SQL is a database computer language designed for the retrieval and management of data in relational database. SQL stands for Structured Query Language.

QL tutorial gives unique learning on **Structured Query Language** and it helps to make practice on SQL commands which provides immediate results. SQL is a language of database, it includes database creation, deletion, fetching rows and modifying rows etc.

SQL is an ANSI (American National Standards Institute) standard but there are many different versions of the SQL language.

What is SQL?

SQL is Structured Query Language, which is a computer language for storing, manipulating and retrieving data stored in relational database.

SQL is the standard language for Relation Database System. All relational database management systems like MySQL, MS Access, Oracle, Sybase, Informix, postgres and SQL Server use SQL as standard database language.

Also, they are using different dialects, such as:

* MS SQL Server using T-SQL,
* Oracle using PL/SQL,
* MS Access version of SQL is called JET SQL (native format) etc.

Why SQL?

* Allows users to access data in relational database management systems.
* Allows users to describe the data.
* Allows users to define the data in database and manipulate that data.
* Allows to embed within other languages using SQL modules, libraries & pre-compilers.
* Allows users to create and drop databases and tables.
* Allows users to create view, stored procedure, functions in a database.
* Allows users to set permissions on tables, procedures, and views

History:

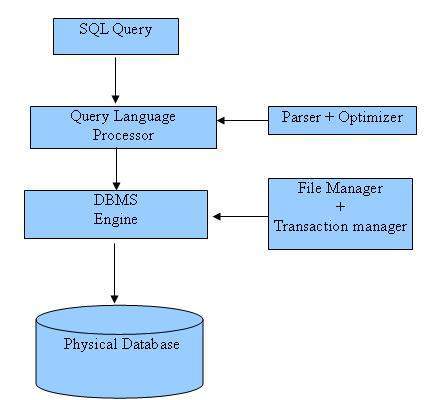
* **1970 --**Dr. Edgar F. "Ted" Codd of IBM is known as the father of relational databases. He described a relational model for databases.
* **1974 --**Structured Query Language appeared.
* **1978 --**IBM worked to develop Codd's ideas and released a product named System/R.
* **1986 --**IBM developed the first prototype of relational database and standardized by ANSI. The first relational database was released by Relational Software and its later becoming Oracle.

SQL Process:

When you are executing an SQL command for any RDBMS, the system determines the best way to carry out your request and SQL engine figures out how to interpret the task.

There are various components included in the process. These components are Query Dispatcher, Optimization Engines, Classic Query Engine and SQL Query Engine, etc. Classic query engine handles all non-SQL queries but SQL query engine won't handle logical files.

Following is a simple diagram showing SQL Architecture:



SQL Commands:

The standard SQL commands to interact with relational databases are CREATE, SELECT, INSERT, UPDATE, DELETE and DROP. These commands can be classified into groups based on their nature:

DDL - Data Definition Language:

|  |  |
| --- | --- |
| **Command** | **Description** |
| CREATE | Creates a new table, a view of a table, or other object in database |
| ALTER | Modifies an existing database object, such as a table. |
| DROP | Deletes an entire table, a view of a table or other object in the database. |

DML - Data Manipulation Language:

|  |  |
| --- | --- |
| **Command** | **Description** |
| SELECT | Retrieves certain records from one or more tables |
| INSERT | Creates a record |
| UPDATE | Modifies records |
| DELETE | Deletes records |

DCL - Data Control Language:

|  |  |
| --- | --- |
| **Command** | **Description** |
| GRANT | Gives a privilege to user |
| REVOKE | Takes back privileges granted from user |

What is RDBMS?

RDBMS stands for **R**elational **D**atabase **M**anagement **S**ystem. RDBMS is the basis for SQL, and for all modern database systems like MS SQL Server, IBM DB2, Oracle, MySQL, and Microsoft Access.

A Relational database management system (RDBMS) is a database management system (DBMS) that is based on the relational model as introduced by E. F. Codd.

What is table?

The data in RDBMS is stored in database objects called **tables**. The table is a collection of related data entries and it consists of columns and rows.

Remember, a table is the most common and simplest form of data storage in a relational database. Following is the example of a CUSTOMERS table:

+----+----------+-----+-----------+----------+

| ID | NAME | AGE | ADDRESS | SALARY |

+----+----------+-----+-----------+----------+

| 1 | Ramesh | 32 | Ahmedabad | 2000.00 |

| 2 | Khilan | 25 | Delhi | 1500.00 |

| 3 | kaushik | 23 | Kota | 2000.00 |

| 4 | Chaitali | 25 | Mumbai | 6500.00 |

| 5 | Hardik | 27 | Bhopal | 8500.00 |

| 6 | Komal | 22 | MP | 4500.00 |

| 7 | Muffy | 24 | Indore | 10000.00 |

+----+----------+-----+-----------+----------+

What is field?

Every table is broken up into smaller entities called fields. The fields in the CUSTOMERS table consist of ID, NAME, AGE, ADDRESS and SALARY.

A field is a column in a table that is designed to maintain specific information about every record in the table.

What is record or row?

A record, also called a row of data, is each individual entry that exists in a table. For example there are 7 records in the above CUSTOMERS table. Following is a single row of data or record in the CUSTOMERS table:

+----+----------+-----+-----------+----------+

| 1 | Ramesh | 32 | Ahmedabad | 2000.00 |

+----+----------+-----+-----------+----------+

A record is a horizontal entity in a table.

What is column?

A column is a vertical entity in a table that contains all information associated with a specific field in a table.

For example, a column in the CUSTOMERS table is ADDRESS, which represents location description and would consist of the following:

+-----------+

| ADDRESS |

+-----------+

| Ahmedabad |

| Delhi |

| Kota |

| Mumbai |

| Bhopal |

| MP |

| Indore |

+----+------+

What is NULL value?

A NULL value in a table is a value in a field that appears to be blank, which means a field with a NULL value is a field with no value.

It is very important to understand that a NULL value is different than a zero value or a field that contains spaces. A field with a NULL value is one that has been left blank during record creation.

SQL Constraints:

Constraints are the rules enforced on data columns on table. These are used to limit the type of data that can go into a table. This ensures the accuracy and reliability of the data in the database.

Constraints could be column level or table level. Column level constraints are applied only to one column where as table level constraints are applied to the whole table.

Following are commonly used constraints available in SQL:

* [NOT NULL Constraint](https://www.tutorialspoint.com/sql/sql-not-null.htm): Ensures that a column cannot have NULL value.
* [DEFAULT Constraint](https://www.tutorialspoint.com/sql/sql-default.htm): Provides a default value for a column when none is specified.
* [UNIQUE Constraint](https://www.tutorialspoint.com/sql/sql-unique.htm): Ensures that all values in a column are different.
* [PRIMARY Key](https://www.tutorialspoint.com/sql/sql-primary-key.htm): Uniquely identified each rows/records in a database table.
* [FOREIGN Key](https://www.tutorialspoint.com/sql/sql-foreign-key.htm): Uniquely identified a rows/records in any another database table.
* [CHECK Constraint](https://www.tutorialspoint.com/sql/sql-check.htm): The CHECK constraint ensures that all values in a column satisfy certain conditions.
* [INDEX](https://www.tutorialspoint.com/sql/sql-index.htm): Use to create and retrieve data from the database very quickly.

Data Integrity:

The following categories of the data integrity exist with each RDBMS:

* **Entity Integrity:** There are no duplicate rows in a table.
* **Domain Integrity:** Enforces valid entries for a given column by restricting the type, the format, or the range of values.
* **Referential integrity:** Rows cannot be deleted, which are used by other records.
* **User-Defined Integrity:** Enforces some specific business rules that do not fall into entity, domain or referential integrity.

Database Normalization

Database normalization is the process of efficiently organizing data in a database. There are two reasons of the normalization process:

* Eliminating redundant data, for example, storing the same data in more than one tables.
* Ensuring data dependencies make sense.

Both of these are worthy goals as they reduce the amount of space a database consumes and ensure that data is logically stored. Normalization consists of a series of guidelines that help guide you in creating a good database structure.

Normalization guidelines are divided into normal forms; think of form as the format or the way a database structure is laid out. The aim of normal forms is to organize the database structure so that it complies with the rules of first normal form, then second normal form, and finally third normal form.

It's your choice to take it further and go to fourth normal form, fifth normal form, and so on, but generally speaking, third normal form is enough.

* [First Normal Form (1NF)](https://www.tutorialspoint.com/sql/first-normal-form.htm)
* [Second Normal Form (2NF)](https://www.tutorialspoint.com/sql/second-normal-form.htm)
* [Third Normal Form (3NF)](https://www.tutorialspoint.com/sql/third-normal-form.htm)

here are many popular RDBMS available to work with. This tutorial gives a brief overview of few most popular RDBMS. This would help you to compare their basic features.

## MySQL

MySQL is an open source SQL database, which is developed by Swedish company MySQL AB. MySQL is pronounced "my ess-que-ell," in contrast with SQL, pronounced "sequel."

MySQL is supporting many different platforms including Microsoft Windows, the major Linux distributions, UNIX, and Mac OS X.

MySQL has free and paid versions, depending on its usage (non-commercial/commercial) and features. MySQL comes with a very fast, multi-threaded, multi-user, and robust SQL database server.

## History:

* Development of MySQL by Michael Widenius & David Axmark beginning in 1994.
* First internal release on 23 May 1995.
* Windows version was released on 8 January 1998 for Windows 95 and NT.
* Version 3.23: beta from June 2000, production release January 2001.
* Version 4.0: beta from August 2002, production release March 2003 (unions).
* Version 4.01: beta from August 2003, Jyoti adopts MySQL for database tracking.
* Version 4.1: beta from June 2004, production release October 2004.
* Version 5.0: beta from March 2005, production release October 2005.
* Sun Microsystems acquired MySQL AB on 26 February 2008.
* Version 5.1: production release 27 November 2008.

## Features:

* High Performance.
* High Availability.
* Scalability and Flexibility Run anything.
* Robust Transactional Support.
* Web and Data Warehouse Strengths.
* Strong Data Protection.
* Comprehensive Application Development.
* Management Ease.
* Open Source Freedom and 24 x 7 Support.
* Lowest Total Cost of Ownership.

## MS SQL Server

MS SQL Server is a Relational Database Management System developed by Microsoft Inc. Its primary query languages are:

* T-SQL.
* ANSI SQL.

## History:

* 1987 - Sybase releases SQL Server for UNIX.
* 1988 - Microsoft, Sybase, and Aston-Tate port SQL Server to OS/2.
* 1989 - Microsoft, Sybase, and Aston-Tate release SQL Server 1.0 for OS/2.
* 1990 - SQL Server 1.1 is released with support for Windows 3.0 clients.
* Aston - Tate drops out of SQL Server development.
* 2000 - Microsoft releases SQL Server 2000.
* 2001 - Microsoft releases XML for SQL Server Web Release 1 (download).
* 2002 - Microsoft releases SQLXML 2.0 (renamed from XML for SQL Server).
* 2002 - Microsoft releases SQLXML 3.0.
* 2005 - Microsoft releases SQL Server 2005 on November 7th, 2005.

## Features:

* High Performance.
* High Availability.
* Database mirroring.
* Database snapshots.
* CLR integration.
* Service Broker.
* DDL triggers.
* Ranking functions.
* Row version-based isolation levels.
* XML integration.
* TRY...CATCH.
* Database Mail.

## ORACLE

It is a very large and multi-user database management system. Oracle is a relational database management system developed by 'Oracle Corporation'.

Oracle works to efficiently manage its resource, a database of information, among the multiple clients requesting and sending data in the network.

It is an excellent database server choice for client/server computing. Oracle supports all major operating systems for both clients and servers, including MSDOS, NetWare, UnixWare, OS/2 and most UNIX flavors.

## History:

Oracle began in 1977 and celebrating its 32 wonderful years in the industry (from 1977 to 2009).

* 1977 - Larry Ellison, Bob Miner and Ed Oates founded Software Development Laboratories to undertake development work.
* 1979 - Version 2.0 of Oracle was released and it became first commercial relational database and first SQL database. The company changed its name to Relational Software Inc. (RSI).
* 1981 - RSI started developing tools for Oracle.
* 1982 - RSI was renamed to Oracle Corporation.
* 1983 - Oracle released version 3.0, rewritten in C language and ran on multiple platforms.
* 1984 - Oracle version 4.0 was released. It contained features like concurrency control - multi-version read consistency, etc.
* 1985 - Oracle version 4.0 was released. It contained features like concurrency control - multi-version read consistency, etc.
* 2007 - Oracle has released Oracle11g. The new version focused on better partitioning, easy migration etc.

## Features:

* Concurrency
* Read Consistency
* Locking Mechanisms
* Quiesce Database
* Portability
* Self-managing database
* SQL\*Plus
* ASM
* Scheduler
* Resource Manager
* Data Warehousing
* Materialized views
* Bitmap indexes
* Table compression
* Parallel Execution
* Analytic SQL
* Data mining
* Partitioning

## MS ACCESS

This is one of the most popular Microsoft products. Microsoft Access is an entry-level database management software. MS Access database is not only an inexpensive but also powerful database for small-scale projects.

MS Access uses the Jet database engine, which utilizes a specific SQL language dialect (sometimes referred to as Jet SQL).

MS Access comes with the professional edition of MS Office package. MS Access has easy-to-use intuitive graphical interface.

* 1992 - Access version 1.0 was released.
* 1993 - Access 1.1 released to improve compatibility with inclusion the Access Basic programming language.
* The most significant transition was from Access 97 to Access 2000
* 2007 - Access 2007, a new database format was introduced ACCDB which supports complex data types such as multi valued and attachment fields.

## Features:

* Users can create tables, queries, forms and reports and connect them together with macros.
* The import and export of data to many formats including Excel, Outlook, ASCII, dBase, Paradox, FoxPro, SQL Server, Oracle, ODBC, etc.
* There is also the Jet Database format (MDB or ACCDB in Access 2007), which can contain the application and data in one file. This makes it very convenient to distribute the entire application to another user, who can run it in disconnected environments.
* Microsoft Access offers parameterized queries. These queries and Access tables can be referenced from other programs like VB6 and .NET through DAO or ADO.
* The desktop editions of Microsoft SQL Server can be used with Access as an alternative to the Jet Database Engine.
* Microsoft Access is a file server-based database. Unlike client-server relational database management systems (RDBMS), Microsoft Access does not implement database triggers, stored procedures, or transaction logging.
* SQL is followed by unique set of rules and guidelines called Syntax. This tutorial gives you a quick start with SQL by listing all the basic SQL Syntax:
* All the SQL statements start with any of the keywords like SELECT, INSERT, UPDATE, DELETE, ALTER, DROP, CREATE, USE, SHOW and all the statements end with a semicolon (;).
* Important point to be noted is that SQL is **case insensitive**, which means SELECT and select have same meaning in SQL statements, but MySQL makes difference in table names. So if you are working with MySQL, then you need to give table names as they exist in the database.
* All the examples given in this tutorial have been tested with MySQL server.

## SQL SELECT Statement:

* SELECT column1, column2....columnN
* FROM table\_name;

## SQL DISTINCT Clause:

* SELECT DISTINCT column1, column2....columnN
* FROM table\_name;

## SQL WHERE Clause:

* SELECT column1, column2....columnN
* FROM table\_name
* WHERE CONDITION;

## SQL AND/OR Clause:

* SELECT column1, column2....columnN
* FROM table\_name
* WHERE CONDITION-1 {AND|OR} CONDITION-2;

## SQL IN Clause:

* SELECT column1, column2....columnN
* FROM table\_name
* WHERE column\_name IN (val-1, val-2,...val-N);

## SQL BETWEEN Clause:

* SELECT column1, column2....columnN
* FROM table\_name
* WHERE column\_name BETWEEN val-1 AND val-2;

## SQL LIKE Clause:

* SELECT column1, column2....columnN
* FROM table\_name
* WHERE column\_name LIKE { PATTERN };

## SQL ORDER BY Clause:

* SELECT column1, column2....columnN
* FROM table\_name
* WHERE CONDITION
* ORDER BY column\_name {ASC|DESC};

## SQL GROUP BY Clause:

* SELECT SUM(column\_name)
* FROM table\_name
* WHERE CONDITION
* GROUP BY column\_name;

## SQL COUNT Clause:

* SELECT COUNT(column\_name)
* FROM table\_name
* WHERE CONDITION;

## SQL HAVING Clause:

* SELECT SUM(column\_name)
* FROM table\_name
* WHERE CONDITION
* GROUP BY column\_name
* HAVING (arithematic function condition);

## SQL CREATE TABLE Statement:

* CREATE TABLE table\_name(
* column1 datatype,
* column2 datatype,
* column3 datatype,
* .....
* columnN datatype,
* PRIMARY KEY( one or more columns )
* );

## SQL DROP TABLE Statement:

* DROP TABLE table\_name;

## SQL CREATE INDEX Statement :

* CREATE UNIQUE INDEX index\_name
* ON table\_name ( column1, column2,...columnN);

## SQL DROP INDEX Statement :

* ALTER TABLE table\_name
* DROP INDEX index\_name;

## SQL DESC Statement :

* DESC table\_name;

## SQL TRUNCATE TABLE Statement:

* TRUNCATE TABLE table\_name;

## SQL ALTER TABLE Statement:

* ALTER TABLE table\_name {ADD|DROP|MODIFY} column\_name {data\_ype};

## SQL ALTER TABLE Statement (Rename) :

* ALTER TABLE table\_name RENAME TO new\_table\_name;

## SQL INSERT INTO Statement:

* INSERT INTO table\_name( column1, column2....columnN)
* VALUES ( value1, value2....valueN);

## SQL UPDATE Statement:

* UPDATE table\_name
* SET column1 = value1, column2 = value2....columnN=valueN
* [ WHERE CONDITION ];

## SQL DELETE Statement:

* DELETE FROM table\_name
* WHERE {CONDITION};

## SQL CREATE DATABASE Statement:

* CREATE DATABASE database\_name;

## SQL DROP DATABASE Statement:

* DROP DATABASE database\_name;

## SQL USE Statement:

* USE database\_name;

## SQL COMMIT Statement:

* COMMIT;

## SQL ROLLBACK Statement:

* ROLLBACK;
* SQL data type is an attribute that specifies type of data of any object. Each column, variable and expression has related data type in SQL.
* You would use these data types while creating your tables. You would choose a particular data type for a table column based on your requirement.
* SQL Server offers six categories of data types for your use −

## Exact Numeric Data Types

|  |  |  |
| --- | --- | --- |
| **DATA TYPE** | **FROM** | **TO** |
| bigint | -9,223,372,036,854,775,808 | 9,223,372,036,854,775,807 |
| int | -2,147,483,648 | 2,147,483,647 |
| smallint | -32,768 | 32,767 |
| tinyint | 0 | 255 |
| bit | 0 | 1 |
| decimal | -10^38 +1 | 10^38 -1 |
| numeric | -10^38 +1 | 10^38 -1 |
| money | -922,337,203,685,477.5808 | +922,337,203,685,477.5807 |
| smallmoney | -214,748.3648 | +214,748.3647 |

## Approximate Numeric Data Types

|  |  |  |
| --- | --- | --- |
| **DATA TYPE** | **FROM** | **TO** |
| float | -1.79E + 308 | 1.79E + 308 |
| real | -3.40E + 38 | 3.40E + 38 |

## Date and Time Data Types

|  |  |  |
| --- | --- | --- |
| **DATA TYPE** | **FROM** | **TO** |
| datetime | Jan 1, 1753 | Dec 31, 9999 |
| smalldatetime | Jan 1, 1900 | Jun 6, 2079 |
| date | Stores a date like June 30, 1991 | |
| time | Stores a time of day like 12:30 P.M. | |

* **Note** − Here, datetime has 3.33 milliseconds accuracy where as smalldatetime has 1 minute accuracy.

## Character Strings Data Types

|  |  |
| --- | --- |
| **DATA TYPE** | **Description** |
| char | Maximum length of 8,000 characters.( Fixed length non-Unicode characters) |
| varchar | Maximum of 8,000 characters.(Variable-length non-Unicode data). |
| varchar(max) | Maximum length of 231characters, Variable-length non-Unicode data (SQL Server 2005 only). |
| text | Variable-length non-Unicode data with a maximum length of 2,147,483,647 characters. |

## Unicode Character Strings Data Types

|  |  |
| --- | --- |
| **DATA TYPE** | **Description** |
| nchar | Maximum length of 4,000 characters.( Fixed length Unicode) |
| nvarchar | Maximum length of 4,000 characters.(Variable length Unicode) |
| nvarchar(max) | Maximum length of 231characters (SQL Server 2005 only).( Variable length Unicode) |
| ntext | Maximum length of 1,073,741,823 characters. ( Variable length Unicode ) |

## Binary Data Types

|  |  |
| --- | --- |
| **DATA TYPE** | **Description** |
| binary | Maximum length of 8,000 bytes(Fixed-length binary data ) |
| varbinary | Maximum length of 8,000 bytes.(Variable length binary data) |
| varbinary(max) | 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) |

## Misc Data Types

|  |  |
| --- | --- |
| **DATA TYPE** | **Description** |
| sql\_variant | Stores values of various SQL Server-supported data types, except text, ntext, and timestamp. |
| timestamp | Stores a database-wide unique number that gets updated every time a row gets updated |
| uniqueidentifier | Stores a globally unique identifier (GUID) |
| xml | Stores XML data. You can store xml instances in a column or a variable (SQL Server 2005 only). |
| cursor | Reference to a cursor object |
| table | Stores a result set for later processing |

What is an Operator in SQL?

An operator is a reserved word or a character used primarily in an SQL statement's WHERE clause to perform operation(s), such as comparisons and arithmetic operations.

Operators are used to specify conditions in an SQL statement and to serve as conjunctions for multiple conditions in a statement.

* Arithmetic operators
* Comparison operators
* Logical operators
* Operators used to negate conditions

SQL Arithmetic Operators:

Assume variable a holds 10 and variable b holds 20, then:

[Show Examples](https://www.tutorialspoint.com/sql/sql-arithmetic-operators.htm)

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| + | Addition - Adds values on either side of the operator | a + b will give 30 |
| - | Subtraction - Subtracts right hand operand from left hand operand | a - b will give -10 |
| \* | Multiplication - Multiplies values on either side of the operator | a \* b will give 200 |
| / | Division - Divides left hand operand by right hand operand | b / a will give 2 |
| % | Modulus - Divides left hand operand by right hand operand and returns remainder | b % a will give 0 |

SQL Comparison Operators:

Assume variable a holds 10 and variable b holds 20, then:

[Show Examples](https://www.tutorialspoint.com/sql/sql-comparison-operators.htm)

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| = | Checks if the values of two operands are equal or not, if yes then condition becomes true. | (a = b) is not true. |
| != | Checks if the values of two operands are equal or not, if values are not equal then condition becomes true. | (a != b) is true. |
| <> | Checks if the values of two operands are equal or not, if values are not equal then condition becomes true. | (a <> b) is true. |
| > | Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true. | (a > b) is not true. |
| < | Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true. | (a < b) is true. |
| >= | Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true. | (a >= b) is not true. |
| <= | Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true. | (a <= b) is true. |
| !< | Checks if the value of left operand is not less than the value of right operand, if yes then condition becomes true. | (a !< b) is false. |
| !> | Checks if the value of left operand is not greater than the value of right operand, if yes then condition becomes true. | (a !> b) is true. |

SQL Logical Operators:

Here is a list of all the logical operators available in SQL.

[Show Examples](https://www.tutorialspoint.com/sql/sql-logical-operators.htm)

|  |  |
| --- | --- |
| **Operator** | **Description** |
| ALL | The ALL operator is used to compare a value to all values in another value set. |
| AND | The AND operator allows the existence of multiple conditions in an SQL statement's WHERE clause. |
| ANY | The ANY operator is used to compare a value to any applicable value in the list according to the condition. |
| BETWEEN | The BETWEEN operator is used to search for values that are within a set of values, given the minimum value and the maximum value. |
| EXISTS | The EXISTS operator is used to search for the presence of a row in a specified table that meets certain criteria. |
| IN | The IN operator is used to compare a value to a list of literal values that have been specified. |
| LIKE | The LIKE operator is used to compare a value to similar values using wildcard operators. |
| NOT | The NOT operator reverses the meaning of the logical operator with which it is used. Eg: NOT EXISTS, NOT BETWEEN, NOT IN, etc. **This is a negate operator.** |
| OR | The OR operator is used to combine multiple conditions in an SQL statement's WHERE clause. |
| IS NULL | The NULL operator is used to compare a value with a NULL value. |
| UNIQUE | The UNIQUE operator searches every row of a specified table for uniqueness (no duplicates). |

# **SQL - Expressions**

Advertisements

[Previous Page](https://www.tutorialspoint.com/sql/sql-operators.htm)

[Next Page](https://www.tutorialspoint.com/sql/sql-create-database.htm)

An expression is a combination of one or more values, operators, and SQL functions that evaluate to a value.

SQL EXPRESSIONs are like formulas and they are written in query language. You can also use them to query the database for specific set of data.

## Syntax:

Consider the basic syntax of the SELECT statement as follows:

SELECT column1, column2, columnN

FROM table\_name

WHERE [CONDITION|EXPRESSION];

There are different types of SQL expressions, which are mentioned below:

## SQL - Boolean Expressions:

SQL Boolean Expressions fetch the data on the basis of matching single value. Following is the syntax:

SELECT column1, column2, columnN

FROM table\_name

WHERE SINGLE VALUE MATCHING EXPRESSION;

Consider the CUSTOMERS table having the following records:

SQL> SELECT \* FROM CUSTOMERS;

+----+----------+-----+-----------+----------+

| ID | NAME | AGE | ADDRESS | SALARY |

+----+----------+-----+-----------+----------+

| 1 | Ramesh | 32 | Ahmedabad | 2000.00 |

| 2 | Khilan | 25 | Delhi | 1500.00 |

| 3 | kaushik | 23 | Kota | 2000.00 |

| 4 | Chaitali | 25 | Mumbai | 6500.00 |

| 5 | Hardik | 27 | Bhopal | 8500.00 |

| 6 | Komal | 22 | MP | 4500.00 |

| 7 | Muffy | 24 | Indore | 10000.00 |

+----+----------+-----+-----------+----------+

7 rows in set (0.00 sec)

Here is simple example showing usage of SQL Boolean Expressions:

SQL> SELECT \* FROM CUSTOMERS WHERE SALARY = 10000;

+----+-------+-----+---------+----------+

| ID | NAME | AGE | ADDRESS | SALARY |

+----+-------+-----+---------+----------+

| 7 | Muffy | 24 | Indore | 10000.00 |

+----+-------+-----+---------+----------+

1 row in set (0.00 sec)

## SQL - Numeric Expression:

This expression is used to perform any mathematical operation in any query. Following is the syntax:

SELECT numerical\_expression as OPERATION\_NAME

[FROM table\_name

WHERE CONDITION] ;

Here numerical\_expression is used for mathematical expression or any formula. Following is a simple examples showing usage of SQL Numeric Expressions:

SQL> SELECT (15 + 6) AS ADDITION

+----------+

| ADDITION |

+----------+

| 21 |

+----------+

1 row in set (0.00 sec)

There are several built-in functions like avg(), sum(), count(), etc., to perform what is known as aggregate data calculations against a table or a specific table column.

SQL> SELECT COUNT(\*) AS "RECORDS" FROM CUSTOMERS;

+---------+

| RECORDS |

+---------+

| 7 |

+---------+

1 row in set (0.00 sec)

## SQL - Date Expressions:

Date Expressions return current system date and time values:

SQL> SELECT CURRENT\_TIMESTAMP;

+---------------------+

| Current\_Timestamp |

+---------------------+

| 2009-11-12 06:40:23 |

+---------------------+

1 row in set (0.00 sec)

Another date expression is as follows:

SQL> SELECT GETDATE();;

+-------------------------+

| GETDATE |

+-------------------------+

| 2009-10-22 12:07:18.140 |

+-------------------------+

1 row in set (0.00 sec)

# **SQL - CREATE Database**

Advertisements

[Previous Page](https://www.tutorialspoint.com/sql/sql-expressions.htm)

[Next Page](https://www.tutorialspoint.com/sql/sql-drop-database.htm)

The SQL **CREATE DATABASE** statement is used to create new SQL database.

## Syntax:

Basic syntax of CREATE DATABASE statement is as follows:

CREATE DATABASE DatabaseName;

Always database name should be unique within the RDBMS.

## Example:

If you want to create new database <testDB>, then CREATE DATABASE statement would be as follows:

SQL> CREATE DATABASE testDB;

Make sure you have admin privilege before creating any database. Once a database is created, you can check it in the list of databases as follows:

SQL> SHOW DATABASES;

+--------------------+

| Database |

+--------------------+

| information\_schema |

| AMROOD |

| TUTORIALSPOINT |

| mysql |

| orig |

| test |

| testDB |

+--------------------+

7 rows in set (0.00 sec)

# **SQL - DROP or DELETE Database**

Advertisements

[Previous Page](https://www.tutorialspoint.com/sql/sql-create-database.htm)

[Next Page](https://www.tutorialspoint.com/sql/sql-select-database.htm)

The SQL **DROP DATABASE** statement is used to drop an existing database in SQL schema.

## Syntax:

Basic syntax of DROP DATABASE statement is as follows:

DROP DATABASE DatabaseName;

Always database name should be unique within the RDBMS.

## Example:

If you want to delete an existing database <testDB>, then DROP DATABASE statement would be as follows:

SQL> DROP DATABASE testDB;

**NOTE:** Be careful before using this operation because by deleting an existing database would result in loss of complete information stored in the database.

Make sure you have admin privilege before dropping any database. Once a database is dropped, you can check it in the list of databases as follows:

SQL> SHOW DATABASES;

+--------------------+

| Database |

+--------------------+

| information\_schema |

| AMROOD |

| TUTORIALSPOINT |

| mysql |

| orig |

| test |

+--------------------+

6 rows in set (0.00 sec)

# **SQL - SELECT Database, USE Statement**

Advertisements

[Previous Page](https://www.tutorialspoint.com/sql/sql-drop-database.htm)

[Next Page](https://www.tutorialspoint.com/sql/sql-create-table.htm)

When you have multiple databases in your SQL Schema, then before starting your operation, you would need to select a database where all the operations would be performed.

