it with aggregate expressions. Having is a very generally used clause in [SQL](#). Similar to WHERE it helps to apply conditions, but HAVING works with groups. If you wish to filter a group, the HAVING clause comes into action.

Some important points:

- Having clause is used to filter data according to the conditions provided.
- Having a clause is generally used in reports of large data.
- Having clause is only used with the SELECT clause.
- The expression in the syntax can only have constants.
- In the query, [ORDER BY](#) is to be placed after the HAVING clause, if any.
- HAVING Clause is implemented in column operation.
- Having clause is generally used after [GROUP BY](#).

Syntax:

AD

```
SELECT col_1, function_name(col_2)
```

FROM tablename

WHERE condition

GROUP BY column1, column2

HAVING Condition

ORDER BY column1, column2;

Here, the function_name is the name of the function used, for example, SUM(), and AVG().

Example 1:

Here first we create a database name as “Company”, then we will create a table named “Employee” in the database. After creating a table we will execute the query.

Step 1: Creating a database

```
CREATE DATABASE Company;
```

Step 2: To use this database

```
USE Company;
```

Step 3: Creating a table

```
CREATE TABLE Employee (
    EmployeeId int,
    Name varchar(20),
    Gender varchar(20),
    Salary int,
    Department varchar(20),
    Experience varchar(20)
);
```

Add value to the table:

```
INSERT INTO Employee (EmployeeId, Name, Gender, Salary, Department, Experience)
VALUES (5, 'Priya Sharma', 'Female', 45000, 'IT', '2 years');
```

```
INSERT INTO Employee (EmployeeId, Name, Gender, Salary, Department, Experience)
VALUES (6, 'Rahul Patel', 'Male', 65000, 'Sales', '5 years');
```

```
INSERT INTO Employee (EmployeeId, Name, Gender, Salary, Department, Experience)
VALUES (7, 'Nisha Gupta', 'Female', 55000, 'Marketing', '4 years');
```

```
INSERT INTO Employee (EmployeeId, Name, Gender, Salary, Department, Experience)
VALUES (8, 'Vikram Singh', 'Male', 75000, 'Finance', '7 years');

INSERT INTO Employee (EmployeeId, Name, Gender, Salary, Department, Experience)
VALUES (9, 'Aarti Desai', 'Female', 50000, 'IT', '3 years');
```

The final table is:

```
SELECT * FROM Employee;
```

Output:

EmployeeId	Name	Gender	Salary	Department	Experience
5	Priya Sharma	Female	45000	IT	2 years
6	Rahul Patel	Male	65000	Sales	5 years
7	Nisha Gupta	Female	55000	Marketing	4 years
8	Vikram Singh	Male	75000	Finance	7 years
9	Aarti Desai	Female	50000	IT	3 years

Step 4: Execute the query

This employee table will help us understand the HAVING Clause. It contains employee IDs, Name, Gender, department, and salary. To Know the sum of salaries, we will write the query:

```
SELECT Department, sum(Salary) as Salary
FROM employee
GROUP BY department;
```

Here is the result,

Output

Department	Salary
Finance	75000
IT	95000
Marketing	55000
Sales	65000

Now if we need to display the departments where the sum of salaries is 50,000 or more. In this condition, we will use the HAVING Clause.

```
SELECT Department, sum(Salary) as Salary
FROM employee
GROUP BY department
HAVING SUM(Salary) >= 50000;
```

Output:

Department	Salary
Finance	75000
IT	95000
Marketing	55000
Sales	65000

Example 2:

Suppose, a teacher wants to announce the toppers in class. For this, she decides to reward every student who scored more than 95%. We need to group the database by name and their percentage and find out who scored more than 95% in that year. So for this first, we create a database name “School”, and then we will create a table named “Student” in the database. After creating a table we will execute the query.

Step 1: Creating a database

```
CREATE DATABASE School;
```

Step 2: To use this database

```
USE School;
```

Step 3: Creating a table

```
CREATE TABLE Student(
    student Varchar(20),
    percentage int
);
```

Add value to the table:

```
INSERT INTO Student VALUES ('Isha Patel', 98)
INSERT INTO Student VALUES ('Harsh Das', 94)
INSERT INTO Student VALUES ('Rachit Sha', 93)
INSERT INTO Student VALUES ('Sumedha', 98)
INSERT INTO Student VALUES ('Rahat Ali', 98)
```

The final table is:

```
SELECT * FROM Student;
```

Output:

	student	percentage
1	Isha Patel	98
2	Harsh Das	94
3	Rachit Sha	93
4	Sumedha	98
5	Rahat Ali	98

Step 4: Execute Query

```
SELECT student, percentage
FROM Student
GROUP BY student, percentage
HAVING percentage > 95;
```

Here, three students named Isha, Sumedha, and Rahat Ali scored more than 95 %.

Output:

	student	percentage
1	Isha Patel	98
2	Rahat Ali	98
3	Sumedha	98

Further, we can also filter rows on multiple values using the HAVING clause. The HAVING clause also permits filtering rows using more than one aggregate condition.

Query:

```
SELECT student
FROM Student
WHERE percentage > 90
GROUP BY student, percentage
HAVING SUM(percentage) < 1000 AND AVG(percentage) > 95;
```

This query returns the students who have more percentage than 95 and the sum of percentage is less than 1000.

Output:

Results	Messages
	student ▾
1	Isha Patel
2	Rahat Ali
3	Sumedha

SQL Having vs WHERE

Having	Where
In the HAVING clause it will check the condition in group of a row.	In the WHERE condition it will check or execute at each row individual.
HAVING clause can only be used with aggregate function.	The WHERE Clause cannot be used with aggregate function like Having
Priority Wise HAVING Clause is executed after Group By.	Priority Wise WHERE is executed before Group By.

Last Updated : 18 Apr, 2023

12

Similar Reads

1. Difference between Having clause and Group by clause

2. SQL query using COUNT and HAVING clause

3. Difference between Where and Having Clause in SQL



SQL | Join (Inner, Left, Right and Full Joins)

[Read](#)
[Discuss](#)
[Courses](#)
[Practice](#)
[Video](#)

SQL Join statement is used to combine data or rows from two or more tables based on a common field between them. Different types of Joins are as follows:

- INNER JOIN
- LEFT JOIN
- RIGHT JOIN
- FULL JOIN
- NATURAL JOIN

Consider the two tables below as follows:

Student

ROLL_NO	NAME	ADDRESS	PHONE	Age
1	HARSH	DELHI	XXXXXXXXXX	18
2	PRATIK	BIHAR	XXXXXXXXXX	19
3	RIYANKA	SILIGURI	XXXXXXXXXX	20
4	DEEP	RAMNAGAR	XXXXXXXXXX	18
5	SAPTARHI	KOLKATA	XXXXXXXXXX	19
6	DHANRAJ	BARABAJAR	XXXXXXXXXX	20
7	ROHIT	BALURGHAT	XXXXXXXXXX	18
8	NIRAJ	ALIPUR	XXXXXXXXXX	19

AD

StudentCourse

COURSE_ID	ROLL_NO
1	1
2	2
2	3
3	4
1	5
4	9
5	10
4	11

The simplest Join is INNER JOIN.

A. INNER JOIN

The INNER JOIN keyword selects all rows from both the tables as long as the condition is satisfied. This keyword will create the result-set by combining all rows from both the tables where the condition satisfies i.e value of the common field will be the same.

Syntax:

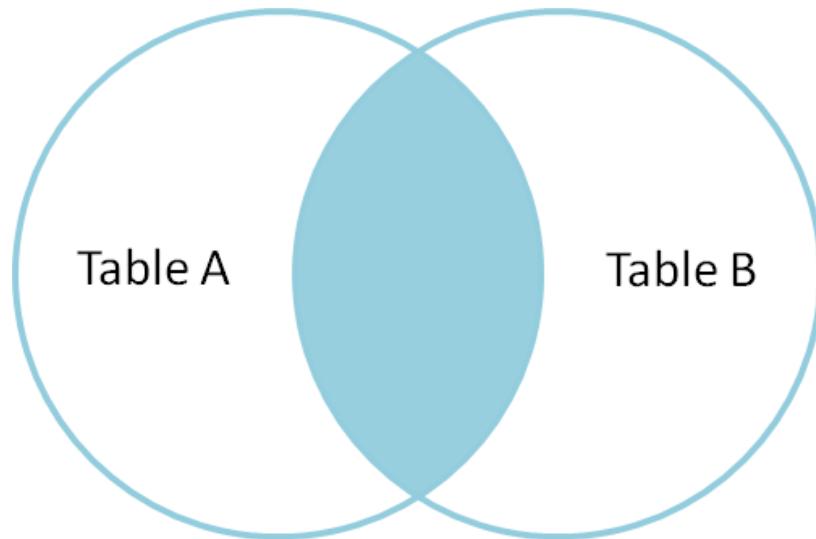
```
SELECT table1.column1,table1.column2,table2.column1,.....
FROM table1
INNER JOIN table2
ON table1.matching_column = table2.matching_column;
```

table1: First table.

table2: Second table

matching_column: Column common to both the tables.

Note: We can also write JOIN instead of INNER JOIN. JOIN is same as INNER JOIN.



Example Queries(INNER JOIN)

This query will show the names and age of students enrolled in different courses.

```
SELECT StudentCourse.COURSE_ID, Student.NAME, Student.AGE FROM Student
INNER JOIN StudentCourse
ON Student.ROLL_NO = StudentCourse.ROLL_NO;
```

Output:

COURSE_ID	NAME	Age
1	HARSH	18
2	PRATIK	19
2	RIYANKA	20
3	DEEP	18
1	SAPTARHI	19

B. LEFT JOIN

This join returns all the rows of the table on the left side of the join and matches rows for the table on the right side of the join. For the rows for which there is no matching row on the right side, the result-set will contain *null*. LEFT JOIN is also known as LEFT OUTER JOIN.

Syntax:

```
SELECT table1.column1,table1.column2,table2.column1,....
FROM table1
```

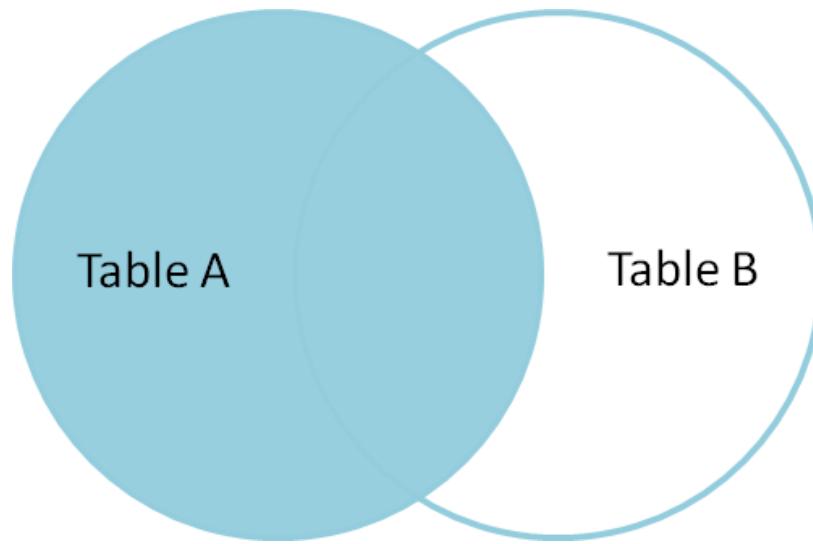
```
LEFT JOIN table2  
ON table1.matching_column = table2.matching_column;
```

table1: First table.

table2: Second table

matching_column: Column common to both the tables.

Note: We can also use LEFT OUTER JOIN instead of LEFT JOIN, both are the same.



Example Queries(LEFT JOIN):

```
SELECT Student.NAME,StudentCourse.COURSE_ID  
FROM Student  
LEFT JOIN StudentCourse  
ON StudentCourse.ROLL_NO = Student.ROLL_NO;
```

Output:

NAME	COURSE_ID
HARSH	1
PRATIK	2
RIYANKA	2
DEEP	3
SAPTARHI	1
DHANRAJ	NULL
ROHIT	NULL
NIRAJ	NULL

C. RIGHT JOIN

RIGHT JOIN is similar to LEFT JOIN. This join returns all the rows of the table on the right side of the join and matching rows for the table on the left side of the join. For the rows for which there is no matching row on the left side, the result-set will contain *null*. RIGHT JOIN is also known as RIGHT OUTER JOIN.

Syntax:

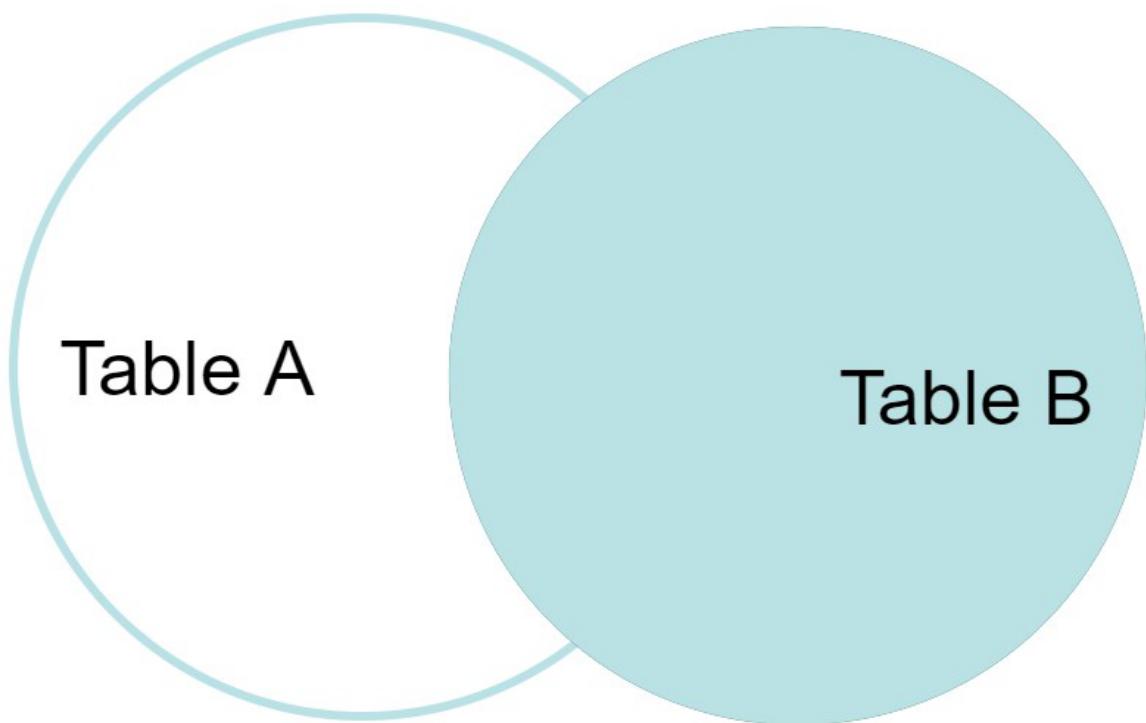
```
SELECT table1.column1,table1.column2,table2.column1,.....
FROM table1
RIGHT JOIN table2
ON table1.matching_column = table2.matching_column;
```

table1: First table.

table2: Second table

matching_column: Column common to both the tables.

Note: We can also use RIGHT OUTER JOIN instead of RIGHT JOIN, both are the same.



Example Queries(RIGHT JOIN):

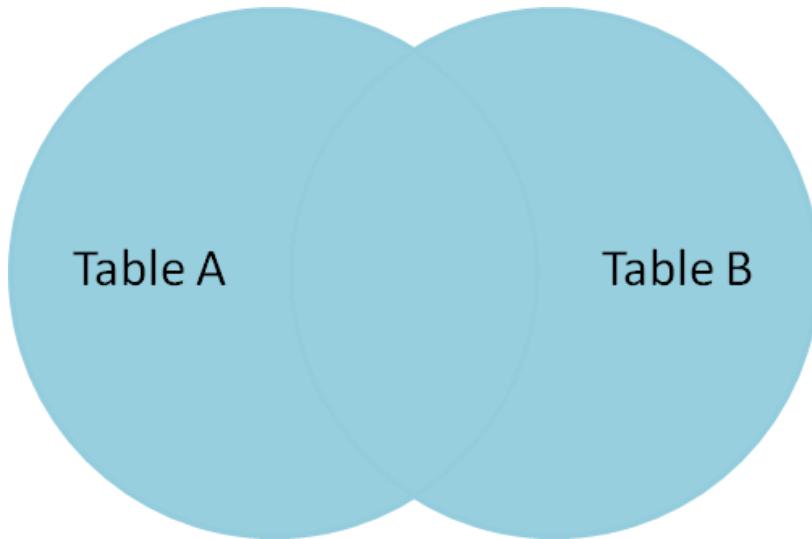
```
SELECT Student.NAME,StudentCourse.COURSE_ID  
FROM Student  
RIGHT JOIN StudentCourse  
ON StudentCourse.ROLL_NO = Student.ROLL_NO;
```

Output:

NAME	COURSE_ID
HARSH	1
PRATIK	2
RIYANKA	2
DEEP	3
SAPTARHI	1
NULL	4
NULL	5
NULL	4

D. FULL JOIN

FULL JOIN creates the result-set by combining results of both **LEFT JOIN** and **RIGHT JOIN**. The result-set will contain all the rows from both tables. For the rows for which there is no matching, the result-set will contain **NULL** values.



Syntax:

```
SELECT table1.column1,table1.column2,table2.column1,....  
FROM table1  
FULL JOIN table2  
ON table1.matching_column = table2.matching_column;
```

table1: First table.

table2: Second table

matching_column: Column common to both the tables.

Example Queries(FULL JOIN):

```
SELECT Student.NAME,StudentCourse.COURSE_ID  
FROM Student  
FULL JOIN StudentCourse  
ON StudentCourse.ROLL_NO = Student.ROLL_NO;
```

Output:

NAME	COURSE_ID
HARSH	1
PRATIK	2

NAME	COURSE_ID
RIYANKA	2
DEEP	3
SAPTARHI	1
DHANRAJ	NULL
ROHIT	NULL
NIRAJ	NULL
NULL	4
NULL	5
NULL	4

[Left JOIN \(Video\)](#)

[Right JOIN \(Video\)](#)

[Full JOIN \(Video\)](#)

[SQL JOIN \(Cartesian Join, Self Join\)](#)

E. Natural join (\bowtie)

Natural join can join tables based on the common columns in the tables being joined. A natural join returns all rows by matching values in common columns having same name and data type of columns and that column should be present in both tables.

Both table must have at list one common column with same column name and same data type.

The two table are joined using Cross join.

DBMS will look for a common column with same name and data type Tuples having exactly same values in common columns are kept in result.

Example:

Employee		
Emp_id	Emp_name	Dept_id
1	Ram	10
2	Jon	30
3	Bob	50

Department	
Dept_id	Dept_name
10	IT
30	HR
40	TIS

Query: Find all Employees and their respective departments.

Solution: (Employee) \bowtie (Department)

Emp_id	Emp_name	Dept_id	Dept_id	Dept_name
1	Ram	10	10	IT
2	Jon	30	30	HR
Employee data			Department data	

This article is contributed by [Harsh Agarwal](#). If you like GeeksforGeeks and would like to contribute, you can also write an article using [write.geeksforgeeks.org](https://www.geeksforgeeks.org/write-for-us/) or mail your article to review-team@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.

Please write comments if you find anything incorrect, or if you want to share more information about the topic discussed above.

[Trending Now](#) [DSA](#) [Data Structures](#) [Algorithms](#) [Interview Preparation](#) [Data Science](#) [Topic-wise Practice](#) [Jobs](#)

SQL Inner Join

 akshatsachan[Read](#)[Discuss](#)[Courses](#)[Practice](#)[Video](#)

Overview :

Structured Query Language or [SQL](#) is a standard Database language that is used to create, maintain and retrieve the data from relational databases like MySQL, Oracle, etc. A join is a combination of a Cartesian product followed by a selection process. A join operation pairs two tuples from different relations if and only if a given join condition is satisfied. An inner join is the one in which only those tuples are included that satisfy some conditions. In this article, we will be using MySQL to demonstrate the working of SQL Inner Join.

Steps to implement the SQL Inner Join :

Here, we will discuss the implementation of SQL Inner Join as follows.

Step-1: Creating Database :

Here, we will create the database by using the following SQL query as follows.

```
CREATE DATABASE geeks;
```

Step-2: Using the Database :

Here, we will use the geeks database.

AD

```
USE geeks;
```

Step-3: Adding Tables :

We will add 2 tables to the database as follows.

1. The first table will be the professor which will contain ID, the name of the professor, and salary.

2. The second table will be taught which will contain the ID of the course, professor's ID, and name of the course.

Adding table professor –

```
CREATE TABLE professor(
    ID int,
    Name varchar(20),
    Salary int
);
```

Adding table teaches –

```
CREATE TABLE teaches(
    course_id int,
    prof_id int,
    course_name varchar(20)
);
```

Step-4: Description of the Tables :

We can get the description of the 2 tables using the following SQL command as follows.

```
DESCRIBE professor
```

Output :

Field	Type	Null	Key	Default	Extra
ID	int	YES		NULL	
Name	varchar(20)	YES		NULL	
Salary	int	YES		NULL	

```
DESCRIBE teaches
```

Output :

Field	Type	Null	Key	Default	Extra
course_id	int	YES		NULL	

Field	Type	Null	Key	Default	Extra
prof_id	int	YES		NULL	
course_name	varchar(20)	YES		NULL	

Step-5: Inserting the rows :

Here, we will insert the rows in both tables one by one as follows.

