1. **CREATE** **DATABASE** database\_name;
2. **DROP** **DATABASE** database\_name;
3. RENAME **DATABASE** old\_db\_name **TO** new\_db\_name;

# **SQL SELECT Database**

1. USE **DATABASE** database\_name;

# **SQL CREATE TABLE Example in MySQL**

1. **CREATE** **TABLE** Employee
2. (
3. EmployeeID **int**,
4. FirstName **varchar**(255),
5. LastName **varchar**(255),
6. Email **varchar**(255),
7. AddressLine **varchar**(255),
8. City **varchar**(255)
9. );
10. **DROP** **TABLE** "table\_name";

First we verify STUDENTS table and then we would delete it from the database.

1. SQL> **DESC** STUDENTS;
2. Query OK, 0 rows affected (0.01 sec)  // means table is deleted

The DELETE statement is used to delete rows from a table. If you want to remove a specific row from a table you should use WHERE condition.

1. **DELETE** **FROM** table\_name [**WHERE** condition];

But if you do not specify the WHERE condition it will remove all the rows from the table.

1. **DELETE** **FROM** table\_name;

# **Difference between DELETE and TRUNCATE statements**

There is a slight difference b/w delete and truncate statement. The **DELETE statement** only deletes the rows from the table based on the condition defined by WHERE clause or delete all the rows from the table when condition is not specified.

But it does not free the space containing by the table.

The **TRUNCATE statement:** it is used to delete all the rows from the table **and free the containing space.**

# **Difference b/w DROP and TRUNCATE statements**

When you use the drop statement it deletes the table's row together with the table's structure so all the relationships of that table with other tables will no longer be valid.

On the other hand when we **TRUNCATE** a table, the table structure remains the same, so you will not face any of the above problems.

1. RENAME TABLE old\_table \_name **To** new\_table\_name;  or
2. **ALTER** **TABLE** table\_name
3. RENAME **TO** new\_table\_name;

If you want to copy a SQL table into another table in the same SQL server database, it is possible by using the select statement.

The syntax of copying table from one to another is given below:

1. **SELECT** \* **INTO** <destination\_table> **FROM** <source\_table>

# **SQL TEMP TABLE**

The concept of temporary table is introduced by SQL server. It helps developers in many ways:

**Temporary tables** can be created at run-time and can do all kinds of operations that a normal table can do. These temporary tables are created inside tempdb database.

There are two types of temp tables based on the behavior and scope.

1. Local Temp Variable
2. Global Temp Variable

# **Local Temp Variable**

Local temp tables are only available at current connection time. It is automatically deleted when user disconnects from instances. It is started with hash (#) sign.

1. **CREATE** **TABLE** #**local** **temp** **table** (
2. User id **int**,
3. Username **varchar** (50),
4. User address **varchar** (150)
5. )

# **Global Temp Variable**

Global temp tables name starts with double hash (##). Once this table is created, it is like a permanent table. It is always ready for all users and not deleted until the total connection is withdrawn.

1. **CREATE** **TABLE** ##new **global** **temp** **table** (
2. User id **int**,
3. User **name** **varchar** (50),
4. User address **varchar** (150)
5. )

# **SQL ALTER TABLE Add Column**

If you want to add columns in SQL table, the SQL alter table syntax is given below:

1. **ALTER** **TABLE** table\_name **ADD** column\_name **column**-definition;
2. **ALTER** **TABLE** table\_name
3. **ADD** (column\_1 **column**-definition,
4. column\_2 **column**-definition,
5. .....
6. column\_n **column**-definition);

# **SQL ALTER TABLE Modify Column**

If you want to modify an existing column in SQL table, syntax is given below:

1. **ALTER** **TABLE** table\_name **MODIFY** column\_name column\_type;

If you want to modify multiple columns in table, the SQL table will be

1. **ALTER** **TABLE** table\_name
2. **MODIFY** (column\_1 column\_type,
3. column\_2 column\_type,
4. .....
5. column\_n column\_type);

# **SQL ALTER TABLE DROP Column**

The syntax of alter table drop column is given below:

1. **ALTER** **TABLE** table\_name **DROP** **COLUMN** column\_name;

**SQL ALTER TABLE RENAME Column**

The syntax of alter table rename column is given below:

1. **ALTER** **TABLE** table\_name
2. RENAME **COLUMN** old\_name **to** new\_name;

**SQL SELECT UNIQUE**

Actually, there is no difference between DISTINCT and UNIQUE.

**SELECT UNIQUE** is an old syntax which was used in oracle description but later ANSI standard defines DISTINCT as the official keyword. In simple words, we can say that SELECT UNIQUE statementLet's see the syntax of select unique statement.

1. **SELECT** **UNIQUE** column\_name
2. **FROM** table\_name;

SQL SELECT DISTINCT statement can also be used for the same cause.

The **SQL COUNT()** function is used to return the number of rows in a query.

**For example:** If you have a record of the voters in selected area and want to count the number of voters then it is very difficult to do it manually but you can do it easily by using the SQL SELECT COUNT query.

1. **SELECT** COUNT (expression)
2. **FROM** tables
3. **WHERE** conditions;

**SQL SELECT COUNT(column\_name)**

1. **SELECT** COUNT(**name**) **FROM** employee\_table;

It will return the total number of names of employee\_table. But null fields will not be counted.