The SQL **USE** statement is used to select any existing database in SQL schema.

## Syntax:

Basic syntax of USE statement is as follows:

USE DatabaseName;

Always database name should be unique within the RDBMS.

## Example:

You can check available databases as follows:

SQL> SHOW DATABASES;

+--------------------+

| Database |

+--------------------+

| information\_schema |

| AMROOD |

| TUTORIALSPOINT |

| mysql |

| orig |

| test |

+--------------------+

6 rows in set (0.00 sec)

Now, if you want to work with AMROOD database, then you can execute the following SQL command and start working with AMROOD database:

SQL> USE AMROOD;

Creating a basic table involves naming the table and defining its columns and each column's data type.

The SQL **CREATE TABLE** statement is used to create a new table.

## Syntax:

Basic syntax of CREATE TABLE statement is as follows:

CREATE TABLE table\_name(

column1 datatype,

column2 datatype,

column3 datatype,

.....

columnN datatype,

PRIMARY KEY( one or more columns )

);

CREATE TABLE is the keyword telling the database system what you want to do. In this case, you want to create a new table. The unique name or identifier for the table follows the CREATE TABLE statement.

Then in brackets comes the list defining each column in the table and what sort of data type it is. The syntax becomes clearer with an example below.

A copy of an existing table can be created using a combination of the CREATE TABLE statement and the SELECT statement. You can check complete details at [Create Table Using another Table.](https://www.tutorialspoint.com/sql/sql-create-table-using-tables.htm)

## Example:

Following is an example, which creates a CUSTOMERS table with ID as primary key and NOT NULL are the constraints showing that these fields can not be NULL while creating records in this table:

SQL> CREATE TABLE CUSTOMERS(

ID INT NOT NULL,

NAME VARCHAR (20) NOT NULL,

AGE INT NOT NULL,

ADDRESS CHAR (25) ,

SALARY DECIMAL (18, 2),

PRIMARY KEY (ID)

);

You can verify if your table has been created successfully by looking at the message displayed by the SQL server, otherwise you can use **DESC** command as follows:

SQL> DESC CUSTOMERS;

+---------+---------------+------+-----+---------+-------+

| Field | Type | Null | Key | Default | Extra |

+---------+---------------+------+-----+---------+-------+

| ID | int(11) | NO | PRI | | |

| NAME | varchar(20) | NO | | | |

| AGE | int(11) | NO | | | |

| ADDRESS | char(25) | YES | | NULL | |

| SALARY | decimal(18,2) | YES | | NULL | |

+---------+---------------+------+-----+---------+-------+

5 rows in set (0.00 sec)

Now, you have CUSTOMERS table available in your database which you can use to store required information related to customers.

The SQL **DROP TABLE** statement is used to remove a table definition and all data, indexes, triggers, constraints, and permission specifications for that table.

**NOTE:** You have to be careful while using this command because once a table is deleted then all the information available in the table would also be lost forever.

## Syntax:

Basic syntax of DROP TABLE statement is as follows:

DROP TABLE table\_name;

## Example:

Let us first verify CUSTOMERS table and then we would delete it from the database:

SQL> DESC CUSTOMERS;

+---------+---------------+------+-----+---------+-------+

| Field | Type | Null | Key | Default | Extra |

+---------+---------------+------+-----+---------+-------+

| ID | int(11) | NO | PRI | | |

| NAME | varchar(20) | NO | | | |

| AGE | int(11) | NO | | | |

| ADDRESS | char(25) | YES | | NULL | |

| SALARY | decimal(18,2) | YES | | NULL | |

+---------+---------------+------+-----+---------+-------+

5 rows in set (0.00 sec)

This means CUSTOMERS table is available in the database, so let us drop it as follows:

SQL> DROP TABLE CUSTOMERS;

Query OK, 0 rows affected (0.01 sec)

Now, if you would try DESC command, then you would get error as follows:

SQL> DESC CUSTOMERS;

ERROR 1146 (42S02): Table 'TEST.CUSTOMERS' doesn't exist

Here, TEST is database name which we are using for our examples.

The SQL **INSERT INTO** Statement is used to add new rows of data to a table in the database.

## Syntax:

There are two basic syntaxes of INSERT INTO statement as follows:

INSERT INTO TABLE\_NAME (column1, column2, column3,...columnN)

VALUES (value1, value2, value3,...valueN);

Here, column1, column2,...columnN are the names of the columns in the table into which you want to insert data.

You may not need to specify the column(s) name in the SQL query if you are adding values for all the columns of the table. But make sure the order of the values is in the same order as the columns in the table. The SQL INSERT INTO syntax would be as follows:

INSERT INTO TABLE\_NAME VALUES (value1,value2,value3,...valueN);

## Example:

Following statements would create six records in CUSTOMERS table:

INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY)

VALUES (1, 'Ramesh', 32, 'Ahmedabad', 2000.00 );

INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY)

VALUES (2, 'Khilan', 25, 'Delhi', 1500.00 );

INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY)

VALUES (3, 'kaushik', 23, 'Kota', 2000.00 );

INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY)

VALUES (4, 'Chaitali', 25, 'Mumbai', 6500.00 );

INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY)

VALUES (5, 'Hardik', 27, 'Bhopal', 8500.00 );

INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY)

VALUES (6, 'Komal', 22, 'MP', 4500.00 );

You can create a record in CUSTOMERS table using second syntax as follows:

INSERT INTO CUSTOMERS

VALUES (7, 'Muffy', 24, 'Indore', 10000.00 );

All the above statements would produce the following records in CUSTOMERS table:

+----+----------+-----+-----------+----------+

| ID | NAME | AGE | ADDRESS | SALARY |

+----+----------+-----+-----------+----------+

| 1 | Ramesh | 32 | Ahmedabad | 2000.00 |

| 2 | Khilan | 25 | Delhi | 1500.00 |

| 3 | kaushik | 23 | Kota | 2000.00 |

| 4 | Chaitali | 25 | Mumbai | 6500.00 |

| 5 | Hardik | 27 | Bhopal | 8500.00 |

| 6 | Komal | 22 | MP | 4500.00 |

| 7 | Muffy | 24 | Indore | 10000.00 |

+----+----------+-----+-----------+----------+

## Populate one table using another table:

You can populate data into a table through select statement over another table provided another table has a set of fields, which are required to populate first table. Here is the syntax:

INSERT INTO first\_table\_name [(column1, column2, ... columnN)]

SELECT column1, column2, ...columnN

FROM second\_table\_name

[WHERE condition];

SQL **SELECT** statement is used to fetch the data from a database table which returns data in the form of result table. These result tables are called result-sets.

## Syntax:

The basic syntax of SELECT statement is as follows:

SELECT column1, column2, columnN FROM table\_name;

Here, column1, column2...are the fields of a table whose values you want to fetch. If you want to fetch all the fields available in the field, then you can use the following syntax:

SELECT \* FROM table\_name;

## Example:

Consider the CUSTOMERS table having the following records:

+----+----------+-----+-----------+----------+

| ID | NAME | AGE | ADDRESS | SALARY |

+----+----------+-----+-----------+----------+

| 1 | Ramesh | 32 | Ahmedabad | 2000.00 |

| 2 | Khilan | 25 | Delhi | 1500.00 |

| 3 | kaushik | 23 | Kota | 2000.00 |

| 4 | Chaitali | 25 | Mumbai | 6500.00 |

| 5 | Hardik | 27 | Bhopal | 8500.00 |

| 6 | Komal | 22 | MP | 4500.00 |

| 7 | Muffy | 24 | Indore | 10000.00 |

+----+----------+-----+-----------+----------+

Following is an example, which would fetch ID, Name and Salary fields of the customers available in CUSTOMERS table:

SQL> SELECT ID, NAME, SALARY FROM CUSTOMERS;

This would produce the following result:

+----+----------+----------+

| ID | NAME | SALARY |

+----+----------+----------+

| 1 | Ramesh | 2000.00 |

| 2 | Khilan | 1500.00 |

| 3 | kaushik | 2000.00 |

| 4 | Chaitali | 6500.00 |

| 5 | Hardik | 8500.00 |

| 6 | Komal | 4500.00 |

| 7 | Muffy | 10000.00 |

+----+----------+----------+

If you want to fetch all the fields of CUSTOMERS table, then use the following query:

SQL> SELECT \* FROM CUSTOMERS;

This would produce the following result:

+----+----------+-----+-----------+----------+

| ID | NAME | AGE | ADDRESS | SALARY |

+----+----------+-----+-----------+----------+

| 1 | Ramesh | 32 | Ahmedabad | 2000.00 |

| 2 | Khilan | 25 | Delhi | 1500.00 |

| 3 | kaushik | 23 | Kota | 2000.00 |

| 4 | Chaitali | 25 | Mumbai | 6500.00 |

| 5 | Hardik | 27 | Bhopal | 8500.00 |

| 6 | Komal | 22 | MP | 4500.00 |

| 7 | Muffy | 24 | Indore | 10000.00 |

+----+----------+-----+-----------+----------+

The SQL **WHERE** clause is used to specify a condition while fetching the data from single table or joining with multiple tables.

If the given condition is satisfied then only it returns specific value from the table. You would use WHERE clause to filter the records and fetching only necessary records.

The WHERE clause is not only used in SELECT statement, but it is also used in UPDATE, DELETE statement, etc., which we would examine in subsequent chapters.

## Syntax:

The basic syntax of SELECT statement with WHERE clause is as follows:

SELECT column1, column2, columnN

FROM table\_name

WHERE [condition]

You can specify a condition using [comparison or logical operators](https://www.tutorialspoint.com/sql/sql-operators.htm) like >, <, =, LIKE, NOT, etc. Below examples would make this concept clear.

## Example:

Consider the CUSTOMERS table having the following records:

+----+----------+-----+-----------+----------+

| ID | NAME | AGE | ADDRESS | SALARY |

+----+----------+-----+-----------+----------+

| 1 | Ramesh | 32 | Ahmedabad | 2000.00 |

| 2 | Khilan | 25 | Delhi | 1500.00 |

| 3 | kaushik | 23 | Kota | 2000.00 |

| 4 | Chaitali | 25 | Mumbai | 6500.00 |

| 5 | Hardik | 27 | Bhopal | 8500.00 |

| 6 | Komal | 22 | MP | 4500.00 |

| 7 | Muffy | 24 | Indore | 10000.00 |

+----+----------+-----+-----------+----------+

Following is an example which would fetch ID, Name and Salary fields from the CUSTOMERS table where salary is greater than 2000:

SQL> SELECT ID, NAME, SALARY

FROM CUSTOMERS

WHERE SALARY > 2000;

This would produce the following result:

+----+----------+----------+

| ID | NAME | SALARY |

+----+----------+----------+

| 4 | Chaitali | 6500.00 |

| 5 | Hardik | 8500.00 |

| 6 | Komal | 4500.00 |

| 7 | Muffy | 10000.00 |

+----+----------+----------+

Following is an example, which would fetch ID, Name and Salary fields from the CUSTOMERS table for a customer with name **Hardik**. Here, it is important to note that all the strings should be given inside single quotes ('') where as numeric values should be given without any quote as in above example:

SQL> SELECT ID, NAME, SALARY

FROM CUSTOMERS

WHERE NAME = 'Hardik';

This would produce the following result:

+----+----------+----------+

| ID | NAME | SALARY |

+----+----------+----------+

| 5 | Hardik | 8500.00 |

+----+----------+----------+

The SQL **AND** and **OR** operators are used to combine multiple conditions to narrow data in an SQL statement. These two operators are called conjunctive operators.

These operators provide a means to make multiple comparisons with different operators in the same SQL statement.

## The AND Operator:

The **AND** operator allows the existence of multiple conditions in an SQL statement's WHERE clause.

## Syntax:

The basic syntax of AND operator with WHERE clause is as follows:

SELECT column1, column2, columnN

FROM table\_name

WHERE [condition1] AND [condition2]...AND [conditionN];

You can combine N number of conditions using AND operator. For an action to be taken by the SQL statement, whether it be a transaction or query, all conditions separated by the AND must be TRUE.

## Example:

Consider the CUSTOMERS table having the following records:

+----+----------+-----+-----------+----------+

| ID | NAME | AGE | ADDRESS | SALARY |

+----+----------+-----+-----------+----------+

| 1 | Ramesh | 32 | Ahmedabad | 2000.00 |

| 2 | Khilan | 25 | Delhi | 1500.00 |

| 3 | kaushik | 23 | Kota | 2000.00 |

| 4 | Chaitali | 25 | Mumbai | 6500.00 |

| 5 | Hardik | 27 | Bhopal | 8500.00 |

| 6 | Komal | 22 | MP | 4500.00 |

| 7 | Muffy | 24 | Indore | 10000.00 |

+----+----------+-----+-----------+----------+

Following is an example, which would fetch ID, Name and Salary fields from the CUSTOMERS table where salary is greater than 2000 AND age is less tan 25 years:

SQL> SELECT ID, NAME, SALARY

FROM CUSTOMERS

WHERE SALARY > 2000 AND age < 25;

This would produce the following result:

+----+-------+----------+

| ID | NAME | SALARY |

+----+-------+----------+

| 6 | Komal | 4500.00 |

| 7 | Muffy | 10000.00 |

+----+-------+----------+

## The OR Operator:

The OR operator is used to combine multiple conditions in an SQL statement's WHERE clause.

## Syntax:

The basic syntax of OR operator with WHERE clause is as follows:

SELECT column1, column2, columnN

FROM table\_name

WHERE [condition1] OR [condition2]...OR [conditionN]

You can combine N number of conditions using OR operator. For an action to be taken by the SQL statement, whether it be a transaction or query, only any ONE of the conditions separated by the OR must be TRUE.

## Example:

Consider the CUSTOMERS table having the following records:

+----+----------+-----+-----------+----------+

| ID | NAME | AGE | ADDRESS | SALARY |

+----+----------+-----+-----------+----------+

| 1 | Ramesh | 32 | Ahmedabad | 2000.00 |

| 2 | Khilan | 25 | Delhi | 1500.00 |

| 3 | kaushik | 23 | Kota | 2000.00 |

| 4 | Chaitali | 25 | Mumbai | 6500.00 |

| 5 | Hardik | 27 | Bhopal | 8500.00 |

| 6 | Komal | 22 | MP | 4500.00 |

| 7 | Muffy | 24 | Indore | 10000.00 |

+----+----------+-----+-----------+----------+

Following is an example, which would fetch ID, Name and Salary fields from the CUSTOMERS table where salary is greater than 2000 OR age is less tan 25 years:

SQL> SELECT ID, NAME, SALARY

FROM CUSTOMERS

WHERE SALARY > 2000 OR age < 25;

This would produce the following result:

+----+----------+----------+

| ID | NAME | SALARY |

+----+----------+----------+

| 3 | kaushik | 2000.00 |

| 4 | Chaitali | 6500.00 |

| 5 | Hardik | 8500.00 |

| 6 | Komal | 4500.00 |

| 7 | Muffy | 10000.00 |

+----+----------+----------+

The SQL **UPDATE** Query is used to modify the existing records in a table.

You can use WHERE clause with UPDATE query to update selected rows otherwise all the rows would be affected.

## Syntax:

The basic syntax of UPDATE query with WHERE clause is as follows:

UPDATE table\_name

SET column1 = value1, column2 = value2...., columnN = valueN

WHERE [condition];

You can combine N number of conditions using AND or OR operators.

## Example:

Consider the CUSTOMERS table having the following records:

+----+----------+-----+-----------+----------+

| ID | NAME | AGE | ADDRESS | SALARY |

+----+----------+-----+-----------+----------+

| 1 | Ramesh | 32 | Ahmedabad | 2000.00 |

| 2 | Khilan | 25 | Delhi | 1500.00 |

| 3 | kaushik | 23 | Kota | 2000.00 |

| 4 | Chaitali | 25 | Mumbai | 6500.00 |

| 5 | Hardik | 27 | Bhopal | 8500.00 |

| 6 | Komal | 22 | MP | 4500.00 |

| 7 | Muffy | 24 | Indore | 10000.00 |

+----+----------+-----+-----------+----------+

Following is an example, which would update ADDRESS for a customer whose ID is 6:

SQL> UPDATE CUSTOMERS

SET ADDRESS = 'Pune'

WHERE ID = 6;

Now, CUSTOMERS table would have the following records:

+----+----------+-----+-----------+----------+

| ID | NAME | AGE | ADDRESS | SALARY |

+----+----------+-----+-----------+----------+

| 1 | Ramesh | 32 | Ahmedabad | 2000.00 |

| 2 | Khilan | 25 | Delhi | 1500.00 |

| 3 | kaushik | 23 | Kota | 2000.00 |

| 4 | Chaitali | 25 | Mumbai | 6500.00 |

| 5 | Hardik | 27 | Bhopal | 8500.00 |

| 6 | Komal | 22 | Pune | 4500.00 |

| 7 | Muffy | 24 | Indore | 10000.00 |

+----+----------+-----+-----------+----------+

If you want to modify all ADDRESS and SALARY column values in CUSTOMERS table, you do not need to use WHERE clause and UPDATE query would be as follows:

SQL> UPDATE CUSTOMERS

SET ADDRESS = 'Pune', SALARY = 1000.00;

Now, CUSTOMERS table would have the following records:

+----+----------+-----+---------+---------+

| ID | NAME | AGE | ADDRESS | SALARY |

+----+----------+-----+---------+---------+

| 1 | Ramesh | 32 | Pune | 1000.00 |

| 2 | Khilan | 25 | Pune | 1000.00 |

| 3 | kaushik | 23 | Pune | 1000.00 |

| 4 | Chaitali | 25 | Pune | 1000.00 |

| 5 | Hardik | 27 | Pune | 1000.00 |

| 6 | Komal | 22 | Pune | 1000.00 |

| 7 | Muffy | 24 | Pune | 1000.00 |

+----+----------+-----+---------+---------+

he SQL **DELETE** Query is used to delete the existing records from a table.

You can use WHERE clause with DELETE query to delete selected rows, otherwise all the records would be deleted.

## Syntax:

The basic syntax of DELETE query with WHERE clause is as follows:

DELETE FROM table\_name

WHERE [condition];

You can combine N number of conditions using AND or OR operators.

## Example:

Consider the CUSTOMERS table having the following records:

+----+----------+-----+-----------+----------+

| ID | NAME | AGE | ADDRESS | SALARY |

+----+----------+-----+-----------+----------+

| 1 | Ramesh | 32 | Ahmedabad | 2000.00 |

| 2 | Khilan | 25 | Delhi | 1500.00 |

| 3 | kaushik | 23 | Kota | 2000.00 |

| 4 | Chaitali | 25 | Mumbai | 6500.00 |

| 5 | Hardik | 27 | Bhopal | 8500.00 |

| 6 | Komal | 22 | MP | 4500.00 |

| 7 | Muffy | 24 | Indore | 10000.00 |

+----+----------+-----+-----------+----------+

Following is an example, which would DELETE a customer, whose ID is 6:

SQL> DELETE FROM CUSTOMERS

WHERE ID = 6;

Now, CUSTOMERS table would have the following records:

+----+----------+-----+-----------+----------+

| ID | NAME | AGE | ADDRESS | SALARY |

+----+----------+-----+-----------+----------+

| 1 | Ramesh | 32 | Ahmedabad | 2000.00 |

| 2 | Khilan | 25 | Delhi | 1500.00 |

| 3 | kaushik | 23 | Kota | 2000.00 |

| 4 | Chaitali | 25 | Mumbai | 6500.00 |

| 5 | Hardik | 27 | Bhopal | 8500.00 |

| 7 | Muffy | 24 | Indore | 10000.00 |

+----+----------+-----+-----------+----------+

If you want to DELETE all the records from CUSTOMERS table, you do not need to use WHERE clause and DELETE query would be as follows:

SQL> DELETE FROM CUSTOMERS;

Now, CUSTOMERS table would not have any record.

The SQL **LIKE** clause is used to compare a value to similar values using wildcard operators. There are two wildcards used in conjunction with the LIKE operator:

* The percent sign (%)
* The underscore (\_)

The percent sign represents zero, one, or multiple characters. The underscore represents a single number or character. The symbols can be used in combinations.

Syntax:

The basic syntax of % and \_ is as follows:

SELECT FROM table\_name

WHERE column LIKE 'XXXX%'

or

SELECT FROM table\_name

WHERE column LIKE '%XXXX%'

or

SELECT FROM table\_name

WHERE column LIKE 'XXXX\_'

or

SELECT FROM table\_name

WHERE column LIKE '\_XXXX'

or

SELECT FROM table\_name

WHERE column LIKE '\_XXXX\_'

You can combine N number of conditions using AND or OR operators. Here, XXXX could be any numeric or string value.

Example:

Here are number of examples showing WHERE part having different LIKE clause with '%' and '\_' operators:

|  |  |
| --- | --- |
| **Statement** | **Description** |
| WHERE SALARY LIKE '200%' | Finds any values that start with 200 |
| WHERE SALARY LIKE '%200%' | Finds any values that have 200 in any position |
| WHERE SALARY LIKE '\_00%' | Finds any values that have 00 in the second and third positions |
| WHERE SALARY LIKE '2\_%\_%' | Finds any values that start with 2 and are at least 3 characters in length |
| WHERE SALARY LIKE '%2' | Finds any values that end with 2 |
| WHERE SALARY LIKE '\_2%3' | Finds any values that have a 2 in the second position and end with a 3 |
| WHERE SALARY LIKE '2\_\_\_3' | Finds any values in a five-digit number that start with 2 and end with 3 |

Let us take a real example, consider the CUSTOMERS table having the following records:

+----+----------+-----+-----------+----------+

| ID | NAME | AGE | ADDRESS | SALARY |

+----+----------+-----+-----------+----------+

| 1 | Ramesh | 32 | Ahmedabad | 2000.00 |

| 2 | Khilan | 25 | Delhi | 1500.00 |

| 3 | kaushik | 23 | Kota | 2000.00 |

| 4 | Chaitali | 25 | Mumbai | 6500.00 |

| 5 | Hardik | 27 | Bhopal | 8500.00 |

| 6 | Komal | 22 | MP | 4500.00 |

| 7 | Muffy | 24 | Indore | 10000.00 |

+----+----------+-----+-----------+----------+

Following is an example, which would display all the records from CUSTOMERS table where SALARY starts with 200:

SQL> SELECT \* FROM CUSTOMERS

WHERE SALARY LIKE '200%';

This would produce the following result:

+----+----------+-----+-----------+----------+

| ID | NAME | AGE | ADDRESS | SALARY |

+----+----------+-----+-----------+----------+

| 1 | Ramesh | 32 | Ahmedabad | 2000.00 |

| 3 | kaushik | 23 | Kota | 2000.00 |

+----+----------+-----+-----------+----------+

The SQL **TOP** clause is used to fetch a TOP N number or X percent records from a table.

**Note:** All the databases do not support TOP clause. For example MySQL supports **LIMIT** clause to fetch limited number of records and Oracle uses **ROWNUM** to fetch limited number of records.

## Syntax:

The basic syntax of TOP clause with SELECT statement would be as follows:

SELECT TOP number|percent column\_name(s)

FROM table\_name

WHERE [condition]

## Example:

Consider the CUSTOMERS table having the following records:

+----+----------+-----+-----------+----------+

| ID | NAME | AGE | ADDRESS | SALARY |

+----+----------+-----+-----------+----------+

| 1 | Ramesh | 32 | Ahmedabad | 2000.00 |

| 2 | Khilan | 25 | Delhi | 1500.00 |

| 3 | kaushik | 23 | Kota | 2000.00 |

| 4 | Chaitali | 25 | Mumbai | 6500.00 |

| 5 | Hardik | 27 | Bhopal | 8500.00 |

| 6 | Komal | 22 | MP | 4500.00 |

| 7 | Muffy | 24 | Indore | 10000.00 |

+----+----------+-----+-----------+----------+

Following is an example on SQL server, which would fetch top 3 records from CUSTOMERS table:

SQL> SELECT TOP 3 \* FROM CUSTOMERS;

This would produce the following result:

+----+---------+-----+-----------+---------+

| ID | NAME | AGE | ADDRESS | SALARY |

+----+---------+-----+-----------+---------+

| 1 | Ramesh | 32 | Ahmedabad | 2000.00 |

| 2 | Khilan | 25 | Delhi | 1500.00 |

| 3 | kaushik | 23 | Kota | 2000.00 |

+----+---------+-----+-----------+---------+

If you are using MySQL server, then here is an equivalent example:

SQL> SELECT \* FROM CUSTOMERS

LIMIT 3;

This would produce the following result:

+----+---------+-----+-----------+---------+

| ID | NAME | AGE | ADDRESS | SALARY |

+----+---------+-----+-----------+---------+

| 1 | Ramesh | 32 | Ahmedabad | 2000.00 |

| 2 | Khilan | 25 | Delhi | 1500.00 |

| 3 | kaushik | 23 | Kota | 2000.00 |

+----+---------+-----+-----------+---------+

If you are using Oracle server, then here is an equivalent example:

SQL> SELECT \* FROM CUSTOMERS

WHERE ROWNUM <= 3;

This would produce the following result:

+----+---------+-----+-----------+---------+

| ID | NAME | AGE | ADDRESS | SALARY |

+----+---------+-----+-----------+---------+

| 1 | Ramesh | 32 | Ahmedabad | 2000.00 |

| 2 | Khilan | 25 | Delhi | 1500.00 |

| 3 | kaushik | 23 | Kota | 2000.00 |

+----+---------+-----+-----------+---------+

The SQL **ORDER BY** clause is used to sort the data in ascending or descending order, based on one or more columns. Some database sorts query results in ascending order by default.

## Syntax:

The basic syntax of ORDER BY clause is as follows:

SELECT column-list

FROM table\_name

[WHERE condition]

[ORDER BY column1, column2, .. columnN] [ASC | DESC];

You can use more than one column in the ORDER BY clause. Make sure whatever column you are using to sort, that column should be in column-list.

## Example:

Consider the CUSTOMERS table having the following records:

+----+----------+-----+-----------+----------+

| ID | NAME | AGE | ADDRESS | SALARY |

+----+----------+-----+-----------+----------+

| 1 | Ramesh | 32 | Ahmedabad | 2000.00 |

| 2 | Khilan | 25 | Delhi | 1500.00 |

| 3 | kaushik | 23 | Kota | 2000.00 |

| 4 | Chaitali | 25 | Mumbai | 6500.00 |

| 5 | Hardik | 27 | Bhopal | 8500.00 |

| 6 | Komal | 22 | MP | 4500.00 |

| 7 | Muffy | 24 | Indore | 10000.00 |

+----+----------+-----+-----------+----------+

Following is an example, which would sort the result in ascending order by NAME and SALARY:

SQL> SELECT \* FROM CUSTOMERS

ORDER BY NAME, SALARY;

This would produce the following result:

+----+----------+-----+-----------+----------+

| ID | NAME | AGE | ADDRESS | SALARY |

+----+----------+-----+-----------+----------+

| 4 | Chaitali | 25 | Mumbai | 6500.00 |

| 5 | Hardik | 27 | Bhopal | 8500.00 |

| 3 | kaushik | 23 | Kota | 2000.00 |

| 2 | Khilan | 25 | Delhi | 1500.00 |

| 6 | Komal | 22 | MP | 4500.00 |

| 7 | Muffy | 24 | Indore | 10000.00 |

| 1 | Ramesh | 32 | Ahmedabad | 2000.00 |

+----+----------+-----+-----------+----------+

Following is an example, which would sort the result in descending order by NAME:

SQL> SELECT \* FROM CUSTOMERS

ORDER BY NAME DESC;

This would produce the following result:

+----+----------+-----+-----------+----------+

| ID | NAME | AGE | ADDRESS | SALARY |

+----+----------+-----+-----------+----------+

| 1 | Ramesh | 32 | Ahmedabad | 2000.00 |

| 7 | Muffy | 24 | Indore | 10000.00 |

| 6 | Komal | 22 | MP | 4500.00 |

| 2 | Khilan | 25 | Delhi | 1500.00 |

| 3 | kaushik | 23 | Kota | 2000.00 |

| 5 | Hardik | 27 | Bhopal | 8500.00 |

| 4 | Chaitali | 25 | Mumbai | 6500.00 |

+----+----------+-----+-----------+----------+

The SQL **GROUP BY**clause is used in collaboration with the SELECT statement to arrange identical data into groups.

The GROUP BY clause follows the WHERE clause in a SELECT statement and precedes the ORDER BY clause.

## Syntax:

The basic syntax of GROUP BY clause is given below. The GROUP BY clause must follow the conditions in the WHERE clause and must precede the ORDER BY clause if one is used.