Inserting rows inside professor table –

```
INSERT INTO professor VALUES (1, 'Rohan', 57000);
INSERT INTO professor VALUES (2, 'Aryan', 45000);
INSERT INTO professor VALUES (3, 'Arpit', 60000);
INSERT INTO professor VALUES (4, 'Harsh', 50000);
INSERT INTO professor VALUES (5, 'Tara', 55000);
```

Output :

Action Output			Message
#	Time	Action	
20	23:12:08	INSERT INTO professor VALUES (1, 'Rohan', 57000)	1 row(s) affected
21	23:12:08	INSERT INTO professor VALUES (2, 'Aryan', 45000)	1 row(s) affected
22	23:12:08	INSERT INTO professor VALUES (3, 'Arpit', 60000)	1 row(s) affected
23	23:12:08	INSERT INTO professor VALUES (4, 'Harsh', 50000)	1 row(s) affected
24	23:12:08	INSERT INTO professor VALUES (5, 'Tara', 55000)	1 row(s) affected

Inserting rows inside teaches table –

```
INSERT INTO teaches VALUES (1, 1, 'English');
INSERT INTO teaches VALUES (1, 3, 'Physics');
INSERT INTO teaches VALUES (2, 4, 'Chemistry');
INSERT INTO teaches VALUES (2, 5, 'Mathematics');
```

Output :

Action Output			Message
#	Time	Action	
✓ 26	23:30:19	INSERT INTO teaches VALUES (1, 1, 'English')	1 row(s) affected
✓ 27	23:30:19	INSERT INTO teaches VALUES (1, 3, 'Physics')	1 row(s) affected
✓ 28	23:30:19	INSERT INTO teaches VALUES (2, 4, 'Chemistry')	1 row(s) affected
✓ 29	23:30:19	INSERT INTO teaches VALUES (2, 5, 'Mathematics')	1 row(s) affected

Step-6: Current State of the Tables :

Verifying the data in both tables as follows.

professor Table –

```
SELECT * FROM professor;
```

Output :

ID	Name	Salary
1	Rohan	57000
2	Aryan	45000
3	Arpit	60000
4	Harsh	50000
5	Tara	55000

teaches Table –

```
SELECT * FROM teaches;
```

Output :

course_id	prof_id	course_name
1	1	English

course_id	prof_id	course_name
1	3	Physics
2	4	Chemistry
2	5	Mathematics

Step-7: INNER JOIN Query :

Syntax :

```
SELECT comma_separated_column_names
FROM table1 INNER JOIN table2 ON condition
```

Example –

```
SELECT teaches.course_id, teaches.prof_id, professor.Name, professor.Salary
FROM professor INNER JOIN teaches ON professor.ID = teaches.prof_id;
```

Output :

Using the Inner Join we are able to combine the information in the two tables based on a condition and the tuples in the Cartesian product of the two tables that do not satisfy the required condition are not included in the resulting table.

course_id	prof_id	Name	Salary
1	1	Rohan	57000
1	3	Arpit	60000
2	4	Harsh	50000
2	5	Tara	55000

Last Updated : 10 May, 2021

3

Similar Reads

1. Difference between Inner Join and Outer Join in SQL

2. Difference between Natural join and Inner Join in SQL

[Trending Now](#) [DSA](#) [Data Structures](#) [Algorithms](#) [Interview Preparation](#) [Data Science](#) [Topic-wise Practice](#) [Jobs](#)

SQL Outer Join

 neeraj kumar 13[Read](#) [Discuss](#) [Courses](#) [Practice](#)

In a [relational DBMS](#), we follow the principles of normalization that allows us to minimize the large tables into small tables. By using a select statement in Joins, we can retrieve the big table back. Outer joins are of following three types.

1. Left outer join
2. Right outer join
3. Full outer join

Creating a database : Run the following command to create a database.

```
Create database testdb;
```

Using the database : Run the following command to use a database.

```
use testdb;
```

Adding table to the database : Run the following command to add tables to a database.

AD

```
CREATE TABLE Students (
    StudentID int,
    LastName varchar(255),
    FirstName varchar(255),
    Address varchar(255),
    City varchar(255)
);
```

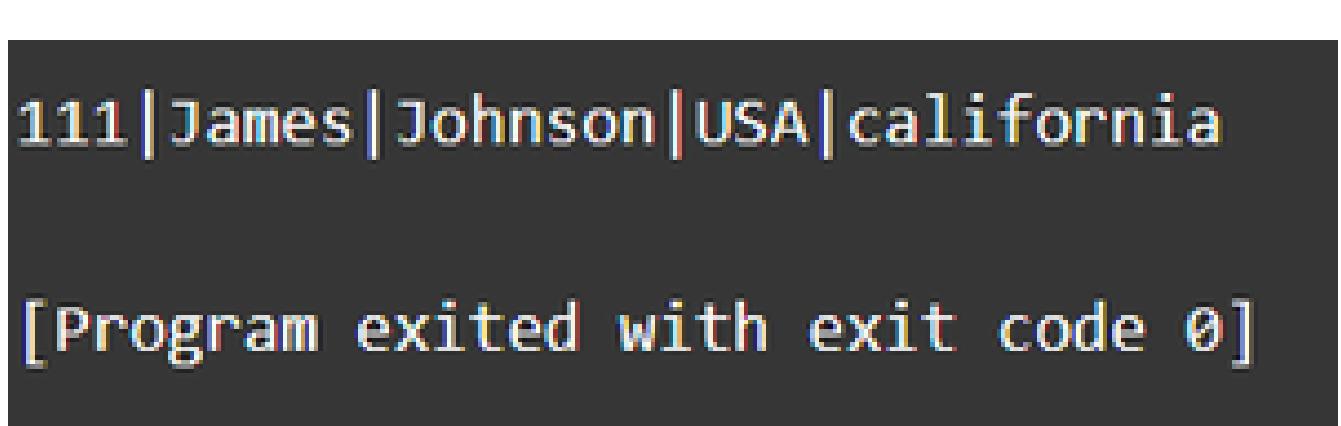
Inserting rows into database :

```
INSERT INTO students (
StudentID,
LastName,
FirstName,
Address,
City
)
VALUES
(
111,
'James',
'Johnson',
'USA',
california
);
```

Output of database :

Type the following command to get output.

```
SELECT * FROM students;
```

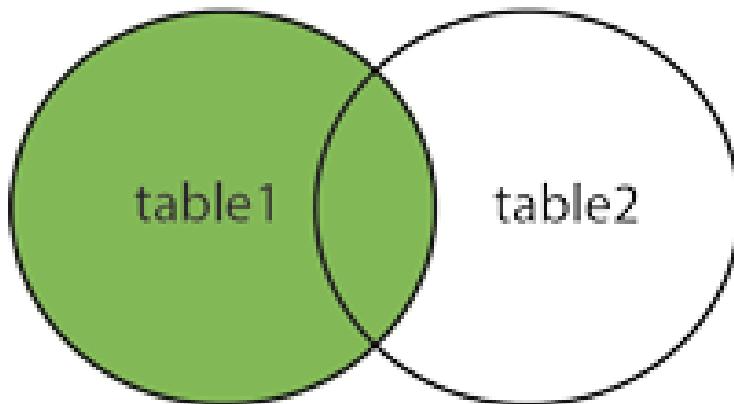
A screenshot of a terminal window with a black background and white text. It displays the output of a SQL query. The first line shows the row inserted into the 'students' table: '111|James|Johnson|USA|california'. The second line shows the terminal's exit message: '[Program exited with exit code 0]'.

```
111|James|Johnson|USA|california
[Program exited with exit code 0]
```

Types of outer join :

1. Left Outer Join : The left join operation returns all record from left table and matching records from the right table. On a matching element not found in right table, NULL is represented in that case.

LEFT JOIN

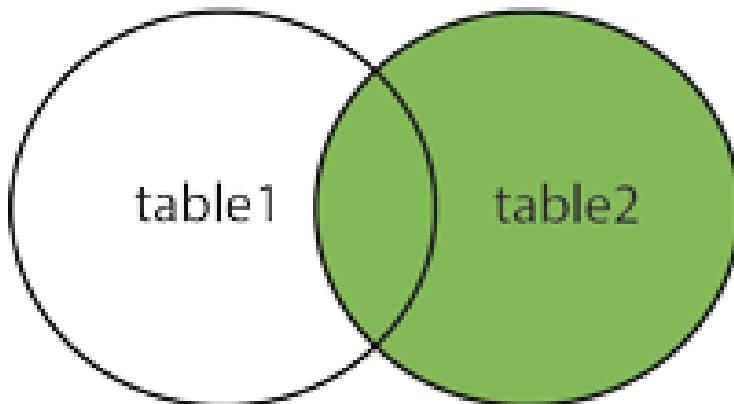


Syntax :

```
SELECT column_name(s)
FROM table1
LEFT JOIN Table2
ON Table1.Column_Name=table2.column_name;
```

2. Right Outer Join : The right join operation returns all record from right table and matching records from the left table. On a matching element not found in left table, NULL is represented in that case.

RIGHT JOIN

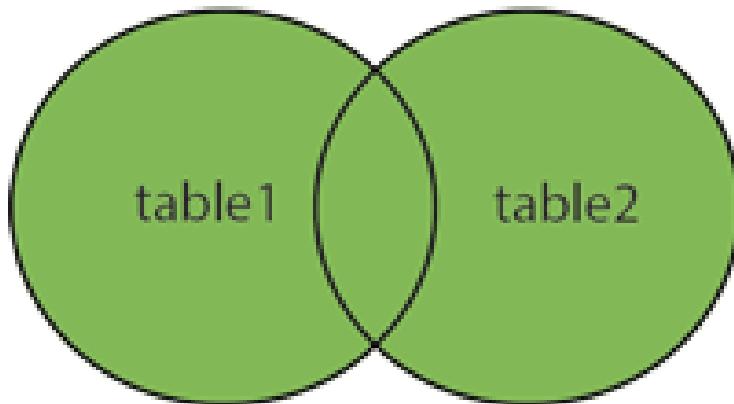


Syntax :

```
SELECT column_name(s)
FROM table1
RIGHT JOIN table2
ON table1.column_name = table2.column_name;
```

3. Full Outer Join : The full outer Join keyword returns all records when there is a match in left or right table records.

FULL OUTER JOIN



Syntax:

```
SELECT column_name
FROM table1
FULL OUTER JOIN table2
ON table1.columnName = table2.columnName
WHERE condition;
```

Example :

Creating 1st Sample table students.

```
CREATE TABLE students (
    id INTEGER,
    name TEXT NOT NULL,
    gender TEXT NOT NULL
);
-- insert some values
INSERT INTO students VALUES (1, 'Ryan', 'M');
INSERT INTO students VALUES (2, 'Joanna', 'F');
INSERT INTO students Values (3, 'Moana', 'F');
```

Creating 2nd sample table college.

```
CREATE TABLE college (
    id INTEGER,
    classTeacher TEXT NOT NULL,
    Strength TEXT NOT NULL
);
-- insert some values
```

```
INSERT INTO college VALUES (1, 'Alpha', '50');
INSERT INTO college VALUES (2, 'Romeo', '60');
INSERT INTO college Values (3, 'Charlie', '55');
```

Performing outer join on above two tables.

```
SELECT College.classTeacher, students.id
FROM College
FULL OUTER JOIN College ON College.id=students.id
ORDER BY College.classTeacher;
```

The above code will perform a full outer join on tables students and college and will return the output that matches the id of college with id of students. The output will be class Teacher from college table and id from students table. The table will be ordered by class Teacher from college table.

Class Teacher	Id
Alpha	1
Romeo	2
Charlie	3

Last Updated : 13 Apr, 2021

9

Similar Reads

1. SQL Full Outer Join Using Left and Right Outer Join and Union Clause

2. Difference between Inner Join and Outer Join in SQL

3. Difference between “INNER JOIN” and “OUTER JOIN”

4. Inner Join vs Outer Join

5. SQL | Join (Cartesian Join & Self Join)

6. SQL Full Outer Join Using Union Clause

7. SQL Full Outer Join Using Where Clause



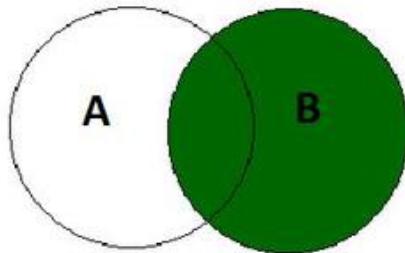
SQL Right Join



disha55handa

[Read](#)[Discuss](#)[Courses](#)[Practice](#)[Video](#)

The RIGHT JOIN keyword in SQL returns the all **matching records(or rows)** and the **records(or rows) which are present in the right table but not in the left table**. That means that, if a certain row is present in the right table but not in the left, the result will include this row but with a NULL value in each column from the left. If a record from the left table is not in the right, it will not be included in the result.

*RIGHT JOIN*

The syntax for a RIGHT JOIN is :-

```
SELECT column_name(s)
FROM tableA
RIGHT JOIN tableB ON tableA.column_name = tableB.column_name;
```

SQL RIGHT JOIN EXAMPLE :

In this example we will consider two tables **employee** table containing details of the employees working in the particular department and **department** table containing the details of the department

AD

employee table :

emp_no	emp_name	dept_no
E1	Varun Singhal	D1
E2	Amrita Aggarwal	D2
E3	Ravi Anand	D3

department table :

dept_no	d_name	location
D1	IT	Delhi
D2	HR	Hyderabad
D3	Finance	Pune
D4	Testing	Noida
D5	Marketing	Mathura

To perform right- join on these two tables we will use the following SQL query:

```
select emp_no , emp_name ,d_name, location
from employee
right join dept on employee.dept_no = department.dept_no;
```

The output that we will get is as follows :

emp_no	emp_name	d_name	location
E1	Varun Singhal	IT	Delhi
E2	Amrita Aggarwal	HR	Hyderabad
E3	Ravi Anand	Finance	Pune
[NULL]	[NULL]	Testing	Noida
[NULL]	[NULL]	Marketing	Mathura

As right join gives the matching rows and the rows that are present in the left table but not in the right table .Here in this example ,we see that the department that contains no employee contains [NULL] values of emp_no and emp_name after performing the right join.

Last Updated : 26 Apr, 2021

2

Similar Reads

1. [SQL | Join \(Cartesian Join & Self Join\)](#)

2. [SQL Full Outer Join Using Left and Right Outer Join and Union Clause](#)

3. [Left join and Right join in MS SQL Server](#)

4. [Difference between Inner Join and Outer Join in SQL](#)

5. [Difference between Natural join and Inner Join in SQL](#)

6. [Difference between Natural join and Cross join in SQL](#)

7. [Full join and Inner join in MS SQL Server](#)

8. [Self Join and Cross Join in MS SQL Server](#)

9. [SQL | EQUI Join and NON EQUI JOIN](#)

10. [Implicit Join vs Explicit Join in SQL](#)

[Engineering Mathematics](#) [Discrete Mathematics](#) [Digital Logic and Design](#) [Computer Organization and Architecture](#)

SQL Left Join

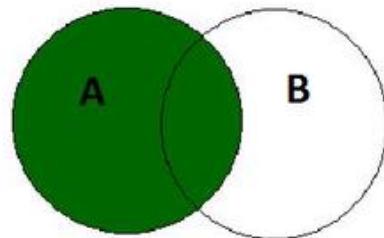


disha55handa

[Read](#)[Discuss](#)[Courses](#)[Practice](#)[Video](#)

The LEFT JOIN keyword in SQL returns all matching records(or rows) and the records(or rows) that are present in the left table but not in the right table.

That means that, if a certain row is present in the left table but not in the right, the result will include this row but with a NULL value in each column from the right. If a record from the right table is not on the left, it will not be included in the result.

*LEFT JOIN*

Syntax

SELECT column_name(s)

AD

FROM tableA

LEFT JOIN tableB ON tableA.column_name = tableB.column_name;

Example

In this example, we will consider two tables Emp containing details of the Employee working in the particular department, and department table containing the details of the department

Employee Table

Query:

```
CREATE TABLE Emp (
    EmpID INT PRIMARY KEY,
    Name VARCHAR(50),
    Country VARCHAR(50),
    Age INT,
    Salary INT,
    department_id INT
);

INSERT INTO Emp (EmpID, Name, Country, Age, Salary, department_id)
VALUES (1, 'Shubham', 'India', 23, 30000, 101),
       (2, 'Aman', 'Australia', 21, 45000, 102),
       (3, 'Naveen', 'Sri Lanka', 24, 40000, 103),
       (4, 'Aditya', 'Austria', 21, 35000, 104),
       (5, 'Nishant', 'Spain', 22, 25000, 101);
```

Output:

EmpID	Name	Country	Age	Salary	department_id
1	Shubham	India	23	30000	101
2	Aman	Australia	21	45000	102
3	Naveen	Sri Lanka	24	40000	103
4	Aditya	Austria	21	35000	104
5	Nishant	Spain	22	25000	101

Department Table

Query:

```
CREATE TABLE department (
    department_id INT PRIMARY KEY,
    department_name VARCHAR(50),
    department_head VARCHAR(50),
    location VARCHAR(50)
```

);

```
INSERT INTO department (department_id, department_name, department_head,
location)
VALUES (101, 'Sales', 'Sarah', 'New York'),
       (102, 'Marketing', 'Jay', 'London'),
       (103, 'Finance', 'Lavish', 'San Francisco'),
       (104, 'Engineering', 'Kabir', 'Bangalore');
Select * from department;
```

Output:

department_id	department_name	department_head	location
101	Sales	Sarah	New York
102	Marketing	Jay	London
103	Finance	Lavish	San Francisco
104	Engineering	Kabir	Bangalore

To perform left-join on these two tables we will use the following SQL query :

```
SELECT Emp.EmpID, Emp.Name, department.
       department_name, department.department_head,
       department.location
FROM Emp
LEFT JOIN department ON Emp.department_id
= department.department_id;
```

Output:

EmpID	Name	department_name	department_head	location
1	Shubham	Sales	Sarah	New York
2	Aman	Marketing	Jay	London
3	Naveen	Finance	Lavish	San Francisco
4	Aditya	Engineering	Kabir	Bangalore
5	Nishant	Sales	Sarah	New York

As left join gives the matching rows and the rows that are present in the left table but not in the right table. Here in this example, we see that the employees that do not work in a particular department, i.e, having dept no values as [NULL], contain [NULL] values of dept name and location after the left join.

SQL Join as Aliases

Now in this query, we will use the aliases “e” for the Emp table and “d” for the department table. Then the SELECT statement then references these aliases for each of the columns being returned, making our query easier to read and type out. Aliases are especially useful when working with tables that have long or complicated names, as they can help simplify the code and make it easier to understand.

Query:

```
SELECT e.EmpID, e.Name, d.department_name,
d.department_head, d.location
FROM Emp e
LEFT JOIN department d ON
e.department_id = d.department_id;
```

Output:

EmpID	Name	department_name	department_head	location
1	Shubham	Sales	Sarah	New York
2	Aman	Marketing	Jay	London
3	Naveen	Finance	Lavish	San Francisco
4	Aditya	Engineering	Kabir	Bangalore
5	Nishant	Sales	Sarah	New York

SQL Join with WHERE Clause

Now in this query, we will add a WHERE clause that specifies to only return results where the “location” column in the department table equals ‘Bangalore’. This will filter the results to only show employees who belong to a department located in Bangalore, and departments that have no employees will not be returned in the results.

Query:

```
SELECT e.EmpID, e.Name, d.department_name,
d.department_head, d.location
FROM Emp e
LEFT JOIN department d ON e.department_id
= d.department_id
WHERE d.location = 'Bangalore';
```

Output:

EmpID	Name	department_name	department_head	location
4	Aditya	Engineering	Kabir	Bangalore

Last Updated : 14 Apr, 2023

3

Similar Reads

1. SQL | Join (Cartesian Join & Self Join)

2. SQL Full Outer Join Using Left and Right Outer Join and Union Clause

3. Left join and Right join in MS SQL Server

4. Difference between Inner Join and Outer Join in SQL

5. Difference between Natural join and Inner Join in SQL

6. Difference between Natural join and Cross join in SQL

7. Full join and Inner join in MS SQL Server

8. Self Join and Cross Join in MS SQL Server

9. SQL | EQUI Join and NON EQUI JOIN

10. Implicit Join vs Explicit Join in SQL

[Previous](#)[Next](#)

Article Contributed By :



disha55handa

disha55handa

Vote for difficulty

Current difficulty : [Hard](#)



SQL | Join (Cartesian Join & Self Join)

[Read](#)
[Discuss](#)
[Courses](#)
[Practice](#)
[Video](#)

[SQL | JOIN\(Inner, Left, Right and Full Joins\)](#)

In this article, we will discuss about the remaining two JOINS:

- **CARTESIAN JOIN**
- **SELF JOIN**

Consider the two tables below:

Student

ROLL_NO	NAME	ADDRESS	PHONE	Age
1	Ram	Delhi	XXXXXXXXXX	18
2	RAMESH	GURGAON	XXXXXXXXXX	18
3	SUJIT	ROHTAK	XXXXXXXXXX	20
4	SURESH	Delhi	XXXXXXXXXX	18

StudentCourse

COURSE_ID	ROLL_NO
1	1
2	2
2	3
3	4

AD

1. CARTESIAN JOIN: The CARTESIAN JOIN is also known as CROSS JOIN. In a CARTESIAN JOIN there is a join for each row of one table to every row of another table. This usually happens when the matching column or WHERE condition is not specified.