**SQL SELECT COUNT(\*)**

1. **SELECT** COUNT(\*) **FROM** employee\_table;

The "select count(\*) from table" is used to return the number of records in table.

**SQL SELECT COUNT(DISTINCT column\_name)**

1. **SELECT** COUNT(**DISTINCT** **name**) **FROM** employee\_table;

It will return the total distinct names of **employee**\_table.

The SQL SELECT TOP Statement is used to select top data from a table. The top clause specifies that how many rows are returned.

There is an example of employee table:

|  |  |  |  |
| --- | --- | --- | --- |
| **EMP\_ID** | **NAME** | **SIR\_NAME** | **USER\_NAME** |
| 1 | RAHUL | OJHA | ra@jha |
| 2 | ANU | SHARMA | anusha1 |
| 3 | RAVI | SINGHAL | ravin |

Let's see the syntax for the select top statement.

1. **SELECT** COUNT (expression)

Let's see the example of sql select top statement.

1. **SELECT** **TOP** 2 \* **FROM** employee

It will return the following table:

|  |  |  |  |
| --- | --- | --- | --- |
| **EMP\_ID** | **NAME** | **SIR\_NAME** | **USER\_NAME** |
| 1 | RAHUL | OJHA | ra@jha |
| 2 | ANU | SHARMA | anusha1 |

The SQL first() function is used to return the first value of the selected column.

Let's see the syntax of sql select first() function:

1. **SELECT** **FIRST**(column\_name) **FROM** table\_name;

Here a point is notable that first function is only supported by MS Access.

If you want to retrieve the first value of the "customer\_name" column from the "customers" table, you need to write following query:

1. **SELECT** **FIRST**(customer\_name) **AS** first\_customer **FROM** customers;

Let us take the example of CUSTOMERS to examine SQL SELECT FIRST command:

Table CUSTOMERS

|  |  |  |  |
| --- | --- | --- | --- |
| **CUSTOMER\_NAME** | **AGE** | **ADDRESS** | **EXPENDITURE** |
| KAMAL SHARMA | 26 | GHAZIABAD | 6000 |
| ROBERT PETT | 23 | NEWYORK | 26000 |
| SHIKHA SRIVASTAV | 22 | DELHI | 9000 |

If you want to retrieve the first value of the "customer\_name" column from the "customers" table, you need to write following query:

Let's see the syntax of sql select first() function:

1. **SELECT** **FIRST** (CUSTOMER\_NAME) **AS** first\_customer **FROM** CUSTOMERS;
2. **After** that query, you will find the result:
3. KAMAL SHARMA

#### Note: The SELECT FIRST statement is only supported by MS Access. This statement doesn?t work with other databases like Oracle, MySQL etc.

Let us take the example of CUSTOMERS to examine SQL SELECT LAST command:

Table CUSTOMERS

|  |  |  |  |
| --- | --- | --- | --- |
| **CUSTOMER\_NAME** | **AGE** | **ADDRESS** | **EXPENDITURE** |
| KAMAL SHARMA | 26 | GHAZIABAD | 6000 |
| ROBERT PETT | 23 | NEWYORK | 26000 |
| SHIKHA SRIVASTAV | 22 | DELHI | 9000 |

If you want to retrieve the last value of the "customer\_name" column from the "customers" table, you need to write following query:

1. **SELECT** **LAST** (CUSTOMER\_NAME) **AS** LAST\_CUSTOMER **FROM** CUSTOMERS;
2. **After** that query, you will find the result:
3. SHIKHA SRIVASTAV

The SQL SELECT RANDOM() function returns the random row. It can be used in online exam to display the random questions.

If you want to select a random row with **MY SQL**:

1. **SELECT** **column** **FROM** **table**
2. **ORDER** **BY** RAND ( )
3. LIMIT 1

**SQL AS** is used to assign temporarily a new name to a table column.

It makes easy presentation of query results and allows the developer to label results more accurately without permanently renaming table columns.

Let's see the example of select as:

1. **SELECT** day\_of\_order **AS** "Date"
2. Customer **As** "Client",
3. Product,
4. Quantity,
5. **FROM** orders;

Let us take a table named orders, it contains:

|  |  |  |  |
| --- | --- | --- | --- |
| **Day\_of\_order** | **Customer** | **Product** | **Quantity** |
| 11-09-2001 | Osama bin qasim | airplane | 2 |
| 13-12-2001 | Mukhtar mahmood | RDX | 20 |
| 26-12-2004 | Balaswamy | Water cannon | 35 |

After applying this SQL AS example syntax

1. **SELECT** day\_of\_order **AS** "Date"
2. Customer **As** "Client",
3. Product,
4. Quantity,
5. **FROM** orders;

Result will be shown as this table:

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Client** | **Product** | **Quantity** |
| 11-09-2001 | Osama bin qasim | airplane | 2 |
| 13-12-2001 | Mukhtar mahmood | RDX | 20 |
| 26-12-2004 | Balaswamy | Water cannon | 35 |

Note: SQL AS is same as SQL ALIAS

# **SQL SELECT IN**

SQL IN is an operator used in a SQL query to help reduce the need to use multiple SQL "OR" conditions.

It is used in SELECT, INSERT, UPDATE or DELETE statement.

# **Advantage of SQL SELECT IN**

It minimizes the use of SQL OR operator.