SELECT column1, column2

FROM table\_name

WHERE [ conditions ]

GROUP BY column1, column2

ORDER BY column1, column2

## Example:

Consider the CUSTOMERS table is having the following records:

+----+----------+-----+-----------+----------+

| ID | NAME | AGE | ADDRESS | SALARY |

+----+----------+-----+-----------+----------+

| 1 | Ramesh | 32 | Ahmedabad | 2000.00 |

| 2 | Khilan | 25 | Delhi | 1500.00 |

| 3 | kaushik | 23 | Kota | 2000.00 |

| 4 | Chaitali | 25 | Mumbai | 6500.00 |

| 5 | Hardik | 27 | Bhopal | 8500.00 |

| 6 | Komal | 22 | MP | 4500.00 |

| 7 | Muffy | 24 | Indore | 10000.00 |

+----+----------+-----+-----------+----------+

If you want to know the total amount of salary on each customer, then GROUP BY query would be as follows:

SQL> SELECT NAME, SUM(SALARY) FROM CUSTOMERS

GROUP BY NAME;

This would produce the following result:

+----------+-------------+

| NAME | SUM(SALARY) |

+----------+-------------+

| Chaitali | 6500.00 |

| Hardik | 8500.00 |

| kaushik | 2000.00 |

| Khilan | 1500.00 |

| Komal | 4500.00 |

| Muffy | 10000.00 |

| Ramesh | 2000.00 |

+----------+-------------+

Now, let us have following table where CUSTOMERS table has the following records with duplicate names:

+----+----------+-----+-----------+----------+

| ID | NAME | AGE | ADDRESS | SALARY |

+----+----------+-----+-----------+----------+

| 1 | Ramesh | 32 | Ahmedabad | 2000.00 |

| 2 | Ramesh | 25 | Delhi | 1500.00 |

| 3 | kaushik | 23 | Kota | 2000.00 |

| 4 | kaushik | 25 | Mumbai | 6500.00 |

| 5 | Hardik | 27 | Bhopal | 8500.00 |

| 6 | Komal | 22 | MP | 4500.00 |

| 7 | Muffy | 24 | Indore | 10000.00 |

+----+----------+-----+-----------+----------+

Now again, if you want to know the total amount of salary on each customer, then GROUP BY query would be as follows:

SQL> SELECT NAME, SUM(SALARY) FROM CUSTOMERS

GROUP BY NAME;

This would produce the following result:

+---------+-------------+

| NAME | SUM(SALARY) |

+---------+-------------+

| Hardik | 8500.00 |

| kaushik | 8500.00 |

| Komal | 4500.00 |

| Muffy | 10000.00 |

| Ramesh | 3500.00 |

+---------+-------------+

The SQL **DISTINCT** keyword is used in conjunction with SELECT statement to eliminate all the duplicate records and fetching only unique records.

There may be a situation when you have multiple duplicate records in a table. While fetching such records, it makes more sense to fetch only unique records instead of fetching duplicate records.

## Syntax:

The basic syntax of DISTINCT keyword to eliminate duplicate records is as follows:

SELECT DISTINCT column1, column2,.....columnN

FROM table\_name

WHERE [condition]

## Example:

Consider the CUSTOMERS table having the following records:

+----+----------+-----+-----------+----------+

| ID | NAME | AGE | ADDRESS | SALARY |

+----+----------+-----+-----------+----------+

| 1 | Ramesh | 32 | Ahmedabad | 2000.00 |

| 2 | Khilan | 25 | Delhi | 1500.00 |

| 3 | kaushik | 23 | Kota | 2000.00 |

| 4 | Chaitali | 25 | Mumbai | 6500.00 |

| 5 | Hardik | 27 | Bhopal | 8500.00 |

| 6 | Komal | 22 | MP | 4500.00 |

| 7 | Muffy | 24 | Indore | 10000.00 |

+----+----------+-----+-----------+----------+

First, let us see how the following SELECT query returns duplicate salary records:

SQL> SELECT SALARY FROM CUSTOMERS

ORDER BY SALARY;

This would produce the following result where salary 2000 is coming twice which is a duplicate record from the original table.

+----------+

| SALARY |

+----------+

| 1500.00 |

| 2000.00 |

| 2000.00 |

| 4500.00 |

| 6500.00 |

| 8500.00 |

| 10000.00 |

+----------+

Now, let us use DISTINCT keyword with the above SELECT query and see the result:

SQL> SELECT DISTINCT SALARY FROM CUSTOMERS

ORDER BY SALARY;

This would produce the following result where we do not have any duplicate entry:

+----------+

| SALARY |

+----------+

| 1500.00 |

| 2000.00 |

| 4500.00 |

| 6500.00 |

| 8500.00 |

| 10000.00 |

+----------+

The SQL **ORDER BY** clause is used to sort the data in ascending or descending order, based on one or more columns. Some database sorts query results in ascending order by default.

## Syntax:

The basic syntax of ORDER BY clause which would be used to sort result in ascending or descending order is as follows:

SELECT column-list

FROM table\_name

[WHERE condition]

[ORDER BY column1, column2, .. columnN] [ASC | DESC];

You can use more than one column in the ORDER BY clause. Make sure whatever column you are using to sort, that column should be in column-list.

## Example:

Consider the CUSTOMERS table having the following records:

+----+----------+-----+-----------+----------+

| ID | NAME | AGE | ADDRESS | SALARY |

+----+----------+-----+-----------+----------+

| 1 | Ramesh | 32 | Ahmedabad | 2000.00 |

| 2 | Khilan | 25 | Delhi | 1500.00 |

| 3 | kaushik | 23 | Kota | 2000.00 |

| 4 | Chaitali | 25 | Mumbai | 6500.00 |

| 5 | Hardik | 27 | Bhopal | 8500.00 |

| 6 | Komal | 22 | MP | 4500.00 |

| 7 | Muffy | 24 | Indore | 10000.00 |

+----+----------+-----+-----------+----------+

Following is an example, which would sort the result in ascending order by NAME and SALARY:

SQL> SELECT \* FROM CUSTOMERS

ORDER BY NAME, SALARY;

This would produce the following result:

+----+----------+-----+-----------+----------+

| ID | NAME | AGE | ADDRESS | SALARY |

+----+----------+-----+-----------+----------+

| 4 | Chaitali | 25 | Mumbai | 6500.00 |

| 5 | Hardik | 27 | Bhopal | 8500.00 |

| 3 | kaushik | 23 | Kota | 2000.00 |

| 2 | Khilan | 25 | Delhi | 1500.00 |

| 6 | Komal | 22 | MP | 4500.00 |

| 7 | Muffy | 24 | Indore | 10000.00 |

| 1 | Ramesh | 32 | Ahmedabad | 2000.00 |

+----+----------+-----+-----------+----------+

Following is an example, which would sort the result in descending order by NAME:

SQL> SELECT \* FROM CUSTOMERS

ORDER BY NAME DESC;

This would produce the following result:

+----+----------+-----+-----------+----------+

| ID | NAME | AGE | ADDRESS | SALARY |

+----+----------+-----+-----------+----------+

| 1 | Ramesh | 32 | Ahmedabad | 2000.00 |

| 7 | Muffy | 24 | Indore | 10000.00 |

| 6 | Komal | 22 | MP | 4500.00 |

| 2 | Khilan | 25 | Delhi | 1500.00 |

| 3 | kaushik | 23 | Kota | 2000.00 |

| 5 | Hardik | 27 | Bhopal | 8500.00 |

| 4 | Chaitali | 25 | Mumbai | 6500.00 |

+----+----------+-----+-----------+----------+

To fetch the rows with own preferred order, the SELECT query would as follows:

SQL> SELECT \* FROM CUSTOMERS

ORDER BY (CASE ADDRESS

WHEN 'DELHI' THEN 1

WHEN 'BHOPAL' THEN 2

WHEN 'KOTA' THEN 3

WHEN 'AHMADABAD' THEN 4

WHEN 'MP' THEN 5

ELSE 100 END) ASC, ADDRESS DESC;

This would produce the following result:

+----+----------+-----+-----------+----------+

| ID | NAME | AGE | ADDRESS | SALARY |

+----+----------+-----+-----------+----------+

| 2 | Khilan | 25 | Delhi | 1500.00 |

| 5 | Hardik | 27 | Bhopal | 8500.00 |

| 3 | kaushik | 23 | Kota | 2000.00 |

| 6 | Komal | 22 | MP | 4500.00 |

| 4 | Chaitali | 25 | Mumbai | 6500.00 |

| 7 | Muffy | 24 | Indore | 10000.00 |

| 1 | Ramesh | 32 | Ahmedabad | 2000.00 |

+----+----------+-----+-----------+----------+

This will sort customers by ADDRESS in your ownoOrder of preference first and in a natural order for the remaining addresses. Also remaining Addresses will be sorted in the reverse alpha order.

Constraints are the rules enforced on data columns on table. These are used to limit the type of data that can go into a table. This ensures the accuracy and reliability of the data in the database.

Constraints could be column level or table level. Column level constraints are applied only to one column, whereas table level constraints are applied to the whole table.

Following are commonly used constraints available in SQL. These constraints have already been discussed in [SQL - RDBMS Concepts](https://www.tutorialspoint.com/sql/sql-rdbms-concepts.htm) chapter but its worth to revise them at this point.

* [NOT NULL Constraint](https://www.tutorialspoint.com/sql/sql-not-null.htm): Ensures that a column cannot have NULL value.
* [DEFAULT Constraint](https://www.tutorialspoint.com/sql/sql-default.htm): Provides a default value for a column when none is specified.
* [UNIQUE Constraint](https://www.tutorialspoint.com/sql/sql-unique.htm): Ensures that all values in a column are different.
* [PRIMARY Key](https://www.tutorialspoint.com/sql/sql-primary-key.htm): Uniquely identified each rows/records in a database table.
* [FOREIGN Key](https://www.tutorialspoint.com/sql/sql-foreign-key.htm): Uniquely identified a rows/records in any another database table.
* [CHECK Constraint](https://www.tutorialspoint.com/sql/sql-check.htm): The CHECK constraint ensures that all values in a column satisfy certain conditions.
* [INDEX](https://www.tutorialspoint.com/sql/sql-index.htm): Use to create and retrieve data from the database very quickly.

Constraints can be specified when a table is created with the CREATE TABLE statement or you can use ALTER TABLE statement to create constraints even after the table is created.

## Dropping Constraints:

Any constraint that you have defined can be dropped using the ALTER TABLE command with the DROP CONSTRAINT option.

For example, to drop the primary key constraint in the EMPLOYEES table, you can use the following command:

ALTER TABLE EMPLOYEES DROP CONSTRAINT EMPLOYEES\_PK;

Some implementations may provide shortcuts for dropping certain constraints. For example, to drop the primary key constraint for a table in Oracle, you can use the following command:

ALTER TABLE EMPLOYEES DROP PRIMARY KEY;

Some implementations allow you to disable constraints. Instead of permanently dropping a constraint from the database, you may want to temporarily disable the constraint and then enable it later.

## Integrity Constraints:

Integrity constraints are used to ensure accuracy and consistency of data in a relational database. Data integrity is handled in a relational database through the concept of referential integrity.

There are many types of integrity constraints that play a role in referential integrity (RI). These constraints include Primary Key, Foreign Key, Unique Constraints and other constraints mentioned above.

The SQL **Joins** clause is used to combine records from two or more tables in a database. A JOIN is a means for combining fields from two tables by using values common to each.

Consider the following two tables, (a) CUSTOMERS table is as follows:

+----+----------+-----+-----------+----------+

| ID | NAME | AGE | ADDRESS | SALARY |

+----+----------+-----+-----------+----------+

| 1 | Ramesh | 32 | Ahmedabad | 2000.00 |

| 2 | Khilan | 25 | Delhi | 1500.00 |

| 3 | kaushik | 23 | Kota | 2000.00 |

| 4 | Chaitali | 25 | Mumbai | 6500.00 |

| 5 | Hardik | 27 | Bhopal | 8500.00 |

| 6 | Komal | 22 | MP | 4500.00 |

| 7 | Muffy | 24 | Indore | 10000.00 |

+----+----------+-----+-----------+----------+

(b) Another table is ORDERS as follows:

+-----+---------------------+-------------+--------+

|OID | DATE | CUSTOMER\_ID | AMOUNT |

+-----+---------------------+-------------+--------+

| 102 | 2009-10-08 00:00:00 | 3 | 3000 |

| 100 | 2009-10-08 00:00:00 | 3 | 1500 |

| 101 | 2009-11-20 00:00:00 | 2 | 1560 |

| 103 | 2008-05-20 00:00:00 | 4 | 2060 |

+-----+---------------------+-------------+--------+

Now, let us join these two tables in our SELECT statement as follows:

SQL> SELECT ID, NAME, AGE, AMOUNT

FROM CUSTOMERS, ORDERS

WHERE CUSTOMERS.ID = ORDERS.CUSTOMER\_ID;

This would produce the following result:

+----+----------+-----+--------+

| ID | NAME | AGE | AMOUNT |

+----+----------+-----+--------+

| 3 | kaushik | 23 | 3000 |

| 3 | kaushik | 23 | 1500 |

| 2 | Khilan | 25 | 1560 |

| 4 | Chaitali | 25 | 2060 |

+----+----------+-----+--------+

Here, it is noticeable that the join is performed in the WHERE clause. Several operators can be used to join tables, such as =, <, >, <>, <=, >=, !=, BETWEEN, LIKE, and NOT; they can all be used to join tables. However, the most common operator is the equal symbol.

## SQL Join Types:

There are different types of joins available in SQL:

* [INNER JOIN:](https://www.tutorialspoint.com/sql/sql-inner-joins.htm) returns rows when there is a match in both tables.
* [LEFT JOIN:](https://www.tutorialspoint.com/sql/sql-left-joins.htm) returns all rows from the left table, even if there are no matches in the right table.
* [RIGHT JOIN:](https://www.tutorialspoint.com/sql/sql-right-joins.htm) returns all rows from the right table, even if there are no matches in the left table.
* [FULL JOIN:](https://www.tutorialspoint.com/sql/sql-full-joins.htm) returns rows when there is a match in one of the tables.
* [SELF JOIN:](https://www.tutorialspoint.com/sql/sql-self-joins.htm) is used to join a table to itself as if the table were two tables, temporarily renaming at least one table in the SQL statement.
* [CARTESIAN JOIN:](https://www.tutorialspoint.com/sql/sql-cartesian-joins.htm) returns the Cartesian product of the sets of records from the two or more joined tables.

The SQL **UNION** clause/operator is used to combine the results of two or more SELECT statements without returning any duplicate rows.

To use UNION, each SELECT must have the same number of columns selected, the same number of column expressions, the same data type, and have them in the same order, but they do not have to be the same length.

Syntax:

The basic syntax of **UNION** is as follows:

SELECT column1 [, column2 ]

FROM table1 [, table2 ]

[WHERE condition]

UNION

SELECT column1 [, column2 ]

FROM table1 [, table2 ]

[WHERE condition]

Here given condition could be any given expression based on your requirement.

Example:

Consider the following two tables, (a) CUSTOMERS table is as follows:

+----+----------+-----+-----------+----------+

| ID | NAME | AGE | ADDRESS | SALARY |

+----+----------+-----+-----------+----------+

| 1 | Ramesh | 32 | Ahmedabad | 2000.00 |

| 2 | Khilan | 25 | Delhi | 1500.00 |

| 3 | kaushik | 23 | Kota | 2000.00 |

| 4 | Chaitali | 25 | Mumbai | 6500.00 |

| 5 | Hardik | 27 | Bhopal | 8500.00 |

| 6 | Komal | 22 | MP | 4500.00 |

| 7 | Muffy | 24 | Indore | 10000.00 |

+----+----------+-----+-----------+----------+

(b) Another table is ORDERS as follows:

+-----+---------------------+-------------+--------+

|OID | DATE | CUSTOMER\_ID | AMOUNT |

+-----+---------------------+-------------+--------+

| 102 | 2009-10-08 00:00:00 | 3 | 3000 |

| 100 | 2009-10-08 00:00:00 | 3 | 1500 |

| 101 | 2009-11-20 00:00:00 | 2 | 1560 |

| 103 | 2008-05-20 00:00:00 | 4 | 2060 |

+-----+---------------------+-------------+--------+

Now, let us join these two tables in our SELECT statement as follows:

SQL> SELECT ID, NAME, AMOUNT, DATE

FROM CUSTOMERS

LEFT JOIN ORDERS

ON CUSTOMERS.ID = ORDERS.CUSTOMER\_ID

UNION

SELECT ID, NAME, AMOUNT, DATE

FROM CUSTOMERS

RIGHT JOIN ORDERS

ON CUSTOMERS.ID = ORDERS.CUSTOMER\_ID;

This would produce the following result:

+------+----------+--------+---------------------+

| ID | NAME | AMOUNT | DATE |

+------+----------+--------+---------------------+

| 1 | Ramesh | NULL | NULL |

| 2 | Khilan | 1560 | 2009-11-20 00:00:00 |

| 3 | kaushik | 3000 | 2009-10-08 00:00:00 |

| 3 | kaushik | 1500 | 2009-10-08 00:00:00 |

| 4 | Chaitali | 2060 | 2008-05-20 00:00:00 |

| 5 | Hardik | NULL | NULL |

| 6 | Komal | NULL | NULL |

| 7 | Muffy | NULL | NULL |

+------+----------+--------+---------------------+

The UNION ALL Clause:

The UNION ALL operator is used to combine the results of two SELECT statements including duplicate rows.

The same rules that apply to UNION apply to the UNION ALL operator.

Syntax:

The basic syntax of **UNION ALL** is as follows:

SELECT column1 [, column2 ]

FROM table1 [, table2 ]

[WHERE condition]

UNION ALL

SELECT column1 [, column2 ]

FROM table1 [, table2 ]

[WHERE condition]

Here given condition could be any given expression based on your requirement.

Example:

Consider the following two tables, (a) CUSTOMERS table is as follows:

+----+----------+-----+-----------+----------+

| ID | NAME | AGE | ADDRESS | SALARY |

+----+----------+-----+-----------+----------+

| 1 | Ramesh | 32 | Ahmedabad | 2000.00 |

| 2 | Khilan | 25 | Delhi | 1500.00 |

| 3 | kaushik | 23 | Kota | 2000.00 |

| 4 | Chaitali | 25 | Mumbai | 6500.00 |

| 5 | Hardik | 27 | Bhopal | 8500.00 |

| 6 | Komal | 22 | MP | 4500.00 |

| 7 | Muffy | 24 | Indore | 10000.00 |

+----+----------+-----+-----------+----------+

(b) Another table is ORDERS as follows:

+-----+---------------------+-------------+--------+

|OID | DATE | CUSTOMER\_ID | AMOUNT |

+-----+---------------------+-------------+--------+

| 102 | 2009-10-08 00:00:00 | 3 | 3000 |

| 100 | 2009-10-08 00:00:00 | 3 | 1500 |

| 101 | 2009-11-20 00:00:00 | 2 | 1560 |

| 103 | 2008-05-20 00:00:00 | 4 | 2060 |

+-----+---------------------+-------------+--------+

Now, let us join these two tables in our SELECT statement as follows:

SQL> SELECT ID, NAME, AMOUNT, DATE

FROM CUSTOMERS

LEFT JOIN ORDERS

ON CUSTOMERS.ID = ORDERS.CUSTOMER\_ID

UNION ALL

SELECT ID, NAME, AMOUNT, DATE

FROM CUSTOMERS

RIGHT JOIN ORDERS

ON CUSTOMERS.ID = ORDERS.CUSTOMER\_ID;

This would produce the following result:

+------+----------+--------+---------------------+

| ID | NAME | AMOUNT | DATE |

+------+----------+--------+---------------------+

| 1 | Ramesh | NULL | NULL |

| 2 | Khilan | 1560 | 2009-11-20 00:00:00 |

| 3 | kaushik | 3000 | 2009-10-08 00:00:00 |

| 3 | kaushik | 1500 | 2009-10-08 00:00:00 |

| 4 | Chaitali | 2060 | 2008-05-20 00:00:00 |

| 5 | Hardik | NULL | NULL |

| 6 | Komal | NULL | NULL |

| 7 | Muffy | NULL | NULL |

| 3 | kaushik | 3000 | 2009-10-08 00:00:00 |

| 3 | kaushik | 1500 | 2009-10-08 00:00:00 |

| 2 | Khilan | 1560 | 2009-11-20 00:00:00 |

| 4 | Chaitali | 2060 | 2008-05-20 00:00:00 |

+------+----------+--------+---------------------+

There are two other clauses (i.e., operators), which are very similar to UNION clause:

* SQL [INTERSECT Clause](https://www.tutorialspoint.com/sql/sql-intersect-clause.htm): is used to combine two SELECT statements, but returns rows only from the first SELECT statement that are identical to a row in the second SELECT statement.
* SQL [EXCEPT Clause](https://www.tutorialspoint.com/sql/sql-except-clause.htm) : combines two SELECT statements and returns rows from the first SELECT statement that are not returned by the second SELECT statement.
* The SQL **NULL** is the term used to represent a missing value. A NULL value in a table is a value in a field that appears to be blank.
* A field with a NULL value is a field with no value. It is very important to understand that a NULL value is different than a zero value or a field that contains spaces.

## Syntax:

* The basic syntax of **NULL** while creating a table:
* SQL> CREATE TABLE CUSTOMERS(
* ID INT NOT NULL,
* NAME VARCHAR (20) NOT NULL,
* AGE INT NOT NULL,
* ADDRESS CHAR (25) ,
* SALARY DECIMAL (18, 2),
* PRIMARY KEY (ID)
* );
* Here, **NOT NULL** signifies that column should always accept an explicit value of the given data type. There are two columns where we did not use NOT NULL, which means these columns could be NULL.
* A field with a NULL value is one that has been left blank during record creation.

## Example:

* The NULL value can cause problems when selecting data, however, because when comparing an unknown value to any other value, the result is always unknown and not included in the final results.
* You must use the **IS NULL** or **IS NOT NULL** operators in order to check for a NULL value.
* Consider the following table, CUSTOMERS having the following records:
* +----+----------+-----+-----------+----------+
* | ID | NAME | AGE | ADDRESS | SALARY |
* +----+----------+-----+-----------+----------+
* | 1 | Ramesh | 32 | Ahmedabad | 2000.00 |
* | 2 | Khilan | 25 | Delhi | 1500.00 |
* | 3 | kaushik | 23 | Kota | 2000.00 |
* | 4 | Chaitali | 25 | Mumbai | 6500.00 |
* | 5 | Hardik | 27 | Bhopal | 8500.00 |
* | 6 | Komal | 22 | MP | |
* | 7 | Muffy | 24 | Indore | |
* +----+----------+-----+-----------+----------+
* Now, following is the usage of **IS NOT NULL** operator:
* SQL> SELECT ID, NAME, AGE, ADDRESS, SALARY
* FROM CUSTOMERS
* WHERE SALARY IS NOT NULL;
* This would produce the following result:
* +----+----------+-----+-----------+----------+
* | ID | NAME | AGE | ADDRESS | SALARY |
* +----+----------+-----+-----------+----------+
* | 1 | Ramesh | 32 | Ahmedabad | 2000.00 |
* | 2 | Khilan | 25 | Delhi | 1500.00 |
* | 3 | kaushik | 23 | Kota | 2000.00 |
* | 4 | Chaitali | 25 | Mumbai | 6500.00 |
* | 5 | Hardik | 27 | Bhopal | 8500.00 |
* +----+----------+-----+-----------+----------+
* Now, following is the usage of **IS NULL** operator:
* SQL> SELECT ID, NAME, AGE, ADDRESS, SALARY
* FROM CUSTOMERS
* WHERE SALARY IS NULL;
* This would produce the following result:
* +----+----------+-----+-----------+----------+
* | ID | NAME | AGE | ADDRESS | SALARY |
* +----+----------+-----+-----------+----------+
* | 6 | Komal | 22 | MP | |
* | 7 | Muffy | 24 | Indore | |
* +----+----------+-----+-----------+----------+
* You can rename a table or a column temporarily by giving another name known as alias.
* The use of table aliases means to rename a table in a particular SQL statement. The renaming is a temporary change and the actual table name does not change in the database.
* The column aliases are used to rename a table's columns for the purpose of a particular SQL query.

## Syntax:

* The basic syntax of **table** alias is as follows:
* SELECT column1, column2....
* FROM table\_name AS alias\_name
* WHERE [condition];
* The basic syntax of **column** alias is as follows:
* SELECT column\_name AS alias\_name
* FROM table\_name
* WHERE [condition];

## Example:

* Consider the following two tables, (a) CUSTOMERS table is as follows:
* +----+----------+-----+-----------+----------+
* | ID | NAME | AGE | ADDRESS | SALARY |
* +----+----------+-----+-----------+----------+
* | 1 | Ramesh | 32 | Ahmedabad | 2000.00 |
* | 2 | Khilan | 25 | Delhi | 1500.00 |
* | 3 | kaushik | 23 | Kota | 2000.00 |
* | 4 | Chaitali | 25 | Mumbai | 6500.00 |
* | 5 | Hardik | 27 | Bhopal | 8500.00 |
* | 6 | Komal | 22 | MP | 4500.00 |
* | 7 | Muffy | 24 | Indore | 10000.00 |
* +----+----------+-----+-----------+----------+
* (b) Another table is ORDERS as follows:
* +-----+---------------------+-------------+--------+
* |OID | DATE | CUSTOMER\_ID | AMOUNT |
* +-----+---------------------+-------------+--------+
* | 102 | 2009-10-08 00:00:00 | 3 | 3000 |
* | 100 | 2009-10-08 00:00:00 | 3 | 1500 |
* | 101 | 2009-11-20 00:00:00 | 2 | 1560 |
* | 103 | 2008-05-20 00:00:00 | 4 | 2060 |
* +-----+---------------------+-------------+--------+
* Now, following is the usage of **table alias**:
* SQL> SELECT C.ID, C.NAME, C.AGE, O.AMOUNT
* FROM CUSTOMERS AS C, ORDERS AS O
* WHERE C.ID = O.CUSTOMER\_ID;
* This would produce the following result:
* +----+----------+-----+--------+
* | ID | NAME | AGE | AMOUNT |
* +----+----------+-----+--------+
* | 3 | kaushik | 23 | 3000 |
* | 3 | kaushik | 23 | 1500 |
* | 2 | Khilan | 25 | 1560 |
* | 4 | Chaitali | 25 | 2060 |
* +----+----------+-----+--------+
* Following is the usage of **column alias**:
* SQL> SELECT ID AS CUSTOMER\_ID, NAME AS CUSTOMER\_NAME
* FROM CUSTOMERS
* WHERE SALARY IS NOT NULL;
* This would produce the following result:
* +-------------+---------------+
* | CUSTOMER\_ID | CUSTOMER\_NAME |
* +-------------+---------------+
* | 1 | Ramesh |
* | 2 | Khilan |
* | 3 | kaushik |
* | 4 | Chaitali |
* | 5 | Hardik |
* | 6 | Komal |
* | 7 | Muffy |
* +-------------+---------------+

Indexes are special lookup tables that the database search engine can use to speed up data retrieval. Simply put, an index is a pointer to data in a table. An index in a database is very similar to an index in the back of a book.

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

An index helps speed up SELECT queries and WHERE clauses, but it slows down data input, with UPDATE and INSERT statements. Indexes can be created or dropped with no effect on the data.

Creating an index involves the CREATE INDEX statement, which allows you to name the index, to specify the table and which column or columns to index, and to indicate whether the index is in ascending or descending order.

Indexes can also be unique, similar to the UNIQUE constraint, in that the index prevents duplicate entries in the column or combination of columns on which there's an index.

## The CREATE INDEX Command:

The basic syntax of **CREATE INDEX** is as follows:

CREATE INDEX index\_name ON table\_name;

## Single-Column Indexes:

A single-column index is one that is created based on only one table column. The basic syntax is as follows:

CREATE INDEX index\_name

ON table\_name (column\_name);

## Unique Indexes:

Unique indexes are used not only for performance, but also for data integrity. A unique index does not allow any duplicate values to be inserted into the table. The basic syntax is as follows:

CREATE UNIQUE INDEX index\_name

on table\_name (column\_name);

## Composite Indexes:

A composite index is an index on two or more columns of a table. The basic syntax is as follows:

CREATE INDEX index\_name

on table\_name (column1, column2);

Whether to create a single-column index or a composite index, take into consideration the column(s) that you may use very frequently in a query's WHERE clause as filter conditions.

Should there be only one column used, a single-column index should be the choice. Should there be two or more columns that are frequently used in the WHERE clause as filters, the composite index would be the best choice.

## Implicit Indexes:

Implicit indexes are indexes that are automatically created by the database server when an object is created. Indexes are automatically created for primary key constraints and unique constraints.

## The DROP INDEX Command:

An index can be dropped using SQL **DROP** command. Care should be taken when dropping an index because performance may be slowed or improved.

The basic syntax is as follows:

DROP INDEX index\_name;

You can check [INDEX Constraint](https://www.tutorialspoint.com/sql/sql-index.htm) chapter to see actual examples on Indexes.

## When should indexes be avoided?

Although indexes are intended to enhance a database's performance, there are times when they should be avoided. The following guidelines indicate when the use of an index should be reconsidered:

* Indexes should not be used on small tables.
* Tables that have frequent, large batch update or insert operations.
* Indexes should not be used on columns that contain a high number of NULL values.
* Columns that are frequently manipulated should not be indexed.
* The SQL **ALTER TABLE** command is used to add, delete or modify columns in an existing table.
* You would also use ALTER TABLE command to add and drop various constraints on a an existing table.