- In the absence of a WHERE condition the CARTESIAN JOIN will behave like a CARTESIAN PRODUCT . i.e., the number of rows in the result-set is the product of the number of rows of the two tables.
- In the presence of WHERE condition this JOIN will function like a INNER JOIN.
- Generally speaking, Cross join is similar to an inner join where the join-condition will always evaluate to True

Syntax:

```
SELECT table1.column1 , table1.column2, table2.column1...
FROM table1
CROSS JOIN table2;
```

table1: First table.

table2: Second table

Example Queries(CARTESIAN JOIN):

- In the below query we will select NAME and Age from Student table and COURSE_ID from StudentCourse table. In the output you can see that each row of the table Student is joined with every row of the table StudentCourse. The total rows in the result-set = $4 * 4 = 16$.

```
SELECT Student.NAME, Student.AGE, StudentCourse.COURSE_ID
FROM Student
CROSS JOIN StudentCourse;
```

Output:

NAME	AGE	COURSE_ID
Ram	18	1
Ram	18	2
Ram	18	2
Ram	18	3
RAMESH	18	1
RAMESH	18	2
RAMESH	18	2
RAMESH	18	3
SUJIT	20	1
SUJIT	20	2
SUJIT	20	2
SUJIT	20	3
SURESH	18	1
SURESH	18	2
SURESH	18	2
SURESH	18	3

2. **SELF JOIN:** As the name signifies, in SELF JOIN a table is joined to itself. That is, each row of the table is joined with itself and all other rows depending on some conditions. In other words we can say that it is a join between two copies of the same table.
- Syntax:**

```
SELECT a.column1 , b.column2
FROM table_name a, table_name b
WHERE some_condition;
```

table_name: Name of the table.

some_condition: Condition for selecting the rows.

Example Queries(SELF JOIN):

```
SELECT a.ROLL_NO , b.NAME
FROM Student a, Student b
WHERE a.ROLL_NO < b.ROLL_NO;
```

Output:

ROLL_NO	NAME
1	RAMESH
1	SUJIT
2	SUJIT
1	SURESH
2	SURESH
3	SURESH

10. Self Join (Top 50 SQL Interview Questions)| GeeksforGeeks



This article is contributed by [Harsh Agarwal](#). If you like GeeksforGeeks and would like to contribute, you can also write an article using contribute.geeksforgeeks.org or mail your article to contribute@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Last Updated : 09 Nov, 2020

69

Similar Reads

1. [Self Join and Cross Join in MS SQL Server](#)
2. [SQL Query to Avoid Cartesian Product](#)
3. [SQL Full Outer Join Using Left and Right Outer Join and Union Clause](#)
4. [Difference between Inner Join and Outer Join in SQL](#)
5. [Difference between Natural join and Inner Join in SQL](#)
6. [Difference between Natural join and Cross join in SQL](#)
7. [Full join and Inner join in MS SQL Server](#)
8. [Left join and Right join in MS SQL Server](#)
9. [SQL | EQUI Join and NON EQUI JOIN](#)
10. [Implicit Join vs Explicit Join in SQL](#)

Previous

Next

Article Contributed By :


[Trending Now](#) [DSA](#) [Data Structures](#) [Algorithms](#) [Interview Preparation](#) [Data Science](#) [Topic-wise Practice](#) [Jobs](#)

Combining aggregate and non-aggregate values in SQL using Joins and Over clause



ishaan bhatnagar

[Read](#)[Discuss](#)[Courses](#)[Practice](#)

Prerequisite – [Aggregate functions in SQL](#), [Joins in SQL](#)

Aggregate functions perform a calculation on a set of values and return a single value. Now, consider an employee table EMP and a department table DEPT with following structure:

Table – EMPLOYEE TABLE

Name	Null	Type
EMPNO	NOT NULL	NUMBER(4)
ENAME		VARCHAR2(10)
JOB		VARCHAR2(9)
MGR		NUMBER(4)
HIREDATE		DATE
SAL		NUMBER(7, 2)
COMM		NUMBER(7, 2)
DEPTNO		NUMBER(2)

Table – DEPARTMENT TABLE

Name	Null	Type
DEPTNO		NUMBER(2)

Name	Null	Type
DNAME		VARCHAR2(14)
LOC		VARCHAR2(13)

And the following results are needed:

AD

1. DISPLAY NAME, SAL, JOB OF EMP ALONG WITH MAX, MIN, AVG, TOTAL SAL OF THE EMPS DOING THE SAME JOB.
2. DISPLAY DEPTNAME WITH NUMBER OF EMP WORKING IN IT.

The aggregated values can't be directly used with non-aggregated values to obtain a result. Thus one can use the following concepts:

1. Using Joins –

1. Create a sub-table containing the result of aggregated values.
2. Using Join, use the results from the sub-table to display them with non-aggregated values.

Solutions for problem 1 using JOIN:

```

SELECT ENAME, SAL, EMP.JOB,
       SUBTABLE.MAXSAL, SUBTABLE.MINSAL,
       SUBTABLE.AVGSAL, SUBTABLE.SUMSAL
FROM EMP
INNER JOIN
  (SELECT JOB, MAX(SAL) MAXSAL, MIN(SAL)
   MINSAL, AVG(SAL) AVGSAL, SUM(SAL) SUMSAL
  FROM EMP
  GROUP BY JOB) SUBTABLE
ON EMP.JOB = SUBTABLE.JOB;

```

Output for sample data:

Ename	Sal	Job	MaxSal	MinSal	AvgSal	SumSal
SCOTT	3300	ANALYST	3300	1925	2841.67	8525
HENRY	1925	ANALYST	3300	1925	2841.67	8525
FORD	3300	ANALYST	3300	1925	2841.67	8525
SMITH	3300	CLERK	3300	1045	1746.25	6985
MILLER	1430	CLERK	3300	1045	1746.25	6985

2. Using 'Over' clause –

1. OVER CLAUSE ALONG WITH PARTITION BY IS USED TO BRAKE UP DATA INTO PARTITIONS.
2. THE SPECIFIED FUNCTION OPERATES FOR EACH PARTITION.

Solutions for problem 2 using OVER Clause:

```
SELECT DISTINCT(DNAME),
COUNT(ENAME) OVER (PARTITION BY EMP.DEPTNO) EMP
FROM EMP
RIGHT OUTER JOIN DEPT
ON EMP.DEPTNO=DEPT.DEPTNO
ORDER BY EMP DESC;
```

Dname	Emp
SALES	6
RESEARCH	5
ACCOUNTING	3
OPERATIONS	0
OTHERS	0


[Trending Now](#) [DSA](#) [Data Structures](#) [Algorithms](#) [Interview Preparation](#) [Data Science](#) [Topic-wise Practice](#) [Jobs](#)

SQL Server Mathematical functions (SQRT, PI, SQUARE, ROUND, CEILING & FLOOR)



Sam007

[Read](#)[Discuss](#)[Courses](#)[Practice](#)

Mathematical functions are present in SQL server which can be used to perform mathematical calculations. Some commonly used mathematical functions are given below:

1. SQRT():

SQRT() function is the most commonly used function. It takes any numeric values and returns the square root value of that number.

Syntax:

```
SELECT SQRT(..value..)
```

Example:

```

SELECT SQRT(100)

SELECT PI()

SELECT SQUARE(25)
SELECT SQUARE(10.12)

SELECT ROUND(125.315, 2);
SELECT ROUND(125.315, 1);

SELECT CEILING(45.56)
SELECT FLOOR(45.56)

```

Results	Messages
(No column name)	
10	

2. PI(): There are calculations which require use of pi. Using pi() function, value of PI can be used anywhere in the query.

AD

Syntax:

```
SELECT PI()
```

Example:

```

SELECT SQRT(100)
SELECT PI()
SELECT SQUARE(25)
SELECT SQUARE(10.12)

SELECT ROUND(125.315, 2);
SELECT ROUND(125.315, 1);

SELECT CEILING(45.56)
SELECT FLOOR(45.56)

```

Results Messages
(No column name)
3.14159265358979

3. SQUARE(): SQUARE() function is used to find the square of any number.**Syntax:**

```
SELECT SQUARE(..value..)
```

Example:

```

SELECT SQUARE(25)
SELECT SQUARE(10.12)

SELECT ROUND(125.315, 2);
SELECT ROUND(125.315, 1);

SELECT CEILING(45.56)
SELECT FLOOR(45.56)

```

Results Messages
(No column name)
625

(No column name)
102.4144

4. ROUND(): ROUND() function is used to round a value to the nearest specified decimal place.

Syntax:

```
SELECT ROUND(..value.., number_of_decimal_places)
```

Example:

```
SELECT ROUND(125.315, 2);
SELECT ROUND(125.315, 1);

SELECT CEILING(45.56)
SELECT FLOOR(45.56)

76 % ▾
Results Messages
(No column name)
1 125.320
```



```
(No column name)
1 125.300
```

5. CEILING() and FLOOR()

CEILING(): CEILING() function is used to find the next highest value (integer).

Syntax:

```
SELECT CEILING(..value..)
```

FLOOR(): FLOOR() function returns the next lowest value (integer).

Syntax:

```
SELECT FLOOR(..value..)
```

Example:

```
SELECT CEILING(45.56)
SELECT FLOOR(45.56)

76 % ▾
Results Messages
(No column name)
1 46
```



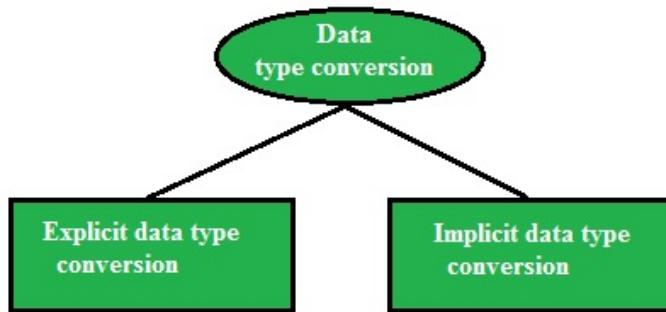
```
(No column name)
1 45
```



SQL | Conversion Function



MrinalVerma

[Read](#) [Discuss](#) [Courses](#) [Practice](#)


In some cases, the Server uses data of one type where it expects data of a different data type. This can happen when the Server can automatically convert the data to the expected data type. This data type conversion can be done implicitly by the Server, or explicitly by the user.

Implicit Data-Type Conversion :

In this type of conversion the data is converted from one type to another implicitly (by itself/automatically).

From	To
VARCHAR2 or CHAR	NUMBER
VARCHAR2 or CHAR	DATE
DATE	VARCHAR2
NUMBER	VARCHAR2

1. QUERY:

```
SELECT employee_id,first_name,salary
FROM employees
WHERE salary > 15000;
```

1. OUTPUT :

Employee_ID	FIRST_NAME	SALARY
100	Steven	24000
101	Neena	17000
102	lex	17000

1. QUERY:

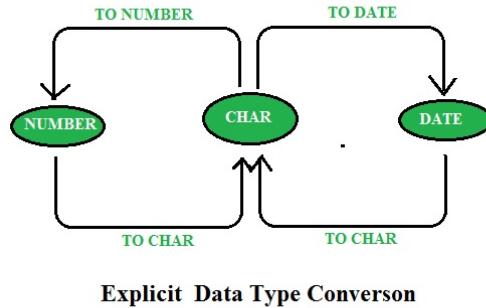
```
SELECT employee_id,first_name,salary
FROM employees
WHERE salary > '15000';
```

1. OUTPUT :

Employee_ID	FIRST_NAME	SALARY
100	Steven	24000
101	Neena	17000
102	lex	17000

1. Here we see the output of both queries came out to be same,inspite of 2nd query using '15000' as text, it is automatically converted into **int** data type.

Explicit Data-Type Conversion :



AD

TO_CHAR Function :

TO_CHAR function is used to typecast a numeric or date input to character type with a format model (optional).

SYNTAX :

```
TO_CHAR(number1, [format], [nls_parameter])
```

Using the TO_CHAR Function with Dates :

SYNTAX :

```
TO_CHAR(date, 'format_model')
```

The format model:

- Must be enclosed in single quotation marks and is case sensitive
- Can include any valid date format element
- Has an fm element to remove padded blanks or suppress leading zeros
- Is separated from the date value by a comma

EXAMPLE :

```
SELECT employee_id, TO_CHAR(hire_date, 'MM/YY') Month_Hired
FROM employees
WHERE last_name = 'Higgins';
```

OUTPUT :

EMPLOYEE_ID	MONTH_HIRED
205	06/94

Elements of the Date Format Model :

YYYY	Full year in Numbers
YEAR	Year spelled out
MM	Two digit value for month
MONTH	Full name of the month
MON	Three Letter abbreviation of the month
DY	Three letter abbreviation of the day of the week
DAY	Full Name of the Day
DD	Numeric day of the month

Elements of the Date Format Model :**Date Format Elements – Time Formats :**

Use the formats listed in the following tables to display time information and literals and to change numerals to spelled numbers.

ELEMENT	DESCRIPTION
AM or PM	Meridian indicator
A.M. or P.M.	Meridian indicator with periods
HH or HH12 or HH24	Hour of day,or hour (1-12),or hour (0-23)
MI	Minute 0-59
SS	Second 0-59
SSSSS	Second past Mid Night 0-86399

Other Formats :

ELEMENT	DESCRIPTION
/ . ,	Punctuation is reproduced in the result
“of the”	Quoted string is reproduced in the result

Specifying Suffixes to Influence Number Display :

ELEMENT	DESCRIPTION
TH	Ordinal Number (for example DDTH for 4TH)
SP	Spelled out number (for example DDSP for FOUR)
SPTH or THSP	spelled out ordinal numbers (for example DDSPTH for FOURTH)

EXAMPLE :

```
SELECT last_name,
TO_CHAR(hire_date, 'fmDD Month YYYY')
AS HIREDATE
FROM employees;
```

OUTPUT :

LASTNAME	HIREDATE
Austin	25 January 2005
Shubham	20 June 2004
Nishant	15 January 1999
Ankit	15 July 1995
Vanshika	5 August 2004
Kusum	10 June 1994
Faviet	11 March 2005
King	9 April 1996

Using the TO_CHAR Function with Numbers :**SYNTAX :**

```
TO_CHAR(number, 'format_model')
```

These are some of the format elements you can use with the TO_CHAR function to display a number value as a character :

9	Represent a number
0	Forces a zero to be displayed

\$	places a floating dollar sign
L	Uses the floating local currency symbol
.	Print a decimal point
,	Prints a Thousand indicator

EXAMPLE :

```
SELECT TO_CHAR(salary, '$99,999.00') SALARY
FROM employees
WHERE last_name = 'Ernst';
```

OUTPUT :

SALARY
\$5000

Using the TO_NUMBER and TO_DATE Functions :

Convert a character string to a number format using the **TO_NUMBER** function :

```
TO_NUMBER(char[, 'format_model'])
```

Convert a character string to a date format using the **TO_DATE** function:

```
TO_DATE(char[, 'format_model'])
```

These functions have an **fx** modifier. This modifier specifies the exact matching for the character argument and date format model of a **TO_DATE** function.

EXAMPLE :

```
SELECT last_name, hire_date
FROM employees
WHERE hire_date = TO_DATE('May 24, 1999', 'fxMonth DD, YYYY');
```

OUTPUT :

LASTNAME	HIREDATE
Kumar	24-MAY-99

Last Updated : 28 Mar, 2023

16

Similar Reads

1. Difference between Structured Query Language (SQL) and Transact-SQL (T-SQL)

2. Configure SQL Jobs in SQL Server using T-SQL

3. MS SQL Server - Type Conversion

4. SQL | Procedures in PL/SQL

5. SQL | Difference between functions and stored procedures in PL/SQL

6. SQL SERVER – Input and Output Parameter For Dynamic SQL

7. Difference between SQL and T-SQL

8. SQL Server | Convert tables in T-SQL into XML

9. SQL SERVER | Bulk insert data from csv file using T-SQL command

10. SQL - SELECT from Multiple Tables with MS SQL Server

[Previous](#)[Next](#)**Article Contributed By :****MrinalVerma**

MrinalVerma

Vote for difficultyCurrent difficulty : [Easy](#)


[Trending Now](#) [DSA](#) [Data Structures](#) [Algorithms](#) [Interview Preparation](#) [Data Science](#) [Topic-wise Practice](#) [Jobs](#)

SQL general functions | NVL, NVL2, DECODE, COALESCE, NULLIF, LNNVL and NANVL



shubham_rana_77

[Read](#) [Discuss](#) [Courses](#) [Practice](#)

In this article, we'll be discussing some powerful SQL general functions, which are – NVL, NVL2, DECODE, COALESCE, NULLIF, LNNVL and NANVL.

These functions work with any data type and pertain to the use of null values in the expression list. These are all **single row function** i.e. provide one result per row.

- **NVL(expr1, expr2)** : In SQL, NVL() converts a null value to an actual value. Data types that can be used are date, character and number. Data type must match with each other i.e. expr1 and expr2 must of same data type.

Syntax –

`NVL (expr1, expr2)`

expr1 is the source value or expression that may contain a null.

expr2 is the target value for converting the null.

AD

Example –

```
SELECT salary, NVL(commission_pct, 0),
       (salary*12) + (salary*12*NVL(commission_pct, 0))
    annual_salary FROM employees;
```

Output :

SALARY	COMMISSION_PCT	NVL(COMMISSION_PCT,0)	ANNUAL_SALARY	Recent Comments	Upvotes	Downvotes
Example :-						
8400	Input : SELECT last_name, salary, commission_pct, (salary*.12) + (salary*.12*NVL(commission_pct, 0)) AS SAL FROM employees;	.2	120960	Mithilesh Upadhyay	96600	0
7000		.15	96600	ques . Updated.	81840	0
6200	Output:	.1	81840	GATE GATE-CS-2016	38400	0
3200	SALARY COMMISSION_PCT NVL(COMMISSION_PCT,0) ANNUAL_SALARY		38400	tanu hey bro ! check	37200	0
3100	8400	.2	120960	Loops & Control Structure in	30000	0
2500	7000	.15	96600	surendra sharma	30000	0
	6200	.1	81840			
	3200		38400			
	3100		37200			
	2500		30000			

- NVL2(expr1, expr2, expr3)** : The NVL2 function examines the first expression. If the first expression is not null, then the NVL2 function returns the second expression. If the first expression is null, then the third expression is returned i.e. If expr1 is not null, NVL2 returns expr2. If expr1 is null, NVL2 returns expr3. The argument expr1 can have any data type.

Syntax –

```
NVL2 (expr1, expr2, expr3)
```

expr1 is the source value or expression that may contain null

expr2 is the value returned if expr1 is not null

expr3 is the value returned if expr1 is null

Example –

```
SELECT last_name, salary, commission_pct,
       NVL2(commission_pct, 'SAL+COMM', 'SAL')
    income FROM employees;
```

Output :

LAST_NAME	SALARY	COMMISSION_PCT	INCOME
Output:			
Livingston	8400	.2	SAL+COMM
Grant	7000	.15	SAL+COMM
Johnson	6200	.1	SAL+COMM
Taylor	3200		SAL
Fleaur	3100		SAL
Sullivan	2500		SAL

- DECODE()** : Facilitates conditional inquiries by doing the work of a CASE or IF-THEN-ELSE statement.

The DECODE function decodes an expression in a way similar to the IF-THEN-ELSE logic

used in various languages. The DECODE function decodes expression after comparing it to each search value. If the expression is the same as search, result is returned.

If the default value is omitted, a null value is returned where a search value does not match any of the result values.

Syntax –

```
DECODE(col|expression, search1, result1
      [, search2, result2,...,][, default])
```

Example –

```
SELECT last_name, job_id, salary,
       DECODE(job_id, 'IT_PROG', 1.10*salary,
              'ST_CLERK', 1.15*salary,
              'SA REP', 1.20*salary,salary)
       REVISED_SALARY FROM employees;
```

Output :

LAST_NAME	JOB_ID	SALARY	REVISED_SALARY
Lorentz	IT_PROG	4200	4620
Sarchand	SH_CLERK	4200	4200
Whalen	AD_ASST	4400	4400
Austin	IT_PROG	4800	5280
Pataballa	IT_PROG	4800	5280
Mourgos	ST_MAN	5800	5800
Ernst	IT_PROG	6000	6600
Fay	MK_REP	6000	6000
Kumar	SA REP	6100	7320
Banda	SA REP	6200	7440

- **COALESCE()** : The COALESCE() function examines the first expression, if the first expression is not null, it returns that expression; Otherwise, it does a COALESCE of the remaining expressions.

The advantage of the COALESCE() function over the NVL() function is that the COALESCE function can take multiple alternate values. In simple words COALESCE() function returns the first non-null expression in the list.

Syntax –

```
COALESCE (expr_1, expr_2, ... expr_n)
```

Examples –

```
SELECT last_name,
       COALESCE(commission_pct, salary, 10) comm
       FROM employees ORDER BY commission_pct;
```

Output :

LAST_NAME	</pre>	COMM
Livingston	This article is on	2
Grant	amp; list=pract.	15
Johnson	contribute, you can	.1
Taylor	3200	
Fleaur	3100	
Sullivan	contribute@gfgeeks.	2500
Geoni	2800	
Sarchand	4200	
Bull	4100	
Dellinger	Please write when	3400
Cabrio	topic discussed abo	3000

- NULLIF()** : The NULLIF function compares two expressions. If they are equal, the function returns null. If they are not equal, the function returns the first expression. You cannot specify the literal NULL for first expression.

Syntax –

```
NULLIF (expr_1, expr_2)
```

Examples –

```
SELECT LENGTH(first_name) "expr1",
       LENGTH(last_name) "expr2",
       NULLIF(LENGTH(first_name), LENGTH(last_name))
    result FROM employees;
```

Output :

LENGTH(FIRST_NAME)	LENGTH(LAST_NAME)	RESULT
8	5	8
6	5	6
5	5	
7	9	7
7	7	
6	6	
3	3	
5	6	5
4	6	4
6	3	6
4	5	4

- LNNVL()** : LNNVL evaluate a condition when one or both operands of the condition may be null. The function can be used only in the WHERE clause of a query. It takes as an argument a condition and returns TRUE if the condition is FALSE or UNKNOWN and FALSE if the condition is TRUE.

Syntax –

`LNNVL(condition(s))`

Examples –

```
SELECT COUNT(*) FROM employees
WHERE commission_pct < .2;
```

Output :

```
SQL> select count(*) from employees where commission_pct<0.2;
      COUNT(*)
-----
      11
```

83

4 Previous Page

Now the above examples does not considered those employees who have no commission at all.