Let's see the syntax for SQL IN:

1. Expression IN (value 1, value 2 ... value n);

Take an example with character values.

1. **SELECT** \*
2. **FROM** students
3. **WHERE** students\_name IN ( Amit , Raghav, Rajeev)

Let?s take another example with numeric values.

1. **SELECT** \*
2. **FROM** marks
3. **WHERE** roll\_no IN (001, 023, 024);

# **SQL SELECT from Multiple Tables**

This statement is used to retrieve fields from multiple tables. To do so, we need to use join query to get data from multiple tables.

Let's see the example for the select from multiple tables:

1. **SELECT** orders.order\_id, suppliers.**name**
2. **FROM** suppliers
3. **INNER** JOIN orders
4. **ON** suppliers.supplier\_id = orders.supplier\_id
5. **ORDER** **BY** order\_id;

# **SQL SELECT DATE**

SQL SELECT DATE is used to retrieve a date from a database. If you want to find a particular date from a database, you can use this statement.

**For example:** let's see the query to get all the records after '2013-12-12'.

1. **SELECT** \* **FROM**
2. **table**-**name** **WHERE** your **date**-**column** >= '2013-12-12'

Let's see the another query to get all the records after '2013-12-12' and before '2013-12-13' date.

1. **SELECT**\* **FROM**
2. **table**-**name** **where** your **date**-**column** < '2013-12-13' and your **date**-**column** >= '2013-12-12'

If you want to compare the dates within the query, you should use BETWEEN operator to compare the dates.

1. **SELECT** \* **FROM**
2. table\_name **WHERE** yourdate BETWEEN ?2012-12-12? and ?2013-12-12?

# **SQL SELECT SUM**

It is also known as SQL SUM() function. It is used in a SQL query to return summed value of an expression.

Let's see the Syntax for the select sum function:

1. **SELECT** SUM (expression)
2. **FROM** tables
3. **WHERE** conditions;

expression may be numeric field or formula.

This would produce the following result.

|  |  |  |
| --- | --- | --- |
| **ID** | **EMPLOYEE\_NAME** | **SALARY** |
| 1 | JACK REACHER | 32000 |
| 2 | PADMA MAHESHWARI | 22000 |
| 3 | JOE PETRA | 41000 |
| 4 | AMBUJ AGRAWAL | 21000 |

After using this SQL SELECT SUM example, it will produce the result containing the sum of the salary greater than 20000.

Total salary: 116,000

# **SQL SUM EXAMPLE with single field:**

If you want to know how the combined total salary of all employee whose salary is above 20000 per month.

1. **SELECT** SUM (salary) **AS** "Total Salary"
2. **FROM** employees
3. **WHERE** salary > 20000;

In this example, you will find the expression as "Total Salary" when the result set is returned.

# **SQL SUM EXAMPLE with SQL DISTINCT:**

You can also use SQL DISTINCT clause with SQL SUM function.

1. **SELECT** SUM (**DISTINCT** salary) **AS** "Total Salary"
2. **FROM** employees
3. **WHERE** salary > 20000;

# **SQL SUM EXAMPLE with SQL GROUP BY:**

Sometimes there is a need to use the SQL GROUP BY statement with the SQL SUM function.

For example, we could also use the SQL SUM function to return the name of department and the total sales related to department.

1. **SELECT** department, SUM (sales) **AS** "Total Sales"
2. **FROM** order\_details
3. **GROUP** **BY** department;

Let us take a table named order\_details

|  |  |  |  |
| --- | --- | --- | --- |
| **ID** | **DEPARTMENT** | **DATE** | **DAILY SALES** |
| 1 | Mechanical | 2012-08-13 | 360 |
| 2 | Electrical | 2012-08-13 | 100 |
| 2 | Electrical | 2012-08-14 | 110 |
| 3 | Electronics | 2012-08-13 | 150 |
| 3 | Electronics | 2012-08-14 | 170 |

After using the SQL GROUP BY statement with SUM, you will find the following result.

|  |  |
| --- | --- |
| **DEPARTMENT** | **SUM(DAILY SALES)** |
| Mechanical | 360 |
| Electrical | 210 |
| electronics | 320 |

# **SQL SELECT NULL**

First of all we should know that what null value is? Null values are used to represent missing unknown data.

There can be two conditions:

1. Where SQL is NULL
2. Where SQL is NOT NULL

If in a table, a column is optional, it is very easy to insert data in column or update an existing record without adding a value in this column. This means that field has null value.

#### Note: we should not compare null value with 0. They are not equivalent.

# **Where SQL is NULL:**

How to select records with null values only? (in the marks column)

There is an example of student table:

|  |  |  |
| --- | --- | --- |
| **SIR\_NAME** | **NAME** | **MARKS** |
| TYAGI | SEEMA |  |
| SINGH | RAMAN | 5.5 |
| SHARMA | AMAR |  |
| JAISWAL | VICKY | 6.2 |

Let's see the query to get all the records where marks is NULL:

1. **SELECT** SIR\_NAME, **NAME**, MARKS **FROM** STUDENTS
2. **WHERE** MARKS **IS** NULL

It will return the following records:

|  |  |  |
| --- | --- | --- |
| **SIR\_NAME** | **NAME** | **MARKS** |
| SHARMA | AMAR |  |
| TYAGI | SEEMA |  |

# **Where SQL is NOT NULL:**

How to select records with no null values(in marks column)? Let's see the query to get all the records where marks is NOT NULL

1. **SELECT** SIR\_NAME, FIRSTNAME, MARKS **FROM** STUDENTS
2. **WHERE** MARKS **IS** NOT NULL

|  |  |  |
| --- | --- | --- |
| **SIR\_NAME** | **NAME** | **MARKS** |
| SINGH | RAMAN | 5.5 |
| JAISWAL | VICKY | 6.2 |

# **SQL WHERE**

A **WHERE clause** in SQL is a data manipulation language statement.