## Syntax:

* The basic syntax of **ALTER TABLE** to add a new column in an existing table is as follows:
* ALTER TABLE table\_name ADD column\_name datatype;
* The basic syntax of ALTER TABLE to **DROP COLUMN** in an existing table is as follows:
* ALTER TABLE table\_name DROP COLUMN column\_name;
* The basic syntax of ALTER TABLE to change the **DATA TYPE** of a column in a table is as follows:
* ALTER TABLE table\_name MODIFY COLUMN column\_name datatype;
* The basic syntax of ALTER TABLE to add a **NOT NULL** constraint to a column in a table is as follows:
* ALTER TABLE table\_name MODIFY column\_name datatype NOT NULL;
* The basic syntax of ALTER TABLE to **ADD UNIQUE CONSTRAINT** to a table is as follows:
* ALTER TABLE table\_name
* ADD CONSTRAINT MyUniqueConstraint UNIQUE(column1, column2...);
* The basic syntax of ALTER TABLE to **ADD CHECK CONSTRAINT** to a table is as follows:
* ALTER TABLE table\_name
* ADD CONSTRAINT MyUniqueConstraint CHECK (CONDITION);
* The basic syntax of ALTER TABLE to **ADD PRIMARY KEY** constraint to a table is as follows:
* ALTER TABLE table\_name
* ADD CONSTRAINT MyPrimaryKey PRIMARY KEY (column1, column2...);
* The basic syntax of ALTER TABLE to **DROP CONSTRAINT** from a table is as follows:
* ALTER TABLE table\_name
* DROP CONSTRAINT MyUniqueConstraint;
* If you're using MySQL, the code is as follows:
* ALTER TABLE table\_name
* DROP INDEX MyUniqueConstraint;
* The basic syntax of ALTER TABLE to **DROP PRIMARY KEY** constraint from a table is as follows:
* ALTER TABLE table\_name
* DROP CONSTRAINT MyPrimaryKey;
* If you're using MySQL, the code is as follows:
* ALTER TABLE table\_name
* DROP PRIMARY KEY;

## Example:

* Consider the CUSTOMERS table having the following records:
* +----+----------+-----+-----------+----------+
* | ID | NAME | AGE | ADDRESS | SALARY |
* +----+----------+-----+-----------+----------+
* | 1 | Ramesh | 32 | Ahmedabad | 2000.00 |
* | 2 | Khilan | 25 | Delhi | 1500.00 |
* | 3 | kaushik | 23 | Kota | 2000.00 |
* | 4 | Chaitali | 25 | Mumbai | 6500.00 |
* | 5 | Hardik | 27 | Bhopal | 8500.00 |
* | 6 | Komal | 22 | MP | 4500.00 |
* | 7 | Muffy | 24 | Indore | 10000.00 |
* +----+----------+-----+-----------+----------+
* Following is the example to ADD a new column in an existing table:
* ALTER TABLE CUSTOMERS ADD SEX char(1);
* Now, CUSTOMERS table is changed and following would be output from SELECT statement:
* +----+---------+-----+-----------+----------+------+
* | ID | NAME | AGE | ADDRESS | SALARY | SEX |
* +----+---------+-----+-----------+----------+------+
* | 1 | Ramesh | 32 | Ahmedabad | 2000.00 | NULL |
* | 2 | Ramesh | 25 | Delhi | 1500.00 | NULL |
* | 3 | kaushik | 23 | Kota | 2000.00 | NULL |
* | 4 | kaushik | 25 | Mumbai | 6500.00 | NULL |
* | 5 | Hardik | 27 | Bhopal | 8500.00 | NULL |
* | 6 | Komal | 22 | MP | 4500.00 | NULL |
* | 7 | Muffy | 24 | Indore | 10000.00 | NULL |
* +----+---------+-----+-----------+----------+------+
* Following is the example to DROP sex column from existing table:
* ALTER TABLE CUSTOMERS DROP SEX;
* Now, CUSTOMERS table is changed and following would be output from SELECT statement:
* +----+---------+-----+-----------+----------+
* | ID | NAME | AGE | ADDRESS | SALARY |
* +----+---------+-----+-----------+----------+
* | 1 | Ramesh | 32 | Ahmedabad | 2000.00 |
* | 2 | Ramesh | 25 | Delhi | 1500.00 |
* | 3 | kaushik | 23 | Kota | 2000.00 |
* | 4 | kaushik | 25 | Mumbai | 6500.00 |
* | 5 | Hardik | 27 | Bhopal | 8500.00 |
* | 6 | Komal | 22 | MP | 4500.00 |
* | 7 | Muffy | 24 | Indore | 10000.00 |
* +----+---------+-----+-----------+----------+
* The SQL **TRUNCATE TABLE** command is used to delete complete data from an existing table.
* You can also use DROP TABLE command to delete complete table but it would remove complete table structure form the database and you would need to re-create this table once again if you wish you store some data.

## Syntax:

* The basic syntax of **TRUNCATE TABLE** is as follows:
* TRUNCATE TABLE table\_name;

## Example:

* Consider the CUSTOMERS table having the following records:
* +----+----------+-----+-----------+----------+
* | ID | NAME | AGE | ADDRESS | SALARY |
* +----+----------+-----+-----------+----------+
* | 1 | Ramesh | 32 | Ahmedabad | 2000.00 |
* | 2 | Khilan | 25 | Delhi | 1500.00 |
* | 3 | kaushik | 23 | Kota | 2000.00 |
* | 4 | Chaitali | 25 | Mumbai | 6500.00 |
* | 5 | Hardik | 27 | Bhopal | 8500.00 |
* | 6 | Komal | 22 | MP | 4500.00 |
* | 7 | Muffy | 24 | Indore | 10000.00 |
* +----+----------+-----+-----------+----------+
* Following is the example to truncate:
* SQL > TRUNCATE TABLE CUSTOMERS;
* Now, CUSTOMERS table is truncated and following would be the output from SELECT statement:
* SQL> SELECT \* FROM CUSTOMERS;
* Empty set (0.00 sec)

 view is nothing more than a SQL statement that is stored in the database with an associated name. A view is actually a composition of a table in the form of a predefined SQL query.

A view can contain all rows of a table or select rows from a table. A view can be created from one or many tables which depends on the written SQL query to create a view.

Views, which are kind of virtual tables, allow users to do the following:

* Structure data in a way that users or classes of users find natural or intuitive.
* Restrict access to the data such that a user can see and (sometimes) modify exactly what they need and no more.
* Summarize data from various tables which can be used to generate reports.

## Creating Views:

Database views are created using the **CREATE VIEW** statement. Views can be created from a single table, multiple tables, or another view.

To create a view, a user must have the appropriate system privilege according to the specific implementation.

The basic CREATE VIEW syntax is as follows:

CREATE VIEW view\_name AS

SELECT column1, column2.....

FROM table\_name

WHERE [condition];

You can include multiple tables in your SELECT statement in very similar way as you use them in normal SQL SELECT query.

## Example:

Consider the CUSTOMERS table having the following records:

+----+----------+-----+-----------+----------+

| ID | NAME | AGE | ADDRESS | SALARY |

+----+----------+-----+-----------+----------+

| 1 | Ramesh | 32 | Ahmedabad | 2000.00 |

| 2 | Khilan | 25 | Delhi | 1500.00 |

| 3 | kaushik | 23 | Kota | 2000.00 |

| 4 | Chaitali | 25 | Mumbai | 6500.00 |

| 5 | Hardik | 27 | Bhopal | 8500.00 |

| 6 | Komal | 22 | MP | 4500.00 |

| 7 | Muffy | 24 | Indore | 10000.00 |

+----+----------+-----+-----------+----------+

Now, following is the example to create a view from CUSTOMERS table. This view would be used to have customer name and age from CUSTOMERS table:

SQL > CREATE VIEW CUSTOMERS\_VIEW AS

SELECT name, age

FROM CUSTOMERS;

Now, you can query CUSTOMERS\_VIEW in similar way as you query an actual table. Following is the example:

SQL > SELECT \* FROM CUSTOMERS\_VIEW;

This would produce the following result:

+----------+-----+

| name | age |

+----------+-----+

| Ramesh | 32 |

| Khilan | 25 |

| kaushik | 23 |

| Chaitali | 25 |

| Hardik | 27 |

| Komal | 22 |

| Muffy | 24 |

+----------+-----+

## The WITH CHECK OPTION:

The WITH CHECK OPTION is a CREATE VIEW statement option. The purpose of the WITH CHECK OPTION is to ensure that all UPDATE and INSERTs satisfy the condition(s) in the view definition.

If they do not satisfy the condition(s), the UPDATE or INSERT returns an error.

The following is an example of creating same view CUSTOMERS\_VIEW with the WITH CHECK OPTION:

CREATE VIEW CUSTOMERS\_VIEW AS

SELECT name, age

FROM CUSTOMERS

WHERE age IS NOT NULL

WITH CHECK OPTION;

The WITH CHECK OPTION in this case should deny the entry of any NULL values in the view's AGE column, because the view is defined by data that does not have a NULL value in the AGE column.

## Updating a View:

A view can be updated under certain conditions:

* The SELECT clause may not contain the keyword DISTINCT.
* The SELECT clause may not contain summary functions.
* The SELECT clause may not contain set functions.
* The SELECT clause may not contain set operators.
* The SELECT clause may not contain an ORDER BY clause.
* The FROM clause may not contain multiple tables.
* The WHERE clause may not contain subqueries.
* The query may not contain GROUP BY or HAVING.
* Calculated columns may not be updated.
* All NOT NULL columns from the base table must be included in the view in order for the INSERT query to function.

So if a view satisfies all the above-mentioned rules then you can update a view. Following is an example to update the age of Ramesh:

SQL > UPDATE CUSTOMERS\_VIEW

SET AGE = 35

WHERE name='Ramesh';

This would ultimately update the base table CUSTOMERS and same would reflect in the view itself. Now, try to query base table, and SELECT statement would produce the following result:

+----+----------+-----+-----------+----------+

| ID | NAME | AGE | ADDRESS | SALARY |

+----+----------+-----+-----------+----------+

| 1 | Ramesh | 35 | Ahmedabad | 2000.00 |

| 2 | Khilan | 25 | Delhi | 1500.00 |

| 3 | kaushik | 23 | Kota | 2000.00 |

| 4 | Chaitali | 25 | Mumbai | 6500.00 |

| 5 | Hardik | 27 | Bhopal | 8500.00 |

| 6 | Komal | 22 | MP | 4500.00 |

| 7 | Muffy | 24 | Indore | 10000.00 |

+----+----------+-----+-----------+----------+

## Inserting Rows into a View:

Rows of data can be inserted into a view. The same rules that apply to the UPDATE command also apply to the INSERT command.

Here we can not insert rows in CUSTOMERS\_VIEW because we have not included all the NOT NULL columns in this view, otherwise you can insert rows in a view in similar way as you insert them in a table.

## Deleting Rows into a View:

Rows of data can be deleted from a view. The same rules that apply to the UPDATE and INSERT commands apply to the DELETE command.

Following is an example to delete a record having AGE= 22.

SQL > DELETE FROM CUSTOMERS\_VIEW

WHERE age = 22;

This would ultimately delete a row from the base table CUSTOMERS and same would reflect in the view itself. Now, try to query base table, and SELECT statement would produce the following result:

+----+----------+-----+-----------+----------+

| ID | NAME | AGE | ADDRESS | SALARY |

+----+----------+-----+-----------+----------+

| 1 | Ramesh | 35 | Ahmedabad | 2000.00 |

| 2 | Khilan | 25 | Delhi | 1500.00 |

| 3 | kaushik | 23 | Kota | 2000.00 |

| 4 | Chaitali | 25 | Mumbai | 6500.00 |

| 5 | Hardik | 27 | Bhopal | 8500.00 |

| 7 | Muffy | 24 | Indore | 10000.00 |

+----+----------+-----+-----------+----------+

## Dropping Views:

Obviously, where you have a view, you need a way to drop the view if it is no longer needed. The syntax is very simple as given below:

DROP VIEW view\_name;

Following is an example to drop CUSTOMERS\_VIEW from CUSTOMERS table:

DROP VIEW CUSTOMERS\_VIEW;

The HAVING clause enables you to specify conditions that filter which group results appear in the final results.

The WHERE clause places conditions on the selected columns, whereas the HAVING clause places conditions on groups created by the GROUP BY clause.

## Syntax:

The following is the position of the HAVING clause in a query:

SELECT

FROM

WHERE

GROUP BY

HAVING

ORDER BY

The HAVING clause must follow the GROUP BY clause in a query and must also precede the ORDER BY clause if used. The following is the syntax of the SELECT statement, including the HAVING clause:

SELECT column1, column2

FROM table1, table2

WHERE [ conditions ]

GROUP BY column1, column2

HAVING [ conditions ]

ORDER BY column1, column2

## Example:

Consider the CUSTOMERS table having the following records:

+----+----------+-----+-----------+----------+

| ID | NAME | AGE | ADDRESS | SALARY |

+----+----------+-----+-----------+----------+

| 1 | Ramesh | 32 | Ahmedabad | 2000.00 |

| 2 | Khilan | 25 | Delhi | 1500.00 |

| 3 | kaushik | 23 | Kota | 2000.00 |

| 4 | Chaitali | 25 | Mumbai | 6500.00 |

| 5 | Hardik | 27 | Bhopal | 8500.00 |

| 6 | Komal | 22 | MP | 4500.00 |

| 7 | Muffy | 24 | Indore | 10000.00 |

+----+----------+-----+-----------+----------+

Following is the example, which would display record for which similar age count would be more than or equal to 2:

SQL > SELECT ID, NAME, AGE, ADDRESS, SALARY

FROM CUSTOMERS

GROUP BY age

HAVING COUNT(age) >= 2;

This would produce the following result:

+----+--------+-----+---------+---------+

| ID | NAME | AGE | ADDRESS | SALARY |

+----+--------+-----+---------+---------+

| 2 | Khilan | 25 | Delhi | 1500.00 |

+----+--------+-----+---------+---------+

A transaction is a unit of work that is performed against a database. Transactions are units or sequences of work accomplished in a logical order, whether in a manual fashion by a user or automatically by some sort of a database program.

A transaction is the propagation of one or more changes to the database. For example, if you are creating a record or updating a record or deleting a record from the table, then you are performing transaction on the table. It is important to control transactions to ensure data integrity and to handle database errors.

Practically, you will club many SQL queries into a group and you will execute all of them together as a part of a transaction.

Properties of Transactions:

Transactions have the following four standard properties, usually referred to by the acronym ACID:

* **Atomicity:** ensures that all operations within the work unit are completed successfully; otherwise, the transaction is aborted at the point of failure, and previous operations are rolled back to their former state.
* **Consistency:** ensures that the database properly changes states upon a successfully committed transaction.
* **Isolation:** enables transactions to operate independently of and transparent to each other.
* **Durability:** ensures that the result or effect of a committed transaction persists in case of a system failure.

Transaction Control:

There are following commands used to control transactions:

* **COMMIT:** to save the changes.
* **ROLLBACK:** to rollback the changes.
* **SAVEPOINT:** creates points within groups of transactions in which to ROLLBACK
* **SET TRANSACTION:** Places a name on a transaction.

Transactional control commands are only used with the DML commands INSERT, UPDATE and DELETE only. They can not be used while creating tables or dropping them because these operations are automatically commited in the database.

The COMMIT Command:

The COMMIT command is the transactional command used to save changes invoked by a transaction to the database.

The COMMIT command saves all transactions to the database since the last COMMIT or ROLLBACK command.

The syntax for COMMIT command is as follows:

COMMIT;

Example:

Consider the CUSTOMERS table having the following records:

+----+----------+-----+-----------+----------+

| ID | NAME | AGE | ADDRESS | SALARY |

+----+----------+-----+-----------+----------+

| 1 | Ramesh | 32 | Ahmedabad | 2000.00 |

| 2 | Khilan | 25 | Delhi | 1500.00 |

| 3 | kaushik | 23 | Kota | 2000.00 |

| 4 | Chaitali | 25 | Mumbai | 6500.00 |

| 5 | Hardik | 27 | Bhopal | 8500.00 |

| 6 | Komal | 22 | MP | 4500.00 |

| 7 | Muffy | 24 | Indore | 10000.00 |

+----+----------+-----+-----------+----------+

Following is the example which would delete records from the table having age = 25 and then COMMIT the changes in the database.

SQL> DELETE FROM CUSTOMERS

WHERE AGE = 25;

SQL> COMMIT;

As a result, two rows from the table would be deleted and SELECT statement would produce the following result:

+----+----------+-----+-----------+----------+

| ID | NAME | AGE | ADDRESS | SALARY |

+----+----------+-----+-----------+----------+

| 1 | Ramesh | 32 | Ahmedabad | 2000.00 |

| 3 | kaushik | 23 | Kota | 2000.00 |

| 5 | Hardik | 27 | Bhopal | 8500.00 |

| 6 | Komal | 22 | MP | 4500.00 |

| 7 | Muffy | 24 | Indore | 10000.00 |

+----+----------+-----+-----------+----------+

The ROLLBACK Command:

The ROLLBACK command is the transactional command used to undo transactions that have not already been saved to the database.

The ROLLBACK command can only be used to undo transactions since the last COMMIT or ROLLBACK command was issued.

The syntax for ROLLBACK command is as follows:

ROLLBACK;

Example:

Consider the CUSTOMERS table having the following records:

+----+----------+-----+-----------+----------+

| ID | NAME | AGE | ADDRESS | SALARY |

+----+----------+-----+-----------+----------+

| 1 | Ramesh | 32 | Ahmedabad | 2000.00 |

| 2 | Khilan | 25 | Delhi | 1500.00 |

| 3 | kaushik | 23 | Kota | 2000.00 |

| 4 | Chaitali | 25 | Mumbai | 6500.00 |

| 5 | Hardik | 27 | Bhopal | 8500.00 |

| 6 | Komal | 22 | MP | 4500.00 |

| 7 | Muffy | 24 | Indore | 10000.00 |

+----+----------+-----+-----------+----------+

Following is the example, which would delete records from the table having age = 25 and then ROLLBACK the changes in the database.

SQL> DELETE FROM CUSTOMERS

WHERE AGE = 25;

SQL> ROLLBACK;

As a result, delete operation would not impact the table and SELECT statement would produce the following result:

+----+----------+-----+-----------+----------+

| ID | NAME | AGE | ADDRESS | SALARY |

+----+----------+-----+-----------+----------+

| 1 | Ramesh | 32 | Ahmedabad | 2000.00 |

| 2 | Khilan | 25 | Delhi | 1500.00 |

| 3 | kaushik | 23 | Kota | 2000.00 |

| 4 | Chaitali | 25 | Mumbai | 6500.00 |

| 5 | Hardik | 27 | Bhopal | 8500.00 |

| 6 | Komal | 22 | MP | 4500.00 |

| 7 | Muffy | 24 | Indore | 10000.00 |

+----+----------+-----+-----------+----------+

The SAVEPOINT Command:

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

The syntax for SAVEPOINT command is as follows:

SAVEPOINT SAVEPOINT\_NAME;

This command serves only in the creation of a SAVEPOINT among transactional statements. The ROLLBACK command is used to undo a group of transactions.

The syntax for rolling back to a SAVEPOINT is as follows:

ROLLBACK TO SAVEPOINT\_NAME;

Following is an example where you plan to delete the three different records from the CUSTOMERS table. You want to create a SAVEPOINT before each delete, so that you can ROLLBACK to any SAVEPOINT at any time to return the appropriate data to its original state:

Example:

Consider the CUSTOMERS table having the following records:

+----+----------+-----+-----------+----------+

| ID | NAME | AGE | ADDRESS | SALARY |

+----+----------+-----+-----------+----------+

| 1 | Ramesh | 32 | Ahmedabad | 2000.00 |

| 2 | Khilan | 25 | Delhi | 1500.00 |

| 3 | kaushik | 23 | Kota | 2000.00 |

| 4 | Chaitali | 25 | Mumbai | 6500.00 |

| 5 | Hardik | 27 | Bhopal | 8500.00 |

| 6 | Komal | 22 | MP | 4500.00 |

| 7 | Muffy | 24 | Indore | 10000.00 |

+----+----------+-----+-----------+----------+

Now, here is the series of operations:

SQL> SAVEPOINT SP1;

Savepoint created.

SQL> DELETE FROM CUSTOMERS WHERE ID=1;

1 row deleted.

SQL> SAVEPOINT SP2;

Savepoint created.

SQL> DELETE FROM CUSTOMERS WHERE ID=2;

1 row deleted.

SQL> SAVEPOINT SP3;

Savepoint created.

SQL> DELETE FROM CUSTOMERS WHERE ID=3;

1 row deleted.

Now that the three deletions have taken place, say you have changed your mind and decided to ROLLBACK to the SAVEPOINT that you identified as SP2. Because SP2 was created after the first deletion, the last two deletions are undone:

SQL> ROLLBACK TO SP2;

Rollback complete.

Notice that only the first deletion took place since you rolled back to SP2:

SQL> SELECT \* FROM CUSTOMERS;

+----+----------+-----+-----------+----------+

| ID | NAME | AGE | ADDRESS | SALARY |

+----+----------+-----+-----------+----------+

| 2 | Khilan | 25 | Delhi | 1500.00 |

| 3 | kaushik | 23 | Kota | 2000.00 |

| 4 | Chaitali | 25 | Mumbai | 6500.00 |

| 5 | Hardik | 27 | Bhopal | 8500.00 |

| 6 | Komal | 22 | MP | 4500.00 |

| 7 | Muffy | 24 | Indore | 10000.00 |

+----+----------+-----+-----------+----------+

6 rows selected.

The RELEASE SAVEPOINT Command:

The RELEASE SAVEPOINT command is used to remove a SAVEPOINT that you have created.

The syntax for RELEASE SAVEPOINT is as follows:

RELEASE SAVEPOINT SAVEPOINT\_NAME;

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

The SET TRANSACTION Command:

The SET TRANSACTION command can be used to initiate a database transaction. This command is used to specify characteristics for the transaction that follows.

For example, you can specify a transaction to be read only, or read write.

The syntax for SET TRANSACTION is as follows:

SET TRANSACTION [ READ WRITE | READ ONLY ];

We already have discussed SQL **LIKE** operator, which is used to compare a value to similar values using wildcard operators.

SQL supports following two wildcard operators in conjunction with the LIKE operator:

|  |  |
| --- | --- |
| **Wildcards** | **Description** |
| The percent sign (%) | Matches one or more characters. Note that MS Access uses the asterisk (\*) wildcard character instead of the percent sign (%) wildcard character. |
| The underscore (\_) | Matches one character. Note that MS Access uses a question mark (?) instead of the underscore (\_) to match any one character. |

The percent sign represents zero, one, or multiple characters. The underscore represents a single number or character. The symbols can be used in combinations.

## Syntax:

The basic syntax of '%' and '\_' is as follows:

SELECT FROM table\_name

WHERE column LIKE 'XXXX%'

or

SELECT FROM table\_name

WHERE column LIKE '%XXXX%'

or

SELECT FROM table\_name

WHERE column LIKE 'XXXX\_'

or

SELECT FROM table\_name

WHERE column LIKE '\_XXXX'

or

SELECT FROM table\_name

WHERE column LIKE '\_XXXX\_'

You can combine N number of conditions using AND or OR operators. Here, XXXX could be any numeric or string value.

## Example:

Here are number of examples showing WHERE part having different LIKE clause with '%' and '\_' operators:

|  |  |
| --- | --- |
| **Statement** | **Description** |
| WHERE SALARY LIKE '200%' | Finds any values that start with 200 |
| WHERE SALARY LIKE '%200%' | Finds any values that have 200 in any position |
| WHERE SALARY LIKE '\_00%' | Finds any values that have 00 in the second and third positions |
| WHERE SALARY LIKE '2\_%\_%' | Finds any values that start with 2 and are at least 3 characters in length |
| WHERE SALARY LIKE '%2' | Finds any values that end with 2 |
| WHERE SALARY LIKE '\_2%3' | Finds any values that have a 2 in the second position and end with a 3 |
| WHERE SALARY LIKE '2\_\_\_3' | Finds any values in a five-digit number that start with 2 and end with 3 |

Let us take a real example, consider the CUSTOMERS table having the following records:

+----+----------+-----+-----------+----------+

| ID | NAME | AGE | ADDRESS | SALARY |

+----+----------+-----+-----------+----------+

| 1 | Ramesh | 32 | Ahmedabad | 2000.00 |

| 2 | Khilan | 25 | Delhi | 1500.00 |

| 3 | kaushik | 23 | Kota | 2000.00 |

| 4 | Chaitali | 25 | Mumbai | 6500.00 |

| 5 | Hardik | 27 | Bhopal | 8500.00 |

| 6 | Komal | 22 | MP | 4500.00 |

| 7 | Muffy | 24 | Indore | 10000.00 |

+----+----------+-----+-----------+----------+

Following is an example, which would display all the records from CUSTOMERS table where SALARY starts with 200:

SQL> SELECT \* FROM CUSTOMERS

WHERE SALARY LIKE '200%';

This would produce the following result:

+----+----------+-----+-----------+----------+

| ID | NAME | AGE | ADDRESS | SALARY |

+----+----------+-----+-----------+----------+

| 1 | Ramesh | 32 | Ahmedabad | 2000.00 |

| 3 | kaushik | 23 | Kota | 2000.00 |

+----+----------+-----+-----------+----------+

Following is a list of all important Date and Time related functions available through SQL. There are various other functions supported by your RDBMS. Given list is based on MySQL RDBMS.

|  |  |
| --- | --- |
| **Name** | **Description** |
| [**ADDDATE()**](https://www.tutorialspoint.com/sql/sql-date-functions.htm#function_adddate) | Adds dates |
| [**ADDTIME()**](https://www.tutorialspoint.com/sql/sql-date-functions.htm#function_addtime) | Adds time |
| [**CONVERT\_TZ()**](https://www.tutorialspoint.com/sql/sql-date-functions.htm#function_convert-tz) | Converts from one timezone to another |
| [**CURDATE()**](https://www.tutorialspoint.com/sql/sql-date-functions.htm#function_curdate) | Returns the current date |
| [**CURRENT\_DATE(), CURRENT\_DATE**](https://www.tutorialspoint.com/sql/sql-date-functions.htm#function_current-date) | Synonyms for CURDATE() |
| [**CURRENT\_TIME(), CURRENT\_TIME**](https://www.tutorialspoint.com/sql/sql-date-functions.htm#function_current-time) | Synonyms for CURTIME() |
| [**CURRENT\_TIMESTAMP(), CURRENT\_TIMESTAMP**](https://www.tutorialspoint.com/sql/sql-date-functions.htm#function_current-timestamp) | Synonyms for NOW() |
| [**CURTIME()**](https://www.tutorialspoint.com/sql/sql-date-functions.htm#function_curtime) | Returns the current time |
| [**DATE\_ADD()**](https://www.tutorialspoint.com/sql/sql-date-functions.htm#function_date-add) | Adds two dates |
| [**DATE\_FORMAT()**](https://www.tutorialspoint.com/sql/sql-date-functions.htm#function_date-format) | Formats date as specified |
| [**DATE\_SUB()**](https://www.tutorialspoint.com/sql/sql-date-functions.htm#function_date-sub) | Subtracts two dates |
| [**DATE()**](https://www.tutorialspoint.com/sql/sql-date-functions.htm#function_date) | Extracts the date part of a date or datetime expression |
| [**DATEDIFF()**](https://www.tutorialspoint.com/sql/sql-date-functions.htm#function_datediff) | Subtracts two dates |
| [**DAY()**](https://www.tutorialspoint.com/sql/sql-date-functions.htm#function_day) | Synonym for DAYOFMONTH() |
| [**DAYNAME()**](https://www.tutorialspoint.com/sql/sql-date-functions.htm#function_dayname) | Returns the name of the weekday |
| [**DAYOFMONTH()**](https://www.tutorialspoint.com/sql/sql-date-functions.htm#function_dayofmonth) | Returns the day of the month (1-31) |
| [**DAYOFWEEK()**](https://www.tutorialspoint.com/sql/sql-date-functions.htm#function_dayofweek) | Returns the weekday index of the argument |
| [**DAYOFYEAR()**](https://www.tutorialspoint.com/sql/sql-date-functions.htm#function_dayofyear) | Returns the day of the year (1-366) |
| [**EXTRACT**](https://www.tutorialspoint.com/sql/sql-date-functions.htm#function_extract) | Extracts part of a date |
| [**FROM\_DAYS()**](https://www.tutorialspoint.com/sql/sql-date-functions.htm#function_from-days) | Converts a day number to a date |
| [**FROM\_UNIXTIME()**](https://www.tutorialspoint.com/sql/sql-date-functions.htm#function_from-unixtime) | Formats date as a UNIX timestamp |
| [**HOUR()**](https://www.tutorialspoint.com/sql/sql-date-functions.htm#function_hour) | Extracts the hour |
| [**LAST\_DAY**](https://www.tutorialspoint.com/sql/sql-date-functions.htm#function_last-day) | Returns the last day of the month for the argument |
| [**LOCALTIME(), LOCALTIME**](https://www.tutorialspoint.com/sql/sql-date-functions.htm#function_localtime) | Synonym for NOW() |
| [**LOCALTIMESTAMP, LOCALTIMESTAMP()**](https://www.tutorialspoint.com/sql/sql-date-functions.htm#function_localtimestamp) | Synonym for NOW() |
| [**MAKEDATE()**](https://www.tutorialspoint.com/sql/sql-date-functions.htm#function_makedate) | Creates a date from the year and day of year |
| [**MAKETIME**](https://www.tutorialspoint.com/sql/sql-date-functions.htm#function_maketime) | MAKETIME() |
| [**MICROSECOND()**](https://www.tutorialspoint.com/sql/sql-date-functions.htm#function_microsecond) | Returns the microseconds from argument |
| [**MINUTE()**](https://www.tutorialspoint.com/sql/sql-date-functions.htm#function_minute) | Returns the minute from the argument |
| [**MONTH()**](https://www.tutorialspoint.com/sql/sql-date-functions.htm#function_month) | Return the month from the date passed |
| [**MONTHNAME()**](https://www.tutorialspoint.com/sql/sql-date-functions.htm#function_monthname) | Returns the name of the month |
| [**NOW()**](https://www.tutorialspoint.com/sql/sql-date-functions.htm#function_now) | Returns the current date and time |
| [**PERIOD\_ADD()**](https://www.tutorialspoint.com/sql/sql-date-functions.htm#function_period-add) | Adds a period to a year-month |
| [**PERIOD\_DIFF()**](https://www.tutorialspoint.com/sql/sql-date-functions.htm#function_period-diff) | Returns the number of months between periods |
| [**QUARTER()**](https://www.tutorialspoint.com/sql/sql-date-functions.htm#function_quarter) | Returns the quarter from a date argument |
| [**SEC\_TO\_TIME()**](https://www.tutorialspoint.com/sql/sql-date-functions.htm#function_sec-to-time) | Converts seconds to 'HH:MM:SS' format |
| [**SECOND()**](https://www.tutorialspoint.com/sql/sql-date-functions.htm#function_second) | Returns the second (0-59) |
| [**STR\_TO\_DATE()**](https://www.tutorialspoint.com/sql/sql-date-functions.htm#function_str-to-date) | Converts a string to a date |
| [**SUBDATE()**](https://www.tutorialspoint.com/sql/sql-date-functions.htm#function_subdate) | When invoked with three arguments a synonym for DATE\_SUB() |
| [**SUBTIME()**](https://www.tutorialspoint.com/sql/sql-date-functions.htm#function_subtime) | Subtracts times |
| [**SYSDATE()**](https://www.tutorialspoint.com/sql/sql-date-functions.htm#function_sysdate) | Returns the time at which the function executes |
| [**TIME\_FORMAT()**](https://www.tutorialspoint.com/sql/sql-date-functions.htm#function_time-format) | Formats as time |
| [**TIME\_TO\_SEC()**](https://www.tutorialspoint.com/sql/sql-date-functions.htm#function_time-to-sec) | Returns the argument converted to seconds |
| [**TIME()**](https://www.tutorialspoint.com/sql/sql-date-functions.htm#function_time) | Extracts the time portion of the expression passed |
| [**TIMEDIFF()**](https://www.tutorialspoint.com/sql/sql-date-functions.htm#function_timediff) | Subtracts time |
| [**TIMESTAMP()**](https://www.tutorialspoint.com/sql/sql-date-functions.htm#function_timestamp) | With a single argument, this function returns the date or datetime expression. With two arguments, the sum of the arguments |
| [**TIMESTAMPADD()**](https://www.tutorialspoint.com/sql/sql-date-functions.htm#function_timestampadd) | Adds an interval to a datetime expression |
| [**TIMESTAMPDIFF()**](https://www.tutorialspoint.com/sql/sql-date-functions.htm#function_timestampdiff) | Subtracts an interval from a datetime expression |
| [**TO\_DAYS()**](https://www.tutorialspoint.com/sql/sql-date-functions.htm#function_to-days) | Returns the date argument converted to days |
| [**UNIX\_TIMESTAMP()**](https://www.tutorialspoint.com/sql/sql-date-functions.htm#function_unix-timestamp) | Returns a UNIX timestamp |
| [**UTC\_DATE()**](https://www.tutorialspoint.com/sql/sql-date-functions.htm#function_utc-date) | Returns the current UTC date |
| [**UTC\_TIME()**](https://www.tutorialspoint.com/sql/sql-date-functions.htm#function_utc-time) | Returns the current UTC time |
| [**UTC\_TIMESTAMP()**](https://www.tutorialspoint.com/sql/sql-date-functions.htm#function_utc-timestamp) | Returns the current UTC date and time |
| [**WEEK()**](https://www.tutorialspoint.com/sql/sql-date-functions.htm#function_week) | Returns the week number |
| [**WEEKDAY()**](https://www.tutorialspoint.com/sql/sql-date-functions.htm#function_weekday) | Returns the weekday index |
| [**WEEKOFYEAR()**](https://www.tutorialspoint.com/sql/sql-date-functions.htm#function_weekofyear) | Returns the calendar week of the date (1-53) |
| [**YEAR()**](https://www.tutorialspoint.com/sql/sql-date-functions.htm#function_year) | Returns the year |
| [**YEARWEEK()**](https://www.tutorialspoint.com/sql/sql-date-functions.htm#function_yearweek) | Returns the year and week |