To include them as well we use LNNVL()

```
SELECT COUNT(*) FROM employees
WHERE LNNVL(commission_pct >= .2);
```

Output :

```
SQL> select count(*) from employees where LNNVL(commission_pct>=0.2);
      COUNT(*)
-----
      83
```

- **NANVL()** : The NANVL function is useful only for floating-point numbers of type BINARY_FLOAT or BINARY_DOUBLE. It instructs the Database to return an alternative value n2 if the input value n1 is NaN (not a number). If n1 is not NaN, then database returns n1. This function is useful for mapping NaN values to NULL.

Syntax –

NANVL(n1 , n2)

Consider the following table named nanvl_demo :

BIN_FLOAT
5.056E+001
Nan

Example –

```
SELECT bin_float, NANVL(bin_float,0)
  FROM nanvl_demo;
```

Output :

BIN_FLOAT	NANVL(BIN_FLOAT,0)
5.056E+001	5.056E+001
Nan	0

Reference: Introduction to Oracle9i SQL(Volume-1 Book)

Last Updated : 18 Dec, 2019

11

Similar Reads

1. NULLIF() Function in SQL Server

2. Use of COALESCE() function in SQL Server

3. MySQL | NULLIF() Function



SQL | Conditional Expressions

[Read](#)
[Discuss](#)
[Courses](#)
[Practice](#)

Following are Conditional Expressions in SQL

1. **The CASE Expression:** Let you use IF-THEN-ELSE statements without having to invoke procedures.

In a simple CASE expression, the SQL searches for the first WHEN.....THEN pair for which expr is equal to comparison_expr and returns return_expr. If above condition is not satisfied, an ELSE clause exists, the SQL returns else_expr. Otherwise, returns NULL.

We cannot specify literal null for the return_expr and the else_expr. All of the expressions(expr, comparison_expr, return_expr) must be of the same data type.

Syntax:

```
CASE expr WHEN comparison_expr1 THEN return_expr1
            [WHEN comparison_expr2 THEN return_expr2
             .
             .
             .
             WHEN comparison_exprn THEN return_exprn
             ELSE else_expr]
END
```

Example:

Input :

```
SELECT first_name, department_id, salary,
       CASE department_id WHEN 50 THEN 1.5*salary
                           WHEN 12 THEN 2.0*salary
                           ELSE salary
       END "REVISED SALARY"
FROM Employee;
```

Output :

The screenshot shows a MySQL command-line interface. The query is:

```
SELECT first_name, department_id, salary,
CASE department_id WHEN 50 THEN 1.5*salary
                  WHEN 12 THEN 2.0*salary
                ELSE salary
END "REVISED SALARY"
FROM Employee;
```

The results table has columns: FIRST_NAME, DEPARTMENT_ID, SALARY, REVISED SALARY. The data is:

FIRST_NAME	DEPARTMENT_ID	SALARY	REVISED SALARY
Vipul	50	30000	45000
Amit	12	27000	54000
Satish	12	3500	7000
Manuel	23	7300	7300
Archit	50	2950	4425

5 rows returned in 0.00 seconds

AD

Explanation: In above SQL statements, the value of department_id is decoded. If it is 50 then salary is made 1.5 times, if it is 12 then salary is made 2 times, else there is no change in salary.

2. The DECODE Function : Facilitates conditional inquiries by doing the work of a CASE or IF-THEN-ELSE statement.

The DECODE function decodes an expression in a way similar to the IF-THEN-ELSE logic used in various languages. The DECODE function decodes expression after comparing it to each search value. If the expression is the same as search, result is returned.

If the default value is omitted, a null value is returned where a search value does not match any of the result values.

Syntax:

```
DECODE(col/expression, search1, result1
      [, search2, result2,.....,]
      [, default])
```

Input :

```
SELECT first_name, department_id, salary,
       DECODE(department_id, 50, 1.5*salary,
              12, 2.0*salary,
              salary)
```

```
"REVISED SALARY"
FROM Employee;
```

Output :

```
SELECT first_name, department_id, salary,
       DECODE(department_id, 50, 1.5*salary,
              12, 2.0*salary,
              salary)
    "REVISED SALARY"
   FROM Employee;
```

FIRST_NAME	DEPARTMENT_ID	SALARY	REVISED SALARY
Vipul	50	30000	45000
Amit	12	27000	54000
Sales	12	3500	7000
Manish	23	7300	7300
Archit	50	2950	4425

5 rows returned in 0.00 seconds

Explanation: In above SQL statements, the value of department_id is tested. If it is 50 then salary is made 1.5 times, if it is 12 then salary is made 2 times, else there is no change in salary.

3. **COALESCE** : Returns the first non-null argument. Null is returned only if all arguments are null. It is often used to substitute a default value for null values when data is retrieved for display.

NOTE: Same as CASE expressions, COALESCE also will not evaluate the arguments to the right of the first non-null argument found.

Syntax:

```
COALESCE(value [ , .....] )
```

Input:

```
SELECT COALESCE(last_name, '- NA -')
  from Employee;
```

Output:

The screenshot shows a SQL query window in Oracle Database Express Edition. The query is:

```
select COALESCE(last_name, '- NA -')
from employee;
```

The results show the last names of employees. Rows where last_name is null are replaced by '- NA -'.

COALESCE(LAST_NAME, '- NA -')
Shah
Garg
Kumar
- NA -
Ramya

6 rows returned in 0.00 seconds

Explanation: “- NA -” will be displayed in place where last name is null else respective last names will be shown.

4. **GREATEST:** Returns the largest value from a list of any number of expressions. Comparison is case sensitive. If datatypes of all the expressions in the list are not same, rest all expressions are converted to the datatype of the first expression for comparison and if this conversion is not possible, SQL will throw an error.

NOTE: Returns null if any expression in the list is null.

Syntax:

GREATEST(expr1, expr2 [,])

- **Input:**

```
SELECT GREATEST('XYZ', 'xyz')
from dual;
```

Output:

```
GREATEST('XYZ', 'xyz')
xyz
```

Explanation: ASCII value of small alphabets is greater.

- **Input:**

```
SELECT GREATEST('XYZ', null, 'xyz')
from dual;
```

Output:

```
GREATEST('XYZ', null, 'xyz')
```

-

Explanation: Since null is present hence, null will be shown as output (as mentioned to note in description above).

5. **IFNULL:** If expr1 is not NULL, returns expr1; otherwise it returns expr2. Returns a numeric or string value, depending on the context in which it is used.

Syntax:

```
IFNULL(expr1, expr2)
```

- **Input:**

```
SELECT IFNULL(1,0)
FROM dual;
```

Output:

-

1

Explanation : Since, no expression is null.

- **Input:**

```
SELECT IFNULL(NULL,10)
FROM dual;
```

Output:

--

10

Explanation: Since, expr1 is null hence, expr2 is shown.

6. **IN:** Checks whether a value is present within a set of values and can be used with WHERE, CHECK and creation of views.

NOTE: Same as CASE and COALESCE expressions, IN also will not evaluate the arguments to the right of the first non-null argument found.

Syntax:

```
WHERE column IN (x1, x2, x3 [ ,..... ] )
```

Input:

```
SELECT * from Employee
WHERE department_id IN(50, 12);
```

Output:

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	CONTACT_NUMBER	EXTENSION	DEPARTMENT_ID	
101	Vimal	Issac	vimalissac@gmail.com	9895509800	33-AUG-11	50	Manager
102	Jani	Goyal	jani.goyal@gmail.com	9895409800	16-JUN-11	12	Manager
103	Satish	Kumar	satish.kumar@gmail.com	9895467897	25-APR-11	12	Manager
105	Ajith	Ramamoorthy	ajith.ramamoorthy@gmail.com	9895420170	01-JUL-11	50	Manager

4 rows returned in 0.00 seconds

Explanation: All data of Employees is shown with department ID 50 or 12.

7. **LEAST:** Returns the smallest value from a list of any number of expressions. Comparison is case sensitive. If datatypes of all the expressions in the list are not same, rest all expressions are converted to the datatype of the first expression for comparison and if this conversion is not possible, SQL will throw an error.

NOTE: Returns null if any expression in the list is null.

Syntax:

LEAST(expr1, expr2 [,])

-

Input:

```
SELECT LEAST('XYZ', 'xyz')
from dual;
```

Output:

```
LEAST('XYZ', 'xyz')
XYZ
```

Explanation: ASCII value of capital alphabets is smaller.

-

Input:

```
SELECT LEAST('XYZ', null, 'xyz')
from dual;
```

Output:

```
LEAST('XYZ', null, 'xyz')
```

Explanation: Since null is present hence, null will be shown as output (as mentioned to note in description above).

8. **NULIF:** Returns a null value if value1=value2, otherwise it returns value1.

Syntax:

```
NULIF(value1, value2)
```

Example:

Input:

```
SELECT NULIF(9995463931, contact_num)
from Employee;
```

Output:

Result
NULLIF(9995463931, CONTACT_NUM)
9995463931
9995463951
9995463831

3 rows returned in 0.00 seconds

Explanation: NULL is displayed for the Employee whose number is matched with the given number. For rest of the Employees value1 is returned.

This article is contributed by [akanshgupta](#). If you like GeeksforGeeks and would like to contribute, you can also write an article using contribute.geeksforgeeks.org or mail your article to contribute@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Last Updated : 29 Apr, 2022

11

Similar Reads

1. [SQL SERVER | Conditional Statements](#)

[Trending Now](#) [DSA](#) [Data Structures](#) [Algorithms](#) [Interview Preparation](#) [Data Science](#) [Topic-wise Practice](#) [Jobs](#)

SQL | Character Functions with Examples

[Read](#) [Discuss](#) [Courses](#) [Practice](#)

Character functions accept character inputs and can return either characters or number values as output. SQL provides a number of different character datatypes which includes – CHAR, VARCHAR, VARCHAR2, LONG, RAW, and LONG RAW. The various datatypes are categorized into three different datatypes :

1. **VARCHAR2** – A variable-length character datatype whose data is converted by the RDBMS.
2. **CHAR** – The fixed-length datatype.
3. **RAW** – A variable-length datatype whose data is not converted by the RDBMS, but left in “raw” form.

Note : When a character function returns a character value, that value is always of type VARCHAR2 (variable length), with the following two exceptions: UPPER and LOWER. These functions convert to upper and to lower case, respectively, and return the CHAR values (fixed length) if the strings they are called on to convert are **fixed-length** CHAR arguments.

Character Functions

SQL provides a rich set of character functions that allow you to get information about strings and modify the contents of those strings in multiple ways. Character functions are of the following two types:

1. Case-Manipulative Functions (LOWER, UPPER and INITCAP)
2. Character-Manipulative Functions (CONCAT, LENGTH, SUBSTR, INSTR, LPAD, RPAD, TRIM and REPLACE)

AD

Case-Manipulative Functions

1. LOWER : This function converts alpha character values to lowercase. LOWER will actually return a fixed-length string if the incoming string is fixed-length. LOWER will not change any characters in the string that are not letters, since case is irrelevant for numbers and special characters, such as the dollar sign (\$) or modulus (%).

Syntax:

```
LOWER(SQL course)
```

Input1: SELECT LOWER('GEEKSFORGEEKS') FROM DUAL;

Output1: geeksforgeeks

Input2: SELECT LOWER('DATABASE@456') FROM DUAL;

Output2: database@456

2. UPPER : This function converts alpha character values to uppercase. Also UPPER function too, will actually return a fixed-length string if the incoming string is fixed-length. UPPER will not change any characters in the string that are not letters, since case is irrelevant for numbers and special characters, such as the dollar sign (\$) or modulus (%).

Syntax:

```
UPPER(SQL course)
```

Input1: SELECT UPPER('geeksforgeeks') FROM DUAL;

Output1: GEEKSFORGEEKS

Input2: SELECT UPPER('dbms\$508%7') FROM DUAL;

Output2: DBMS\$508%7

3. INITCAP : This function converts alpha character values to uppercase for the first letter of each word and all others in lowercase. The words in the string is must be separated by either # or _ or space.

Syntax:

```
INITCAP(SQL course)
```

Input1: SELECT INITCAP('geeksforgeeks is a computer science portal for geeks') FROM DUAL;

Output1: Geeksforgeeks Is A Computer Science Portal For Geeks

Input2: SELECT INITCAP('PRACTICE_CODING_FOR_EFFICIENCY') FROM DUAL;

Output2: Practice_Coding_For_Efficiency

Character-Manipulative Functions

1. CONCAT : This function always appends (concatenates) string2 to the end of string1. If either of the string is NULL, CONCAT function returns the non-NULL argument. If both

strings are NULL, CONCAT returns NULL.

Syntax:

```
CONCAT('String1', 'String2')
```

Input1: SELECT CONCAT('computer' , 'science') FROM DUAL;

Output1: computerscience

Input2: SELECT CONCAT(NULL , 'Android') FROM DUAL;

Output2: Android

Input3: SELECT CONCAT(NULL ,NULL) FROM DUAL;

Output3: -

2. **LENGTH :** This function returns the length of the input string. If the input string is NULL, then LENGTH function returns NULL and not Zero. Also, if the input string contains extra spaces at the start, or in between or at the end of the string, then the LENGTH function includes the extra spaces too and returns the complete length of the string.

Syntax:

```
LENGTH(Column|Expression)
```

Input1: SELECT LENGTH('Learning Is Fun') FROM DUAL;

Output1: 15

Input2: SELECT LENGTH(' Write an Interview Experience ') FROM DUAL;

Output2: 34

Input3: SELECT LENGTH('') FROM DUAL; or SELECT LENGTH(NULL) FROM DUAL;

Output3: -

3. **SUBSTR :** This function returns a portion of a string from a given start point to an end point. If a substring length is not given, then SUBSTR returns all the characters till the end of string (from the starting position specified).

Syntax:

```
SUBSTR('String',start-index,length_of_extracted_string)
```

Input1: SELECT SUBSTR('Database Management System', 9) FROM DUAL;

Output1: Management System

Input2: SELECT SUBSTR('Database Management System', 9, 7) FROM DUAL;

Output2: Manage

4. **INSTR :** This function returns numeric position of a character or a string in a given string. Optionally, you can provide a position *m* to start searching, and the occurrence *n* of string.

Also, if the starting position is not given, then it starts search from index 1, by default. If after searching in the string, no match is found then, INSTR function returns 0.

Syntax: INSTR(Column|Expression, 'String', [,m], [n])

Input: SELECT INSTR('Google apps are great applications','app',1,2) FROM DUAL;
Output: 23

5. LPAD and RPAD : These functions return the strings padded to the left or right (as per the use) ; hence the “L” in “LPAD” and the “R” in “RPAD” ; to a specified length, and with a specified pad string. If the pad string is not specified, then the given string is padded on the left or right (as per the use) with spaces.

Syntax:

LPAD(Column|Expression, n, 'String')

Syntax: RPAD(Column|Expression, n, 'String')

LPAD Input1: SELECT LPAD('100',5,'*') FROM DUAL;

LPAD Output1: **100

LPAD Input2: SELECT LPAD('hello', 21, 'geek') FROM DUAL;

LPAD Output2: geekgeekgeekgeekhello

RPAD Input1: SELECT RPAD('5000',7,'*') FROM DUAL;

RPAD Output1: 5000***

RPAD Input1: SELECT RPAD('earn', 19, 'money') FROM DUAL;

RPAD Output1: earnmoneymoneymoney

6. TRIM : This function trims the string input from the start or end (or both). If no string or char is specified to be trimmed from the string and there exists some extra space at start or end of the string, then those extra spaces are trimmed off.

Syntax:

TRIM(Leading|Trailing|Both, trim_character FROM trim_source)

Input1: SELECT TRIM('G' FROM 'GEEKS') FROM DUAL;

Output1: EEEKS

Input2: SELECT TRIM(' geeksforgeeks ') FROM DUAL;

Output2:geeksforgeeks

7. REPLACE : This function searches for a character string and, if found, replaces it with a given replacement string at all the occurrences of the string. REPLACE is useful for searching patterns of characters and then changing all instances of that pattern in a single

function call.

If a replacement string is not given, then REPLACE function removes all the occurrences of that character string in the input string. If neither a match string nor a replacement string is specified, then REPLACE returns NULL.

Syntax:

```
REPLACE(Text, search_string, replacement_string)
```

Input1: SELECT REPLACE('DATA MANAGEMENT', 'DATA', 'DATABASE') FROM DUAL;

Output1: DATABASE MANAGEMENT

Input2: SELECT REPLACE('abcdeabcccabdddeeabcc', 'abc') FROM DUAL;

Output2: deccabdddeec

This article is contributed by **Anshika Goyal**. If you like GeeksforGeeks and would like to contribute, you can also write an article using contribute.geeksforgeeks.org or mail your article to contribute@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Last Updated : 21 Mar, 2018

20

Similar Reads

1. [SQL | Functions \(Aggregate and Scalar Functions\)](#)

2. [SQL | Difference between functions and stored procedures in PL/SQL](#)

3. [Difference between Structured Query Language \(SQL\) and Transact-SQL \(T-SQL\)](#)

4. [Configure SQL Jobs in SQL Server using T-SQL](#)

5. [SQL | Date functions](#)

6. [SQL | NULL functions](#)

7. [SQL general functions | NVL, NVL2, DECODE, COALESCE, NULLIF, LNNVL and NANVL](#)

8. [SQL Server Mathematical functions \(SQRT, PI, SQUARE, ROUND, CEILING & FLOOR\)](#)

9. [SQL | Numeric Functions](#)



SQL | Date functions

[Read](#) [Discuss](#) [Courses](#) [Practice](#)

In [SQL](#), dates are complicated for newbies, since while working with a database, the format of the data in the table must be matched with the input data to insert. In various scenarios instead of date, datetime (time is also involved with date) is used.

For storing a date or a date and time value in a database, [MySQL](#) offers the following data types:

DATE	format YYYY-MM-DD
DATETIME	format: YYYY-MM-DD HH:MI: SS
TIMESTAMP	format: YYYY-MM-DD HH:MI: SS
YEAR	format YYYY or YY

Now, come to some popular functions in SQL date functions.

NOW()

Returns the current date and time.

AD

Query:

```
SELECT NOW();
```

Output:

Number of Records: 1

NOW()

2023-04-04 07:29:38

CURDATE()

Returns the current date.

Query:

```
SELECT CURDATE();
```

Output:

Number of Records: 1

CURDATE()

2023-04-04

CURTIME()

Returns the current time.

Query:

```
SELECT CURTIME();
```

Output:

Number of Records: 1

CURTIME()

07:32:24

DATE()

Extracts the date part of a date or date/time expression. Example: For the below table named 'Test'

Id	Name	BirthTime
4120	Pratik	1996-09-26 16:44:15.581

Query:

```
SELECT Name, DATE(BirthTime)
AS BirthDate FROM Test;
```

Output:

Name	BirthDate
Pratik	1996-09-26

EXTRACT()

Returns a single part of a date/time.

Syntax

EXTRACT(unit FROM date);

Several units can be considered but only some are used such as **MICROSECOND**, **SECOND**, **MINUTE**, **HOUR**, **DAY**, **WEEK**, **MONTH**, **QUARTER**, **YEAR**, etc. And 'date' is a valid date expression. Example: For the below table named 'Test'

Id	Name	BirthTime
4120	Pratik	1996-09-26 16:44:15.581

Query:

```
SELECT Name, Extract(DAY FROM
BirthTime) AS BirthDay FROM Test;
```

Output:

Name	Birthday
Pratik	26

Query:

```
SELECT Name, Extract(YEAR FROM BirthTime)
AS BirthYear FROM Test;
```

Output:

Name	BirthYear
Pratik	1996

Query:

```
SELECT Name, Extract(SECOND FROM
BirthTime) AS BirthSecond FROM Test;
```

Output:

Name	BirthSecond
Pratik	581

DATE_ADD()

Adds a specified time interval to a date.

Syntax:

```
DATE_ADD(date, INTERVAL expr type);
```

Where, date – valid date expression, and expr is the number of intervals we want to add. and type can be one of the following: MICROSECOND, SECOND, MINUTE, HOUR, DAY, WEEK, MONTH, QUARTER, YEAR, etc. Example: For the below table named ‘Test’

Id	Name	BirthTime
4120	Pratik	1996-09-26 16:44:15.581

Query:

```
SELECT Name, DATE_ADD(BirthTime, INTERVAL  
1 YEAR) AS BirthTimeModified FROM Test;
```

Output:

Name	BirthTimeModified
Pratik	1997-09-26 16:44:15.581

Query:

```
SELECT Name, DATE_ADD(BirthTime,  
INTERVAL 30 DAY) AS BirthDayModified FROM Test;
```

Output:

Name	BirthDayModified
Pratik	1996-10-26 16:44:15.581

Query:

```
SELECT Name, DATE_ADD(BirthTime, INTERVAL  
4 HOUR) AS BirthHourModified FROM Test;
```

Output:

Name	BirthSecond
Pratik	1996-10-26 20:44:15.581

DATE_SUB()

Subtracts a specified time interval from a date. The syntax for DATE_SUB is the same as DATE_ADD just the difference is that DATE_SUB is used to subtract a given interval of date.

DATEDIFF()

Returns the number of days between two dates.

Syntax:

DATEDIFF(date1, date2);
date1 & date2- date/time expression

Query:

```
SELECT DATEDIFF('2017-01-13','2017-01-03') AS DateDiff;
```

Output:

DateDiff
10

DATE_FORMAT()

Displays date/time data in different formats.