WHERE clauses are not mandatory clauses of SQL DML statements. But it can be used to limit the number of rows affected by a SQL DML statement or returned by a query.

Actually. it filters the records. It returns only those queries which fulfill the specific conditions.

WHERE clause is used in SELECT, UPDATE, DELETE statement etc.

Let's see the syntax for sql where:

1. **SELECT** column1, **column** 2, ... **column** n
2. **FROM**    table\_name
3. **WHERE** [conditions]

# **SQL AND**

The SQL AND condition is used in SQL query to create two or more conditions to be met.

It is used in SQL SELECT, INSERT, UPDATE and DELETE statements.

Let's see the syntax for SQL AND:

1. **SELECT** columns
2. **FROM** tables
3. **WHERE** condition 1
4. AND condition 2;

The SQL AND condition requires that both conditions should be met.

The SQL AND condition also can be used to join multiple tables in a SQL statement.

# **SQL "AND" example with "INSERT" statement**

This is how an SQL "AND" condition can be used in the SQL INSERT statement.

For example:

1. **INSERT** **INTO** suppliers
2. (supplier\_id, supplier\_name)
3. **SELECT** account\_no, **name**
4. **FROM** customers
5. **WHERE** customer\_name ='IBM'
6. AND employees =1000;

# **SQL "AND" example with "UPDATE" statement**

This is how the "AND" condition can be used in the SQL UPDATE statement.

For example:

1. **UPDATE** suppliers
2. **SET** supplier\_name = 'HP'
3. **WHERE** supplier\_name = 'IBM'
4. AND offices = 8;

# **SQL "AND" example with "DELETE" statement**

This is how an SQL "AND" condition can be used in the SQL DELETE statement.

For example:

1. **DELETE** **FROM** suppliers
2. **WHERE** supplier\_name = 'IBM'
3. AND product = 'PC computers';

SQL OR same as and

# **SQL ORDER BY Clause**

The SQL ORDER BY clause is used for sorting data in ascending and descending order based on one or more columns.

Some databases sort query results in ascending order by default.

SQL ORDER BY syntax:

1. **SELECT** expressions
2. **FROM** tables
3. **WHERE** conditions
4. **ORDER** **BY** expression [**ASC** | **DESC**];

Let us take a CUSTOMERS table having the following records:

|  |
| --- |
|  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ID** | **NAME** | **AGE** | **ADDRESS** | **SALARY** |
| 1 | Himani gupta | 21 | Modinagar | 22000 |
| 2 | Shiva tiwari | 22 | Bhopal | 21000 |
| 3 | Ajeet bhargav | 45 | Meerut | 65000 |
| 4 | Ritesh yadav | 36 | Azamgarh | 26000 |
| 5 | Balwant singh | 45 | Varanasi | 36000 |
| 6 | Mahesh sharma | 26 | Mathura | 22000 |

This is an example that would sort the result in ascending order by NAME and SALARY.

1. **SELECT** \* **FROM** CUSTOMERS
2. **ORDER** **BY** **NAME**, SALARY;

This would produce the following result.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ID** | **NAME** | **AGE** | **ADDRESS** | **SALARY** |
| 3 | Ajeet bhargav | 45 | Meerut | 65000 |
| 5 | Balwant singh | 45 | Varanasi | 36000 |
| 1 | Himani gupta | 21 | Modinagar | 22000 |
| 6 | Mahesh sharma | 26 | Mathura | 22000 |
| 4 | Ritesh yadav | 36 | Azamgarh | 26000 |
| 2 | Shiva tiwari | 22 | Bhopal | 21000 |

This is an example to sort the result in descending order by NAME.

1. **SELECT** \* **FROM** CUSTOMERS
2. **ORDER** **BY** **NAME** **DESC**;

This would produce the following result.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ID** | **NAME** | **AGE** | **ADDRESS** | **SALARY** |
| 2 | Shiva tiwari | 22 | Bhopal | 21000 |
| 4 | Ritesh yadav | 36 | Azamgarh | 26000 |
| 6 | Mahesh sharma | 26 | Mathura | 22000 |
| 1 | Himani gupta | 21 | Modinagar | 22000 |
| 5 | Balwant singh | 45 | Varanasi | 36000 |
| 3 | Ajeet bhargav | 45 | Meerut | 65000 |

# **SQL INSERT STATEMENT**

SQL INSERT statement is a SQL query. It is used to insert a single or a multiple records in a table.

There are two ways to insert data in a table:

1. By SQL insert into statement
   1. By specifying column names
   2. Without specifying column names
2. By SQL insert into select statement

## **1) Inserting data directly into a table**

You can insert a row in the table by using SQL INSERT INTO command. But there are 2 ways to do this.

You can specify or ignore the column names while using INSERT INTO statement.