## ADDDATE(date,INTERVAL expr unit), ADDDATE(expr,days)

When invoked with the INTERVAL form of the second argument, ADDDATE() is a synonym for DATE\_ADD(). The related function SUBDATE() is a synonym for DATE\_SUB(). For information on the INTERVAL unit argument, see the discussion for DATE\_ADD().

mysql> SELECT DATE\_ADD('1998-01-02', INTERVAL 31 DAY);

+---------------------------------------------------------+

| DATE\_ADD('1998-01-02', INTERVAL 31 DAY) |

+---------------------------------------------------------+

| 1998-02-02 |

+---------------------------------------------------------+

1 row in set (0.00 sec)

mysql> SELECT ADDDATE('1998-01-02', INTERVAL 31 DAY);

+---------------------------------------------------------+

| ADDDATE('1998-01-02', INTERVAL 31 DAY) |

+---------------------------------------------------------+

| 1998-02-02 |

+---------------------------------------------------------+

1 row in set (0.00 sec)

When invoked with the days form of the second argument, MySQL treats it as an integer number of days to be added to expr.

mysql> SELECT ADDDATE('1998-01-02', 31);

+---------------------------------------------------------+

| DATE\_ADD('1998-01-02', INTERVAL 31 DAY) |

+---------------------------------------------------------+

| 1998-02-02 |

+---------------------------------------------------------+

1 row in set (0.00 sec)

## ADDTIME(expr1,expr2)

ADDTIME() adds expr2 to expr1 and returns the result. expr1 is a time or datetime expression, and expr2 is a time expression.

mysql> SELECT ADDTIME('1997-12-31 23:59:59.999999','1 1:1:1.000002');

+---------------------------------------------------------+

| DATE\_ADD('1997-12-31 23:59:59.999999','1 1:1:1.000002') |

+---------------------------------------------------------+

| 1998-01-02 01:01:01.000001 |

+---------------------------------------------------------+

1 row in set (0.00 sec)

## CONVERT\_TZ(dt,from\_tz,to\_tz)

This converts a datetime value dt from the time zone given by from\_tz to the time zone given by to\_tz and returns the resulting value. This function returns NULL if the arguments are invalid.

mysql> SELECT CONVERT\_TZ('2004-01-01 12:00:00','GMT','MET');

+---------------------------------------------------------+

| CONVERT\_TZ('2004-01-01 12:00:00','GMT','MET') |

+---------------------------------------------------------+

| 2004-01-01 13:00:00 |

+---------------------------------------------------------+

1 row in set (0.00 sec)

mysql> SELECT CONVERT\_TZ('2004-01-01 12:00:00','+00:00','+10:00');

+---------------------------------------------------------+

| CONVERT\_TZ('2004-01-01 12:00:00','+00:00','+10:00') |

+---------------------------------------------------------+

| 2004-01-01 22:00:00 |

+---------------------------------------------------------+

1 row in set (0.00 sec)

## CURDATE()

Returns the current date as a value in 'YYYY-MM-DD' or YYYYMMDD format, depending on whether the function is used in a string or numeric context.

mysql> SELECT CURDATE();

+---------------------------------------------------------+

| CURDATE() |

+---------------------------------------------------------+

| 1997-12-15 |

+---------------------------------------------------------+

1 row in set (0.00 sec)

mysql> SELECT CURDATE() + 0;

+---------------------------------------------------------+

| CURDATE() + 0 |

+---------------------------------------------------------+

| 19971215 |

+---------------------------------------------------------+

1 row in set (0.00 sec)

## CURRENT\_DATE and CURRENT\_DATE()

CURRENT\_DATE and CURRENT\_DATE() are synonyms for CURDATE()

## CURTIME()

Returns the current time as a value in 'HH:MM:SS' or HHMMSS format, depending on whether the function is used in a string or numeric context. The value is expressed in the current time zone.

mysql> SELECT CURTIME();

+---------------------------------------------------------+

| CURTIME() |

+---------------------------------------------------------+

| 23:50:26 |

+---------------------------------------------------------+

1 row in set (0.00 sec)

mysql> SELECT CURTIME() + 0;

+---------------------------------------------------------+

| CURTIME() + 0 |

+---------------------------------------------------------+

| 235026 |

+---------------------------------------------------------+

1 row in set (0.00 sec)

## CURRENT\_TIME and CURRENT\_TIME()

CURRENT\_TIME and CURRENT\_TIME() are synonyms for CURTIME().

## CURRENT\_TIMESTAMP and CURRENT\_TIMESTAMP()

CURRENT\_TIMESTAMP and CURRENT\_TIMESTAMP() are synonyms for NOW().

## DATE(expr)

Extracts the date part of the date or datetime expression expr.

mysql> SELECT DATE('2003-12-31 01:02:03');

+---------------------------------------------------------+

| DATE('2003-12-31 01:02:03') |

+---------------------------------------------------------+

| 2003-12-31 |

+---------------------------------------------------------+

1 row in set (0.00 sec)

## DATEDIFF(expr1,expr2)

DATEDIFF() returns expr1 . expr2 expressed as a value in days from one date to the other. expr1 and expr2 are date or date-and-time expressions. Only the date parts of the values are used in the calculation.

mysql> SELECT DATEDIFF('1997-12-31 23:59:59','1997-12-30');

+---------------------------------------------------------+

| DATEDIFF('1997-12-31 23:59:59','1997-12-30') |

+---------------------------------------------------------+

| 1 |

+---------------------------------------------------------+

1 row in set (0.00 sec)

## DATE\_ADD(date,INTERVAL expr unit), DATE\_SUB(date,INTERVAL expr unit)

These functions perform date arithmetic. date is a DATETIME or DATE value specifying the starting date. expr is an expression specifying the interval value to be added or subtracted from the starting date. expr is a string; it may start with a '-' for negative intervals. unit is a keyword indicating the units in which the expression should be interpreted.

The INTERVAL keyword and the unit specifier are not case sensitive.

The following table shows the expected form of the expr argument for each unit value;

|  |  |
| --- | --- |
| unit **Value** | **Expected**expr**Format** |
| MICROSECOND | MICROSECONDS |
| SECOND | SECONDS |
| MINUTE | MINUTES |
| HOUR | HOURS |
| DAY | DAYS |
| WEEK | WEEKS |
| MONTH | MONTHS |
| QUARTER | QUARTERS |
| YEAR | YEARS |
| SECOND\_MICROSECOND | 'SECONDS.MICROSECONDS' |
| MINUTE\_MICROSECOND | 'MINUTES.MICROSECONDS' |
| MINUTE\_SECOND | 'MINUTES:SECONDS' |
| HOUR\_MICROSECOND | 'HOURS.MICROSECONDS' |
| HOUR\_SECOND | 'HOURS:MINUTES:SECONDS' |
| HOUR\_MINUTE | 'HOURS:MINUTES' |
| DAY\_MICROSECOND | 'DAYS.MICROSECONDS' |
| DAY\_SECOND | 'DAYS HOURS:MINUTES:SECONDS' |
| DAY\_MINUTE | 'DAYS HOURS:MINUTES' |
| DAY\_HOUR | 'DAYS HOURS' |
| YEAR\_MONTH | 'YEARS-MONTHS' |

The values QUARTER and WEEK are available beginning with MySQL 5.0.0.

mysql> SELECT DATE\_ADD('1997-12-31 23:59:59',

-> INTERVAL '1:1' MINUTE\_SECOND);

+---------------------------------------------------------+

| DATE\_ADD('1997-12-31 23:59:59', INTERVAL... |

+---------------------------------------------------------+

| 1998-01-01 00:01:00 |

+---------------------------------------------------------+

1 row in set (0.00 sec)

mysql> SELECT DATE\_ADD('1999-01-01', INTERVAL 1 HOUR);

+---------------------------------------------------------+

| DATE\_ADD('1999-01-01', INTERVAL 1 HOUR) |

+---------------------------------------------------------+

| 1999-01-01 01:00:00 |

+---------------------------------------------------------+

1 row in set (0.00 sec)

## DATE\_FORMAT(date,format)

Formats the date value according to the format string.

The following specifiers may be used in the format string. The '%' character is required before format specifier characters.

|  |  |
| --- | --- |
| **Specifier** | **Description** |
| %a | Abbreviated weekday name (Sun..Sat) |
| %b | Abbreviated month name (Jan..Dec) |
| %c | Month, numeric (0..12) |
| %D | Day of the 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) |
| %I | Hour (01..12) |
| %i | Minutes, numeric (00..59) |
| %j | Day of 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, numeric, four digits; used with %V |
| %x | Year for the week, where Monday is the first day of the week, numeric, four digits; used with %v |
| %Y | Year, numeric, four digits |
| %y | Year, numeric (two digits) |
| %% | A literal .%. character |
| %x | x, for any.x. not listed above |

mysql> SELECT DATE\_FORMAT('1997-10-04 22:23:00', '%W %M %Y');

+---------------------------------------------------------+

| DATE\_FORMAT('1997-10-04 22:23:00', '%W %M %Y') |

+---------------------------------------------------------+

| Saturday October 1997 |

+---------------------------------------------------------+

1 row in set (0.00 sec)

mysql> SELECT DATE\_FORMAT('1997-10-04 22:23:00'

-> '%H %k %I %r %T %S %w');

+---------------------------------------------------------+

| DATE\_FORMAT('1997-10-04 22:23:00....... |

+---------------------------------------------------------+

| 22 22 10 10:23:00 PM 22:23:00 00 6 |

+---------------------------------------------------------+

1 row in set (0.00 sec)

## DATE\_SUB(date,INTERVAL expr unit)

This is similar to DATE\_ADD() function.

## DAY(date)

DAY() is a synonym for DAYOFMONTH().

## DAYNAME(date)

Returns the name of the weekday for date.

mysql> SELECT DAYNAME('1998-02-05');

+---------------------------------------------------------+

| DAYNAME('1998-02-05') |

+---------------------------------------------------------+

| Thursday |

+---------------------------------------------------------+

1 row in set (0.00 sec)

## DAYOFMONTH(date)

Returns the day of the month for date, in the range 0 to 31.

mysql> SELECT DAYOFMONTH('1998-02-03');

+---------------------------------------------------------+

| DAYOFMONTH('1998-02-03') |

+---------------------------------------------------------+

| 3 |

+---------------------------------------------------------+

1 row in set (0.00 sec)

## DAYOFWEEK(date)

Returns the weekday index for date (1 = Sunday, 2 = Monday, ., 7 = Saturday). These index values correspond to the ODBC standard.

mysql> SELECT DAYOFWEEK('1998-02-03');

+---------------------------------------------------------+

|DAYOFWEEK('1998-02-03') |

+---------------------------------------------------------+

| 3 |

+---------------------------------------------------------+

1 row in set (0.00 sec)

## DAYOFYEAR(date)

Returns the day of the year for date, in the range 1 to 366.

mysql> SELECT DAYOFYEAR('1998-02-03');

+---------------------------------------------------------+

| DAYOFYEAR('1998-02-03') |

+---------------------------------------------------------+

| 34 |

+---------------------------------------------------------+

1 row in set (0.00 sec)

## EXTRACT(unit FROM date)

The EXTRACT() function uses the same kinds of unit specifiers as DATE\_ADD() or DATE\_SUB(), but extracts parts from the date rather than performing date arithmetic.

mysql> SELECT EXTRACT(YEAR FROM '1999-07-02');

+---------------------------------------------------------+

| EXTRACT(YEAR FROM '1999-07-02') |

+---------------------------------------------------------+

| 1999 |

+---------------------------------------------------------+

1 row in set (0.00 sec)

mysql> SELECT EXTRACT(YEAR\_MONTH FROM '1999-07-02 01:02:03');

+---------------------------------------------------------+

| EXTRACT(YEAR\_MONTH FROM '1999-07-02 01:02:03') |

+---------------------------------------------------------+

| 199907 |

+---------------------------------------------------------+

1 row in set (0.00 sec)

## FROM\_DAYS(N)

Given a day number N, returns a DATE value.

mysql> SELECT FROM\_DAYS(729669);

+---------------------------------------------------------+

| FROM\_DAYS(729669) |

+---------------------------------------------------------+

| 1997-10-07 |

+---------------------------------------------------------+

1 row in set (0.00 sec)

Use FROM\_DAYS() with caution on old dates. It is not intended for use with values that precede the advent of the Gregorian calendar (1582).

## FROM\_UNIXTIME(unix\_timestamp)

## FROM\_UNIXTIME(unix\_timestamp,format)

Returns a representation of the unix\_timestamp argument as a value in 'YYYY-MM-DD HH:MM:SS' or YYYYMMDDHHMMSS format, depending on whether the function is used in a string or numeric context. The value is expressed in the current time zone. unix\_timestamp is an internal timestamp value such as is produced by the UNIX\_TIMESTAMP() function.

If format is given, the result is formatted according to the format string, which is used the same way as listed in the entry for the DATE\_FORMAT() function.

mysql> SELECT FROM\_UNIXTIME(875996580);

+---------------------------------------------------------+

| FROM\_UNIXTIME(875996580) |

+---------------------------------------------------------+

| 1997-10-04 22:23:00 |

+---------------------------------------------------------+

1 row in set (0.00 sec)

## HOUR(time)

Returns the hour for time. The range of the return value is 0 to 23 for time-of-day values. However, the range of TIME values actually is much larger, so HOUR can return values greater than 23.

mysql> SELECT HOUR('10:05:03');

+---------------------------------------------------------+

| HOUR('10:05:03') |

+---------------------------------------------------------+

| 10 |

+---------------------------------------------------------+

1 row in set (0.00 sec)

## LAST\_DAY(date)

Takes a date or datetime value and returns the corresponding value for the last day of the month. Returns NULL if the argument is invalid.

mysql> SELECT LAST\_DAY('2003-02-05');

+---------------------------------------------------------+

| LAST\_DAY('2003-02-05') |

+---------------------------------------------------------+

| 2003-02-28 |

+---------------------------------------------------------+

1 row in set (0.00 sec)

## LOCALTIME and LOCALTIME()

LOCALTIME and LOCALTIME() are synonyms for NOW().

## LOCALTIMESTAMP and LOCALTIMESTAMP()

LOCALTIMESTAMP and LOCALTIMESTAMP() are synonyms for NOW().

## MAKEDATE(year,dayofyear)

Returns a date, given year and day-of-year values. dayofyear must be greater than 0 or the result is NULL.

mysql> SELECT MAKEDATE(2001,31), MAKEDATE(2001,32);

+---------------------------------------------------------+

| MAKEDATE(2001,31), MAKEDATE(2001,32) |

+---------------------------------------------------------+

| '2001-01-31', '2001-02-01' |

+---------------------------------------------------------+

1 row in set (0.00 sec)

## MAKETIME(hour,minute,second)

Returns a time value calculated from the hour, minute, and second arguments.

mysql> SELECT MAKETIME(12,15,30);

+---------------------------------------------------------+

| MAKETIME(12,15,30) |

+---------------------------------------------------------+

| '12:15:30' |

+---------------------------------------------------------+

1 row in set (0.00 sec)

## MICROSECOND(expr)

Returns the microseconds from the time or datetime expression expr as a number in the range from 0 to 999999.

mysql> SELECT MICROSECOND('12:00:00.123456');

+---------------------------------------------------------+

| MICROSECOND('12:00:00.123456') |

+---------------------------------------------------------+

| 123456 |

+---------------------------------------------------------+

1 row in set (0.00 sec)

## MINUTE(time)

Returns the minute for time, in the range 0 to 59.

mysql> SELECT MINUTE('98-02-03 10:05:03');

+---------------------------------------------------------+

| MINUTE('98-02-03 10:05:03') |

+---------------------------------------------------------+

| 5 |

+---------------------------------------------------------+

1 row in set (0.00 sec)

## MONTH(date)

Returns the month for date, in the range 0 to 12.

mysql> SELECT MONTH('1998-02-03')

+---------------------------------------------------------+

| MONTH('1998-02-03') |

+---------------------------------------------------------+

| 2 |

+---------------------------------------------------------+

1 row in set (0.00 sec)

## MONTHNAME(date)

Returns the full name of the month for date.

mysql> SELECT MONTHNAME('1998-02-05');

+---------------------------------------------------------+

| MONTHNAME('1998-02-05') |

+---------------------------------------------------------+

| February |

+---------------------------------------------------------+

1 row in set (0.00 sec)

## NOW()

Returns the current date and time as a value in 'YYYY-MM-DD HH:MM:SS' or YYYYMMDDHHMMSS format, depending on whether the function is used in a string or numeric context. The value is expressed in the current time zone.

mysql> SELECT NOW();

+---------------------------------------------------------+

| NOW() |

+---------------------------------------------------------+

| 1997-12-15 23:50:26 |

+---------------------------------------------------------+

1 row in set (0.00 sec)

## PERIOD\_ADD(P,N)

Adds N months to period P (in the format YYMM or YYYYMM). Returns a value in the format YYYYMM. Note that the period argument P is not a date value.

mysql> SELECT PERIOD\_ADD(9801,2);

+---------------------------------------------------------+

| PERIOD\_ADD(9801,2) |

+---------------------------------------------------------+

| 199803 |

+---------------------------------------------------------+

1 row in set (0.00 sec)

## PERIOD\_DIFF(P1,P2)

Returns the number of months between periods P1 and P2. P1 and P2 should be in the format YYMM or YYYYMM. Note that the period arguments P1 and P2 are not date values.

mysql> SELECT PERIOD\_DIFF(9802,199703);

+---------------------------------------------------------+

| PERIOD\_DIFF(9802,199703) |

+---------------------------------------------------------+

| 11 |

+---------------------------------------------------------+

1 row in set (0.00 sec)

## QUARTER(date)

Returns the quarter of the year for date, in the range 1 to 4.

mysql> SELECT QUARTER('98-04-01');

+---------------------------------------------------------+

| QUARTER('98-04-01') |

+---------------------------------------------------------+

| 2 |

+---------------------------------------------------------+

1 row in set (0.00 sec)

## SECOND(time)

Returns the second for time, in the range 0 to 59.

mysql> SELECT SECOND('10:05:03');

+---------------------------------------------------------+

| SECOND('10:05:03') |

+---------------------------------------------------------+

| 3 |

+---------------------------------------------------------+

1 row in set (0.00 sec)

## SEC\_TO\_TIME(seconds)

Returns the seconds argument, converted to hours, minutes and seconds, as a value in 'HH:MM:SS' or HHMMSS format, depending on whether the function is used in a string or numeric context.

mysql> SELECT SEC\_TO\_TIME(2378);

+---------------------------------------------------------+

| SEC\_TO\_TIME(2378) |

+---------------------------------------------------------+

| 00:39:38 |

+---------------------------------------------------------+

1 row in set (0.00 sec)

## STR\_TO\_DATE(str,format)

This is the inverse of the DATE\_FORMAT() function. It takes a string str and a format string format. STR\_TO\_DATE() returns a DATETIME value if the format string contains both date and time parts or a DATE or TIME value if the string contains only date or time parts.

mysql> SELECT STR\_TO\_DATE('04/31/2004', '%m/%d/%Y');

+---------------------------------------------------------+

| STR\_TO\_DATE('04/31/2004', '%m/%d/%Y') |

+---------------------------------------------------------+

| 2004-04-31 |

+---------------------------------------------------------+

1 row in set (0.00 sec)

## SUBDATE(date,INTERVAL expr unit) and SUBDATE(expr,days)

When invoked with the INTERVAL form of the second argument, SUBDATE() is a synonym for DATE\_SUB(). For information on the INTERVAL unit argument, see the discussion for DATE\_ADD().

mysql> SELECT DATE\_SUB('1998-01-02', INTERVAL 31 DAY);

+---------------------------------------------------------+

| DATE\_SUB('1998-01-02', INTERVAL 31 DAY) |

+---------------------------------------------------------+

| 1997-12-02 |

+---------------------------------------------------------+

1 row in set (0.00 sec)

mysql> SELECT SUBDATE('1998-01-02', INTERVAL 31 DAY);

+---------------------------------------------------------+

| SUBDATE('1998-01-02', INTERVAL 31 DAY) |

+---------------------------------------------------------+

| 1997-12-02 |

+---------------------------------------------------------+

1 row in set (0.00 sec)

## SUBTIME(expr1,expr2)

SUBTIME() returns expr1 . expr2 expressed as a value in the same format as expr1. expr1 is a time or datetime expression, and expr2 is a time.

mysql> SELECT SUBTIME('1997-12-31 23:59:59.999999',

-> '1 1:1:1.000002');

+---------------------------------------------------------+

| SUBTIME('1997-12-31 23:59:59.999999'... |

+---------------------------------------------------------+

| 1997-12-30 22:58:58.999997 |

+---------------------------------------------------------+

1 row in set (0.00 sec)

## SYSDATE()

Returns the current date and time as a value in 'YYYY-MM-DD HH:MM:SS' or YYYYMMDDHHMMSS format, depending on whether the function is used in a string or numeric context.

mysql> SELECT SYSDATE();

+---------------------------------------------------------+

| SYSDATE() |

+---------------------------------------------------------+

| 2006-04-12 13:47:44 |

+---------------------------------------------------------+

1 row in set (0.00 sec)

## TIME(expr)

Extracts the time part of the time or datetime expression expr and returns it as a string.

mysql> SELECT TIME('2003-12-31 01:02:03');

+---------------------------------------------------------+

| TIME('2003-12-31 01:02:03') |

+---------------------------------------------------------+

| 01:02:03 |

+---------------------------------------------------------+

1 row in set (0.00 sec)

## TIMEDIFF(expr1,expr2)

TIMEDIFF() returns expr1 . expr2 expressed as a time value. expr1 and expr2 are time or date-and-time expressions, but both must be of the same type.

mysql> SELECT TIMEDIFF('1997-12-31 23:59:59.000001',

-> '1997-12-30 01:01:01.000002');

+---------------------------------------------------------+

| TIMEDIFF('1997-12-31 23:59:59.000001'..... |

+---------------------------------------------------------+

| 46:58:57.999999 |

+---------------------------------------------------------+

1 row in set (0.00 sec)

## TIMESTAMP(expr), TIMESTAMP(expr1,expr2)

With a single argument, this function returns the date or datetime expression expr as a datetime value. With two arguments, it adds the time expression expr2 to the date or datetime expression expr1 and returns the result as a datetime value.

mysql> SELECT TIMESTAMP('2003-12-31');

+---------------------------------------------------------+

| TIMESTAMP('2003-12-31') |

+---------------------------------------------------------+

| 2003-12-31 00:00:00 |

+---------------------------------------------------------+

1 row in set (0.00 sec)

## TIMESTAMPADD(unit,interval,datetime\_expr)

Adds the integer expression interval to the date or datetime expression datetime\_expr. The unit for interval is given by the unit argument, which should be one of the following values: FRAC\_SECOND, SECOND, MINUTE, HOUR, DAY, WEEK, MONTH, QUARTER or YEAR.

The unit value may be specified using one of keywords as shown or with a prefix of SQL\_TSI\_. For example, DAY and SQL\_TSI\_DAY both are legal.

mysql> SELECT TIMESTAMPADD(MINUTE,1,'2003-01-02');

+---------------------------------------------------------+

| TIMESTAMPADD(MINUTE,1,'2003-01-02') |

+---------------------------------------------------------+

| 2003-01-02 00:01:00 |

+---------------------------------------------------------+

1 row in set (0.00 sec)

## TIMESTAMPDIFF(unit,datetime\_expr1,datetime\_expr2)

Returns the integer difference between the date or datetime expressions datetime\_expr1 and datetime\_expr2. The unit for the result is given by the unit argument. The legal values for unit are the same as those listed in the description of the TIMESTAMPADD() function.

mysql> SELECT TIMESTAMPDIFF(MONTH,'2003-02-01','2003-05-01');

+---------------------------------------------------------+

| TIMESTAMPDIFF(MONTH,'2003-02-01','2003-05-01') |

+---------------------------------------------------------+

| 3 |

+---------------------------------------------------------+

1 row in set (0.00 sec)

## TIME\_FORMAT(time,format)

This is used like the DATE\_FORMAT() function, but the format string may contain format specifiers only for hours, minutes and seconds.

If the time value contains an hour part that is greater than 23, the %H and %k hour format specifiers produce a value larger than the usual range of 0..23. The other hour format specifiers produce the hour value modulo 12.

mysql> SELECT TIME\_FORMAT('100:00:00', '%H %k %h %I %l');

+---------------------------------------------------------+

| TIME\_FORMAT('100:00:00', '%H %k %h %I %l') |

+---------------------------------------------------------+

| 100 100 04 04 4 |

+---------------------------------------------------------+

1 row in set (0.00 sec)

## TIME\_TO\_SEC(time)

Returns the time argument converted to seconds.

mysql> SELECT TIME\_TO\_SEC('22:23:00');

+---------------------------------------------------------+

| TIME\_TO\_SEC('22:23:00') |

+---------------------------------------------------------+

| 80580 |

+---------------------------------------------------------+

1 row in set (0.00 sec)

## TO\_DAYS(date)

Given a date, returns a day number (the number of days since year 0).

mysql> SELECT TO\_DAYS(950501);

+---------------------------------------------------------+

| TO\_DAYS(950501) |

+---------------------------------------------------------+

| 728779 |

+---------------------------------------------------------+

1 row in set (0.00 sec)

## UNIX\_TIMESTAMP(), UNIX\_TIMESTAMP(date)

If called with no argument, returns a Unix timestamp (seconds since '1970-01-01 00:00:00' UTC) as an unsigned integer. If UNIX\_TIMESTAMP() is called with a date argument, it returns the value of the argument as seconds since '1970-01-01 00:00:00' UTC. date may be a DATE string, a DATETIME string, a TIMESTAMP, or a number in the format YYMMDD or YYYYMMDD.

mysql> SELECT UNIX\_TIMESTAMP();

+---------------------------------------------------------+

| UNIX\_TIMESTAMP() |

+---------------------------------------------------------+

| 882226357 |

+---------------------------------------------------------+

1 row in set (0.00 sec)

mysql> SELECT UNIX\_TIMESTAMP('1997-10-04 22:23:00');

+---------------------------------------------------------+

| UNIX\_TIMESTAMP('1997-10-04 22:23:00') |

+---------------------------------------------------------+

| 875996580 |

+---------------------------------------------------------+

1 row in set (0.00 sec)

## UTC\_DATE, UTC\_DATE()

Returns the current UTC date as a value in 'YYYY-MM-DD' or YYYYMMDD format, depending on whether the function is used in a string or numeric context.

mysql> SELECT UTC\_DATE(), UTC\_DATE() + 0;

+---------------------------------------------------------+

| UTC\_DATE(), UTC\_DATE() + 0 |

+---------------------------------------------------------+

| 2003-08-14, 20030814 |

+---------------------------------------------------------+

1 row in set (0.00 sec)

## UTC\_TIME, UTC\_TIME()

Returns the current UTC time as a value in 'HH:MM:SS' or HHMMSS format, depending on whether the function is used in a string or numeric context.

mysql> SELECT UTC\_TIME(), UTC\_TIME() + 0;

+---------------------------------------------------------+

| UTC\_TIME(), UTC\_TIME() + 0 |

+---------------------------------------------------------+

| 18:07:53, 180753 |

+---------------------------------------------------------+

1 row in set (0.00 sec)

## UTC\_TIMESTAMP, UTC\_TIMESTAMP()

Returns the current UTC date and time as a value in 'YYYY-MM-DD HH:MM:SS' or YYYYMMDDHHMMSS format, depending on whether the function is used in a string or numeric context.

mysql> SELECT UTC\_TIMESTAMP(), UTC\_TIMESTAMP() + 0;

+---------------------------------------------------------+

| UTC\_TIMESTAMP(), UTC\_TIMESTAMP() + 0 |

+---------------------------------------------------------+

| 2003-08-14 18:08:04, 20030814180804 |

+---------------------------------------------------------+

1 row in set (0.00 sec)

## WEEK(date[,mode])

This function returns the week number for date. The two-argument form of WEEK() allows you to specify whether the week starts on Sunday or Monday and whether the return value should be in the range from 0 to 53 or from 1 to 53. If the mode argument is omitted, the value of the default\_week\_format system variable is used

|  |  |  |  |
| --- | --- | --- | --- |
| **Mode** | **First Day of week** | **Range** | **Week 1 is the first week.** |
| 0 | Sunday | 0-53 | with a Sunday in this year |
| 1 | Monday | 0-53 | with more than 3 days this year |
| 2 | Sunday | 1-53 | with a Sunday in this year |
| 3 | Monday | 1-53 | with more than 3 days this year |
| 4 | Sunday | 0-53 | with more than 3 days this year |
| 5 | Monday | 0-53 | with a Monday in this year |
| 6 | Sunday | 1-53 | with more than 3 days this year |
| 7 | Monday | 1-53 | with a Monday in this year |

mysql> SELECT WEEK('1998-02-20');

+---------------------------------------------------------+

| WEEK('1998-02-20') |

+---------------------------------------------------------+

| 7 |

+---------------------------------------------------------+

1 row in set (0.00 sec)

## WEEKDAY(date)

Returns the weekday index for date (0 = Monday, 1 = Tuesday, . 6 = Sunday).

mysql> SELECT WEEKDAY('1998-02-03 22:23:00');

+---------------------------------------------------------+

| WEEKDAY('1998-02-03 22:23:00') |

+---------------------------------------------------------+

| 1 |

+---------------------------------------------------------+

1 row in set (0.00 sec)

## WEEKOFYEAR(date)

Returns the calendar week of the date as a number in the range from 1 to 53. WEEKOFYEAR() is a compatibility function that is equivalent to WEEK(date,3).

mysql> SELECT WEEKOFYEAR('1998-02-20');

+---------------------------------------------------------+

| WEEKOFYEAR('1998-02-20') |

+---------------------------------------------------------+

| 8 |

+---------------------------------------------------------+

1 row in set (0.00 sec)

## YEAR(date)

Returns the year for date, in the range 1000 to 9999, or 0 for the .zero. date.

mysql> SELECT YEAR('98-02-03');

+---------------------------------------------------------+

| YEAR('98-02-03') |

+---------------------------------------------------------+

| 1998 |

+---------------------------------------------------------+

1 row in set (0.00 sec)

## YEARWEEK(date), YEARWEEK(date,mode)

Returns year and week for a date. The mode argument works exactly like the mode argument to WEEK(). The year in the result may be different from the year in the date argument for the first and the last week of the year.

mysql> SELECT YEARWEEK('1987-01-01');

+---------------------------------------------------------+

| YEAR('98-02-03')YEARWEEK('1987-01-01') |

+---------------------------------------------------------+

| 198653 |

+---------------------------------------------------------+

1 row in set (0.00 sec)

Note that the week number is different from what the WEEK() function would return (0) for optional arguments 0 or 1, as WEEK() then returns the week in the context of the given year.