Syntax:

DATE_FORMAT(date,format);

the date is a valid date and the format specifies the output format for the date/time. The formats that can be used are:

- %a-Abbreviated weekday name (Sun-Sat)
- %b-Abbreviated month name (Jan-Dec)
- %c-Month, numeric (0-12)
- %D-Day of month with English suffix (0th, 1st, 2nd, 3rd)
- %d-Day of the month, numeric (00-31)
- %e-Day of the month, numeric (0-31)
- %f-Microseconds (000000-999999)
- %H-Hour (00-23)
- %h-Hour (01-12)
- %l-Hour (01-12)
- %i-Minutes, numeric (00-59)
- %j-Day of the year (001-366)
- %k-Hour (0-23)
- %l-Hour (1-12)
- %M-Month name (January-December)
- %m-Month, numeric (00-12)
- %p-AM or PM
- %r-Time, 12-hour (hh:mm: ss followed by AM or PM)
- %S-Seconds (00-59)
- %s-Seconds (00-59)
- %T-Time, 24-hour (hh:mm: ss)
- %U-Week (00-53) where Sunday is the first day of the week
- %u-Week (00-53) where Monday is the first day of the week
- %V-Week (01-53) where Sunday is the first day of the week, used with %X
- %v-Week (01-53) where Monday is the first day of the week, used with %x
- %W-Weekday name (Sunday-Saturday)
- %w-Day of the week (0=Sunday, 6=Saturday)
- %X-Year for the week where Sunday is the first day of the week, four digits, used with %V
- %x-Year for the week where Monday is the first day of the week, four digits, used with %v
- %Y-Year, numeric, four digits
- %y-Year, numeric, two digits

This article is contributed by [Pratik Agarwal](#). If you like GeeksforGeeks and would like to contribute, you can also write an article using write.geeksforgeeks.org or mail your article to review-team@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks. Please write comments if you find anything incorrect, or if you want to share more information about the topic discussed above.

Similar Reads

[Trending Now](#) [DSA](#) [Data Structures](#) [Algorithms](#) [Interview Preparation](#) [Data Science](#) [Topic-wise Practice](#) [Jobs](#)

SQL | Date Functions (Set-1)



Sakshi98

[Read](#) [Discuss](#) [Courses](#) [Practice](#)

In SQL, dates are complicated for newbies, since while working with a database, the format of the date in the table must be matched with the input date in order to insert. In various scenarios instead of date, datetime (time is also involved with date) is used.

Some of the important date functions have been already discussed in the previous [post](#). The basic idea of this post is to know the working or syntax of all the date functions:

Below are the date functions that are used in SQL:

1. **ADDDATE()**: It returns a date after a certain time/date interval has been added.

Syntax: `SELECT ADDTIME("2018-07-16 02:52:47", "2");`

Output: 2018-07-16 02:52:49

2. **ADDTIME()**: It returns a time / date time after a certain time interval has been added.

Syntax: `SELECT ADDTIME("2017-06-15 09:34:21", "2");`

Output: 2017-06-15 09:34:23

3. **CURDATE()**: It returns the current date.

Syntax: `SELECT CURDATE();`

Output: 2018-07-16

4. **CURRENT_DATE()**: It returns the current date.

Syntax: `SELECT CURRENT_DATE();`

Output: 2018-07-16

5. **CURRENT_TIME()**: It returns the current time.

Syntax: `SELECT CURRENT_TIME();`

Output: 02:53:15

6. **CURRENT_TIMESTAMP()**: It returns the current date and time.

Syntax: `SELECT CURRENT_TIMESTAMP();`

Output: 2018-07-16 02:53:21

7. **CURTIME()**: It returns the current time.

Syntax: `SELECT CURTIME();`

Output: 02:53:28

8. **DATE()**: It extracts the date value from a date or date time expression.

Syntax: `SELECT DATE("2017-06-15");`

Output: 2017-06-15

9. **DATEDIFF()**: It returns the difference in days between two date values.

Syntax: `SELECT DATEDIFF("2017-06-25", "2017-06-15");`

Output: 10

10. **DATE_ADD()**: It returns a date after a certain time/date interval has been added.

Syntax: `SELECT DATE_ADD("2018-07-16", INTERVAL 10 DAY);`

Output: 2018-07-16

11. **DATE_FORMAT()**: It formats a date as specified by a format mask.

Syntax: `SELECT DATE_FORMAT("2018-06-15", "%Y");`

Output: 2018

12. **DATE_SUB()**: It returns a date after a certain time/date interval has been subtracted.

Syntax: `SELECT DATE_SUB("2017-06-15", INTERVAL 10 DAY);`

Output: 2018-07-16

13. **DAY()**: It returns the day portion of a date value.

Syntax: `SELECT DAY("2018-07-16");`

Output: 16

14. **DAYNAME()**: It returns the weekday name for a date.

Syntax: `SELECT DAYNAME('2008-05-15');`

Output: Thursday

15. **DAYOFMONTH()**: It returns the day portion of a date value.

Syntax: `SELECT DAYOFMONTH('2018-07-16');`

Output: 16

16. **DAYWEEK()**: It returns the weekday index for a date value.

Syntax: `SELECT WEEKDAY("2018-07-16");`

Output: 0

17. **DAYOFYEAR()**: It returns the day of the year for a date value.

Syntax: `SELECT DAYOFYEAR("2018-07-16");`

Output: 197

18. **EXTRACT()**: It extracts parts from a date.

Syntax: `SELECT EXTRACT(MONTH FROM "2018-07-16");`

Output: 7

19. **FROM_DAYS()**: It returns a date value from a numeric representation of the day.

Syntax: `SELECT FROM_DAYS(685467);`

Output: 1876-09-29

20. **HOUR()**: It returns the hour portion of a date value.

Syntax: `SELECT HOUR("2018-07-16 09:34:00");`

Output: 9

21. **LAST_DAY()**: It returns the last day of the month for a given date.

Syntax: `SELECT LAST_DAY('2018-07-16');`

Output: 2018-07-31

22. **LOCALTIME()**: It returns the current date and time.

Syntax: `SELECT LOCALTIME();`

Output: 2018-07-16 02:56:42

23. **LOCALTIMESTAMP()**: It returns the current date and time.

Syntax: `SELECT LOCALTIMESTAMP();`

Output: 2018-07-16 02:56:48

24. **MAKEDATE()**: It returns the date for a certain year and day-of-year value.

Syntax: `SELECT MAKEDATE(2009, 138);`

Output: 2009-05-18

25. **MAKETIME()**: It returns the time for a certain hour, minute, second combination.

Syntax: `SELECT MAKETIME(11, 35, 4);`



SQL | Date Functions (Set-2)



Sakshi98

[Read](#) [Discuss](#) [Courses](#) [Practice](#)

In SQL, dates are complicated for newbies, since while working with a database, the format of the date in the table must be matched with the input date in order to insert. In various scenarios instead of date, datetime (time is also involved with date) is used.

Some of the date functions have been already discussed in the [Set-1](#). In this post, the remaining date functions have been discussed.

Below are the remaining date functions that are used in SQL:

1. **MICROSECOND()**: It returns the microsecond portion of a date value.

Syntax: `SELECT MICROSECOND("2018-07-18 09:12:00.000345");`

Output: 345

2. **MINUTE()**: It returns the minute portion of a date value.

Syntax: `SELECT MINUTE("2018-07-18 09:12:00");`

Output: 12

3. **MONTH()**: It returns the month portion of a date value.

Syntax: `SELECT MONTH ('2018/07/18')AS MONTH;`

Output: 7

4. **MONTHNAME()**: It returns the full month name for a date.

Syntax: `SELECT MONTHNAME("2018/07/18");`

Output: JULY

5. **NOW()**: It returns the current date and time.

Syntax: `SELECT NOW();`

Output: 2018-07-18 09:14:32

6. **PERIOD_ADD()**: It takes a period and adds a specified number of months to it.

Syntax: `SELECT PERIOD_ADD(201803, 6);`

Output: 201809

7. **PERIOD_DIFF()**: It returns the difference in months between two periods.

Syntax: `SELECT PERIOD_DIFF(201810, 201802);`

Output: 8

8. **QUARTER()**: It returns the quarter portion of a date value.

Syntax: `SELECT QUARTER("2018/07/18");`

Output: 3

9. **SECOND()**: It returns the second portion of a date value.

Syntax: `SELECT SECOND("09:14:00:00032");`

Output: 0

10. **SEC_TO_TIME()**: It converts numeric seconds into a time value.

Syntax: `SELECT SEC_TO_TIME(1);`

Output: 00:00:01

11. **STR_TO_DATE()**: It takes a string and returns a date specified by a format mask.

Syntax: `SELECT STR_TO_DATE("JULY 18 2018", "%M %D %Y");`

Output: 2018-07-18

12. **SUBDATE()**: It returns a date after which a certain time/date interval has been subtracted.

Syntax: `SELECT SUBDATE("2017-06-15", INTERVAL 10 DAY);`

Output: 2017-06-05

13. **SUBTIME()**: It returns a time/date time value after a certain time interval has been subtracted.

Syntax: `SELECT SUBTIME("2018/07/18", INTERVAL 10 DAY);`

Output: 2018-07-18 09:15:17.542768

14. **SYSDATE()**: It returns the current date and time.

Syntax: `SELECT SYSDATE();`

Output: 2018-07-18 09:19:03

15. **TIME()**: It extracts the time value from a time/date time expression.

Syntax: `SELECT TIME("09:16:10");`

Output: 09:16:10

16. **TIME_FORMAT()**: It formats the time as specified by a format mask.

Syntax: `SELECT TIME_FORMAT("09:16:10", "%H %I %S");`

Output: 09 09 10

17. **TIME_TO_SEC()**: It converts a time value into numeric seconds.

Syntax: `SELECT TIME_TO_SEC("09:16:10");`

Output: 33370

18. **TIMEDIFF()**: It returns the difference between two time/datetime values.

Syntax: `SELECT TIMEDIFF("09:16:10", "09:16:04");`

Output: 00:00:06

19. **TIMESTAMP()**: It converts an expression to a date time value and if specified adds an optional time interval to the value.

Syntax: `SELECT TIMESTAMP("2018-07-18", "09:16:10");`

Output: 2018-07-18 09:16:10

20. **TO_DAYS()**: It converts a date into numeric days.

Syntax: `SELECT TO_DAYS("2018-07-18");`

Output: 737258

21. **WEEK()**: It returns the week portion of a date value.

Syntax: `SELECT WEEK("2018-07-18");`

Output: 28

22. **WEEKDAY()**: It returns the weekday index for a date value.

Syntax: `SELECT WEEKDAY("2018-07-18");`

Output: 2

23. **WEEKOFYEAR()**: It returns the week of the year for a date value.

Syntax: `SELECT WEEKOFYEAR("2018-07-18");`

Output: 29

24. **YEAR()**: It returns the year portion of a date value.

Syntax: `SELECT YEAR("2018-07-18");`

Output: 2018

25. **YEARWEEK()**: It returns the year and week for a date value.

Syntax: SELECT YEARWEEK("2018-07-18");

Output: 201828

Last Updated : 19 Jul, 2018

4

Similar Reads

1. [SQL | Date Functions \(Set-1\)](#)
2. [How to Write a SQL Query For a Specific Date Range and Date Time?](#)
3. [SQL | Date functions](#)
4. [Useful Date and Time Functions in PL/SQL](#)
5. [Date manipulating functions in SQL](#)
6. [SQL Query to Check if Date is Greater Than Today in SQL](#)
7. [SQL | Functions \(Aggregate and Scalar Functions\)](#)
8. [SQL | Difference between functions and stored procedures in PL/SQL](#)
9. [Difference between Structured Query Language \(SQL\) and Transact-SQL \(T-SQL\)](#)
10. [Configure SQL Jobs in SQL Server using T-SQL](#)

Previous

Next

Article Contributed By :



Sakshi98

Sakshi98

Vote for difficulty

Current difficulty : [Easy](#)

Article Tags : [SQL-Functions](#), [SQL](#)

Practice Tags : [SQL](#)



SQL | LISTAGG

[Read](#)
[Discuss](#)
[Courses](#)
[Practice](#)

LISTAGG function in DBMS is used to aggregate strings from data in columns in a database table.

- It makes it very easy to concatenate strings. It is similar to concatenation but uses grouping.
- The speciality about this function is that, it also allows to order the elements in the concatenated list.

Syntax:

```
LISTAGG (measure_expr [, 'delimiter']) WITHIN GROUP
(order_by_clause) [OVER query_partition_clause]
measure_expr : The column or expression to concatenate the values.
delimiter : Character in between each measure_expr, which is by default a comma
(,) .
order_by_clause : Order of the concatenated values.
```

Let us have a table named Gfg having two columns showing the subject names and subject number that each subject belongs to, as shown below :

```
SQL> select * from GfG;
```

SUBNO	SUBNAME
D20	Algorithm
D30	DataStructure
D30	C
D20	C++
D30	Python
D30	DBMS
D10	LinkedList
D20	Matrix
D10	String
D30	Graph
D20	Tree

```
11 rows selected.
```

Query 1: Write an SQL query using LISTAGG function to output the subject names in a single field with the values comma delimited.

```
AD
```

```
SQL> SELECT LISTAGG(SubName, ' , ') WITHIN GROUP (ORDER BY SubName) AS SUBJECTS
  2  FROM    GfG ;
```

Output:

```
SUBJECTS
-----
-- 
Algorithm , C , C++ , DBMS , DataStructure , Graph , LinkedList , Matrix , Python
,
String , Tree
```

Query 2: Write an SQL query to group each subject and show each subject in its respective department separated by comma with the help of LISTAGG function.

```
SQL> SELECT SubNo, LISTAGG(SubName, ' , ') WITHIN GROUP (ORDER BY SubName) AS
SUBJECTS
  2  FROM    GfG
  3  GROUP BY SubNo;
```

Output:

```
SUBNO      SUBJECTS
-----      -----
D10        LinkedList , String
D20        Algorithm , C++ , Matrix , Tree
D30        C , DBMS , DataStructure , Graph , Python
```

Query 3: Write an SQL query to show the subjects belonging to each department ordered by the subject number (SUBNO) with the help of LISTAGG function.

```
SQL> SELECT SubNo, LISTAGG(SubName, ',' ,') WITHIN GROUP (ORDER BY SubName) AS
SUBJECTS
2  FROM GfG
3  GROUP BY SubNo
4  ORDER BY SubNo;
```

Output:

SUBNO	SUBJECTS
D10	LinkedList, String
D20	Algorithm, C++, Matrix, Tree
D30	C, DBMS, DataStructure, Graph, Python

This article is contributed by MAZHAR IMAM KHAN. If you like GeeksforGeeks and would like to contribute, you can also write an article using contribute.geeksforgeeks.org or mail your article to contribute@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Last Updated : 22 Jun, 2018

12

Similar Reads

1. Difference between Structured Query Language (SQL) and Transact-SQL (T-SQL)

2. Configure SQL Jobs in SQL Server using T-SQL

3. SQL | Procedures in PL/SQL

4. SQL | Difference between functions and stored procedures in PL/SQL

5. SQL SERVER – Input and Output Parameter For Dynamic SQL

6. Difference between SQL and T-SQL

7. SQL Server | Convert tables in T-SQL into XML

8. SQL SERVER | Bulk insert data from csv file using T-SQL command

9. SQL - SELECT from Multiple Tables with MS SQL Server



Aggregate functions in SQL

[Read](#)[Discuss](#)

In database management an aggregate function is a function where the values of multiple rows are grouped together as input on certain criteria to form a single value of more significant meaning.

Various Aggregate Functions

- 1) Count()
- 2) Sum()
- 3) Avg()
- 4) Min()
- 5) Max()

AD

Now let us understand each Aggregate function with a example:

Id	Name	Salary
<hr/>		
1	A	80
2	B	40
3	C	60
4	D	70
5	E	60
6	F	Null

Count():

Count(*): Returns total number of records .i.e 6.

Count(salary): Return number of Non Null values over the column salary. i.e 5.

Count(Distinct Salary): Return number of distinct Non Null values over the column salary .i.e 4

Sum():

sum(salary): Sum all Non Null values of Column salary i.e., 310

sum(Distinct salary): Sum of all distinct Non-Null values i.e., 250.

Avg():

Avg(salary) = Sum(salary) / count(salary) = 310/5

Avg(Distinct salary) = sum(Distinct salary) / Count(Distinct Salary) = 250/4

Min():

Min(salary): Minimum value in the salary column except NULL i.e., 40.

Max(salary): Maximum value in the salary i.e., 80.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Last Updated : 20 Aug, 2019

115

Similar Reads

1. SQL | Functions (Aggregate and Scalar Functions)
2. Aggregate functions in Cassandra
3. SQL | Difference between functions and stored procedures in PL/SQL
4. SQL vs NO SQL vs NEW SQL



SQL | Functions (Aggregate and Scalar Functions)

[Read](#) [Discuss](#) [Courses](#) [Practice](#)

For doing operations on data SQL has many built-in functions, they are categorized in two categories and further sub-categorized in different seven functions under each category. The categories are:

1. Aggregate functions:

These functions are used to do operations from the values of the column and a single value is returned.

1. AVG()
2. COUNT()
3. FIRST()
4. LAST()
5. MAX()
6. MIN()
7. SUM()

2. Scalar functions:

These functions are based on user input, these too returns single value.

1. UCASE()
2. LCASE()
3. MID()
4. LEN()
5. ROUND()
6. NOW()
7. FORMAT()

Students-Table

ID	NAME	MARKS	AGE
1	Harsh	90	19
2	Suresh	50	20
3	Pratik	80	19
4	Dhanraj	95	21
5	Ram	85	18

Aggregate Functions

AD

AVG(): It returns the average value after calculating from values in a numeric column.

Syntax:

```
SELECT AVG(column_name) FROM table_name;
```

Queries:

- Computing average marks of students.

```
SELECT AVG(MARKS) AS AvgMarks FROM Students;
```

Output:

AvgMarks
80

- Computing average age of students.

```
SELECT AVG(AGE) AS AvgAge FROM Students;
```

Output:

AvgAge
19.4

COUNT(): It is used to count the number of rows returned in a SELECT statement. It can't be used in MS ACCESS.

Syntax:

```
SELECT COUNT(column_name) FROM table_name;
```

Queries:

- Computing total number of students.

```
SELECT COUNT(*) AS NumStudents FROM Students;
```

Output:

NumStudents
5

- Computing number of students with unique/distinct age.

```
SELECT COUNT(DISTINCT AGE) AS NumStudents FROM Students;
```

Output:

NumStudents
4

FIRST(): The FIRST() function returns the first value of the selected column.

Syntax:

```
SELECT FIRST(column_name) FROM table_name;
```

Queries:

- Fetching marks of first student from the Students table.

```
SELECT FIRST(MARKS) AS MarksFirst FROM Students;
```

Output:

MarksFirst
90

- Fetching age of first student from the Students table.

```
SELECT FIRST(AGE) AS AgeFirst FROM Students;
```

Output:

AgeFirst
19

LAST(): The LAST() function returns the last value of the selected column. It can be used only in MS ACCESS.

Syntax:

```
SELECT LAST(column_name) FROM table_name;
```

Queries:

- Fetching marks of last student from the Students table.

```
SELECT LAST(MARKS) AS MarksLast FROM Students;
```

Output:

MarksLast
85

- Fetching age of last student from the Students table.

```
SELECT LAST(AGE) AS AgeLast FROM Students;
```

Output:

AgeLast
18

MAX(): The MAX() function returns the maximum value of the selected column.

Syntax:

```
SELECT MAX(column_name) FROM table_name;
```

Queries:

- Fetching maximum marks among students from the Students table.

```
SELECT MAX(MARKS) AS MaxMarks FROM Students;
```

Output:

MaxMarks
95

- Fetching max age among students from the Students table.

```
SELECT MAX(AGE) AS MaxAge FROM Students;
```

Output:

MaxAge
21

MIN(): The MIN() function returns the minimum value of the selected column.

Syntax:

```
SELECT MIN(column_name) FROM table_name;
```

Queries:

- Fetching minimum marks among students from the Students table.

```
SELECT MIN(MARKS) AS MinMarks FROM Students;
```

Output:

MinMarks
50

- Fetching minimum age among students from the Students table.

```
SELECT MIN(AGE) AS MinAge FROM Students;
```

Output:

MinAge
18

SUM(): The SUM() function returns the sum of all the values of the selected column.

Syntax:

```
SELECT SUM(column_name) FROM table_name;
```

Queries:

- Fetching summation of total marks among students from the Students table.

```
SELECT SUM(MARKS) AS TotalMarks FROM Students;
```

Output:

TotalMarks
400

- Fetching summation of total age among students from the Students table.

```
SELECT SUM(AGE) AS TotalAge FROM Students;
```

Output:

TotalAge
97

Scalar Functions

UCASE(): It converts the value of a field to uppercase.

Syntax:

```
SELECT UCASE(column_name) FROM table_name;
```

Queries:

- Converting names of students from the table Students to uppercase.

```
SELECT UCASE(NAME) FROM Students;
```

Output:

NAME
HARSH
SURESH
PRATIK
DHANRAJ
RAM

LCASE(): It converts the value of a field to lowercase.

Syntax:

```
SELECT LCASE(column_name) FROM table_name;
```

Queries:

- Converting names of students from the table Students to lowercase.

```
SELECT LCASE(NAME) FROM Students;
```

Output:

NAME
harsh
suresh
pratik
dhanraj
ram

MID(): The MID() function extracts texts from the text field.

Syntax:

```
SELECT MID(column_name,start,length) AS some_name FROM table_name;
```

specifying length is optional here, and start signifies start position (starting from 1)

Queries:

- Fetching first four characters of names of students from the Students table.

```
SELECT MID(NAME,1,4) FROM Students;
```

Output:

NAME
HARS
SURE

PRAT
DHAN
RAM

LEN(): The LEN() function returns the length of the value in a text field.

Syntax:

```
SELECT LENGTH(column_name) FROM table_name;
```

Queries:

- Fetching length of names of students from Students table.

```
SELECT LENGTH(NAME) FROM Students;
```

Output:

NAME
5
6
6
7
3

ROUND(): The ROUND() function is used to round a numeric field to the number of decimals specified.
NOTE: Many database systems have adopted the IEEE 754 standard for arithmetic operations, which says that when any numeric .5 is rounded it results to the nearest even integer i.e, 5.5 and 6.5 both gets rounded off to 6.

Syntax:

```
SELECT ROUND(column_name,decimals) FROM table_name;
```

decimals- number of decimals to be fetched.