To insert partial column values, you must have to specify the column names. But if you want to insert all the column values, you can specify or ignore the column names.

If you specify the column names, syntax of the insert into statement will be as follows:

1. **INSERT** **INTO** TABLE\_NAME
2. [(col1, col2, col3,.... col N)]
3. **VALUES** (value1, value2, value 3, .... Value N);

Here col1, col2, col3, .... colN are the columns of the table in which you want to insert data.

##### Note: At the time of inserting a row into table, if you add values for all columns then there is no need to specify the column name in SQL INSERT query. Moreover, you must be sure that you are entering the values in the same order as the columns exist.

But, If you ignore the column names, syntax of the insert into statement will be as follows:

1. **INSERT** **INTO** TABLE\_NAME
2. **VALUES** (value1, value2, value 3, .... Value N);

## **2) Inserting data through SELECT Statement**

**SQL INSERT INTO SELECT Syntax**

1. **INSERT** **INTO** table\_name
2. [(column1, column2, .... **column**)]
3. **SELECT** column1, column2, .... **Column** N
4. **FROM** table\_name [**WHERE** condition];

**Note:** when you add a new row, you should make sure that data type of the value and the column should be matched.

If any integrity constraints are defined for the table, you must follow them.

1. **INSERT** **INTO** STUDENTS (ROLL\_NO, **NAME**, AGE, CITY)
2. **VALUES** (1, ABHIRAM, 22, ALLAHABAD);
3. **INSERT** **INTO** STUDENTS (ROLL\_NO, **NAME**, AGE, CITY)
4. **VALUES** (2, ALKA, 20, GHAZIABAD);
5. **INSERT** **INTO** STUDENTS (ROLL\_NO, **NAME**, AGE, CITY)
6. **VALUES** (3, DISHA, 21, VARANASI);
7. **INSERT** **INTO** STUDENTS (ROLL\_NO, **NAME**, AGE, CITY)
8. **VALUES** (4, ESHA, 21, DELHI);
9. **INSERT** **INTO** STUDENTS (ROLL\_NO, **NAME**, AGE, CITY)
10. **VALUES** (5, MANMEET, 23, JALANDHAR);

# **SQL UPDATE**

The SQL commands (*UPDATE* and *DELETE*) are used to modify the data that is already in the database. The SQL DELETE command uses a WHERE clause.

**SQL UPDATE** statement is used to change the data of the records held by tables. Which rows is to be update, it is decided by a condition. To specify condition, we use WHERE clause.

The UPDATE statement can be written in following form:

1. **UPDATE** table\_name **SET** [column\_name1= value1,... column\_nameN = valueN] [**WHERE** condition]

Let's see the Syntax:

1. **UPDATE** table\_name
2. **SET** column\_name = expression
3. **WHERE** conditions

Let's take an example: here we are going to update an entry in the source table.

SQL statement:

1. **UPDATE** students
2. **SET** User\_Name = 'beinghuman'
3. **WHERE** Student\_Id = '3'

**Source Table:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Student\_Id** | **FirstName** | **LastName** | **User\_Name** |
| 1 | Ada | Sharma | sharmili |
| 2 | Rahul | Maurya | sofamous |
| 3 | James | Walker | jonny |

See the result after updating value:

|  |  |  |  |
| --- | --- | --- | --- |
| **Student\_Id** | **FirstName** | **LastName** | **User\_Name** |
| 1 | Ada | Sharma | sharmili |
| 2 | Rahul | Maurya | sofamous |
| 3 | James | Walker | **beinghuman** |

# **Updating Multiple Fields:**

If you are going to update multiple fields, you should separate each field assignment with a comma.

SQL UPDATE statement for multiple fields:

1. **UPDATE** students
2. **SET** User\_Name = 'beserious', First\_Name = 'Johnny'
3. **WHERE** Student\_Id = '3'

Result of the table is given below:

|  |  |  |  |
| --- | --- | --- | --- |
| **Student\_Id** | **FirstName** | **LastName** | **User\_Name** |
| 1 | Ada | Sharma | sharmili |
| 2 | Rahul | Maurya | sofamous |
| 3 | **Johnny** | Walker | **beserious** |

MYSQL SYNTAX FOR UPDATING TABLE:

1. **UPDATE** table\_name
2. **SET** field1 = new-value1, field2 = new-value2,
3. [**WHERE** CLAUSE]

# **SQL UPDATE with JOIN**

**SQL UPDATE JOIN** means we will update one table using another table and join condition.

Let us take an example of a customer table. I have updated customer table that contains latest customer details from another source system. I want to update the customer table with latest data. In such case, I will perform join between target table and source table using join on customer ID.

Let's see the *syntax* of SQL UPDATE query with JOIN statement.

1. **UPDATE** customer\_table
2. **INNER** JOIN
3. Customer\_table
4. **ON** customer\_table.rel\_cust\_name = customer\_table.cust\_id
5. **SET** customer\_table.rel\_cust\_name = customer\_table.cust\_name

# **How to use multiple tables in SQL UPDATE statement with JOIN**

Let's take two tables, table 1 and table 2.