There are RDBMS, which support temporary tables. Temporary Tables are a great feature that lets you store and process intermediate results by using the same selection, update, and join capabilities that you can use with typical SQL Server tables.

The temporary tables could be very useful in some cases to keep temporary data. The most important thing that should be known for temporary tables is that they will be deleted when the current client session terminates.

Temporary tables are available in MySQL version 3.23 onwards. If you use an older version of MySQL than 3.23, you can't use temporary tables, but you can use heap tables.

As stated earlier, temporary tables will only last as long as the session is alive. If you run the code in a PHP script, the temporary table will be destroyed automatically when the script finishes executing. If you are connected to the MySQL database server through the MySQL client program, then the temporary table will exist until you close the client or manually destroy the table.

## Example:

Here is an example showing you usage of temporary table:

mysql> CREATE TEMPORARY TABLE SALESSUMMARY (

-> product\_name VARCHAR(50) NOT NULL

-> , total\_sales DECIMAL(12,2) NOT NULL DEFAULT 0.00

-> , avg\_unit\_price DECIMAL(7,2) NOT NULL DEFAULT 0.00

-> , total\_units\_sold INT UNSIGNED NOT NULL DEFAULT 0

);

Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO SALESSUMMARY

-> (product\_name, total\_sales, avg\_unit\_price, total\_units\_sold)

-> VALUES

-> ('cucumber', 100.25, 90, 2);

mysql> SELECT \* FROM SALESSUMMARY;

+--------------+-------------+----------------+------------------+

| product\_name | total\_sales | avg\_unit\_price | total\_units\_sold |

+--------------+-------------+----------------+------------------+

| cucumber | 100.25 | 90.00 | 2 |

+--------------+-------------+----------------+------------------+

1 row in set (0.00 sec)

When you issue a **SHOW TABLES** command, then your temporary table would not be listed out in the list. Now if you will log out of the MySQL session and then you will issue a SELECT command, then you will find no data available in the database. Even your temporary table would also not exist.

## Dropping Temporary Tables:

By default, all the temporary tables are deleted by MySQL when your database connection gets terminated. Still you want to delete them in between, then you do so by issuing DROP TABLE command.

Following is the example on dropping a temporary table.

mysql> CREATE TEMPORARY TABLE SALESSUMMARY (

-> product\_name VARCHAR(50) NOT NULL

-> , total\_sales DECIMAL(12,2) NOT NULL DEFAULT 0.00

-> , avg\_unit\_price DECIMAL(7,2) NOT NULL DEFAULT 0.00

-> , total\_units\_sold INT UNSIGNED NOT NULL DEFAULT 0

);

Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO SALESSUMMARY

-> (product\_name, total\_sales, avg\_unit\_price, total\_units\_sold)

-> VALUES

-> ('cucumber', 100.25, 90, 2);

mysql> SELECT \* FROM SALESSUMMARY;

+--------------+-------------+----------------+------------------+

| product\_name | total\_sales | avg\_unit\_price | total\_units\_sold |

+--------------+-------------+----------------+------------------+

| cucumber | 100.25 | 90.00 | 2 |

+--------------+-------------+----------------+------------------+

1 row in set (0.00 sec)

mysql> DROP TABLE SALESSUMMARY;

mysql> SELECT \* FROM SALESSUMMARY;

ERROR 1146: Table 'TUTORIALS.SALESSUMMARY' doesn't exist

There may be a situation when you need an exact copy of a table and CREATE TABLE ... SELECT... doesn't suit your purposes because the copy must include the same indexes, default values, and so forth.

If you are using MySQL RDBMS, you can handle this situation by the following steps:

* Use SHOW CREATE TABLE command to get a CREATE TABLE statement that specifies the source table's structure, indexes and all.
* Modify the statement to change the table name to that of the clone table and execute the statement. This way you will have exact clone table.
* Optionally, if you need the table contents copied as well, issue an INSERT INTO ... SELECT statement, too.

## Example:

Try out the following example to create a clone table for **TUTORIALS\_TBL** whose structure is as follows:

## Step 1:

Get complete structure about table.

SQL> SHOW CREATE TABLE TUTORIALS\_TBL \G;

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 1. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Table: TUTORIALS\_TBL

Create Table: CREATE TABLE `TUTORIALS\_TBL` (

`tutorial\_id` int(11) NOT NULL auto\_increment,

`tutorial\_title` varchar(100) NOT NULL default '',

`tutorial\_author` varchar(40) NOT NULL default '',

`submission\_date` date default NULL,

PRIMARY KEY (`tutorial\_id`),

UNIQUE KEY `AUTHOR\_INDEX` (`tutorial\_author`)

) TYPE=MyISAM

1 row in set (0.00 sec)

## Step 2:

Rename this table and create another table.

SQL> CREATE TABLE `CLONE\_TBL` (

-> `tutorial\_id` int(11) NOT NULL auto\_increment,

-> `tutorial\_title` varchar(100) NOT NULL default '',

-> `tutorial\_author` varchar(40) NOT NULL default '',

-> `submission\_date` date default NULL,

-> PRIMARY KEY (`tutorial\_id`),

-> UNIQUE KEY `AUTHOR\_INDEX` (`tutorial\_author`)

-> ) TYPE=MyISAM;

Query OK, 0 rows affected (1.80 sec)

## Step 3:

After executing step 2, you will clone a table in your database. If you want to copy data from old table, then you can do it by using INSERT INTO... SELECT statement.

SQL> INSERT INTO CLONE\_TBL (tutorial\_id,

-> tutorial\_title,

-> tutorial\_author,

-> submission\_date)

-> SELECT tutorial\_id,tutorial\_title,

-> tutorial\_author,submission\_date,

-> FROM TUTORIALS\_TBL;

Query OK, 3 rows affected (0.07 sec)

Records: 3 Duplicates: 0 Warnings: 0

Finally, you will have exact clone table as you wanted to have.

A Subquery or Inner query or Nested query is a query within another SQL query and embedded within the WHERE clause.

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.

Subqueries can be used with the SELECT, INSERT, UPDATE, and DELETE statements along with the operators like =, <, >, >=, <=, IN, BETWEEN etc.

There are a few rules that subqueries must follow:

* Subqueries must be enclosed within parentheses.
* A subquery can have only one column in the SELECT clause, unless multiple columns are in the main query for the subquery to compare its selected columns.
* An ORDER BY cannot be used in a subquery, although the main query can use an ORDER BY. The GROUP BY can be used to perform the same function as the ORDER BY in a subquery.
* Subqueries that return more than one row can only be used with multiple value operators, such as the IN operator.
* The SELECT list cannot include any references to values that evaluate to a BLOB, ARRAY, CLOB, or NCLOB.
* A subquery cannot be immediately enclosed in a set function.
* The BETWEEN operator cannot be used with a subquery; however, the BETWEEN operator can be used within the subquery.

## Subqueries with the SELECT Statement:

Subqueries are most frequently used with the SELECT statement. The basic syntax is as follows:

SELECT column\_name [, column\_name ]

FROM table1 [, table2 ]

WHERE column\_name OPERATOR

(SELECT column\_name [, column\_name ]

FROM table1 [, table2 ]

[WHERE])

## Example:

Consider the CUSTOMERS table having the following records:

+----+----------+-----+-----------+----------+

| ID | NAME | AGE | ADDRESS | SALARY |

+----+----------+-----+-----------+----------+

| 1 | Ramesh | 35 | Ahmedabad | 2000.00 |

| 2 | Khilan | 25 | Delhi | 1500.00 |

| 3 | kaushik | 23 | Kota | 2000.00 |

| 4 | Chaitali | 25 | Mumbai | 6500.00 |

| 5 | Hardik | 27 | Bhopal | 8500.00 |

| 6 | Komal | 22 | MP | 4500.00 |

| 7 | Muffy | 24 | Indore | 10000.00 |

+----+----------+-----+-----------+----------+

Now, let us check following subquery with SELECT statement:

SQL> SELECT \*

FROM CUSTOMERS

WHERE ID IN (SELECT ID

FROM CUSTOMERS

WHERE SALARY > 4500) ;

This would produce the following result:

+----+----------+-----+---------+----------+

| ID | NAME | AGE | ADDRESS | SALARY |

+----+----------+-----+---------+----------+

| 4 | Chaitali | 25 | Mumbai | 6500.00 |

| 5 | Hardik | 27 | Bhopal | 8500.00 |

| 7 | Muffy | 24 | Indore | 10000.00 |

+----+----------+-----+---------+----------+

## Subqueries with the INSERT Statement:

Subqueries also can be used with INSERT statements. The INSERT statement uses the data returned from the subquery to insert into another table. The selected data in the subquery can be modified with any of the character, date or number functions.

The basic syntax is as follows:

INSERT INTO table\_name [ (column1 [, column2 ]) ]

SELECT [ \*|column1 [, column2 ]

FROM table1 [, table2 ]

[ WHERE VALUE OPERATOR ]

## Example:

Consider a table CUSTOMERS\_BKP with similar structure as CUSTOMERS table. Now to copy complete CUSTOMERS table into CUSTOMERS\_BKP, following is the syntax:

SQL> INSERT INTO CUSTOMERS\_BKP

SELECT \* FROM CUSTOMERS

WHERE ID IN (SELECT ID

FROM CUSTOMERS) ;

## Subqueries with the UPDATE Statement:

The subquery can be used in conjunction with the UPDATE statement. Either single or multiple columns in a table can be updated when using a subquery with the UPDATE statement.

The basic syntax is as follows:

UPDATE table

SET column\_name = new\_value

[ WHERE OPERATOR [ VALUE ]

(SELECT COLUMN\_NAME

FROM TABLE\_NAME)

[ WHERE) ]

## Example:

Assuming, we have CUSTOMERS\_BKP table available which is backup of CUSTOMERS table.

Following example updates SALARY by 0.25 times in CUSTOMERS table for all the customers whose AGE is greater than or equal to 27:

SQL> UPDATE CUSTOMERS

SET SALARY = SALARY \* 0.25

WHERE AGE IN (SELECT AGE FROM CUSTOMERS\_BKP

WHERE AGE >= 27 );

This would impact two rows and finally CUSTOMERS table would have the following records:

+----+----------+-----+-----------+----------+

| ID | NAME | AGE | ADDRESS | SALARY |

+----+----------+-----+-----------+----------+

| 1 | Ramesh | 35 | Ahmedabad | 125.00 |

| 2 | Khilan | 25 | Delhi | 1500.00 |

| 3 | kaushik | 23 | Kota | 2000.00 |

| 4 | Chaitali | 25 | Mumbai | 6500.00 |

| 5 | Hardik | 27 | Bhopal | 2125.00 |

| 6 | Komal | 22 | MP | 4500.00 |

| 7 | Muffy | 24 | Indore | 10000.00 |

+----+----------+-----+-----------+----------+

## Subqueries with the DELETE Statement:

The subquery can be used in conjunction with the DELETE statement like with any other statements mentioned above.

The basic syntax is as follows:

DELETE FROM TABLE\_NAME

[ WHERE OPERATOR [ VALUE ]

(SELECT COLUMN\_NAME

FROM TABLE\_NAME)

[ WHERE) ]

## Example:

Assuming, we have CUSTOMERS\_BKP table available which is backup of CUSTOMERS table.

Following example deletes records from CUSTOMERS table for all the customers whose AGE is greater than or equal to 27:

SQL> DELETE FROM CUSTOMERS

WHERE AGE IN (SELECT AGE FROM CUSTOMERS\_BKP

WHERE AGE >= 27 );

This would impact two rows and finally CUSTOMERS table would have the following records:

+----+----------+-----+---------+----------+

| ID | NAME | AGE | ADDRESS | SALARY |

+----+----------+-----+---------+----------+

| 2 | Khilan | 25 | Delhi | 1500.00 |

| 3 | kaushik | 23 | Kota | 2000.00 |

| 4 | Chaitali | 25 | Mumbai | 6500.00 |

| 6 | Komal | 22 | MP | 4500.00 |

| 7 | Muffy | 24 | Indore | 10000.00 |

+----+----------+-----+---------+----------+

A sequence is a set of integers 1, 2, 3, ... that are generated in order on demand. Sequences are frequently used in databases because many applications require each row in a table to contain a unique value, and sequences provide an easy way to generate them.

This chapter describes how to use sequences in MySQL.

## Using AUTO\_INCREMENT column:

The simplest way in MySQL to use sequences is to define a column as AUTO\_INCREMENT and leave rest of the things to MySQL to take care.

## Example:

Try out the following example. This will create table and after that it will insert few rows in this table where it is not required to give record ID because its auto-incremented by MySQL.

mysql> CREATE TABLE INSECT

-> (

-> id INT UNSIGNED NOT NULL AUTO\_INCREMENT,

-> PRIMARY KEY (id),

-> name VARCHAR(30) NOT NULL, # type of insect

-> date DATE NOT NULL, # date collected

-> origin VARCHAR(30) NOT NULL # where collected

);

Query OK, 0 rows affected (0.02 sec)

mysql> INSERT INTO INSECT (id,name,date,origin) VALUES

-> (NULL,'housefly','2001-09-10','kitchen'),

-> (NULL,'millipede','2001-09-10','driveway'),

-> (NULL,'grasshopper','2001-09-10','front yard');

Query OK, 3 rows affected (0.02 sec)

Records: 3 Duplicates: 0 Warnings: 0

mysql> SELECT \* FROM INSECT ORDER BY id;

+----+-------------+------------+------------+

| id | name | date | origin |

+----+-------------+------------+------------+

| 1 | housefly | 2001-09-10 | kitchen |

| 2 | millipede | 2001-09-10 | driveway |

| 3 | grasshopper | 2001-09-10 | front yard |

+----+-------------+------------+------------+

3 rows in set (0.00 sec)

## Obtain AUTO\_INCREMENT Values:

LAST\_INSERT\_ID( ) is a SQL function, so you can use it from within any client that understands how to issue SQL statements. Otherwise PERL and PHP scripts provide exclusive functions to retrieve auto-incremented value of last record.

## PERL Example:

Use the mysql\_insertid attribute to obtain the AUTO\_INCREMENT value generated by a query. This attribute is accessed through either a database handle or a statement handle, depending on how you issue the query. The following example references it through the database handle:

$dbh->do ("INSERT INTO INSECT (name,date,origin)

VALUES('moth','2001-09-14','windowsill')");

my $seq = $dbh->{mysql\_insertid};

## PHP Example:

After issuing a query that generates an AUTO\_INCREMENT value, retrieve the value by calling mysql\_insert\_id( ):

mysql\_query ("INSERT INTO INSECT (name,date,origin)

VALUES('moth','2001-09-14','windowsill')", $conn\_id);

$seq = mysql\_insert\_id ($conn\_id);

## Renumbering an Existing Sequence:

There may be a case when you have deleted many records from a table and you want to resequence all the records. This can be done by using a simple trick but you should be very careful to do so if your table is having join, with other table.

If you determine that resequencing an AUTO\_INCREMENT column is unavoidable, the way to do it is to drop the column from the table, then add it again. The following example shows how to renumber the id values in the insect table using this technique:

mysql> ALTER TABLE INSECT DROP id;

mysql> ALTER TABLE insect

-> ADD id INT UNSIGNED NOT NULL AUTO\_INCREMENT FIRST,

-> ADD PRIMARY KEY (id);

## Starting a Sequence at a Particular Value:

By default, MySQL will start sequence from 1 but you can specify any other number as well at the time of table creation. Following is the example where MySQL will start sequence from 100.

mysql> CREATE TABLE INSECT

-> (

-> id INT UNSIGNED NOT NULL AUTO\_INCREMENT = 100,

-> PRIMARY KEY (id),

-> name VARCHAR(30) NOT NULL, # type of insect

-> date DATE NOT NULL, # date collected

-> origin VARCHAR(30) NOT NULL # where collected

);

Alternatively, you can create the table and then set the initial sequence value with ALTER TABLE.

mysql> ALTER TABLE t AUTO\_INCREMENT = 100;

There may be a situation when you have multiple duplicate records in a table. While fetching such records, it makes more sense to fetch only unique records instead of fetching duplicate records.

The SQL **DISTINCT** keyword, which we already have discussed, is used in conjunction with SELECT statement to eliminate all the duplicate records and fetching only unique records.

## Syntax:

The basic syntax of DISTINCT keyword to eliminate duplicate records is as follows:

SELECT DISTINCT column1, column2,.....columnN

FROM table\_name

WHERE [condition]

## Example:

Consider the CUSTOMERS table having the following records:

+----+----------+-----+-----------+----------+

| ID | NAME | AGE | ADDRESS | SALARY |

+----+----------+-----+-----------+----------+

| 1 | Ramesh | 32 | Ahmedabad | 2000.00 |

| 2 | Khilan | 25 | Delhi | 1500.00 |

| 3 | kaushik | 23 | Kota | 2000.00 |

| 4 | Chaitali | 25 | Mumbai | 6500.00 |

| 5 | Hardik | 27 | Bhopal | 8500.00 |

| 6 | Komal | 22 | MP | 4500.00 |

| 7 | Muffy | 24 | Indore | 10000.00 |

+----+----------+-----+-----------+----------+

First, let us see how the following SELECT query returns duplicate salary records:

SQL> SELECT SALARY FROM CUSTOMERS

ORDER BY SALARY;

This would produce the following result where salary 2000 is coming twice which is a duplicate record from the original table.

+----------+

| SALARY |

+----------+

| 1500.00 |

| 2000.00 |

| 2000.00 |

| 4500.00 |

| 6500.00 |

| 8500.00 |

| 10000.00 |

+----------+

Now, let us use DISTINCT keyword with the above SELECT query and see the result:

SQL> SELECT DISTINCT SALARY FROM CUSTOMERS

ORDER BY SALARY;

This would produce the following result where we do not have any duplicate entry:

+----------+

| SALARY |

+----------+

| 1500.00 |

| 2000.00 |

| 4500.00 |

| 6500.00 |

| 8500.00 |

| 10000.00 |

+----------+

If you take user input through a webpage and insert it into a SQL database, there's a chance that you have left yourself wide open for a security issue known as SQL Injection.

This lesson will teach you how to help prevent this from happening and help you secure your scripts and SQL statements in your server side scripts such as PERL Script.

Injection usually occurs when you ask a user for input, like their name, and instead of a name they give you a SQL statement that you will unknowingly run on your database.

Never trust user provided data, process this data only after validation; as a rule, this is done by pattern matching.

In the example below, the **name** is restricted to alphanumerical chars plus underscore and to a length between 8 and 20 chars (modify these rules as needed).

if (preg\_match("/^\w{8,20}$/", $\_GET['username'], $matches))

{

$result = mysql\_query("SELECT \* FROM CUSTOMERS

WHERE name=$matches[0]");

}

else

{

echo "user name not accepted";

}

To demonstrate the problem, consider this excerpt:

// supposed input

$name = "Qadir'; DELETE FROM CUSTOMERS;";

mysql\_query("SELECT \* FROM CUSTOMSRS WHERE name='{$name}'");

The function call is supposed to retrieve a record from the CUSTOMERS table where the name column matches the name specified by the user. Under normal circumstances, $name would only contain alphanumeric characters and perhaps spaces, such as the string ilia. But here, by appending an entirely new query to $name, the call to the database turns into disaster: the injected DELETE query removes all records from CUSTOMERS.

Fortunately, if you use MySQL, the mysql\_query() function does not permit query stacking or executing multiple SQL queries in a single function call. If you try to stack queries, the call fails.

However, other PHP database extensions, such as SQLite and PostgreSQL, happily perform stacked queries, executing all of the queries provided in one string and creating a serious security problem.

## Preventing SQL Injection:

You can handle all escape characters smartly in scripting languages like PERL and PHP. The MySQL extension for PHP provides the function mysql\_real\_escape\_string() to escape input characters that are special to MySQL.

if (get\_magic\_quotes\_gpc())

{

$name = stripslashes($name);

}

$name = mysql\_real\_escape\_string($name);

mysql\_query("SELECT \* FROM CUSTOMERS WHERE name='{$name}'");

## The LIKE Quandary:

To address the LIKE quandary, a custom escaping mechanism must convert user-supplied '%' and '\_' characters to literals. Use addcslashes(), a function that let's you specify a character range to escape.

$sub = addcslashes(mysql\_real\_escape\_string("%str"), "%\_");

// $sub == \%str\\_

mysql\_query("SELECT \* FROM messages

WHERE subject LIKE '{$sub}%'");

It takes time to become a Database Expert or an expert Database Administrator. This all comes with lot of experience in various database designs and good trainings.

But the following list may be helpful for the beginners to have a nice database performance −

* Use 3BNF database design explained in this tutorial in RDBMS Concepts chapter.
* Avoid number-to-character conversions because numbers and characters compare differently and lead to performance downgrade.
* While using SELECT statement, only fetch whatever information is required and avoid using \* in your SELECT queries because it would load the system unnecessarily.
* Create your indexes carefully on all the tables where you have frequent search operations. Avoid index on the tables where you have less number of search operations and more number of insert and update operations.
* A full-table scan occurs when the columns in the WHERE clause do not have an index associated with them. You can avoid a full-table scan by creating an index on columns that are used as conditions in the WHERE clause of an SQL statement.
* Be very careful of equality operators with real numbers and date/time values. Both of these can have small differences that are not obvious to the eye but that make an exact match impossible, thus preventing your queries from ever returning rows.
* Use pattern matching judiciously. LIKE COL% is a valid WHERE condition, reducing the returned set to only those records with data starting with the string COL. However, COL%Y does not further reduce the returned results set since %Y cannot be effectively evaluated. The effort to do the evaluation is too large to be considered. In this case, the COL% is used, but the %Y is thrown away. For the same reason, a leading wildcard %COL effectively prevents the entire filter from being used.
* Fine tune your SQL queries examining the structure of the queries (and subqueries), the SQL syntax, to discover whether you have designed your tables to support fast data manipulation and written the query in an optimum manner, allowing your DBMS to manipulate the data efficiently.
* For queries that are executed on a regular basis, try to use procedures. A procedure is a potentially large group of SQL statements. Procedures are compiled by the database engine and then executed. Unlike an SQL statement, the database engine need not optimize the procedure before it is executed.
* Avoid using the logical operator OR in a query if possible. OR inevitably slows down nearly any query against a table of substantial size.
* You can optimize bulk data loads by dropping indexes. Imagine the history table with many thousands of rows. That history table is also likely to have one or more indexes. When you think of an index, you normally think of faster table access, but in the case of batch loads, you can benefit by dropping the index(es).
* When performing batch transactions, perform COMMIT at after a fair number of records creation in stead of creating them after every record creation.
* Plan to defragment the database on a regular basis, even if doing so means developing a weekly routine.

## Built-In Tuning Tools

Oracle has many tools for managing SQL statement performance but among them two are very popular. These two tools are −

* **Explain plan** − tool identifies the access path that will be taken when the SQL statement is executed.
* **tkprof** − measures the performance by time elapsed during each phase of SQL statement processing.

If you want to simply measure the elapsed time of a query in Oracle, you can use the SQL\*Plus command SET TIMING ON.

Check your RDBMS documentation for more detail on the above-mentioned tools and defragmenting the database.

<http://www.1keydata.com/sql/sql.html>

**. What is DBMS?**

A Database Management System (DBMS) is a program that controls creation, maintenance and use of a database. DBMS can be termed as File Manager that manages data in a database rather than saving it in file systems.

**2. What is RDBMS?**

RDBMS stands for Relational Database Management System. RDBMS store the data into the collection of tables, which is related by common fields between the columns of the table. It also provides relational operators to manipulate the data stored into the tables.

**Example: SQL Server.**

**3. What is SQL?**

SQL stands for Structured Query Language , and it is used to communicate with the Database. This is a standard language used to perform tasks such as retrieval, updation, insertion and deletion of data from a database.

Standard SQL Commands are Select.

**4. What is a Database?**

Database is nothing but an organized form of data for easy access, storing, retrieval and managing of data. This is also known as structured form of data which can be accessed in many ways.

Example: School Management Database, Bank Management Database.

**5. What are tables and Fields?**

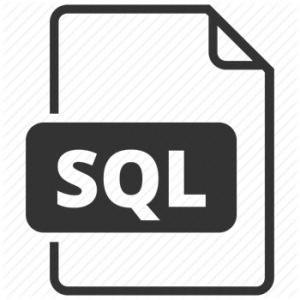
A table is a set of data that are organized in a model with Columns and Rows. Columns can be categorized as vertical, and Rows are horizontal. A table has specified number of column called fields but can have any number of rows which is called record.

Example:.

Table: Employee.

Field: Emp ID, Emp Name, Date of Birth.

Data: 201456, David, 11/15/1960.

[](http://career.guru99.com/wp-content/uploads/2014/07/sql-512.png)

**6. What is a primary key?**

A primary key is a combination of fields which uniquely specify a row. This is a special kind of unique key, and it has implicit NOT NULL constraint. It means, Primary key values cannot be NULL.

**7. What is a unique key?**

A Unique key constraint uniquely identified each record in the database. This provides uniqueness for the column or set of columns.

A Primary key constraint has automatic unique constraint defined on it. But not, in the case of Unique Key.

There can be many unique constraint defined per table, but only one Primary key constraint defined per table.

**8. What is a foreign key?**

A foreign key is one table which can be related to the primary key of another table. Relationship needs to be created between two tables by referencing foreign key with the primary key of another table.

**9. What is a join?**

This is a keyword used to query data from more tables based on the relationship between the fields of the tables. Keys play a major role when JOINs are used.

**10. What are the types of join and explain each?**

There are various types of join which can be used to retrieve data and it depends on the relationship between tables.

**Inner join.**

Inner join return rows when there is at least one match of rows between the tables.

**Right Join.**

Right join return rows which are common between the tables and all rows of Right hand side table. Simply, it returns all the rows from the right hand side table even though there are no matches in the left hand side table.

**Left Join.**

Left join return rows which are common between the tables and all rows of Left hand side table. Simply, it returns all the rows from Left hand side table even though there are no matches in the Right hand side table.

**Full Join.**

Full join return rows when there are matching rows in any one of the tables. This means, it returns all the rows from the left hand side table and all the rows from the right hand side table.

**11. What is normalization?**

Normalization is the process of minimizing redundancy and dependency by organizing fields and table of a database. The main aim of Normalization is to add, delete or modify field that can be made in a single table.

**12. What is Denormalization.**

DeNormalization is a technique used to access the data from higher to lower normal forms of database. It is also process of introducing redundancy into a table by incorporating data from the related tables.

**13. What are all the different normalizations?**

The normal forms can be divided into 5 forms, and they are explained below -.

**First Normal Form (1NF):.**

This should remove all the duplicate columns from the table. Creation of tables for the related data and identification of unique columns.

**Second Normal Form (2NF):.**

Meeting all requirements of the first normal form. Placing the subsets of data in separate tables and Creation of relationships between the tables using primary keys.

**Third Normal Form (3NF):.**

This should meet all requirements of 2NF. Removing the columns which are not dependent on primary key constraints.