Queries:

- Fetching maximum marks among students from the Students table.

```
SELECT ROUND(MARKS,0) FROM table_name;
```

Output:

MARKS
90
50
80
95
85

NOW(): The NOW() function returns the current system date and time.

Syntax:

```
SELECT NOW() FROM table_name;
```

Queries:

- Fetching current system time.

```
SELECT NAME, NOW() AS DateTime FROM Students;
```

Output:

NAME	DateTime
HARSH	1/13/2017 1:30:11 PM
SURESH	1/13/2017 1:30:11 PM
PRATIK	1/13/2017 1:30:11 PM

DHANRAJ	1/13/2017 1:30:11 PM
RAM	1/13/2017 1:30:11 PM

FORMAT(): The FORMAT() function is used to format how a field is to be displayed.

Syntax:

```
SELECT FORMAT(column_name,format) FROM table_name;
```

Queries:

- Formatting current date as 'YYYY-MM-DD'.

```
SELECT NAME, FORMAT(Now(), 'YYYY-MM-DD') AS Date FROM Students;
```

Output:

NAME	Date
HARSH	2017-01-13
SURESH	2017-01-13
PRATIK	2017-01-13
DHANRAJ	2017-01-13
RAM	2017-01-13

This article is contributed by [Pratik Agarwal](#). If you like GeeksforGeeks and would like to contribute, you can also write an article using [write.geeksforgeeks.org](#) or mail your article to review-team@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Similar Reads



SQL | NULL functions

[Read](#)[Discuss](#)[Courses](#)[Practice](#)

In the database, null values serve as placeholders for data that is either missing or not available. A null value is a flexible data type that can be placed in the column of any data type, including string, int, blob, and CLOB datatypes. It is not a component of any specific data type. Null values are helpful when cleaning the data prior to exploratory analysis.

Null values assist us in eradicating data ambiguity. Null values are also useful for maintaining a consistent datatype across the column. We will learn about the necessity and guidelines for using Null values in this article. Now let's use examples to try to better understand null values and null functions in SQL.

Why do We Need NULL Values?

Null functions are required to perform operations on null values stored in the database. With NULL values, we can perform operations that clearly identify whether the value is null or not. With this ability to recognize null data, operations similar to SQL's join methods can be performed on them.

Following are the NULL functions defined in SQL:

AD

ISNULL()

The ISNULL function has different uses in SQL Server and MySQL. In SQL Server, ISNULL() function is used to replace NULL values.

Syntax:

SELECT column(s), ISNULL(column_name, value_to_replace)

FROM table_name;

Example: Consider the following Employee table,

Name	Salary
John	8000
William	NULL

Find the sum of the salary of all Employees, if the Salary of any employee is not available (or NULL value), use salary as 10000.

Query:

```
SELECT SUM(ISNULL(Salary, 10000) AS Salary
FROM Employee;
```

Output:

Salary
18000

In MySQL, ISNULL() function is used to test whether an expression is NULL or not. If the expression is NULL it returns TRUE, else FALSE.

Syntax:

SELECT column(s)

FROM table_name

WHERE ISNULL(column_name);

Example: Consider the following Employee table

Name	Salary
John	8000
William	NULL

Fetch the name of all employees whose salary is available in the table (not NULL).

Query:

```
SELECT Name
FROM Employee
WHERE ISNULL(Salary);
```

Output:

Name
William

IFNULL()

This function is available in MySQL, and not in SQL Server or Oracle. This function takes two arguments. If the first argument is not NULL, the function returns the first argument. Otherwise, the second argument is returned. This function is commonly used to replace NULL value with another value.

Syntax:

```
SELECT column(s), IFNULL(column_name, value_to_replace)
FROM table_name;
```

Example: Consider the following Employee table,

Name	Salary
John	8000
William	NULL

Find the sum of the salary of all Employees, if the Salary of any employee is not available (or NULL value), use salary as 10000.

Query:

```
SELECT SUM(IFNULL(Salary, 10000)) AS Salary
FROM Employee;
```

Output:

Salary
18000

COALESCE()

COALESCE function in SQL returns the first non-NULL expression among its arguments. If all the expressions evaluate to null, then the COALESCE function will return null.

Syntax:

SELECT column(s), COALESCE(expression_1,...,expression_n)

FROM table_name;

Example:

Consider the following Contact_info table,

Name	Phone1	Phone2
John	1234567897	1258741235
William	NULL	7897894561

Fetch the name and contact number of each employee.

Query:

```
SELECT Name, COALESCE(Phone1, Phone2) AS Contact
FROM Contact_info;
```

Output:

Name	Contact
John	1258741235
William	7897894561

NULLIF()

The NULLIF function takes two arguments. If the two arguments are equal, then NULL is returned. Otherwise, the first argument is returned.

Syntax:

SELECT column(s), NULLIF(expression1, expression2)

FROM table_name;

Example: Consider the following Sales table

Store	Actual	Goal
Store_A	50	50
Store_B	80	100

```
SELECT Store, NULLIF(Actual, Goal)
FROM Sales;
```

Output:

Store	NULLIF (Actual, Goal)
Store_A	NULL
Store_B	80

Conclusion

In this article, we learned what null values are and why we should use them. We now know that using NULL values is fundamental to databases and is done so in order to preserve their integrity. Following this, we learned more about the different functions that can be used with NULL values.

This article is contributed by [Anuj Chauhan](#). If you like GeeksforGeeks and would like to contribute, you can also write an article using [write.geeksforgeeks.org](#) or mail your article to review-team@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Last Updated : 22 May, 2023

3

Similar Reads

1. How to Alter a Column from Null to Not Null in SQL Server?

[Trending Now](#) [DSA](#) [Data Structures](#) [Algorithms](#) [Interview Preparation](#) [Data Science](#) [Topic-wise Practice](#) [Jobs](#)

SQL | Numeric Functions



Sakshi98

[Read](#) [Discuss](#) [Courses](#) [Practice](#)

Numeric Functions are used to perform operations on numbers and return numbers. Following are the numeric functions defined in SQL:

ABS(): It returns the absolute value of a number.

Syntax: `SELECT ABS(-243.5);`

Output: 243.5

```
SQL> SELECT ABS(-10);
+-----+
| ABS(10) |
+-----+
| 10      |
+-----+
```

ACOS(): It returns the cosine of a number, in radians.

AD

Syntax: `SELECT ACOS(0.25);`

Output: 1.318116071652818

ASIN(): It returns the arc sine of a number, in radians.

Syntax: `SELECT ASIN(0.25);`

Output: 0.25268025514207865

ATAN(): It returns the arc tangent of a number, in radians.

Syntax: `SELECT ATAN(2.5);`

Output: 1.1902899496825317

CEIL(): It returns the smallest integer value that is greater than or equal to a number.

Syntax: `SELECT CEIL(25.75);`

Output: 26

CEILING(): It returns the smallest integer value that is greater than or equal to a number.

Syntax: `SELECT CEILING(25.75);`

Output: 26

COS(): It returns the cosine of a number, in radians.

Syntax: `SELECT COS(30);`

Output: 0.15425144988758405

COT(): It returns the cotangent of a number, in radians.

Syntax: `SELECT COT(6);`

Output: -3.436353004180128

DEGREES(): It converts a radian value into degrees.

Syntax: `SELECT DEGREES(1.5);`

Output: 85.94366926962348

```
SQL>SELECT DEGREES(PI());
+-----+
| DEGREES(PI())
+-----+
| 180.000000
+-----+
```

DIV(): It is used for integer division.

Syntax: `SELECT 10 DIV 5;`

Output: 2

EXP(): It returns e raised to the power of a number.

Syntax: `SELECT EXP(1);`

Output: 2.718281828459045

FLOOR(): It returns the largest integer value that is less than or equal to a number.

Syntax: `SELECT FLOOR(25.75);`

Output: 25

GREATEST(): It returns the greatest value in a list of expressions.

Syntax: `SELECT GREATEST(30, 2, 36, 81, 125);`

Output: 125

LEAST(): It returns the smallest value in a list of expressions.

Syntax: `SELECT LEAST(30, 2, 36, 81, 125);`

Output: 2

LN(): It returns the natural logarithm of a number.

Syntax: `SELECT LN(2);`

Output: 0.6931471805599453

LOG10(): It returns the base-10 logarithm of a number.

Syntax: `SELECT LOG(2);`

Output: 0.6931471805599453

LOG2(): It returns the base-2 logarithm of a number.

Syntax: `SELECT LOG2(6);`

Output: 2.584962500721156

MOD(): It returns the remainder (aka. modulus) of n divided by m.

Syntax: `SELECT MOD(18, 4);`

Output: 2

PI(): It returns the value of Pi and displays 6 decimal places.

Syntax: `SELECT PI();`

Output: 3.141593

POWER(m, n): It returns m raised to the nth power.

Syntax: `SELECT POWER(4, 2);`

Output: 16

RADIANS(): It converts a value in degrees to radians.

Syntax: `SELECT RADIANS(180);`

Output: 3.141592653589793

RAND(): It returns a random number between 0 (inclusive) and 1 (exclusive).

Syntax: `SELECT RAND();`

Output: 0.33623238684258644

ROUND(): It returns a number rounded to a certain number of decimal places.

Syntax: `SELECT ROUND(5.553);`

Output: 6

SIGN(): It returns a value indicating the sign of a number. A return value of 1 means positive; 0 means negative.

Syntax: `SELECT SIGN(255.5);`

Output: 1

SIN(): It returns the sine of a number in radians.

Syntax: `SELECT SIN(2);`

Output: 0.9092974268256817

SQRT(): It returns the square root of a number.

Syntax: `SELECT SQRT(25);`

Output: 5

TAN(): It returns the tangent of a number in radians.

Syntax: `SELECT TAN(1.75);`

Output: -5.52037992250933

ATAN2(): It returns the arctangent of the x and y coordinates, as an angle and expressed in radians.

Syntax: `SELECT ATAN2(7);`

Output: 1.42889927219073

TRUNCATE(): This doesn't work for SQL Server. It returns 7.53635 truncated to n places right of the decimal point.

Syntax: `SELECT TRUNCATE(7.53635, 2);`

Output: 7.53

Last Updated : 12 Feb, 2023

11

Similar Reads

1. [SQL | Functions \(Aggregate and Scalar Functions\)](#)
2. [SQL | Difference between functions and stored procedures in PL/SQL](#)
3. [Difference between Structured Query Language \(SQL\) and Transact-SQL \(T-SQL\)](#)
4. [Configure SQL Jobs in SQL Server using T-SQL](#)
5. [Various String, Numeric, and Date & Time functions in MySQL](#)
6. [Numeric and Date-time data types in SQL Server](#)
7. [PHP - My SQL : abs\(\) - numeric function](#)
8. [SQL Query to convert NUMERIC to NVARCHAR](#)
9. [SQL | Date functions](#)
10. [SQL | NULL functions](#)

Previous

Next



SQL | String functions



Sakshi98

[Read](#)[Discuss](#)[Courses](#)[Practice](#)

String functions

are used to perform an operation on input string and return an output string.

Following are the string functions defined in SQL:

1. **ASCII()**: This function is used to find the ASCII value of a character.

Syntax: `SELECT ascii('t');`

Output: 116

2. **CHAR_LENGTH()**: Doesn't work for SQL Server. Use LEN() for SQL Server. This function is used to find the length of a word.

Syntax: `SELECT char_length('Hello!');`

Output: 6

3. **CHARACTER_LENGTH()**: Doesn't work for SQL Server. Use LEN() for SQL Server. This function is used to find the length of a line.

Syntax: `SELECT CHARACTER_LENGTH('geeks for geeks');`

Output: 15

4. **CONCAT()**: This function is used to add two words or strings.

Syntax: `SELECT 'Geeks' || ' ' || 'forGeeks' FROM dual;`

Output: 'GeeksforGeeks'

5. **CONCAT_WS()**: This function is used to add two words or strings with a symbol as concatenating symbol.

Syntax: `SELECT CONCAT_WS('_', 'geeks', 'for', 'geeks');`

Output: geeks_for_geeks

6. **FIND_IN_SET()**: This function is used to find a symbol from a set of symbols.

Syntax: `SELECT FIND_IN_SET('b', 'a, b, c, d, e, f');`

Output: 2

7. **FORMAT()**: This function is used to display a number in the given format.

Syntax: Format("0.981", "Percent");

Output: '98.10%

8. INSERT(): This function is used to insert the data into a database.

Syntax: INSERT INTO database (geek_id, geek_name) VALUES (5000, 'abc');

Output: successfully updated

9. INSTR(): This function is used to find the occurrence of an alphabet.

Syntax: INSTR('geeks for geeks', 'e');

Output: 2 (the first occurrence of 'e')

Syntax: INSTR('geeks for geeks', 'e', 1, 2);

Output: 3 (the second occurrence of 'e')

10. LCASE(): This function is used to convert the given string into lower case.

Syntax: LCASE ("GeeksFor Geeks To Learn");

Output: geeksforgeeks to learn

11. LEFT(): This function is used to SELECT a sub string from the left of given size or characters.

Syntax: SELECT LEFT('geeksforgeeks.org', 5);

Output: geeks

12. LENGTH(): This function is used to find the length of a word.

Syntax: LENGTH('GeeksForGeeks');

Output: 13

13. LOCATE(): This function is used to find the nth position of the given word in a string.

Syntax: SELECT LOCATE('for', 'geeksforgeeks', 1);

Output: 6

14. LOWER(): This function is used to convert the upper case string into lower case.

Syntax: SELECT LOWER('GEEKSFORGEEKS.ORG');

Output: geeksforgeeks.org

15. LPAD(): This function is used to make the given string of the given size by adding the given symbol.

Syntax: LPAD('geeks', 8, '0');

Output:

000geeks

16. LTRIM(): This function is used to cut the given sub string from the original string.

Syntax: LTRIM('123123geeks', '123');

Output: geeks

17. **MID():** This function is to find a word from the given position and of the given size.

Syntax: Mid ("geeksforgeeks", 6, 2);

Output: for

18. **POSITION():** This function is used to find position of the first occurrence of the given alphabet.

Syntax: SELECT POSITION('e' IN 'geeksforgeeks');

Output: 2

19. **REPEAT():** This function is used to write the given string again and again till the number of times mentioned.

Syntax: SELECT REPEAT('geeks', 2);

Output: geeksgeeks

20. **REPLACE():** This function is used to cut the given string by removing the given sub string.

Syntax: REPLACE('123geeks123', '123');

Output: geeks

21. **REVERSE():** This function is used to reverse a string.

Syntax: SELECT REVERSE('geeksforgeeks.org');

Output: ‘gro.skeegrofskeeg’

22. **RIGHT():** This function is used to SELECT a sub string from the right end of the given size.

Syntax: SELECT RIGHT('geeksforgeeks.org', 4);

Output: ‘.org’

23. **RPAD():** This function is used to make the given string as long as the given size by adding the given symbol on the right.

Syntax: RPAD('geeks', 8, '0');

Output: ‘geeks000’

24. **RTRIM():** This function is used to cut the given sub string from the original string.

Syntax: RTRIM('geeksxyzzyy', 'xyz');

Output: ‘geeks’

25. **SPACE():** This function is used to write the given number of spaces.

Syntax: SELECT SPACE(7);

Output: ‘ ’

26. STRCMP(): This function is used to compare 2 strings.

- If string1 and string2 are the same, the STRCMP function will return 0.
- If string1 is smaller than string2, the STRCMP function will return -1.
- If string1 is larger than string2, the STRCMP function will return 1.

Syntax: SELECT STRCMP('google.com', 'geeksforgeeks.com');

Output: -1

27. SUBSTR(): This function is used to find a sub string from the a string from the given position.

Syntax: SUBSTR('geeksforgeeks', 1, 5);

Output: 'geeks'

28. SUBSTRING(): This function is used to find an alphabet from the mentioned size and the given string.

Syntax: SELECT SUBSTRING('GeeksForGeeks.org', 9, 1);

Output: 'G'

29. SUBSTRING_INDEX(): This function is used to find a sub string before the given symbol.

Syntax: SELECT SUBSTRING_INDEX('www.geeksforgeeks.org', '.', 1);

Output: 'www'

30. TRIM(): This function is used to cut the given symbol from the string.

Syntax: TRIM(LEADING '0' FROM '000123');

Output: 123

31. UCASE(): This function is used to make the string in upper case.

Syntax: UCASE ("GeeksForGeeks");

Output:

GEEKSFORGEEEKS

Last Updated : 30 Dec, 2019

29

Similar Reads

1. SQL | Functions (Aggregate and Scalar Functions)

2. SQL | Difference between functions and stored procedures in PL/SQL

3. Difference between Structured Query Language (SQL) and Transact-SQL (T-SQL)

4. Configure SQL Jobs in SQL Server using T-SQL

[Trending Now](#) [DSA](#) [Data Structures](#) [Algorithms](#) [Interview Preparation](#) [Data Science](#) [Topic-wise Practice](#) [Jobs](#)

SQL | Advanced Functions



Sakshi98

[Read](#)[Discuss](#)[Courses](#)[Practice](#)

SQL (Structured Query Language) offers a wide range of advanced functions that allow you to perform complex calculations, transformations, and aggregations on your data.

Aggregate Functions

In database management an aggregate function is a function where the values of multiple rows are grouped together as input on certain criteria to form a single value of more significant meaning.

- **SUM()**: Calculates the sum of values in a column.
- **AVG()**: Computes the average of values in a column.
- **COUNT()**: Returns the number of rows or non-null values in a column.
- **MIN()**: Finds the minimum value in a column.
- **MAX()**: Retrieves the maximum value in a column.

Conditional Functions

- **CASE WHEN**: Allows conditional logic to be applied in the **SELECT** statement.
- **COALESCE()**: Returns the first non-null value in a list.
- **NULLIF()**: Compares two expressions and returns null if they are equal; otherwise, returns the first expression.

Mathematical Functions

Mathematical functions are present in SQL which can be used to perform mathematical calculations. Some commonly used mathematical functions are given below:

- **ABS()**: Returns the absolute value of a number.
- **ROUND()**: Rounds a number to a specified number of decimal places.
- **POWER()**: Raises a number to a specified power.
- **SQRT()**: Calculates the square root of a number.

Advanced Functions in SQL

BIN(): It converts a decimal number to a binary number.

AD

Query:

```
SELECT BIN(18);
```

Output:

BIN(18)
10010

BINARY(): It converts a value to a binary string.

Query:

```
SELECT BINARY "GeeksforGeeks";
```

Output:

BINARY "GeeksforGeeks"
GeeksforGeeks

COALESCE(): It returns the first non-null expression in a list.

Query:

```
SELECT COALESCE(NULL,NULL,'GeeksforGeeks',NULL,'Geeks');
```

Output:

COALESCE(NULL,NULL,'GeeksforGeeks',NULL,'Geeks')
GeeksforGeeks

CONNECTION_ID(): It returns the unique connection ID for the current connection.

Query:

```
SELECT CONNECTION_ID();
```

Output:

CONNECTION_ID()
9

CURRENT_USER(): It returns the user name and hostname for the MySQL account used by the server to authenticate the current client.

Query:

```
SELECT CURRENT_USER();
```

Output:

CURRENT_USER()
root@localhost

DATABASE(): It returns the name of the default database.

Query:

```
SELECT DATABASE();
```

Output:

DATABASE()
NULL

IF(): It returns one value if a condition is TRUE, or another value if a condition is FALSE.

Query:

```
SELECT IF(200<500, "YES", "NO");
```

Output:

IF(200<500, "YES", "NO")
YES

LAST_INSERT_ID(): It returns the first AUTO_INCREMENT value that was set by the most recent INSERT or UPDATE statement.

Query:

```
SELECT LAST_INSERT_ID();
```

Output:

LAST_INSERT_ID()
0

Query:

```
SELECT NULLIF(25.11, 25);
```

Output:

NULLIF(25.11, 25)
25.11

Query:

```
SELECT NULLIF(115, 115);
```

Output:

NULLIF(115, 115)
NULL

SESSION_USER(): It returns the user name and host name for the current MySQL user.

Query:

```
SELECT SESSION_USER();
```

Output:

SESSION_USER()
root@localhost

SYSTEM_USER(): It returns the user name and host name for the current MySQL user.

Query:

```
SELECT SYSTEM_USER();
```

Output:

SYSTEM_USER()
root@localhost

USER(): It returns the user name and host name for the current MySQL user.

Query:

```
SELECT USER();
```

Output:

USER()
root@localhost

VERSION(): It returns the version of the MySQL database.

Query:

```
SELECT VERSION();
```

Output:

VERSION()
8.0.11



SQL | Subquery

[Read](#) [Discuss](#) [Courses](#) [Practice](#)

In SQL a Subquery can be simply defined as a query within another query. In other words we can say that a Subquery is a query that is embedded in WHERE clause of another SQL query. Important rules for Subqueries:

- You can place the Subquery in a number of SQL clauses: [WHERE](#) clause, [HAVING](#) clause, FROM clause. Subqueries can be used with SELECT, UPDATE, INSERT, DELETE statements along with expression operator. It could be equality operator or comparison operator such as =, >, =, <= and Like operator.
- A subquery is a query within another query. The outer query is called as **main query** and inner query is called as **subquery**.
- The subquery generally executes first when the subquery doesn't have any **co-relation** with the **main query**, when there is a co-relation the parser takes the decision **on the fly** on which query to execute on **precedence** and uses the output of the subquery accordingly.
- Subquery must be enclosed in parentheses.
- Subqueries are on the right side of the comparison operator.
- [ORDER BY](#) command **cannot** be used in a Subquery. [GROUPBY](#) command can be used to perform same function as ORDER BY command.
- Use single-row operators with singlerow Subqueries. Use multiple-row operators with multiple-row Subqueries.