**Create table1**

1. **CREATE** **TABLE** table1 (column1 **INT**, column2 **INT**, column3 **VARCHAR** (100))
2. **INSERT** **INTO** table1 (col1, col2, col3)
3. **SELECT** 1, 11, 'FIRST'
4. **UNION** ALL
5. **SELECT** 11,12, 'SECOND'
6. **UNION** ALL
7. **SELECT** 21, 13, 'THIRD'
8. **UNION** ALL
9. **SELECT** 31, 14, 'FOURTH'

**Create table2**

1. **CREATE** **TABLE** table2 (column1 **INT**, column2 **INT**, column3 **VARCHAR** (100))
2. **INSERT** **INTO** table2 (col1, col2, col3)
3. **SELECT** 1, 21, 'TWO-ONE'
4. **UNION** ALL
5. **SELECT** 11, 22, 'TWO-TWO'
6. **UNION** ALL
7. **SELECT** 21, 23, 'TWO-THREE'
8. **UNION** ALL
9. **SELECT** 31, 24, 'TWO-FOUR'

Now check the content in the table.

1. **SELECT** \* **FROM** table\_1
2. **SELECT** \* **FROM** table\_2

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Col 1** | **Col 2** | **Col 3** |
| 1 | 1 | 11 | First |
| 2 | 11 | 12 | Second |
| 3 | 21 | **13** | **Third** |
| 4 | 31 | **14** | **Fourth** |

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Col 1** | **Col 2** | **Col 3** |
| 1 | 1 | 21 | Two-One |
| 2 | 11 | 22 | Two-Two |
| 3 | 21 | **23** | **Two-Three** |
| 4 | 31 | **24** | **Two-Four** |

Our requirement is that we have table 2 which has two rows where Col 1 is 21 and 31. We want to update the value from table 2 to table 1 for the rows where Col 1 is 21 and 31.

We want to also update the values of Col 2 and Col 3 only.

The most easiest and common way is to use join clause in the update statement and use multiple tables in the update statement.

1. **UPDATE** **table** 1
2. **SET** Col 2 = t2.Col2,
3. Col 3 = t2.Col3
4. **FROM** table1 t1
5. **INNER** JOIN **table** 2 t2 **ON** t1.Col1 = t2.col1
6. **WHERE** t1.Col1 IN (21,31)

Check the content of the table

SELECT FROM table 1

SELECT FROM table 2

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Col 1** | **Col 2** | **Col 3** |
| 1 | 1 | 11 | First |
| 2 | 11 | 12 | Second |
| 3 | 21 | **23** | **Two-Three** |
| 4 | 31 | **24** | **Two-Four** |

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Col 1** | **Col 2** | **Col 3** |
| 1 | 1 | 21 | First |
| 2 | 11 | 22 | Second |
| 3 | 21 | **23** | **Two-Three** |
| 4 | 31 | **24** | **Two-Four** |

Here we can see that using join clause in update statement. We have merged two tables by the use of join clause.

# **SQL DELETE**

The **SQL DELETE statement** is used to delete rows from a table. Generally DELETE statement removes one or more records from a table.

# **SQL DELETE Example**

Let us take a table, named ?EMPLOYEE? table.

|  |  |  |  |
| --- | --- | --- | --- |
| **ID** | **EMP\_NAME** | **CITY** | **SALARY** |
| 101 | Adarsh Singh | Obra | **20000** |
| 102 | Sanjay Singh | Meerut | **21000** |
| 103 | Priyanka Sharma | Raipur | **25000** |
| 104 | Esha Singhal | Delhi | **26000** |

Example of delete with WHERE clause is given below:

1. **DELETE** **FROM** EMPLOYEE **WHERE** ID=101;

Resulting table after the query:

|  |  |  |  |
| --- | --- | --- | --- |
| **ID** | **EMP\_NAME** | **CITY** | **SALARY** |
| 102 | Sanjay Singh | Meerut | **21000** |
| 103 | Priyanka Sharma | Raipur | **25000** |
| 104 | Esha Singhal | Delhi | **26000** |

Another example of delete statement is given below

1. **DELETE** **FROM** EMPLOYEE;

Resulting table after the query:

|  |  |  |  |
| --- | --- | --- | --- |
| **ID** | **EMP\_NAME** | **CITY** | **SALARY** |

It will delete all the records of EMPLOYEE table.

It will delete the all the records of EMPLOYEE table where ID is 101.

The WHERE clause in the SQL DELETE statement is optional and it identifies the rows in the column that gets deleted.

WHERE clause is used to prevent the deletion of all the rows in the table, If you don't use the WHERE clause you might loss all the rows.

**DROP DATABASE Statement:**

The drop database statement is used to delete a database.

1. **DROP** **DATABASE** database\_name

# **SQL DELETE VIEW**

Before knowing about what is SQL delete view, it is important to know -

**What is SQL view?**

A view is a result set of a stored query on the data.

The SQL view is a table which does not physically exist. It is only a virtual table.

SQL VIEW can be created by a SQL query by joining one or more table.

**Syntax for SQL create view -**

1. **CREATE** **VIEW** view\_name **AS**
2. **SELECT** columns
3. **FROM** tables
4. **WHERE** conditions;

If you want to delete a SQL view, It is done by SQL DROP command you should use the following syntax:

**SQL DROP VIEW syntax:**

1. **DROP** **VIEW** view\_name

**Why use the SQL DROP VIEW statement?**

When a view no longer useful you may drop the view permanently. Also if a view needs change within it, it would be dropped and then created again with changes in appropriate places.