**Fourth Normal Form (3NF):.**

Meeting all the requirements of third normal form and it should not have multi- valued dependencies.

**14. What is a View?**

A view is a virtual table which consists of a subset of data contained in a table. Views are not virtually present, and it takes less space to store. View can have data of one or more tables combined, and it is depending on the relationship.

**15. What is an Index?**

An index is performance tuning method of allowing faster retrieval of records from the table. An index creates an entry for each value and it will be faster to retrieve data.

**16. What are all the different types of indexes?**

There are three types of indexes -.

**Unique Index.**

This indexing does not allow the field to have duplicate values if the column is unique indexed. Unique index can be applied automatically when primary key is defined.

**Clustered Index.**

This type of index reorders the physical order of the table and search based on the key values. Each table can have only one clustered index.

**NonClustered Index.**

NonClustered Index does not alter the physical order of the table and maintains logical order of data. Each table can have 999 nonclustered indexes.

**17. What is a Cursor?**

A database Cursor is a control which enables traversal over the rows or records in the table. This can be viewed as a pointer to one row in a set of rows. Cursor is very much useful for traversing such as retrieval, addition and removal of database records.

**18. What is a relationship and what are they?**

Database Relationship is defined as the connection between the tables in a database. There are various data basing relationships, and they are as follows:.

* One to One Relationship.
* One to Many Relationship.
* Many to One Relationship.
* Self-Referencing Relationship.

**19. What is a query?**

A DB query is a code written in order to get the information back from the database. Query can be designed in such a way that it matched with our expectation of the result set. Simply, a question to the Database.

**20. What is subquery?**

A subquery is a query within another query. The outer query is called as main query, and inner query is called subquery. SubQuery is always executed first, and the result of subquery is passed on to the main query.

**21. What are the types of subquery?**

There are two types of subquery – Correlated and Non-Correlated.

A correlated subquery cannot be considered as independent query, but it can refer the column in a table listed in the FROM the list of the main query.

A Non-Correlated sub query can be considered as independent query and the output of subquery are substituted in the main query.

**22. What is a stored procedure?**

Stored Procedure is a function consists of many SQL statement to access the database system. Several SQL statements are consolidated into a stored procedure and execute them whenever and wherever required.

**23. What is a trigger?**

A DB trigger is a code or programs that automatically execute with response to some event on a table or view in a database. Mainly, trigger helps to maintain the integrity of the database.

Example: When a new student is added to the student database, new records should be created in the related tables like Exam, Score and Attendance tables.

**24. What is the difference between DELETE and TRUNCATE commands?**

DELETE command is used to remove rows from the table, and WHERE clause can be used for conditional set of parameters. Commit and Rollback can be performed after delete statement.

TRUNCATE removes all rows from the table. Truncate operation cannot be rolled back.

**25. What are local and global variables and their differences?**

Local variables are the variables which can be used or exist inside the function. They are not known to the other functions and those variables cannot be referred or used. Variables can be created whenever that function is called.

Global variables are the variables which can be used or exist throughout the program. Same variable declared in global cannot be used in functions. Global variables cannot be created whenever that function is called.

**26. What is a constraint?**

Constraint can be used to specify the limit on the data type of table. Constraint can be specified while creating or altering the table statement. Sample of constraint are.

* NOT NULL.
* CHECK.
* DEFAULT.
* UNIQUE.
* PRIMARY KEY.
* FOREIGN KEY.

**27. What is data Integrity?**

Data Integrity defines the accuracy and consistency of data stored in a database. It can also define integrity constraints to enforce business rules on the data when it is entered into the application or database.

**28. What is Auto Increment?**

Auto increment keyword allows the user to create a unique number to be generated when a new record is inserted into the table. AUTO INCREMENT keyword can be used in Oracle and IDENTITY keyword can be used in SQL SERVER.

Mostly this keyword can be used whenever PRIMARY KEY is used.

**29. What is the difference between Cluster and Non-Cluster Index?**

Clustered index is used for easy retrieval of data from the database by altering the way that the records are stored. Database sorts out rows by the column which is set to be clustered index.

A nonclustered index does not alter the way it was stored but creates a complete separate object within the table. It point back to the original table rows after searching.

**30. What is Datawarehouse?**

Datawarehouse is a central repository of data from multiple sources of information. Those data are consolidated, transformed and made available for the mining and online processing. Warehouse data have a subset of data called Data Marts.

**31. What is Self-Join?**

Self-join is set to be query used to compare to itself. This is used to compare values in a column with other values in the same column in the same table. ALIAS ES can be used for the same table comparison.

**32. What is Cross-Join?**

Cross join defines as Cartesian product where number of rows in the first table multiplied by number of rows in the second table. If suppose, WHERE clause is used in cross join then the query will work like an INNER JOIN.

**33. What is user defined functions?**

User defined functions are the functions written to use that logic whenever required. It is not necessary to write the same logic several times. Instead, function can be called or executed whenever needed.

**34. What are all types of user defined functions?**

Three types of user defined functions are.

* Scalar Functions.
* Inline Table valued functions.
* Multi statement valued functions.

Scalar returns unit, variant defined the return clause. Other two types return table as a return.

**35. What is collation?**

Collation is defined as set of rules that determine how character data can be sorted and compared. This can be used to compare A and, other language characters and also depends on the width of the characters.

ASCII value can be used to compare these character data.

**36. What are all different types of collation sensitivity?**

Following are different types of collation sensitivity -.

* Case Sensitivity – A and a and B and b.
* Accent Sensitivity.
* Kana Sensitivity – Japanese Kana characters.
* Width Sensitivity – Single byte character and double byte character.

**37. Advantages and Disadvantages of Stored Procedure?**

Stored procedure can be used as a modular programming – means create once, store and call for several times whenever required. This supports faster execution instead of executing multiple queries. This reduces network traffic and provides better security to the data.

Disadvantage is that it can be executed only in the Database and utilizes more memory in the database server.

**38. What is Online Transaction Processing (OLTP)?**

Online Transaction Processing or OLTP manages transaction based applications which can be used for data entry and easy retrieval processing of data. This processing makes like easier on simplicity and efficiency. It is faster, more accurate results and expenses with respect to OTLP.

Example – Bank Transactions on a daily basis.

**39. What is CLAUSE?**

SQL clause is defined to limit the result set by providing condition to the query. This usually filters some rows from the whole set of records.

Example – Query that has WHERE condition

Query that has HAVING condition.

**40. What is recursive stored procedure?**

A stored procedure which calls by itself until it reaches some boundary condition. This recursive function or procedure helps programmers to use the same set of code any number of times.

**41. What is Union, minus and Interact commands?**

UNION operator is used to combine the results of two tables, and it eliminates duplicate rows from the tables.

MINUS operator is used to return rows from the first query but not from the second query. Matching records of first and second query and other rows from the first query will be displayed as a result set.

INTERSECT operator is used to return rows returned by both the queries.

**42. What is an ALIAS command?**

ALIAS name can be given to a table or column. This alias name can be referred in WHERE clause to identify the table or column.

Example-.



|  |  |
| --- | --- |
| 1 | Select st.StudentID, Ex.Result from student st, Exam as Ex where st.studentID = Ex. StudentID |

Here, st refers to alias name for student table and Ex refers to alias name for exam table.

**43. What is the difference between TRUNCATE and DROP statements?**

TRUNCATE removes all the rows from the table, and it cannot be rolled back. DROP command removes a table from the database and operation cannot be rolled back.

**44. What are aggregate and scalar functions?**

Aggregate functions are used to evaluate mathematical calculation and return single values. This can be calculated from the columns in a table. Scalar functions return a single value based on the input value.

Example -.

Aggregate – max(), count – Calculated with respect to numeric.

Scalar – UCASE(), NOW() – Calculated with respect to strings.

**45. How can you create an empty table from an existing table?**

Example will be -.



|  |  |
| --- | --- |
| 1 | Select \* into studentcopy from student where 1=2 |

Here, we are copying student table to another table with the same structure with no rows copied.

**46. How to fetch common records from two tables?**

Common records result set can be achieved by -.



|  |  |
| --- | --- |
| 1 | Select studentID from student. <strong>INTERSECT </strong> Select StudentID from Exam |

**47. How to fetch alternate records from a table?**

Records can be fetched for both Odd and Even row numbers -.

To display even numbers-.



|  |  |
| --- | --- |
| 1 | Select studentId from (Select rowno, studentId from student) where mod(rowno,2)=0 |

To display odd numbers-.



|  |  |
| --- | --- |
| 1 | Select studentId from (Select rowno, studentId from student) where mod(rowno,2)=1 |

from (Select rowno, studentId from student) where mod(rowno,2)=1.[/sql]

**48. How to select unique records from a table?**

Select unique records from a table by using DISTINCT keyword.



|  |  |
| --- | --- |
| 1 | Select DISTINCT StudentID, StudentName from Student. |

**49. What is the command used to fetch first 5 characters of the string?**

There are many ways to fetch first 5 characters of the string -.



|  |  |
| --- | --- |
| 1 | Select SUBSTRING(StudentName,1,5) as studentname from student |



|  |  |
| --- | --- |
| 1 | Select RIGHT(Studentname,5) as studentname from student |

**50. Which operator is used in query for pattern matching?**

LIKE operator is used for pattern matching, and it can be used as -.

1. % – Matches zero or more characters.
2. \_(Underscore) – Matching exactly one character.

**Example -.**



|  |  |
| --- | --- |
| 1 | Select \* from Student where studentname like ‘a%’ |



|  |  |
| --- | --- |
| 1 | Select \* from Student where studentname like ‘ami\_’ |

## nterview Questions and answers on Database Basics

**1. What is DBMS ?**

The database management system is a collection of programs that enables user to store, retrieve, update and delete information from a database.

**2. What is RDBMS ?**

Relational Database Management system (RDBMS) is a database management system (DBMS) that is based on the relational model. Data from relational database can be accessed or reassembled in many different ways without having to reorganize the database tables. Data from relational database can be accessed using an API , Structured Query Language (SQL).

**3. What is SQL ?**

Structured Query Language(SQL) is a language designed specifically for communicating with databases. SQL is an ANSI (American National Standards Institute) standard.

**4. What are the different type of SQL's statements ?**

This is one of the most frequently asked SQL Interview Questions for freshers. SQL statements are broadly classified into three. They are

**1. DDL – Data Definition Language**

DDL is used to define the structure that holds the data. For example, Create, Alter, Drop and Truncate table.

**2. DML – Data Manipulation Language**

DML is used for manipulation of the data itself. Typical operations are Insert, Delete, Update and retrieving the data from the table. The Select statement is considered as a limited version of the DML, since it can't change the data in the database. But it can perform operations on data retrieved from the DBMS, before the results are returned to the calling function.  
  
**3. DCL – Data Control Language**   
DCL is used to control the visibility of data like granting database access and set privileges to create tables, etc. Example - Grant, Revoke access permission to the user to access data in the database.

**5. What are the Advantages of SQL ?**

1. **SQL is not a proprietary language** used by specific database vendors. Almost every major DBMS supports SQL, so learning this one language will enable programmers to interact with any database like ORACLE, SQL ,MYSQL etc.

2. **SQL is easy to learn**. The statements are all made up of descriptive English words, and there aren't that many of them.

3. SQL is actually a very powerful language and by using its language elements you can perform very **complex and sophisticated database operations**.

**6. what is a field in a database ?**

A field is an area within a record reserved for a specific piece of data.   
**Examples**: Employee Name, Employee ID, etc.



**Also Read**

1. [Top SQL Query Interview Questions and Answers](http://a4academics.com/interview-questions/53-database-and-sql/397-top-100-database-sql-interview-questions-and-answers-examples-queries)    
  
2. [SQL Tutorial](http://a4academics.com/tutorials/24-sql-tutorial)

**7. What is a Record in a database ?**

A record is the collection of values / fields of a specific entity: i.e. an Employee, Salary etc.

**8. What is a Table in a database ?**

A table is a collection of records of a specific type. For example, employee table, salary table etc.

**9. What is a database transaction?**

Database transaction takes database from one consistent state to another. At the end of the transaction the system must be in the prior state if the transaction fails or the status of the system should reflect the successful completion if the transaction goes through.

**10. What are properties of a transaction?**

Expect this SQL Interview Questions as a part of an any interview, irrespective of your experience. Properties of the transaction can be summarized as ACID Properties.

**1. Atomicity**

A transaction consists of many steps. When all the steps in a transaction get completed, it will get reflected in DB or if any step fails, all the transactions are rolled back.

**2. Consistency**

The database will move from one consistent state to another, if the transaction succeeds and remain in the original state, if the transaction fails.

**3. Isolation**

Every transaction should operate as if it is the only transaction in the system.

**4. Durability**

Once a transaction has completed successfully, the updated rows/records must be available for all other transactions on a permanent basis.

**11. What is a Database Lock ?**

Database lock tells a transaction, if the data item in questions is currently being used by other transactions.

**12. What are the type of locks ?**

**1. Shared Lock**

When a shared lock is applied on data item, other transactions can only read the item, but can't write into it.

**2. Exclusive Lock**

When an exclusive lock is applied on data item, other transactions can't read or write into the data item.

## Database Normalization Interview Questions

**13. What are the different type of normalization?**

In database design, we start with one single table, with all possible columns. A lot of redundant data would be present since it’s a single table. **The process of removing the redundant data, by splitting up the table in a well defined fashion is called normalization.**

**1. First Normal Form (1NF)**

A relation is said to be in first normal form if and only if all underlying domains contain atomic values only. After 1NF, we can still have redundant data.

**2. Second Normal Form (2NF)**

A relation is said to be in 2NF if and only if it is in 1NF and every non key attribute is fully dependent on the primary key. After 2NF, we can still have redundant data.

**3. Third Normal Form (3NF)**

A relation is said to be in 3NF, if and only if it is in 2NF and every non key attribute is non-transitively dependent on the primary key.

## Database Keys and Constraints SQL Interview Questions

**14. What is a primary key?**

A primary key is a column whose values **uniquely identify every row** in a table. Primary key values can never be reused. If a row is deleted from the table, its primary key may not be assigned to any new rows in the future. To define a field as primary key, following conditions had to be met :

1. No two rows can have the same primary key value.

2. Every row must have a primary key value.

3. The primary key field cannot be null.

4. Value in a primary key column can never be modified or updated, if any foreign key refers to that primary key.

**15. What is a Composite Key ?**

A Composite primary key is a type of candidate key, which represents a set of columns whose values uniquely identify every row in a table.  
 **For example -** if "Employee\_ID" and "Employee Name" in a table is combined to uniquely identify a row its called a Composite Key.

**16. What is a Composite Primary Key ?**

A Composite primary key is a set of columns whose values uniquely identify every row in a table. What it means is that, a table which contains composite primary key will be indexed based on the columns specified in the primary key. This key will be referred in Foreign Key tables.  
  
**For example** - if the combined effect of columns, "Employee\_ID" and "Employee Name" in a table is required to uniquely identify a row, its called a Composite Primary Key. In this case, both the columns will be represented as primary key.

**17. What is a Foreign Key ?**

When a "one" table's primary key field is added to a related "many" table in order to create the common field which relates the two tables, it is called a foreign key in the "many" table.  
  
For example, the salary of an employee is stored in salary table. The relation is established via foreign key column “Employee\_ID\_Ref” which refers “Employee\_ID” field in the Employee table.

**18. What is a Unique Key ?**

Unique key is same as primary with the difference being the existence of null. Unique key field allows one value as NULL value.

## SQL Insert, Update and Delete Commands Interview Questions

**19. Define SQL Insert Statement ?**

SQL INSERT statement is used to add rows to a table. For a full row insert, SQL Query should start with “insert into “ statement followed by table name and values command, followed by the values that need to be inserted into the table. The insert can be used in several ways:

1. To insert a single complete row.

2. To insert a single partial row.

**20. Define SQL Update Statement ?**

SQL Update is used to update data in a row or set of rows specified in the filter condition.

The basic format of an SQL UPDATE statement is, Update command followed by table to be updated and SET command followed by column names and their new values followed by filter condition that determines which rows should be updated.

**21. Define SQL Delete Statement ?**

SQL Delete is used to delete a row or set of rows specified in the filter condition.  
  
The basic format of an SQL DELETE statement is, DELETE FROM command followed by table name followed by filter condition that determines which rows should be updated.

**2. What are wild cards used in database for Pattern Matching ?**

**SQL Like** operator is used for pattern matching. SQL 'Like' command takes more time to process. So before using "like" operator, consider suggestions given below on when and where to use wild card search.   
  
1) Don't overuse wild cards. If another search operator will do, use it instead.   
2) When you do use wild cards, try not to use them at the beginning of the search pattern, unless absolutely necessary. Search patterns that begin with wild cards are the slowest to process.   
3) Pay careful attention to the placement of the wild card symbols. If they are misplaced, you might not return the data you intended.

## SQL Joins Interview Questions and answers

**23. Define Join and explain different type of joins?**

Another frequently asked SQL Interview Questions on Joins. In order to avoid data duplication, data is stored in related tables. **Join** keyword is used to fetch data from related tables. "Join" return rows when there is at least one match in both table. Type of joins are

**Right Join**

Return all rows from the right table, even if there are no matches in the left table.

**Outer Join**

**Left Join**

Return all rows from the left table, even if there are no matches in the right table.

**Full Join**

Return rows when there is a match in one of the tables.

**24. What is Self-Join?**

Self-join is query used to **join a table to itself**. Aliases should be used for the same table comparison.

**25. What is Cross Join?**

Cross Join will return all records where each row from the first table is combined with each row from the second table.

## Database Views Interview Questions

**26. What is a view?**

The views are virtual tables. Unlike tables that contain data, views simply contain queries that dynamically retrieve data when used.

**27. What is a materialized view?**

Materialized views are also a view but are disk based. **Materialized views** get updates on specific duration, base upon the interval specified in the query definition. We can index materialized view.

**28. What are the advantages and disadvantages of views in a database?**

**Advantages**:

1. Views don't store data in a physical location.

2. The view can be used to hide some of the columns from the table.

3. Views can provide Access Restriction, since data insertion, update and deletion is not possible with the view.

**Disadvantages**:

1. When a table is dropped, associated view become irrelevant.

2. Since the view is created when a query requesting data from view is triggered, its a bit slow.

3. When views are created for large tables, it occupies more memory.

**29. What is a stored procedure?**

Stored Procedure is a function which contains a collection of SQL Queries. The procedure can take inputs , process them and send back output.

**30. What are the advantages of a stored procedure?**

Stored Procedures are precomplied and stored in the database. This enables the database to execute the queries much faster. Since many queries can be included in a stored procedure, round trip time to execute multiple queries from source code to database and back is avoided.

**31. What is a trigger?**

Database triggers are sets of commands that get executed when an event(Before Insert, After Insert, On Update, On delete of a row) occurs on a table, views.

**32. Explain the difference between DELETE , TRUNCATE and DROP commands?**

Once **delete operation** is performed, Commit and Rollback can be performed to retrieve data.  
  
Once the **truncate** statement is executed, Commit and Rollback statement cannot be performed. Where condition can be used along with delete statement but it can't be used with truncate statement.  
  
**Drop** command is used to drop the table or keys like primary,foreign from a table.

**33. What is the difference between Cluster and Non cluster Index?**

A **clustered index** reorders the way records in the table are physically stored. There can be only one clustered index per table. It makes data retrieval faster.  
  
A **non clustered index** does not alter the way it was stored but creates a completely separate object within the table. As a result insert and update command will be faster.

**34. What is Union, minus and Interact commands?**

MINUS operator is used to return rows from the first query but not from the second query. INTERSECT operator is used to return rows returned by both the queries.

**able Name : Employee**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Employee\_id** | **First\_name** | **Last\_name** | **Salary** | **Joining\_date** | **Department** |
| 1 | John | Abraham | 1000000 | 01-JAN-13 12.00.00 AM | Banking |
| 2 | Michael | Clarke | 800000 | 01-JAN-13 12.00.00 AM | Insurance |
| 3 | Roy | Thomas | 700000 | 01-FEB-13 12.00.00 AM | Banking |
| 4 | Tom | Jose | 600000 | 01-FEB-13 12.00.00 AM | Insurance |
| 5 | Jerry | Pinto | 650000 | 01-FEB-13 12.00.00 AM | Insurance |
| 6 | Philip | Mathew | 750000 | 01-JAN-13 12.00.00 AM | Services |
| 7 | TestName1 | 123 | 650000 | 01-JAN-13 12.00.00 AM | Services |
| 8 | TestName2 | Lname% | 600000 | 01-FEB-13 12.00.00 AM | Insurance |

**Table Name : Incentives**

|  |  |  |
| --- | --- | --- |
| **Employee\_ref\_id** | **Incentive\_date** | **Incentive\_amount** |
| 1 | 01-FEB-13 | 5000 |
| 2 | 01-FEB-13 | 3000 |
| 3 | 01-FEB-13 | 4000 |
| 1 | 01-JAN-13 | 4500 |
| 2 | 01-JAN-13 | 3500 |

## SQL Queries Interview Questions and Answers on "SQL Select"

**1. Get all employee details from the employee table**

Select \* from employee

**2. Get First\_Name,Last\_Name from employee table**

Select first\_name, Last\_Name from employee

**3. Get First\_Name from employee table using alias name “Employee Name”**

Select first\_name Employee Name from employee

**4. Get First\_Name from employee table in upper case**

Select upper(FIRST\_NAME) from EMPLOYEE

**5. Get First\_Name from employee table in lower case**

Select lower(FIRST\_NAME) from EMPLOYEE

**6. Get unique DEPARTMENT from employee table**

select distinct DEPARTMENT from EMPLOYEE

**Don't Miss** - [SQL and Database theory Interview Questions](http://a4academics.com/interview-questions/53-database-and-sql/411-sql-interview-questions-and-answers-database)

**7. Select first 3 characters of FIRST\_NAME from EMPLOYEE**

**Oracle Equivalent of SQL Server SUBSTRING is SUBSTR**, Query : select substr(FIRST\_NAME,0,3) from employee  
 **SQL Server Equivalent of Oracle SUBSTR is SUBSTRING**, Query : select substring(FIRST\_NAME,1,3) from employee  
 **MySQL Server Equivalent of Oracle SUBSTR is SUBSTRING**. In MySQL start position is 1, Query : select substring(FIRST\_NAME,1,3) from employee

**8. Get position of 'o' in name 'John' from employee table**

**Oracle Equivalent of SQL Server CHARINDEX is INSTR**, Query : Select instr(FIRST\_NAME,'o') from employee where first\_name='John'  
 **SQL Server Equivalent of Oracle INSTR is CHARINDEX**, Query: Select CHARINDEX('o',FIRST\_NAME,0) from employee where first\_name='John'  
 **MySQL Server Equivalent of Oracle INSTR is LOCATE**, Query: Select LOCATE('o',FIRST\_NAME) from employee where first\_name='John'

**9. Get FIRST\_NAME from employee table after removing white spaces from right side**

select RTRIM(FIRST\_NAME) from employee

**10. Get FIRST\_NAME from employee table after removing white spaces from left side**

select LTRIM(FIRST\_NAME) from employee

**11. Get length of FIRST\_NAME from employee table**

**Oracle,MYSQL Equivalent of SQL Server Len is Length** , Query :select length(FIRST\_NAME) from employee  
 **SQL Server Equivalent of Oracle,MYSQL Length is Len**, Query :select len(FIRST\_NAME) from employee

**12. Get First\_Name from employee table after replacing 'o' with '$'**

select REPLACE(FIRST\_NAME,'o','$') from employee

**13. Get First\_Name and Last\_Name as single column from employee table separated by a '\_'**

**Oracle Equivalent of MySQL concat is '||'**, Query : Select FIRST\_NAME|| '\_' ||LAST\_NAME from EMPLOYEE  
 **SQL Server Equivalent of MySQL concat is '+'**, Query : Select FIRST\_NAME + '\_' +LAST\_NAME from EMPLOYEE  
 **MySQL Equivalent of Oracle '||' is concat**, Query : Select concat(FIRST\_NAME,'\_',LAST\_NAME) from EMPLOYEE

**14. Get FIRST\_NAME ,Joining year,Joining Month and Joining Date from employee table**

**SQL Queries in Oracle**, Select FIRST\_NAME, to\_char(joining\_date,'YYYY') JoinYear , to\_char(joining\_date,'Mon'), to\_char(joining\_date,'dd') from EMPLOYEE  
 **SQL Queries in SQL Server**, select SUBSTRING (convert(varchar,joining\_date,103),7,4) , SUBSTRING (convert(varchar,joining\_date,100),1,3) , SUBSTRING (convert(varchar,joining\_date,100),5,2) from EMPLOYEE  
 **SQL Queries in MySQL**, select year(joining\_date),month(joining\_date), DAY(joining\_date) from EMPLOYEE

**15. Get all employee details from the employee table order by First\_Name Ascending**

Select \* from employee order by FIRST\_NAME asc

**16. Get all employee details from the employee table order by First\_Name descending**

Select \* from employee order by FIRST\_NAME desc

**17. Get all employee details from the employee table order by First\_Name Ascending and Salary descending**

Select \* from employee order by FIRST\_NAME asc,SALARY desc

## "SQL Where Condition" Interview Questions

**18. Get employee details from employee table whose employee name is “John”**

Select \* from EMPLOYEE where FIRST\_NAME='John'

**19. Get employee details from employee table whose employee name are “John” and “Roy”**

Select \* from EMPLOYEE where FIRST\_NAME in ('John','Roy')

**20. Get employee details from employee table whose employee name are not “John” and “Roy”**

Select \* from EMPLOYEE where FIRST\_NAME not in ('John','Roy')

## "SQL Wild Card Search" Interview Questions

**21. Get employee details from employee table whose first name starts with 'J'**

Select \* from EMPLOYEE where FIRST\_NAME like 'J%'

**22. Get employee details from employee table whose first name contains 'o'**

Select \* from EMPLOYEE where FIRST\_NAME like '%o%'

**23. Get employee details from employee table whose first name ends with 'n'**

Select \* from EMPLOYEE where FIRST\_NAME like '%n'

## "SQL Pattern Matching" Interview Questions

**24. Get employee details from employee table whose first name ends with 'n' and name contains 4 letters**

Select \* from EMPLOYEE where FIRST\_NAME like '\_\_\_n' (Underscores)

**25. Get employee details from employee table whose first name starts with 'J' and name contains 4 letters**

Select \* from EMPLOYEE where FIRST\_NAME like 'J\_\_\_' (Underscores)

**26. Get employee details from employee table whose Salary greater than 600000**

Select \* from EMPLOYEE where Salary >600000

**27. Get employee details from employee table whose Salary less than 800000**

Select \* from EMPLOYEE where Salary <800000

**28. Get employee details from employee table whose Salary between 500000 and 800000**

Select \* from EMPLOYEE where Salary between 500000 and 800000

**29. Get employee details from employee table whose name is 'John' and 'Michael'**

Select \* from EMPLOYEE where FIRST\_NAME in ('John','Michael')

**30. Get employee details from employee table whose joining year is “2013”**

**SQL Queries in Oracle**, Select \* from EMPLOYEE where to\_char(joining\_date,'YYYY')='2013'  
  
**SQL Queries in SQL Server**, Select \* from EMPLOYEE where SUBSTRING(convert(varchar,joining\_date,103),7,4)='2013'  
  
**SQL Queries in MySQL**, Select \* from EMPLOYEE where year(joining\_date)='2013'

**31. Get employee details from employee table whose joining month is “January”**

**SQL Queries in Oracle**, Select \* from EMPLOYEE where to\_char(joining\_date,'MM')='01' or Select \* from EMPLOYEE where to\_char(joining\_date,'Mon')='Jan'  
  
**SQL Queries in SQL Server**, Select \* from EMPLOYEE where SUBSTRING(convert(varchar,joining\_date,100),1,3)='Jan'  
  
**SQL Queries in MySQL**, Select \* from EMPLOYEE where month(joining\_date)='01'

**32. Get employee details from employee table who joined before January 1st 2013**

**SQL Queries in Oracle**, Select \* from EMPLOYEE where JOINING\_DATE <to\_date('01/01/2013','dd/mm/yyyy')  
  
**SQL Queries in SQL Server** (Format - “MM/DD/YYYY”), Select \* from EMPLOYEE where joining\_date <'01/01/2013'  
  
**SQL Queries in MySQL** (Format - “YYYY-DD-MM”), Select \* from EMPLOYEE where joining\_date <'2013-01-01'

**33. Get employee details from employee table who joined after January 31st**

**SQL Queries in Oracle**, Select \* from EMPLOYEE where JOINING\_DATE >to\_date('31/01/2013','dd/mm/yyyy')  
  
**SQL Queries in SQL Server and MySQL** (Format - “MM/DD/YYYY”), Select \* from EMPLOYEE where joining\_date >'01/31/2013'  
  
**SQL Queries in MySQL** (Format - “YYYY-DD-MM”), Select \* from EMPLOYEE where joining\_date >'2013-01-31'

**35. Get Joining Date and Time from employee table**

**SQL Queries in Oracle**, select to\_char(JOINING\_DATE,'dd/mm/yyyy hh:mi:ss') from EMPLOYEE  
  
**SQL Queries in SQL Server**, Select convert(varchar(19),joining\_date,121) from EMPLOYEE  
  
**SQL Queries in MySQL**, Select CONVERT(DATE\_FORMAT(joining\_date,'%Y-%m-%d-%H:%i:00'),DATETIME) from EMPLOYEE

**36. Get Joining Date,Time including milliseconds from employee table**

**SQL Queries in Oracle**, select to\_char(JOINING\_DATE,'dd/mm/yyyy HH:mi:ss.ff') from EMPLOYEE . Column Data Type should be “TimeStamp”  
  
**SQL Queries in SQL Server**, select convert(varchar,joining\_date,121) from EMPLOYEE  
  
**SQL Queries in MySQL**, Select MICROSECOND(joining\_date) from EMPLOYEE

**37. Get difference between JOINING\_DATE and INCENTIVE\_DATE from employee and incentives table**

Select FIRST\_NAME,INCENTIVE\_DATE - JOINING\_DATE from employee a inner join incentives B on A.EMPLOYEE\_ID=B.EMPLOYEE\_REF\_ID

**38. Get database date**

**SQL Queries in Oracle**, select sysdate from dual  
  
**SQL Queries in SQL Server**, select getdate()  
  
**SQL Query in MySQL**, select now()

**39. Get names of employees from employee table who has '%' in Last\_Name. Tip : Escape character for special characters in a query.**

**SQL Queries in Oracle**, Select FIRST\_NAME from employee where Last\_Name like '%?%%'

**SQL Queries in SQL Server**, Select FIRST\_NAME from employee where Last\_Name like '%[%]%'

**SQL Queries in MySQL**, Select FIRST\_NAME from employee where Last\_Name like '%\%%'

**40. Get Last Name from employee table after replacing special character with white space**

**SQL Queries in Oracle**, Select translate(LAST\_NAME,'%',' ') from employee  
  
**SQL Queries in SQL Server and MySQL**, Select REPLACE(LAST\_NAME,'%',' ') from employee

## "SQL Group By Query" Interview Questions and Answers

**41. Get department,total salary with respect to a department from employee table.**

Select DEPARTMENT,sum(SALARY) Total\_Salary from employee group by department

**42. Get department,total salary with respect to a department from employee table order by total salary descending**

Select DEPARTMENT,sum(SALARY) Total\_Salary from employee group by DEPARTMENT order by Total\_Salary descending

## SQL Queries Interview Questions and Answers on "SQL Mathematical Operations using Group By"

**43. Get department,no of employees in a department,total salary with respect to a department from employee table order by total salary descending**

Select DEPARTMENT,count(FIRST\_NAME),sum(SALARY) Total\_Salary from employee group by DEPARTMENT order by Total\_Salary descending

**44. Get department wise average salary from employee table order by salary ascending**

select DEPARTMENT,avg(SALARY) AvgSalary from employee group by DEPARTMENT order by AvgSalary asc

**45. Get department wise maximum salary from employee table order by salary ascending**

select DEPARTMENT,max(SALARY) MaxSalary from employee group by DEPARTMENT order by MaxSalary asc

**46. Get department wise minimum salary from employee table order by salary ascending**

select DEPARTMENT,min(SALARY) MinSalary from employee group by DEPARTMENT order by MinSalary asc

**47. Select no of employees joined with respect to year and month from employee table**

**SQL Queries in Oracle**, select to\_char (JOINING\_DATE,'YYYY') Join\_Year,to\_char (JOINING\_DATE,'MM') Join\_Month,count(\*) Total\_Emp from employee group by to\_char (JOINING\_DATE,'YYYY'),to\_char(JOINING\_DATE,'MM')  
  
**SQL Queries in SQL Server**, select datepart (YYYY,JOINING\_DATE) Join\_Year,datepart (MM,JOINING\_DATE) Join\_Month,count(\*) Total\_Emp from employee group by datepart(YYYY,JOINING\_DATE), datepart(MM,JOINING\_DATE)  
  
**SQL Queries in MySQL**, select year (JOINING\_DATE) Join\_Year,month (JOINING\_DATE) Join\_Month,count(\*) Total\_Emp from employee group by year(JOINING\_DATE), month(JOINING\_DATE)

**48. Select department,total salary with respect to a department from employee table where total salary greater than 800000 order by Total\_Salary descending**

Select DEPARTMENT,sum(SALARY) Total\_Salary from employee group by DEPARTMENT having sum(SALARY) >800000 order by Total\_Salary desc

**49. Select employee details from employee table if data exists in incentive table ?**

select \* from EMPLOYEE where exists (select \* from INCENTIVES)

**Explanation** : Here "exists" statement helps us to do the job of If statement. Main query will get executed if the sub query returns at least one row. So we can consider the sub query as "If condition" and the main query as "code block" inside the If condition. We can use any SQL commands (Joins, Group By , having etc) in sub query. This command will be useful in queries which need to detect an event and do some activity.