Syntax: There is not any general syntax for Subqueries. However, Subqueries are seen to be used most frequently with SELECT statement as shown below:

```
SELECT column_name
FROM table_name
WHERE column_name expression operator
( SELECT COLUMN_NAME from TABLE_NAME WHERE ... );
```

Sample Table:

DATABASE

AD

NAME	ROLL_NO	LOCATION	PHONE_NUMBER
Ram	101	Chennai	9988775566
Raj	102	Coimbatore	8877665544
Sasi	103	Madurai	7766553344
Ravi	104	Salem	8989898989
Sumathi	105	Kanchipuram	8989856868

STUDENT

NAME	ROLL_NO	SECTION
Ravi	104	A
Sumathi	105	B
Raj	102	A

Sample Queries

:

- To display NAME, LOCATION, PHONE_NUMBER of the students from DATABASE table whose section is A

```
Select NAME, LOCATION, PHONE_NUMBER from DATABASE
WHERE ROLL_NO IN
(SELECT ROLL_NO from STUDENT where SECTION='A');
```

- Explanation :** First subquery executes “ SELECT ROLL_NO from STUDENT where SECTION='A' ” returns ROLL_NO from STUDENT table whose SECTION is ‘A’. Then outer-query executes it and return the NAME, LOCATION, PHONE_NUMBER from the DATABASE table of the student whose ROLL_NO is returned from inner subquery. Output:

NAME	ROLL_NO	LOCATION	PHONE_NUMBER

Ravi	104	Salem	8989898989
Raj	102	Coimbatore	8877665544

- Insert Query Example:

Table1: Student1

NAME	ROLL_NO	LOCATION	PHONE_NUMBER
Ram	101	chennai	9988773344
Raju	102	coimbatore	9090909090
Ravi	103	salem	8989898989

Table2: Student2

NAME	ROLL_NO	LOCATION	PHONE_NUMBER
Raj	111	chennai	8787878787
Sai	112	mumbai	6565656565
Sri	113	coimbatore	7878787878

- To insert Student2 into Student1 table:

```
INSERT INTO Student1 SELECT * FROM Student2;
```

- Output:

NAME	ROLL_NO	LOCATION	PHONE_NUMBER
Ram	101	chennai	9988773344
Raju	102	coimbatore	9090909090
Ravi	103	salem	8989898989
Raj	111	chennai	8787878787
Sai	112	mumbai	6565656565
Sri	113	coimbatore	7878787878

- To delete students from Student2 table whose rollno is same as that in Student1 table and having location as chennai

```
DELETE FROM Student2
WHERE ROLL_NO IN ( SELECT ROLL_NO
                    FROM Student1
                  WHERE LOCATION = 'chennai');
```

- Output:

1 row delete successfully.

- Display Student2 table:

NAME	ROLL_NO	LOCATION	PHONE_NUMBER
Sai	112	mumbai	6565656565
Sri	113	coimbatore	7878787878

- To update name of the students to geeks in Student2 table whose location is same as Raju,Ravi in Student1 table

```
UPDATE Student2
SET NAME='geeks'
WHERE LOCATION IN ( SELECT LOCATION
                      FROM Student1
                    WHERE NAME IN ('Raju','Ravi'));
```

- Output:

1 row updated successfully.

- Display Student2 table:

NAME	ROLL_NO	LOCATION	PHONE_NUMBER
Sai	112	mumbai	6565656565
geeks	113	coimbatore	7878787878

This article is contributed by **RanjaniRavi**. If you like GeeksforGeeks and would like to contribute, you can also write an article using write.geeksforgeeks.org or mail your article to review-team@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.


[Trending Now](#) [DSA](#) [Data Structures](#) [Algorithms](#) [Interview Preparation](#) [Data Science](#) [Topic-wise Practice](#) [Jobs](#)

SQL | Sub queries in From Clause



Mayank Kumar 12

[Read](#) [Discuss](#) [Courses](#) [Practice](#)

From clause can be used to specify a sub-query expression in SQL. The relation produced by the sub-query is then used as a new relation on which the outer query is applied.

- Sub queries in the from clause are supported by most of the SQL implementations.
- The correlation variables from the relations in from clause cannot be used in the sub-queries in the from clause.

Syntax:

```
SELECT column1, column2 FROM
(SELECT column_x as C1, column_y FROM table WHERE PREDICATE_X)
as table2, table1
WHERE PREDICATE;
```

Note: The sub-query in the from clause is evaluated first and then the results of evaluation are stored in a new temporary relation.

Next, the outer query is evaluated, selecting only those tuples from the temporary relation that satisfies the predicate in the where clause of the outer query.

AD

Query

Example 1: Find all professors whose salary is greater than the average budget of all the departments.

Instructor relation:

InstructorID	Name	Department	Salary
44547	Smith	Computer Science	95000
44541	Bill	Electrical	55000
47778	Sam	Humanities	44000
48147	Erik	Mechanical	80000
411547	Melisa	Information Technology	65000
48898	Jena	Civil	50000

Department relation:

Department Name	Budget
Computer Science	100000
Electrical	80000
Humanities	50000
Mechanical	40000
Information Technology	90000
Civil	60000

Query:

```
select I.ID, I.NAME, I.DEPARTMENT, I.SALARY from
(select avg(BUDGET) as averageBudget from DEPARTMENT) as BUDGET, Instructor as I
where I.SALARY > BUDGET.averageBudget;
```

Output

InstructorID	Name	Department	Salary
44547	Smith	Computer Science	95000
48147	Erik	Mechanical	80000

Explanation: The average budget of all departments from the department relation is 70000. Erik and Smith are the only instructors in the instructor relation whose salary is more than 70000 and therefore are present in the output relation.

Last Updated : 11 Apr, 2022

40

Similar Reads

1. Difference between Having clause and Group by clause

2. SQL | Distinct Clause

3. SQL | WHERE Clause

4. Top Clause in Microsoft SQL Server

5. SQL | Union Clause

6. SQL | WITH clause

7. SQL | Except Clause

8. SQL | OFFSET-FETCH Clause

9. SQL | LIMIT Clause

10. SQL | Intersect & Except clause

Previous

Next

Article Contributed By :



Mayank Kumar 12

Mayank Kumar 12

Vote for difficulty

Current difficulty : Easy

Easy	Normal	Medium	Hard	Expert
------	--------	--------	------	--------

Improved By : Dharmesh Singh 2, 19eucs091



Nested Queries in SQL

[Read](#)[Discuss](#)

Prerequisites : [Basics of SQL](#)

In nested queries, a query is written inside a query. The result of inner query is used in execution of outer query. We will use **STUDENT**, **COURSE**, **STUDENT_COURSE** tables for understanding nested queries.

AD

STUDENT

S_ID	S_NAME	S_ADDRESS	S_PHONE	S AGE
S1	RAM	DELHI	9455123451	18
S2	RAMESH	GURGAON	9652431543	18
S3	SUJIT	ROHTAK	9156253131	20
S4	SURESH	DELHI	9156768971	18

COURSE

C_ID	C_NAME
C1	DSA
C2	Programming
C3	DBMS

STUDENT_COURSE

S_ID	C_ID
S1	C1
S1	C3
S2	C1
S3	C2
S4	C2
S4	C3

There are mainly two types of nested queries:

- **Independent Nested Queries:** In independent nested queries, query execution starts from innermost query to outermost queries. The execution of inner query is independent of outer query, but the result of inner query is used in execution of outer query. Various operators like IN, NOT IN, ANY, ALL etc are used in writing independent nested queries.

IN: If we want to find out **S_ID** who are enrolled in **C_NAME** 'DSA' or 'DBMS', we can write it with the help of independent nested query and IN operator. From **COURSE** table, we can find out **C_ID** for **C_NAME** 'DSA' or DBMS' and we can use these **C_IDs** for finding **S_IDs** from **STUDENT_COURSE** TABLE.

STEP 1: Finding **C_ID** for **C_NAME** ='DSA' or 'DBMS'

Select **C_ID** from **COURSE** where **C_NAME** = ‘DSA’ or **C_NAME** = ‘DBMS’

STEP 2: Using **C_ID** of step 1 for finding **S_ID**

Select **S_ID** from **STUDENT_COURSE** where **C_ID** IN

(SELECT **C_ID** from **COURSE** where **C_NAME** = ‘DSA’ or **C_NAME**=‘DBMS’);

The inner query will return a set with members C1 and C3 and outer query will return those **S_IDs** for which **C_ID** is equal to any member of set (C1 and C3 in this case). So, it will return S1, S2 and S4.

Note: If we want to find out names of **STUDENTs** who have either enrolled in ‘DSA’ or ‘DBMS’, it can be done as:

Select **S_NAME** from **STUDENT** where **S_ID** IN

(Select **S_ID** from **STUDENT_COURSE** where **C_ID** IN

(SELECT **C_ID** from **COURSE** where **C_NAME**=‘DSA’ or **C_NAME**=‘DBMS’));

NOT IN: If we want to find out **S_IDs** of **STUDENTs** who have neither enrolled in ‘DSA’ nor in ‘DBMS’, it can be done as:

Select **S_ID** from **STUDENT** where **S_ID** NOT IN

(Select **S_ID** from **STUDENT_COURSE** where **C_ID** IN

(SELECT **C_ID** from **COURSE** where **C_NAME**=‘DSA’ or **C_NAME**=‘DBMS’));

The innermost query will return a set with members C1 and C3. Second inner query will return those **S_IDs** for which **C_ID** is equal to any member of set (C1 and C3 in this case) which are S1, S2 and S4. The outermost query will return those **S_IDs** where **S_ID** is not a member of set (S1, S2 and S4). So it will return S3.

- **Co-related Nested Queries:** In co-related nested queries, the output of inner query depends on the row which is being currently executed in outer query. e.g.; If we want to find out **S_NAME** of **STUDENTs** who are enrolled in **C_ID** ‘C1’, it can be done with the help of co-related nested query as:

Select **S_NAME** from **STUDENT** S where EXISTS

(select * from **STUDENT_COURSE** SC where **S.S_ID**=**SC.S_ID** and **SC.C_ID**=‘C1’);

For each row of **STUDENT** S, it will find the rows from **STUDENT_COURSE** where **S.S_ID = SC.S_ID** and **SC.C_ID='C1'**. If for a **S_ID** from **STUDENT** S, atleast a row exists in **STUDENT_COURSE** SC with **C_ID='C1'**, then inner query will return true and corresponding **S_ID** will be returned as output.

This article has been contributed by Sonal Tuteja.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Last Updated : 28 Jun, 2021

109

Similar Reads

1. Configure SQL Jobs in SQL Server using T-SQL
2. SQL vs NO SQL vs NEW SQL
3. Decision Making in PL/SQL (if-then , if-then-else, Nested if-then, if-then-elsif-then-else)
4. SQL queries on clustered and non-clustered Indexes
5. SQL queries on FIML Database
6. Production databases in SQL queries
7. SQL Concepts and Queries
8. How to Compare Two Queries in SQL
9. SQL | Difference between functions and stored procedures in PL/SQL
10. Difference between T-SQL and PL-SQL

Previous

Next

Article Contributed By :



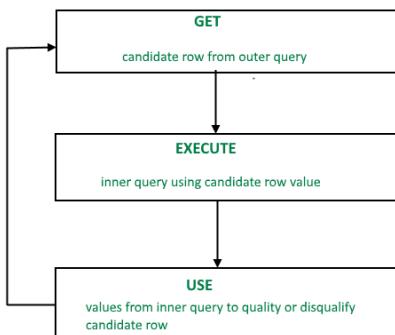
SQL Correlated Subqueries



MrinalVerma

[Read](#) [Discuss](#) [Courses](#) [Practice](#)

Correlated subqueries are used for row-by-row processing. Each subquery is executed once for every row of the outer query.



A correlated subquery is evaluated once for each row processed by the parent statement. The parent statement can be a **SELECT**, **UPDATE**, or **DELETE** statement.

```

SELECT column1, column2, ....
FROM table1 outer
WHERE column1 operator
      (SELECT column1, column2
       FROM table2
       WHERE expr1 =
             outer.expr2);
  
```

A correlated subquery is one way of reading every row in a table and comparing values in each row against related data. It is used whenever a subquery must return a different result or set of results for each candidate row considered by the main query. In other words, you can use a correlated subquery to answer a multipart question whose answer depends on the value in each row processed by the parent statement.

Nested Subqueries Versus Correlated Subqueries :

With a normal nested subquery, the inner **SELECT** query runs first and executes once, returning values to be used by the main query. A correlated subquery, however, executes once for each candidate row considered by the outer query. In other words, the inner query is driven by the outer query.

NOTE: You can also use the **ANY** and **ALL** operator in a correlated subquery. **EXAMPLE of Correlated Subqueries :** Find all the employees who earn more than the average salary in their department.

AD

```
SELECT last_name, salary, department_id
FROM employees outer
WHERE salary >
      (SELECT AVG(salary)
       FROM employees
       WHERE department_id =
             outer.department_id group by department_id);
```

Other use of correlation is in **UPDATE** and **DELETE**

CORRELATED UPDATE :

```
UPDATE table1 alias1
SET column = (SELECT expression
               FROM table2 alias2
               WHERE alias1.column =
                     alias2.column);
```

Use a correlated subquery to update rows in one table based on rows from another table.

CORRELATED DELETE :

```
DELETE FROM table1 alias1
WHERE column1 operator
      (SELECT expression
```

```
FROM table2 alias2
WHERE alias1.column = alias2.column);
```

Use a correlated subquery to delete rows in one table based on the rows from another table.

Using the EXISTS Operator :

The EXISTS operator tests for existence of rows in the results set of the subquery. If a subquery row value is found the condition is flagged **TRUE** and the search does not continue in the inner query, and if it is not found then the condition is flagged **FALSE** and the search continues in the inner query.

EXAMPLE of using EXIST operator :

Find employees who have at least one person reporting to them.

```
SELECT employee_id, last_name, job_id, department_id
FROM employees outer
WHERE EXISTS ( SELECT 'X'
    FROM employees
    WHERE manager_id =
        outer.employee_id);
```

OUTPUT :

EMPLOYEE_ID	LAST_NAME	JOB_ID	DEPARTMENT_ID
100	King	AD_PRES	90
101	Kochhar	AD_VP	90
102	De Haan	AD_VP	90
103	Hunold	IT_PROG	60
108	Greenberg	FI_MGR	100
114	Raphaely	PU_MAN	30
120	Weiss	ST_MAN	50
121	Fripp	ST_MAN	50
122	Kaufling	ST_MAN	50
123	Vollman	ST_MAN	50

More than 10 rows available. Increase rows selector to view more rows.

10 rows returned in 0.05 seconds [CSV Export](#)

EXAMPLE of using NOT EXIST operator :

Find all departments that do not have any employees.

```
SELECT department_id, department_name
FROM departments d
WHERE NOT EXISTS (SELECT 'X'
    FROM employees
```

```
WHERE department_id
= d.department_id);
```

OUTPUT :

DEPARTMENT_ID	DEPARTMENT_NAME
120	Treasury
130	Corporate Tax
140	Control And Credit
150	Shareholder Services
160	Benefits
170	Manufacturing
180	Construction
190	Contracting
200	Operations
210	IT Support
More than 10 rows available. Increase rows selector to view more rows.	

10 rows returned in 0.18 seconds

[CSV Export](#)

Last Updated : 11 Dec, 2022

39

Similar Reads

1. [Difference between Nested Subquery, Correlated Subquery and Join Operation](#)
2. [Difference between Structured Query Language \(SQL\) and Transact-SQL \(T-SQL\)](#)
3. [Configure SQL Jobs in SQL Server using T-SQL](#)
4. [SQL | Procedures in PL/SQL](#)
5. [SQL | Difference between functions and stored procedures in PL/SQL](#)
6. [SQL SERVER – Input and Output Parameter For Dynamic SQL](#)
7. [Difference between SQL and T-SQL](#)
8. [SQL Server | Convert tables in T-SQL into XML](#)
9. [SQL SERVER | Bulk insert data from csv file using T-SQL command](#)
10. [SQL - SELECT from Multiple Tables with MS SQL Server](#)

[Previous](#)[Next](#)



SQL Join vs Subquery



harikamoluguram27

[Read](#)[Discuss](#)[Courses](#)[Practice](#)

What are Joins?

A join is a query that combines records from two or more tables. A join will be performed whenever multiple tables appear in the FROM clause of the query. The select list of the query can select any columns from any of these tables. If join condition is omitted or invalid then a Cartesian product is formed. If any two of these tables have a column name in common, then must qualify these columns throughout the query with table or table alias names to avoid ambiguity. Most join queries contain at least one join condition, either in the FROM clause or in the WHERE clause.

what is Subquery?

AD

A Subquery or Inner query or Nested query is a query within SQL query and embedded within the WHERE clause. A Subquery is a SELECT statement that is embedded in a clause of another SQL statement. They can be very useful to select rows from a table with a condition that depends on the data in the same or another table. A Subquery is used to return data that will be used in the main query as a condition to further restrict the data to be retrieved. The subquery can be placed in the following SQL clauses they are WHERE clause, HAVING clause, FROM clause.

Advantages Of Joins:

- The advantage of a join includes that it executes faster.
- The retrieval time of the query using joins almost always will be faster than that of a subquery.

- By using joins, you can minimize the calculation burden on the database i.e., instead of multiple queries using one join query. This means you can make better use of the database's abilities to search through, filter, sort, etc.

Disadvantages Of Joins:

- Disadvantage of using joins includes that they are not as easy to read as subqueries.
- More joins in a query means the database server has to do more work, which means that it is more time consuming process to retrieve data
- As there are different types of joins, it can be confusing as to which join is the appropriate type of join to use to yield the correct desired result set.
- Joins cannot be avoided when retrieving data from a normalized database, but it is important that joins are performed correctly, as incorrect joins can result in serious performance degradation and inaccurate query results.

Advantages Of Subquery:

- Subqueries divide the complex query into isolated parts so that a complex query can be broken down into a series of logical steps.
- It is easy to understand and code maintenance is also at ease.
- Subqueries allow you to use the results of another query in the outer query.
- In some cases, subqueries can replace complex joins and unions.

Disadvantages of Subquery:

- The optimizer is more mature for MYSQL for joins than for subqueries, so in many cases a statement that uses a subquery can be executed more efficiently if you rewrite it as join.
- We cannot modify a table and select from the same table within a subquery in the same SQL statement.

Conclusion : A subquery is easier to write, but a joint might be better optimized by the server.

For example a Left Outer join typically works faster because servers optimize it.

Last Updated : 07 Nov, 2022

17

Similar Reads

1. Difference between Nested Subquery, Correlated Subquery and Join Operation
2. SQL | Join (Cartesian Join & Self Join)



Difference between Nested Subquery, Correlated Subquery and Join Operation



deekshajaindodya

[Read](#)[Discuss](#)[Courses](#)[Practice](#)

Join Operation :

Join operation is a binary operation used to combine data or rows from two or more tables based on a common field between them. INNER JOIN, LEFT JOIN, RIGHT JOIN, FULL JOIN are different types of Joins.

Example –

```
Orders (OrderID, CustomerID, OrderDate);  
Customers (CustomerID, CustomerName, ContactName, Country);
```

Find details of customers who have ordered.

```
SELECT * from Customers JOIN Orders  
ON Orders.CustomerID=Customers.CustomerID;
```

Subquery

When a query is included inside another query, the Outer query is known as Main Query, and Inner query is known as Subquery.

AD

- **Nested Query –**

In Nested Query, Inner query runs first, and only once. Outer query is executed with result from Inner query.Hence, Inner query is used in execution of Outer query.

Example –

```
Orders (OrderID, CustomerID, OrderDate);
```

```
Customers (CustomerID, CustomerName, ContactName, Country);
```

Find details of customers who have ordered.

```
SELECT * FROM Customers WHERE
CustomerID IN (SELECT CustomerID FROM Orders);
```

- **Correlated Query –**

In Correlated Query, Outer query executes first and for every Outer query row Inner query is executed. Hence, Inner query uses values from Outer query.

Example –

```
Orders (OrderID, CustomerID, OrderDate);
Customers (CustomerID, CustomerName, ContactName, Country);
```

Find details of customers who have ordered.

```
SELECT * FROM Customers where
EXISTS (SELECT CustomerID FROM Orders
WHERE Orders.CustomerID=Customers.CustomerID);
```

Application of Join Operation and Subquery :

To understand the difference between Nested Subquery, Correlated Subquery and Join Operation firstly we have to understand where we use subqueries and where to use joins.

- When we want to get data from multiple tables we use join operation.

Example: Let's consider two relations as:

```
Employee (eId, eName, eSalary, dId);
Department (dId, dName, dLocation);
```

Now, we have to find employee names and Department name working at London Location. Here, we have to display eName from employee table and dName from Department table. Hence we have to use Join Operation.

```
SELECT e.eName, d.dName from Employee e,
Department d
where e.dId=d.dId and d.dLocation="London";
```

- When we want to get data from one table and condition is based on another table we can either use Join or Subquery. Now, we have to find employee names working at London Location.

Here, we have to display only eName from employee table hence we can use either Join Operation or Subquery

Using Join Operation –

```
SELECT e.eName from Employee e, Department d
where e.dId=d.dId and d.dLocation="London";
```

Using Subquery –

```
SELECT eName from Employee
where dId=(SELECT dId from Department where dLocation="London");
```

After understanding the basic difference between Join and Subqueries, Now we will understand the difference between Nested Subquery, Correlated Subquery and Join Operation.

Difference between Nested Query, Correlated Query and Join Operation :

Parameters	Nested Query	Correlated Query	Join Operation
Definition	In Nested query, a query is written inside another query and the result of inner query is used in execution of outer query.	In Correlated query, a query is nested inside another query and inner query uses values from outer query.	Join operation is used to combine data or rows from two or more tables based on a common field between them. INNER JOIN, LEFT JOIN, RIGHT JOIN, FULL JOIN are different types of Joins.
Approach	Bottom up approach i.e. Inner query runs first, and only once. Outer query is executed with result from Inner query.	Top to Down Approach i.e. Outer query executes first and for every Outer query row Inner query is executed.	It is basically cross product satisfying a condition.
Dependency	Inner query execution is not dependent on Outer query.	Inner query is dependent on Outer query.	There is no Inner Query or Outer Query. Hence, no dependency is there.
Performance	Performs better than Correlated	Performs slower than both Nested	By using joins we maximize the calculation

Parameters	Nested Query	Correlated Query	Join Operation
	Query but is slower than Join Operation.	Query and Join operations as for every outer query inner query is executed.	burden on the database but joins are better optimized by the server so the retrieval time of the query using joins will almost always be faster than that of a subquery.