# **SQL JOIN**

As the name shows, JOIN means *to combine something*. In case of SQL, JOIN means **"to combine two or more tables"**.

The SQL JOIN clause takes records from two or more tables in a database and combines it together.

**ANSI standard SQL** defines five types of JOIN :

1. inner join,
2. left outer join,
3. right outer join,
4. full outer join, and
5. cross join.

In the process of joining, rows of both tables are combined in a single table.

# **Why SQL JOIN is used?**

If you want to access more than one table through a select statement.

If you want to combine two or more table then SQL JOIN statement is used .it combines rows of that tables in one table and one can retrieve the information by a SELECT statement.

The joining of two or more tables is based on common field between them.

SQL INNER JOIN also known as simple join is the most common type of join.

# **SQL PRIMARY KEY**

A column or columns is called **primary key (PK)** that *uniquely identifies each row in the table*.

If you want to create a primary key, you should define a PRIMARY KEY constraint when you create or modify a table.

When multiple columns are used as a primary key, it is known as **composite primary key**.

In designing the composite primary key, you should use as few columns as possible. It is good for storage and performance both, the more columns you use for primary key the more storage space you require.

Inn terms of performance, less data means the database can process faster.

# **Points to remember for primary key:**

* Primary key enforces the entity integrity of the table.
* Primary key always has unique data.
* A primary key length cannot be exceeded than 900 bytes.
* A primary key cannot have null value.
* There can be no duplicate value for a primary key.
* A table can contain only one primary key constraint.

# **SQL primary key for one column:**

The following SQL command creates a PRIMARY KEY on the "S\_Id" column when the "students" table is created.

**MySQL:**

1. **CREATE** **TABLE** students
2. (
3. S\_Id **int** NOT NULL,
4. LastName **varchar** (255) NOT NULL,
5. FirstName **varchar** (255),
6. Address **varchar** (255),
7. City **varchar** (255),
8. **PRIMARY** **KEY** (S\_Id)
9. )

# **How to DROP a PRIMARY KEY constraint?**

If you want to DROP (remove) a primary key constraint, you should use following syntax:

**MySQL:**

1. **ALTER** **TABLE** students
2. **DROP** **PRIMARY** **KEY**

# **SQL FOREIGN KEY**

In the relational databases, a foreign key is a field or a column that is used to establish a link between two tables.

In simple words you can say that, a foreign key in one table used to point primary key in another table.

Let us take an example to explain it:

Here are two tables first one is students table and second is orders table.

Here orders are given by students.

**First table:**

|  |  |  |  |
| --- | --- | --- | --- |
| **S\_Id** | **LastName** | **FirstName** | **CITY** |
| 1 | MAURYA | AJEET | ALLAHABAD |
| 2 | JAISWAL | RATAN | GHAZIABAD |
| 3 | ARORA | SAUMYA | MODINAGAR |

**Second table:**

|  |  |  |
| --- | --- | --- |
| **O\_Id** | **OrderNo** | **S\_Id** |
| 1 | 99586465 | 2 |
| 2 | 78466588 | 2 |
| 3 | 22354846 | 3 |
| 4 | 57698656 | 1 |

Here you see that "S\_Id" column in the "Orders" table points to the "S\_Id" column in "Students" table.

* The "S\_Id" column in the "Students" table is the PRIMARY KEY in the "Students" table.
* The "S\_Id" column in the "Orders" table is a FOREIGN KEY in the "Orders" table.

The foreign key constraint is generally prevents action that destroy links between tables.

It also prevents invalid data to enter in foreign key column.

# **SQL FOREIGN KEY constraint ON CREATE TABLE:**

(Defining a foreign key constraint on single column)

To create a foreign key on the "S\_Id" column when the "Orders" table is created:

**MySQL:**

1. **CREATE** **TABLE** orders
2. (
3. O\_Id **int** NOT NULL,
4. Order\_No  **int** NOT NULL,
5. S\_Id **int**,
6. PRIMAY **KEY** (O\_Id),
7. **FOREIGN** **KEY** (S\_Id) **REFERENCES** Persons (S\_Id)
8. )

**SQL Server /Oracle / MS Access:**

1. **CREATE** **TABLE** Orders
2. (
3. O\_Id **int** NOT NULL PRIMAY **KEY**,
4. Order\_No **int** NOT NULL,
5. S\_Id **int** **FOREIGN** **KEY** **REFERENCES** persons (S\_Id)
6. )

# **SQL FOREIGN KEY constraint for ALTER TABLE:**

If the Order table is already created and you want to create a FOREIGN KEY constraint on the ?S\_Id? column, you should write the following syntax:

**Defining a foreign key constraint on single column:**

**MySQL / SQL Server / Oracle / MS Access:**

1. **ALTER** **TABLE** Orders
2. **ADD** **CONSTRAINT** fk\_PerOrders
3. **FOREIGN** **KEY**(S\_Id)
4. **REFERENCES** Students (S\_Id)

# **DROP SYNTAX for FOREIGN KEY COSTRAINT:**

If you want to drop a FOREIGN KEY constraint, use the following syntax:

**MySQL:**

1. **ALTER** **TABLE** Orders
2. ROP **FOREIGN** **KEY** fk\_PerOrders

**SQL Server / Oracle / MS Access:**

1. **ALTER** **TABLE** Orders
2. **DROP** **CONSTRAINT** fk\_PerOrders

# **Difference between primary key and foreign key in SQL:**

These are some important difference between primary key and foreign key in SQL-

Primary key cannot be null on the other hand foreign key can be null.