Last Updated : 28 Dec, 2020

26

Similar Reads

1. Difference between Nested Loop Join and Hash Join

2. Difference between Nested Loop join and Sort Merge Join

3. SQL | Join (Cartesian Join & Self Join)

4. SQL Join vs Subquery

5. Join operation Vs Nested query in DBMS

6. Difference between Inner Join and Outer Join in SQL

7. Difference between Natural join and Inner Join in SQL

8. Difference between Natural join and Cross join in SQL

9. Difference between Hash Join and Sort Merge Join

10. Difference between “INNER JOIN” and “OUTER JOIN”

[Previous](#)[Next](#)

Article Contributed By :



deekshajaindodya
deekshajaindodya



MySQL | Regular expressions (Regexp)



shreyanshi_arun

[Read](#)[Discuss](#)[Courses](#)[Practice](#)

MySQL supports another type of pattern matching operation based on the regular expressions and the REGEXP operator.

1. It provides a powerful and flexible pattern match that can help us implement powerful search utilities for our database systems.
2. REGEXP is the operator used when performing regular expression pattern matches. RLIKE is the synonym.
3. It also supports a number of metacharacters which allow more flexibility and control when performing pattern matching.
4. The backslash is used as an escape character. It's only considered in the pattern match if double backslashes have been used.
5. Not case sensitive.

Pattern	What the Pattern matches
*	Zero or more instances of string preceding it
+	One or more instances of strings preceding it

[Engineering Mathematics](#) [Discrete Mathematics](#) [Digital Logic and Design](#) [Computer Organization and Architecture](#)

?	Match zero or one instances of the strings preceding it.
^	caret(^) matches Beginning of string
\$	End of string
[abc]	Any character listed between the square brackets
[^abc]	Any character not listed between the square brackets
[A-Z]	match any upper case letter.



Pattern	What the Pattern matches
[a-z]	match any lower case letter
[0-9]	match any digit from 0 through to 9.
[:<:]	matches the beginning of words.
[:>:]	matches the end of words.
[:class:]	matches a character class i.e. [:alpha:] to match letters, [:space:] to match white space, [:punct:] is match punctuations and [:upper:] for upper class letters.
p1 p2 p3	Alternation; matches any of the patterns p1, p2, or p3
{n}	n instances of preceding element
{m,n}	m through n instances of preceding element

Examples with explanation :

- **Match beginning of string(^):** Gives all the names starting with 'sa'. Example- sam,samarth.

```
SELECT name FROM student_tbl WHERE name REGEXP '^sa';
```

- **Match the end of a string(\$):** Gives all the names ending with 'on'. Example – norton,merton.

```
SELECT name FROM student_tbl WHERE name REGEXP 'on$';
```

- **Match zero or one instance of the strings preceding it(?):** Gives all the titles containing 'com'. Example – comedy , romantic comedy.

```
SELECT title FROM movies_tbl WHERE title REGEXP 'com?';
```

- **matches any of the patterns p1, p2, or p3(p1|p2|p3):** Gives all the names containing 'be' or 'ae'. Example – Abel, Baer.

```
SELECT name FROM student_tbl WHERE name REGEXP 'be|ae' ;
```

- **Matches any character listed between the square brackets([abc]):** Gives all the names containing 'j' or 'z'. Example – Lorentz, Rajs.

```
SELECT name FROM student_tbl WHERE name REGEXP '[jz]' ;
```

- Matches any lower case letter between ‘a’ to ‘z’- ([a-z]) ([a-z] and (.)):** Retrieve all names that contain a letter in the range of ‘b’ and ‘g’, followed by any character, followed by the letter ‘a’. Example – Tobias, sewall. Matches any single character(.)

```
SELECT name FROM student_tbl WHERE name REGEXP '[b-g].[a]' ;
```

- Matches any character not listed between the square brackets.([abc]):** Gives all the names not containing ‘j’ or ‘z’. Example – nerton, sewall.

```
SELECT name FROM student_tbl WHERE name REGEXP '[^jz]' ;
```

- Matches the end of words[[>]]:** Gives all the titles ending with character “ack”. Example – Black.

```
SELECT title FROM movies_tbl WHERE REGEXP 'ack[[>]]';
```

- Matches the beginning of words[[<]]:** Gives all the titles starting with character “for”. Example – Forgetting Sarah Marshal.

```
SELECT title FROM movies_tbl WHERE title REGEXP '[[<]]for';
```

- Matches a character class[:class]:** i.e [:lower:] – lowercase character ,[:digit:] – digit characters etc. Gives all the titles containing alphabetic character only. Example – stranger things, Avengers.

```
SELECT title FROM movies_tbl WHERE REGEXP '[:alpha:]' ;
```

- Matches the beginning of all words by any character listed between the square brackets. (^[abc]):** Gives all the names starting with ‘n’ or ‘s’. Example – nerton, sewall.

```
SELECT name FROM student_tbl WHERE name REGEXP '^[ns]' ;
```

Similar Reads

1. MySQL | Recursive CTE (Common Table Expressions)

2. MySQL | Common MySQL Queries



IFNULL in MySQL



DevanshuAgarwal

[Read](#)[Discuss](#)[Courses](#)[Practice](#)

Given a TABLE, in this TABLE, it prints entry of the table. If table is empty then it gives NULL.

Examples:

QUESTION : Given an employee table, print name from the given table which id equals to 2.

id	Name
1	Geek1
2	Geek2
3	Geek3
4	Geek4

Output : Geek2

QUESTION : Given same Employee table, print name from the given table which id equals to 5.

Output : NULL

AD

Approach: In this case, we use here IFNULL. IFNULL print the null if the table is an empty or other condition.

Query:-

```
SELECT  
IFNULL(  
    (SELECT NAME  
     from employee  
     where id = 2),  
    'NULL') as NAME;
```

Output:-

Geek2

Query:-

```
SELECT  
IFNULL(  
    (SELECT NAME  
     from employee  
     where id = 5),  
    'NULL') as NAME;
```

Output:-

NULL

Last Updated : 03 Mar, 2018

7

Similar Reads

1. [MySQL | Common MySQL Queries](#)

2. [MySQL | LEAD\(\) and LAG\(\) Function](#)

3. [PHP | MySQL UPDATE Query](#)

4. [PHP | MySQL Database Introduction](#)

5. [PHP | MySQL \(Creating Database \)](#)

6. [PHP | MySQL \(Creating Table \)](#)

7. [PHP | Inserting into MySQL database](#)



MySQL | LAST_DAY() Function



Shubrodeep Banerjee

[Read](#)[Discuss](#)[Courses](#)[Practice](#)

The `LAST_DAY()` function in MySQL can be used to know the last day of the month for a given date or a datetime. The `LAST_DAY()` function takes a *date* value as argument and returns the last day of month in that *date*. The *date* argument represents a valid date or datetime.

Syntax:

```
LAST_DAY( Date );
```

If the date or datetime value is invalid, the function returns `NULL`.

Parameters Used:

Date: The `LAST_DAY()` function takes a single argument *Date*. It is the date or datetime value for which you want to extract the last day of the month.

AD

Trending Now DSA Data Structures Algorithms Interview Preparation Data Science Topic-wise Practice J:

The `LAST_DAY()` function returns the last day of the month for a valid *Date* argument. If the argument *Date* is invalid or null, then the function will also return `NULL`.

Below are some common examples or usages of the `LAST_DAY()` function:

1. Extracting last day from a given date:

To know the last date of the month December 2017, the `LAST_DAY()` function can be executed in the following way:

Syntax :



```
mysql> SELECT LAST_DAY('2017-12-25');
```

Output :

'2017-12-31'

2. Extracting the last day from a given datetime:

To know the last date of the month December using datetime format, the LAST_DAY() function can be executed in the following way:

Syntax :

```
mysql> SELECT LAST_DAY('2017-12-25 08:21:05');
```

Output :

'2017-12-31'

3. Checking whether it is a leap year or not:

To know whether the year is a leap year or not, we can use the LAST_DAY() function to check the last day of the month February of that year. If it is the 29th day then that year is leap otherwise not.

Syntax :

```
mysql> SELECT LAST_DAY('2016-02-17');
```

Output :

'2016-02-29'

4. Extracting the last day for the current month:

To know the last date of the current month ,the LAST_DAY() function is combined with the NOW() or CURDATE() function and can be executed in the following way:

Using the NOW() function: The NOW() function in MySQL returns the current date-time stamp.

Syntax :

```
mysql> SELECT LAST_DAY(NOW());
```

Output :

'2017-12-31'

5. Using the CURDATE() function: The CURDATE() function in MySQL return the current date in Date format.**Syntax :**

```
mysql> SELECT LAST_DAY(CURDATE());
```

Output :

'2017-12-31'

6. Extracting the last day of the next month:

To know the last date of the next month, the last_day() function can be executed in the following way:

Syntax :

```
mysql> SELECT LAST_DAY(CURDATE() + INTERVAL 1 MONTH);
```

Output :

'2018-01-31'

Last Updated : 21 Mar, 2018

Similar Reads

1. [PLSQL | LAST_DAY Function](#)

2. [LOCALTIME\(\) and LAST_DAY\(\) Function in MariaDB](#)

3. [MySQL | Common MySQL Queries](#)



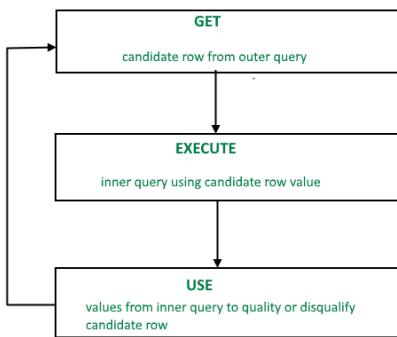
SQL Correlated Subqueries



MrinalVerma

[Read](#) [Discuss](#) [Courses](#) [Practice](#)

Correlated subqueries are used for row-by-row processing. Each subquery is executed once for every row of the outer query.



A correlated subquery is evaluated once for each row processed by the parent statement. The parent statement can be a **SELECT**, **UPDATE**, or **DELETE** statement.

```

SELECT column1, column2, ....
FROM table1 outer
WHERE column1 operator
      (SELECT column1, column2
       FROM table2
       WHERE expr1 =
             outer.expr2);
  
```

A correlated subquery is one way of reading every row in a table and comparing values in each row against related data. It is used whenever a subquery must return a different result or set of results for each candidate row considered by the main query. In other words, you can use a correlated subquery to answer a multipart question whose answer depends on the value in each row processed by the parent statement.

Nested Subqueries Versus Correlated Subqueries :



With a normal nested subquery, the inner **SELECT** query runs first and executes once, returning values to be used by the main query. A correlated subquery, however, executes once for each candidate row considered by the outer query. In other words, the inner query is driven by the outer query.

NOTE: You can also use the **ANY** and **ALL** operator in a correlated subquery. **EXAMPLE of Correlated Subqueries :** Find all the employees who earn more than the average salary in their department.

AD

```
SELECT last_name, salary, department_id
FROM employees outer
WHERE salary >
      (SELECT AVG(salary)
       FROM employees
       WHERE department_id =
             outer.department_id group by department_id);
```

Other use of correlation is in **UPDATE** and **DELETE**

CORRELATED UPDATE :

```
UPDATE table1 alias1
SET column = (SELECT expression
              FROM table2 alias2
              WHERE alias1.column =
                    alias2.column);
```

Use a correlated subquery to update rows in one table based on rows from another table.

CORRELATED DELETE :

```
DELETE FROM table1 alias1
WHERE column1 operator
      (SELECT expression
```

```
FROM table2 alias2
WHERE alias1.column = alias2.column);
```

Use a correlated subquery to delete rows in one table based on the rows from another table.

Using the EXISTS Operator :

The EXISTS operator tests for existence of rows in the results set of the subquery. If a subquery row value is found the condition is flagged **TRUE** and the search does not continue in the inner query, and if it is not found then the condition is flagged **FALSE** and the search continues in the inner query.

EXAMPLE of using EXIST operator :

Find employees who have at least one person reporting to them.

```
SELECT employee_id, last_name, job_id, department_id
FROM employees outer
WHERE EXISTS ( SELECT 'X'
                FROM employees
               WHERE manager_id =
outer.employee_id);
```

OUTPUT :

EMPLOYEE_ID	LAST_NAME	JOB_ID	DEPARTMENT_ID
100	King	AD_PRES	90
101	Kochhar	AD_VP	90
102	De Haan	AD_VP	90
103	Hunold	IT_PROG	60
108	Greenberg	FI_MGR	100
114	Raphaely	PU_MAN	30
120	Weiss	ST_MAN	50
121	Fripp	ST_MAN	50
122	Kaufling	ST_MAN	50
123	Vollman	ST_MAN	50

More than 10 rows available. Increase rows selector to view more rows.

10 rows returned in 0.05 seconds

[CSV Export](#)

```
SELECT department_id, department_name
FROM departments d
WHERE NOT EXISTS (SELECT 'X'
                  FROM employees
```

```
WHERE department_id
= d.department_id);
```

OUTPUT :

DEPARTMENT_ID	DEPARTMENT_NAME
120	Treasury
130	Corporate Tax
140	Control And Credit
150	Shareholder Services
160	Benefits
170	Manufacturing
180	Construction
190	Contracting
200	Operations
210	IT Support
More than 10 rows available. Increase rows selector to view more rows.	

10 rows returned in 0.18 seconds

[CSV Export](#)

Last Updated : 11 Dec, 2022

39

Similar Reads

1. [Difference between Nested Subquery, Correlated Subquery and Join Operation](#)
2. [Difference between Structured Query Language \(SQL\) and Transact-SQL \(T-SQL\)](#)
3. [Configure SQL Jobs in SQL Server using T-SQL](#)
4. [SQL | Procedures in PL/SQL](#)
5. [SQL | Difference between functions and stored procedures in PL/SQL](#)
6. [SQL SERVER – Input and Output Parameter For Dynamic SQL](#)
7. [Difference between SQL and T-SQL](#)
8. [SQL Server | Convert tables in T-SQL into XML](#)
9. [SQL SERVER | Bulk insert data from csv file using T-SQL command](#)
10. [SQL - SELECT from Multiple Tables with MS SQL Server](#)

[Previous](#)[Next](#)



How to find Nth highest salary from a table?

[Read](#)[Discuss](#)[Courses](#)[Practice](#)[Video](#)

Structured Query Language is a computer language that we use to interact with a relational database. Finding Nth highest salary in a table is the most common question asked in interviews. Here is a way to do this task using the dense_rank() function.

Consider the following table:

Trending Now DSA Data Structures Algorithms Interview Preparation Data Science Topic-wise Practice J:

CREATE TABLE:

AD

```
CREATE TABLE emp (
    emp_name VARCHAR(50),
    emp_salary DECIMAL(10,2)
);
```

Let's insert some random data with a random name and then we will look at how to calculate the nth highest emp_salary.

Query:

```
CREATE TABLE emp (
    emp_name VARCHAR(50),
    emp_salary DECIMAL(10,2)
);
INSERT INTO emp (emp_name, emp_salary) VALUES
('Shubham Thakur', 50000.00),
('Aman Chopra', 60000.50),
('Naveen Tulasi', 75000.75),
```



```
('Bhavika uppala', 45000.25),
('Nishant jain', 80000.00);
```

Output:

emp_name	emp_salary
Shubham Thakur	50000
Aman Chopra	60000.5
Naveen Tulasi	75000.75
Bhavika uppala	45000.25
Nishant jain	80000

Query:

```
select * from(
select emp_name, emp_salary, dense_rank()
over(order by emp_salary desc)r from Emp)
where r=3;
```

1. To find the 2nd highest sal set n = 2
2. To find the 3rd highest sal set n = 3 and so on.

Let's check to find 3rd highest salary:

Output:

emp_name	emp_salary	r
Aman Chopra	60000.5	3

Using DENSE_RANK

1. DENSE_RANK computes the rank of a row in an ordered group of rows and returns the rank as a NUMBER. The ranks are consecutive integers beginning with 1.
2. This function accepts arguments as any numeric data type and returns NUMBER.
3. As an analytic function, DENSE_RANK computes the rank of each row returned from a query with respect to the other rows, based on the values of the value_exprs in the order_by_clause.
4. In the above query, the rank is returned based on the sal of the employee table. In the case of a tie, it assigns equal rank to all the rows.

Alternate Solution

```
CREATE TABLE `Employee` (
  `ENAME` varchar(225) COLLATE utf8_unicode_ci NOT NULL,
  `SAL` bigint(20) unsigned NOT NULL,
  PRIMARY KEY (`ENAME`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci;
```

To find the 6th highest salary

Query:

```
mysql> select * from ((select * from Employee
   ORDER BY `sal` DESC limit 6 ) AS T)
   ORDER BY T.`sal` ASC limit 1;
```

Alternate Use of Limit

```
select * from Employee ORDER BY `sal` DESC limit 5,1; // will return 6th highest
```

Output:

emp_name	emp_salary	r
Aman Chopra	60000.5	3

Alternate Solution

Suppose the task is to find the employee with the Nth highest salary from the above table. We can do this as follows:

1. Find the employees with top N distinct salaries.
2. Find the lowest salary among the salaries fetched by the above query, this will give us the Nth highest salary.
3. Find the details of the employee whose salary is the lowest salary fetched by the above query.

Query:

```
SELECT * FROM Employee WHERE sal =
(
  SELECT MIN(sal) FROM Employee
  WHERE sal IN (
    SELECT DISTINCT TOP N
      sal FROM Employee
    ORDER BY sal DESC
  ))
```

The above query will fetch the details of the employee with the Nth highest salary. Let us see how:

Consider N = 4.

Starting with the most inner query, the query:

Query:

```
SELECT DISTINCT sal
FROM Employee
ORDER BY sal DESC
LIMIT 4;
```

Output:

emp_salary
80000
75000.75
60000.5
50000

Query:

```
SELECT MIN(sal) FROM Employee WHERE sal IN (
    SELECT DISTINCT sal
    FROM Employee
    ORDER BY sal DESC
    LIMIT 4
);
```

Output:

MIN(emp_salary)
50000

You can see that the above-returned result is the required 4th highest salary.

Another Solution:

Here N = nth Highest Salary eg. 3rd Highest salary: N=3.

Syntax:

SELECT ename,sal from Employee e1 where

$N-1 = (\text{SELECT COUNT(DISTINCT sal) from Employee e2 where e2.sal > e1.sal})$

Using the LIMIT Clause

Syntax:

Select Salary from table_name order by Salary DESC limit n-1,1;

Here we are ordering our salaries in descending order so we will get the highest salary first and then subsequently lower salaries. The limit clause has two components, the First component is to skip a number of rows from the top and the second component is to display the number of rows we want.

Let us see with an example :

To find the 4th Highest salary query will be

Query:

`Select emp_sal from Emp order by emp_sal DESC limit 3,1;`

Output:

emp_salary
70000

Here we are skipping 3 rows from the Top and returning only 1 row after skipping.

You can also find names of employees having Nth Highest Salary

Syntax:

Select Emp_name from table_name where Salary =

(Select Salary from table_name order by Salary DESC limit n-1,1);

There can be another question like finding Nth Lowest Salary. In order to do that, just reverse order using ASC (if you don't specify by default column will be ordered in ascending order).

Syntax:

Select Salary from table_name order by Salary limit n-1,1;

This article is contributed by **Rishav Shandilya**. If you like GeeksforGeeks and would like to contribute, you can also write an article using write.geeksforgeeks.org or mail your article to review-team@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.

Last Updated : 06 May, 2023

80

Similar Reads

1. [SQL Query to Find Monthly Salary of Employee If Annual Salary is Given](#)
2. [SQL Query to Print the Name and Salary of the Person Having Least Salary in the Department](#)
3. [SQL Query to Find the Highest Salary of Each Department](#)
4. [Displaying Department Name Having Highest Average Salary in SQL Server](#)
5. [SQL Query to find an employee whose salary is equal to or greater than a specific number](#)
6. [Finding Average Salary of Each Department in SQL Server](#)
7. [SQL Query to Find the Highest Purchase Amount Ordered by the Each Customer](#)
8. [SQL Query to Display Nth Record from Employee Table](#)
9. [How to Select All Records from One Table That Do Not Exist in Another Table in SQL?](#)
10. [SQL Query to Filter a Table using Another Table](#)

Previous

Next

Article Contributed By :



GeeksforGeeks

Vote for difficulty

Current difficulty : [Medium](#)

SQL - Useful Functions

SQL has many built-in functions for performing processing on string or numeric data. Following is the list of all useful SQL built-in functions –

- SQL COUNT Function - The SQL COUNT aggregate function is used to count the number of rows in a database table.
- SQL MAX Function - The SQL MAX aggregate function allows us to select the highest (maximum) value for a certain column.
- SQL MIN Function - The SQL MIN aggregate function allows us to select the lowest (minimum) value for a certain column.
- SQL AVG Function - The SQL AVG aggregate function selects the average value for certain table column.
- SQL SUM Function - The SQL SUM aggregate function allows selecting the total for a numeric column.
- SQL SQRT Functions - This is used to generate a square root of a given number.
- SQL RAND Function - This is used to generate a random number using SQL command.
- SQL CONCAT Function - This is used to concatenate any string inside any SQL command.
- SQL Numeric Functions - Complete list of SQL functions required to manipulate numbers in SQL.
- SQL String Functions - Complete list of SQL functions required to manipulate strings in SQL.