Primary key is always unique while foreign key can be duplicated.

Primary key uniquely identify a record in a table while foreign key is a field in a table that is primary key in another table.

There is only one primary key in the table on the other hand we can have more than one foreign key in the table.

By default primary key adds a clustered index on the other hand foreign key does not automatically create an index, clustered or non-clustered. You must manually create an index for foreign key.

# **SQL Composite Key**

A composite key is a combination of two or more columns in a table that can be used to uniquely identify each row in the table when the columns are combined uniqueness is guaranteed, but when it taken individually it does not guarantee uniqueness.

Sometimes more than one attributes are needed to uniquely identify an entity. A primary key that is made by the combination of more than one attribute is known as a composite key.

In other words we can say that:

Composite key is a key which is the combination of more than one field or column of a given table. It may be a candidate key or primary key.

Columns that make up the composite key can be of different data types.

**SQL Syntax to specify composite key:**

1. **CREATE** **TABLE** TABLE\_NAME
2. (COLUMN\_1, DATA\_TYPE\_1,
3. COLUMN\_2, DATA\_TYPE\_2,
4. ???
5. **PRIMARY** **KEY** (COLUMN\_1, COLUMN\_2, ...));

In all cases composite key created consist of COLUMN1 and COLUMN2.

MySQL:

1. **CREATE** **TABLE** SAMPLE\_TABLE
2. (COL1 **integer**,
3. COL2 **varchar**(30),
4. COL3 **varchar**(50),
5. **PRIMARY** **KEY** (COL1, COL2));

MySQL:

1. **CREATE** **TABLE** SAMPLE\_TABLE
2. (COL1 **integer**,
3. COL2 **varchar**(30),
4. COL3 **varchar**(50),
5. **PRIMARY** **KEY** (COL1, COL2));

Oracle:

1. **CREATE** **TABLE** SAMPLE\_TABLE
2. **CREATE** **TABLE** SAMPLE\_TABLE
3. (COL1 **integer**,
4. COL2 **varchar**(30),
5. COL3 **varchar**(50),
6. **PRIMARY** **KEY** (COL1, COL2));

**SQL Server:**

Let's see the Syntax for the select top statement:

1. **CREATE** **TABLE** SAMPLE\_TABLE
2. (COL1 **integer**,
3. COL2 nvarchar(30),
4. COL3 nvarchar(50),
5. **PRIMARY** **KEY** (COL1, COL2));

# **Unique Key in SQL**

A unique key is a set of one or more than one fields/columns of a table that uniquely identify a record in a database table.

You can say that it is little like primary key but it can accept only one null value and it cannot have duplicate values.

The unique key and primary key both provide a guarantee for uniqueness for a column or a set of columns.

There is an automatically defined unique key constraint within a primary key constraint.

There may be many unique key constraints for one table, but only one PRIMARY KEY constraint for one table.

**SQL UNIQUE KEY constraint on CREATE TABLE:**

If you want to create a UNIQUE constraint on the ?S\_Id? column when the ?students? table is created, use the following SQL syntax:

**SQL Server / Oracle / MS Access:**

**(Defining a unique key constraint on single column):**

1. **CREATE** **TABLE** students
2. (
3. S\_Id **int** NOT NULL **UNIQUE**,
4. LastName **varchar** (255) NOT NULL,
5. FirstName **varchar** (255),
6. City **varchar** (255)
7. )

**MySQL:**

1. **CREATE** **TABLE** students
2. **CREATE** **TABLE** students
3. (
4. S\_Id **int** NOT NULL,
5. LastName **varchar** (255) NOT NULL,
6. FirstName **varchar** (255),
7. City **varchar** (255),
8. **UNIQUE** (S\_Id)
9. )

**(Defining a unique key constraint on multiple columns):**

**MySQL / SQL Server / Oracle / MS Access:**

1. **CREATE** **TABLE** students
2. (
3. S\_Id **int** NOT NULL,
4. LastName **varchar** (255) NOT NULL,
5. FirstName **varchar** (255),
6. City **varchar** (255),
7. **CONSTRAINT** uc\_studentId **UNIQUE** (S\_Id, LastName)
8. )

**SQL UNIQUE KEY constraint on ALTER TABLE:**

If you want to create a unique constraint on ?S\_Id? column when the table is already created, you should use the following SQL syntax:

**(Defining a unique key constraint on single column):**

**MySQL / SQL Server / Oracle / MS Access:**

1. **ALTER** **TABLE** students
2. **ADD** **UNIQUE** (S\_Id)

**(Defining a unique key constraint on multiple columns):**

**MySQL / SQL Server / Oracle / MS Access:**

1. **ALTER** **TABLE** students
2. **ADD** **CONSTRAINT** uc\_StudentId **UNIQUE**  (S\_Id, LastName)

**DROP SYNTAX FOR A FOREIGN KEY constraint:**

If you want to drop a UNIQUE constraint, use the following SQL syntax:

**MySQL:**

1. **ALTER** **TABLE** students
2. **DROP** **INDEX** uc\_studentID

**SQL Server / Oracle / MS Access:**

1. **ALTER** **TABLE** students
2. **DROP** **CONSTRAINT** uc\_studentID