**50. How to fetch data that are common in two query results ?**

select \* from EMPLOYEE where EMPLOYEE\_ID INTERSECT select \* from EMPLOYEE where EMPLOYEE\_ID < 4

**Explanation** : Here "INTERSECT" command is used to fetch data that are common in 2 queries. In this example, we had taken EMPLOYEE table in both the queries.We can apply INTERSECT command on different tables. The result of the above query will return employee details of "ROY" because, employee id of ROY is 3, and both query results have the information about ROY.

**51. Get Employee ID's of those employees who didn't receive incentives without using sub query ?**

select EMPLOYEE\_ID from EMPLOYEE  
MINUS  
select EMPLOYEE\_REF\_ID from INCENTIVES

**Explanation** : To filter out certain information we use MINUS command. What MINUS Command odes is that, it returns all the results from the first query, that are not part of the second query. In our example, first three employees received the incentives. So query will return employee id's 4 to 8.

**52. Select 20 % of salary from John , 10% of Salary for Roy and for other 15 % of salary from employee table**

SELECT FIRST\_NAME, CASE FIRST\_NAME WHEN 'John' THEN SALARY \* .2 WHEN 'Roy' THEN SALARY \* .10 ELSE SALARY \* .15 END "Deduced\_Amount" FROM EMPLOYEE

**Explanation** : Here, we are using "SQL CASE" statement to achieve the desired results. After case statement, we had to specify the column on which filtering is applied. In our case it is "FIRST\_NAME". And in then condition, specify the name of filter like John, Roy etc. To handle conditions outside our filter, use else block where every one other than John and Roy enters.

**53. Select Banking as 'Bank Dept', Insurance as 'Insurance Dept' and Services as 'Services Dept' from employee table**

SQL Queries in Oracle, SELECT distinct DECODE (DEPARTMENT, 'Banking', 'Bank Dept', 'Insurance', 'Insurance Dept', 'Services', 'Services Dept') FROM EMPLOYEE  
SQL Queries in SQL Server and MySQL, SELECT case DEPARTMENT when 'Banking' then 'Bank Dept' when 'Insurance' then 'Insurance Dept' when 'Services' then 'Services Dept' end FROM EMPLOYEE

**Explanation** : Here "DECODE" keyword is used to specify the alias name. In oracle we had specify, Column Name followed by Actual Name and Alias Name as arguments. In SQL Server and MySQL, we can use the earlier switch case statements for alias names.

**54. Delete employee data from employee table who got incentives in incentive table**

delete from EMPLOYEE where EMPLOYEE\_ID in (select EMPLOYEE\_REF\_ID from INCENTIVES)

**Explanation** : Trick about this question is that we can't delete data from a table based on some condition in another table by joining them. Here to delete multiple entries from EMPLOYEE table, we need to use Subquery. Entries will get deleted based on the result of Subquery.

**55. Insert into employee table Last Name with " ' " (Single Quote - Special Character)**

Tip - Use another single quote before special character  
Insert into employee (LAST\_NAME) values ('Test''')

**56. Select Last Name from employee table which contain only numbers**

Select \* from EMPLOYEE where lower(LAST\_NAME)=upper(LAST\_NAME)

**Explanation** : In order to achieve the desired result, we use "ASCII" property of the database. If we get results for a column using Lower and Upper commands, ASCII of both results will be same for numbers. If there is any alphabets in the column, results will differ.

**57. Write a query to rank employees based on their incentives for a month**

select FIRST\_NAME,INCENTIVE\_AMOUNT,DENSE\_RANK() OVER (PARTITION BY INCENTIVE\_DATE ORDER BY INCENTIVE\_AMOUNT DESC) AS Rank from EMPLOYEE a, INCENTIVES b where a.EMPLOYEE\_ID=b.EMPLOYEE\_REF\_ID

**Explanation** : In order to rank employees based on their rank for a month, "DENSE\_RANK" keyword is used. Here partition by keyword helps us to sort the column with which filtering is done. Rank is provided to the column specified in the order by statement. The above query ranks employees with respect to their incentives for a given month.

**58. Update incentive table where employee name is 'John'**

update INCENTIVES set INCENTIVE\_AMOUNT='9000' where EMPLOYEE\_REF\_ID=(select EMPLOYEE\_ID from EMPLOYEE where FIRST\_NAME='John' )

**Explanation** : We need to join Employee and Incentive Table for updating the incentive amount. But for update statement joining query wont work. We need to use sub query to update the data in the incentive table. SQL Query is as shown below.

**59. Select first\_name, incentive amount from employee and incentives table for those employees who have incentives**

Select FIRST\_NAME,INCENTIVE\_AMOUNT from employee a inner join incentives B on A.EMPLOYEE\_ID=B.EMPLOYEE\_REF\_ID

**60. Select first\_name, incentive amount from employee and incentives table for those employees who have incentives and incentive amount greater than 3000**

Select FIRST\_NAME,INCENTIVE\_AMOUNT from employee a inner join incentives B on A.EMPLOYEE\_ID=B.EMPLOYEE\_REF\_ID and INCENTIVE\_AMOUNT >3000

**61. Select first\_name, incentive amount from employee and incentives table for all employes even if they didn't get incentives**

Select FIRST\_NAME,INCENTIVE\_AMOUNT from employee a left join incentives B on A.EMPLOYEE\_ID=B.EMPLOYEE\_REF\_ID

**62. Select first\_name, incentive amount from employee and incentives table for all employees even if they didn't get incentives and set incentive amount as 0 for those employees who didn't get incentives.**

**SQL Queries in Oracle**, Select FIRST\_NAME,nvl(INCENTIVE\_AMOUNT,0) from employee a left join incentives B on A.EMPLOYEE\_ID=B.EMPLOYEE\_REF\_ID  
  
**SQL Queries in SQL Server**, Select FIRST\_NAME, ISNULL(INCENTIVE\_AMOUNT,0) from employee a left join incentives B on A.EMPLOYEE\_ID=B.EMPLOYEE\_REF\_ID  
  
**SQL Queries in MySQL**, Select FIRST\_NAME, IFNULL(INCENTIVE\_AMOUNT,0) from employee a left join incentives B on A.EMPLOYEE\_ID=B.EMPLOYEE\_REF\_ID

**63. Select first\_name, incentive amount from employee and incentives table for all employees who got incentives using left join**

**SQL Queries in Oracle**, Select FIRST\_NAME,nvl(INCENTIVE\_AMOUNT,0) from employee a right join incentives B on A.EMPLOYEE\_ID=B.EMPLOYEE\_REF\_ID  
  
**SQL Queries in SQL Server**, Select FIRST\_NAME, isnull(INCENTIVE\_AMOUNT,0) from employee a right join incentives B on A.EMPLOYEE\_ID=B.EMPLOYEE\_REF\_ID  
  
**SQL Queries in MySQL**, Select FIRST\_NAME, IFNULL(INCENTIVE\_AMOUNT,0) from employee a right join incentives B on A.EMPLOYEE\_ID=B.EMPLOYEE\_REF\_ID

**64. Select max incentive with respect to employee from employee and incentives table using sub query**

**SQL Queries in Oracle**, select DEPARTMENT,(select nvl(max(INCENTIVE\_AMOUNT),0) from INCENTIVES where EMPLOYEE\_REF\_ID=EMPLOYEE\_ID) Max\_incentive from EMPLOYEE  
  
**SQL Queries in SQL Server**, select DEPARTMENT,(select ISNULL(max(INCENTIVE\_AMOUNT),0) from INCENTIVES where EMPLOYEE\_REF\_ID=EMPLOYEE\_ID) Max\_incentive from EMPLOYEE  
  
**SQL Queries in SQL Server**, select DEPARTMENT,(select IFNULL (max(INCENTIVE\_AMOUNT),0) from INCENTIVES where EMPLOYEE\_REF\_ID=EMPLOYEE\_ID) Max\_incentive from EMPLOYEE

## "Top N Salary" SQL Interview Questions and Answers

**65. Select TOP 2 salary from employee table**

**SQL Queries in Oracle**, select \* from (select \* from employee order by SALARY desc) where rownum <3  
  
**SQL Queries in SQL Server**, select top 2 \* from employee order by salary desc  
  
**SQL Queries in MySQL**, select \* from employee order by salary desc limit 2

**66. Select TOP N salary from employee table**

**SQL Queries in Oracle**, select \* from (select \* from employee order by SALARY desc) where rownum <N + 1  
  
**SQL Queries in SQL Server**, select top N \* from employee  
  
**SQL Queries in MySQL**, select \* from employee order by salary desc limit N

**67. Select 2nd Highest salary from employee table**

**SQL Queries in Oracle**, select min(salary) from (select \* from (select \* from employee order by SALARY desc) where rownum <3)  
  
**SQL Queries in SQL Server**, select min(SALARY) from (select top 2 \* from employee) a  
  
**SQL Queries in MySQL**, select min(SALARY) from (select \* from employee order by salary desc limit 2) a

**68. Select Nth Highest salary from employee table**

**SQL Queries in Oracle**, select min(salary) from (select \* from (select \* from employee order by SALARY desc) where rownum <N + 1)  
  
**SQL Queries in SQL Server**, select min(SALARY) from (select top N \* from employee) a  
  
**SQL Queries in MySQL**, select min(SALARY) from (select \* from employee order by salary desc limit N) a

## "SQL Union" Query Interview Questions

**69. Select First\_Name,LAST\_NAME from employee table as separate rows**

select FIRST\_NAME from EMPLOYEE union select LAST\_NAME from EMPLOYEE

**70. What is the difference between UNION and UNION ALL ?**

Both UNION and UNION ALL is used to select information from structurally similar tables. That means corresponding columns specified in the union should have same data type. For example, in the above query, if FIRST\_NAME is DOUBLE and LAST\_NAME is STRING above query wont work. Since the data type of both the columns are VARCHAR, union is made possible. Difference between UNION and UNION ALL is that , UNION query return only distinct values.

**71. Write create table syntax for employee table**

Oracle -CREATE TABLE EMPLOYEE (  
EMPLOYEE\_ID NUMBER,  
FIRST\_NAME VARCHAR2(20 BYTE),  
LAST\_NAME VARCHAR2(20 BYTE),  
SALARY FLOAT(126),  
JOINING\_DATE TIMESTAMP (6) DEFAULT sysdate,  
DEPARTMENT VARCHAR2(30 BYTE) )  
SQL Server -CREATE TABLE EMPLOYEE(  
EMPLOYEE\_ID int NOT NULL,  
FIRST\_NAME varchar(50) NULL,  
LAST\_NAME varchar(50) NULL,  
SALARY decimal(18, 0) NULL,  
JOINING\_DATE datetime2(7) default getdate(),  
DEPARTMENT varchar(50) NULL)

**72. Write syntax to delete table employee**

DROP table employee;

**73. Write syntax to set EMPLOYEE\_ID as primary key in employee table**

ALTER TABLE EMPLOYEE add CONSTRAINT EMPLOYEE\_PK PRIMARY KEY(EMPLOYEE\_ID)

**74. Write syntax to set 2 fields(EMPLOYEE\_ID,FIRST\_NAME) as primary key in employee table**

ALTER TABLE EMPLOYEE add CONSTRAINT EMPLOYEE\_PK PRIMARY KEY(EMPLOYEE\_ID,FIRST\_NAME)

**75. Write syntax to drop primary key on employee table**

Alter TABLE EMPLOYEE drop CONSTRAINT EMPLOYEE\_PK;

**76. Write Sql Syntax to create EMPLOYEE\_REF\_ID in INCENTIVES table as foreign key with respect to EMPLOYEE\_ID in employee table**

ALTER TABLE INCENTIVES ADD CONSTRAINT INCENTIVES\_FK FOREIGN KEY (EMPLOYEE\_REF\_ID) REFERENCES EMPLOYEE(EMPLOYEE\_ID)

**77. Write SQL to drop foreign key on employee table**

ALTER TABLE INCENTIVES drop CONSTRAINT INCENTIVES\_FK;

**78. Write SQL to create Orcale Sequence**

CREATE SEQUENCE EMPLOYEE\_ID\_SEQ START WITH 0 NOMAXVALUE MINVALUE 0 NOCYCLE NOCACHE NOORDER;

**79. Write Sql syntax to create Oracle Trigger before insert of each row in employee table**

CREATE OR REPLACE TRIGGER EMPLOYEE\_ROW\_ID\_TRIGGER  
BEFORE INSERT ON EMPLOYEE FOR EACH ROW  
DECLARE  
seq\_no number(12);  
BEGIN  
select EMPLOYEE\_ID\_SEQ.nextval into seq\_no from dual ;  
:new EMPLOYEE\_ID :=seq\_no;  
END;  
SHOW ERRORS;

**80. Oracle Procedure81. Oracle View**

An example oracle view script is given below  
create view Employee\_Incentive as select FIRST\_NAME,max(INCENTIVE\_AMOUNT) INCENTIVE\_AMOUNT from EMPLOYEE a, INCENTIVES b where a.EMPLOYEE\_ID=b.EMPLOYEE\_REF\_ID group by FIRST\_NAME

**82. Oracle materialized view - Daily Auto Refresh**

CREATE MATERIALIZED VIEW Employee\_Incentive  
REFRESH COMPLETE  
START WITH SYSDATE  
NEXT SYSDATE + 1 AS  
select FIRST\_NAME,INCENTIVE\_DATE,INCENTIVE\_AMOUNT from EMPLOYEE a, INCENTIVES b   
where a.EMPLOYEE\_ID=b.EMPLOYEE\_REF\_ID

**83. Oracle materialized view - Fast Refresh on Commit**

Create materialized view log for fast refresh. Following materialized view script wont get executed if materialized view log doesn't exists  
  
CREATE MATERIALIZED VIEW MAT\_Employee\_Incentive\_Refresh  
BUILD IMMEDIATE  
REFRESH FAST ON COMMIT AS  
select FIRST\_NAME,max(INCENTIVE\_AMOUNT) from EMPLOYEE a, INCENTIVES b  
where a.EMPLOYEE\_ID=b.EMPLOYEE\_REF\_ID group by FIRST\_NAME

**84. What is SQL Injection ?**

SQL Injection is one of the the techniques uses by hackers to hack a website by injecting SQL commands in data fields.

**1. Compare SQL & PL/SQL**

|  |  |  |
| --- | --- | --- |
| **Criteria** | **SQL** | **PL/SQL** |
| What it is | Single query or command execution | Full programming language |
| What it comprises | Data source for reports, web pages | Application language to build, format and display report, web pages |
| Characteristic | Declarative in nature | Procedural in nature |
| Used for | Manipulating data | Creating applications |

**2. What is BCP? When is it used?**

It is a tool used to duplicate enormous quantity of information from tables and views. It does not facsimile the structures same as foundation to target.  
**BULK INSERT** command helps to bring in a data folder into a record, table or view in a user-specific arrangement.

**3. When is the UPDATE\_STATISTICS command used?**

This command is used, ones the processing of large data is done.  
When we delete a large number of files, alteration or reproduction takes place in the tables, to be concerned of these changes we need to restructure the indexes This is done **UPDATE\_STATISTICS**.

**Get Sql Certification in just 16 Hours**

**GET CERTIFIED**

**4. Explain the steps needed to Create the scheduled job?**

**Steps to create a Scheduled Job :**

1. [Connect to the database of SQL server](https://intellipaat.com/tutorial/oracle-dba-tutorial/oracle-installation/)in SQL Server Management Studio. On the SQL Server Agent, we will find a Jobs folder.
2. Right click on jobs and choose Add New.
3. A New Job window will come into view. Give an associated name for the same.
4. Click next on the “Steps” in the left list of options. An SQL job can have multiple steps either in the form of SQL declaration or a stored practice call.
5. Click on the “Schedules” in the left list of options. An SQL job can comprise of one or supplementary schedules. It is basically the instance at which SQL job will jog itself. We can spell out returning schedules also.

**5. When are we going to use truncate and delete?**

1. TRUNCATE is a DDL command, whereas DELETE is a DML command.
2. We can’t execute a trigger in case of TRUNCATE whilst with DELETE, we can accomplish a trigger.
3. [TRUNCATE is quicker than DELETE](https://intellipaat.com/interview-question/sql-server-interview-questions/), for the reason that when we use DELETE to delete the data, at that time it store the whole statistics in the rollback gap on or after where we can get the data back after removal. In case of TRUNCATE, it will not store data in rollback gap and will unswervingly rub it out. TRUNCATE do not recover the deleted data.
4. We can use any condition in WHERE clause using DELETE but it is not possible with TRUNCATE.5.If a table is referenced by any foreign key constraints, then TRUNCATE won’t work.

**6. Explain correlated query work?**

It’s most important to be attentive of the arrange of operations in an interrelated subquery.  
**First,** a row is processed in the outer doubt.  
**Then,** for that exacting row, the subquery is executed – as a result for each row processed by the outer query, the subquery will also be processed. In correlated subquery, each time a line is worked for Emp1, the subquery will also make a decision on the exacting row’s value for Emp1.Salary and run. And the outer query will move on to the next row, and the subquery will execute for that row’s value of Emp1.Salary.  
It will persist in anticipation of the **“WHERE (1) = (… )”** state is pleased.

[**Wish to Learn SQL? Click Here**](https://intellipaat.com/microsoft-sql-server-certification-training/?utm_source=IQ&utm_campaign=IQ_SQL_CTA&utm_medium=Website#course-content)

**7. When is the Explicit Cursor Used ?**

If the developer needs to perform the row by row operations for the result set containing more than one row, then he unambiguously declares a pointer with a name. They are managed by OPEN, FETCH and CLOSE.%FOUND, %NOFOUND, %ROWCOUNT and %ISOPEN characteristics are used in all types of pointers.

**8. Find What is Wrong in this Query?  
SELECT subject\_code, AVG (marks) FROM students WHERE AVG(marks) > 75 GROUP BY subject\_code; The WHERE clause cannot be used to restrict groups. Instead, the HAVING clause should be used.**

SELECT subject\_code, AVG (marks)

FROM students

HAVING AVG(marks) > 75

GROUP BY subject\_code;

**9. Write the Syntax for STUFF function in an SQL server?**

STUFF (String1, Position, Length, String2)

String1 - String to be overwritten

Position - Starting location for overwriting

Length - Length of substitute string

String2- String to overwrite.

**10. Name some commands that can be used to manipulate text in T-SQL code. For example, a command that obtains only a portion of the text or replace a text string, etc.**

* **CHARINDEX( findTextData, textData, [startingPosition] )** – Returns the starting position of the specified expression in a character string. The starting position is optional.
* **LEFT( character\_expression , integer\_expression )** – Returns the left part of a character string with the specified number of characters.
* **LEN( textData )** – Returns integer value of the length of the string, excluding trailing blanks.
* **LOWER ( character\_expression )** – Returns a character expression after converting uppercase character data to lowercase.
* **LTRIM( textData)** – Removes leading blanks. PATINDEX( findTextData, textData ) – Returns integer value of the starting position of text found in the string.
* **REPLACE( textData, findTextData, replaceWithTextData )** – Replaces occurrences of text found in the string with a new value.
* **REPLICATE( character\_expression , integer\_expression )** – Repeats a character expression for a specified number of times.
* **REVERSE( character\_expression )** – Returns the reverse of a character expression.
* **RTRIM( textData)** – Removes trailing blanks. SPACE( numberOfSpaces ) – Repeats space value specified number of times.
* **STUFF( textData, start , length , insertTextData )** – Deletes a specified length of characters and inserts another set of characters at a specified starting point.
* **SUBSTRING( textData, startPosition, length )** – Returns portion of the string.
* **UPPER( character\_expression )** – Returns a character expression with lowercase character data converted to uppercase.

**11. What are the three ways that Dynamic SQL can be executed?**

* Writing a query with parameters.
* Using EXEC.
* Using sp\_executesql.

**Download Sql Interview questions asked by top MNCs in 2017 ?**

Top of Form



Bottom of Form

**12. In what version of SQL Server were synonyms released? How do synonyms work and explain its use cases? Synonyms were released with SQL Server 2005.**

* Synonyms enable the reference of another object (View, Table, Stored Procedure or Function) potentially on a different server, database or schema in your environment. In simple words, the original object that is referenced in the whole code is using a completely different underlying object, but no coding changes are necessary. Think of this as an alias as a means to simplify migrations and application testing without the need to make any dependent coding changes.
* Synonyms can offer a great deal of value when converting underlying database objects without breaking front end or middle tier code. This could be useful during a re-architecture or upgrade project.

**13. If you are a SQL Developer, how can you delete duplicate records in a table with no primary key?**

Use the SET ROWCOUNT command. For instance,  
if you have 2 duplicate rows, you would SET ROWCOUNT 1, execute DELETE command and then SET ROWCOUNT 0.

**14. Is it possible to import data directly from T-SQL commands without using SQL Server Integration Services? If so, what are the commands?**

**Yes, six commands are available to import data directly in the T-SQL language. These commands include :**

* **BCP :** The bulk copy (bcp) command of Microsoft SQL Server provides you with the ability to insert large numbers of records directly from the command line. In addition to being a great tool for command-line aficionados, bcp is a powerful tool for those seeking to insert data into a SQL Server database from within a batch file or other programmatic method.
* **Bulk Insert :** The BULK INSERT statement was introduced in SQL Server 7 and allows you to interact with bcp (bulk copy program) via a script.
* **OpenRowSet :** The OPENROWSET function can be referenced in the FROM clause of a query as if it were a table name. The OPENROWSET function can also be referenced as the target table of an INSERT, UPDATE, or DELETE statement, subject to the capabilities of the OLE DB provider. Although the query might return multiple result sets, OPENROWSET returns only the first one.
* **OPENDATASOURCE :** Provides ad hoc connection information as part of a four-part object name without using a linked server name.
* **OPENQUERY :** Executes the specified pass-through query on the specified linked server. This server is an OLE DB data source. OPENQUERY can be referenced in the FROM clause of a query as if it were a table name.
* **Linked Servers :** Configure a linked server to enable the SQL Server Database Engine to execute commands against OLE DB data sources outside of the instance of SQL Server. Typically linked servers are configured to enable the Database Engine to execute a Transact-SQL statement that includes tables in another instance of SQL Server, or another database product such as Oracle.

**15. What is the native system stored procedure to execute a command against all databases?**

* The sp\_MSforeachdb system stored procedure accepts the **@Command** parameter which can be exetecuted against all databases. The ‘?’ is used as a placeholder for the database name to execute the same command.
* The alternative is to use a cursor to process specific commands against each database.

**16. How can a SQL Developer prevent T-SQL code from running on a production SQL Server?**

Use IF logic with the **@@SERVERNAME function** compared against a string with a RETURN command before any other logic.

**17. How do you maintain database integrity where deletions from one table will automatically cause deletions in another table?**

You can create a trigger that will automatically delete elements in the second table when elements from the first table are removed.

**18. What port does SQL server run on?**

1433 is the standard port for SQL server.

**19. What is the SQL CASE statement used for? Explain with an example?**

It allows you to embed an if-else like clause in the SELECT clause.

SELECT Employee\_Name, CASE Location

WHEN 'alex' THEN Bonus \* 2

WHEN 'robin' THEN Bonus \*, 5

ELSE Bonus

END

"New Bonus"

FROM Intellipaat\_employee;

**20. What are the risks of storing a hibernate-managed object in cache? How do you overcome the problems?**

The primary problem here is that the object will outlive the session it came from. Lazily loaded properties won’t get loaded if needed later. To overcome the problem, perform cache on the object’s id and class and then retrieve the object in the current session context.

**21. When is the use of UPDATE\_STATISTICS command ?**

Updating statistics ensures that queries compile with up-to-date statistics. However, updating statistics causes queries to recompile. We recommend not updating statistics too often because there is a performance tradeoff between improving query plans and the time it takes to recompile queries. The specific tradeoffs depend on your application. UPDATE STATISTICS can use tempdb to sort the sample of rows for building statistics.

**Syntax**

UPDATE STATISTICS table\_or\_indexed\_view\_name

[

{

{ index\_or\_statistics\_\_name }

| ( { index\_or\_statistics\_name } [ ,...n ] )

}

]

[ WITH

[

FULLSCAN

| SAMPLE number { PERCENT | ROWS }

| RESAMPLE

[ ON PARTITIONS ( { | } [, …n] ) ]

| [ ,...n ]

]

[ [ , ] [ ALL | COLUMNS | INDEX ]

[ [ , ] NORECOMPUTE ]

[ [ , ] INCREMENTAL = { ON | OFF } ]

] ;

::=

[ STATS\_STREAM = stats\_stream ]

[ ROWCOUNT = numeric\_constant ]

[ PAGECOUNT = numeric\_contant ]

**22. What is SQL Profiler?**

Microsoft SQL Server Profiler is a graphical user interface to SQL Trace for monitoring an instance of the Database Engine or Analysis Services. You can capture and save data about each event to a file or table to analyze later.

Use SQL Profiler to monitor only the events in which you are interested.

If traces are becoming too large, you can filter them based on the information you want, so that only a subset of the event data is collected. Monitoring too many events adds overhead to the server and the monitoring process and can cause the trace file or trace table to grow very large, especially when the monitoring process takes place over a long period of time.

**23. What command using Query Analyzer will give you the version of SQL server and operating system?**

SELECT SERVERPROPERTY (‘productversion’), SERVERPROPERTY (‘productlevel’), SERVERPROPERTY (‘edition’).

**24. What does it mean to have QUOTED\_IDENTIFIER ON? What are the implications of having it OFF?**

When **SET QUOTED\_IDENTIFIER** is **ON,** identifiers can be delimited by double quotation marks, and literals must be delimited by single quotation marks. When **SET QUOTED\_IDENTIFIER** is **OFF**, identifiers cannot be quoted and must follow all Transact-SQL rules for identifiers.

**25. What is the STUFF function and how does it differ from the REPLACE function in SQL?**

**Stuff function :** – This function is used to replace string from the given start position, passed as 2nd argument with string passed as last argument. In Stuff function, 3rd argument defines the number of characters which are going to be replaced.  
**Syntax :-**

STUFF ( character\_expression , start , length , replaceWith\_expression )

**For example :-**

Select Stuff ('Intellipaat', 3, 3, 'abc')

This query will return the string **"Iabcllipaat"**. In this example, Stuff function replaces the string **"Intellipaat"** onwards the 3rd position('nte') with 'abc'.

**Replace Function :**– Replace function is used to replace all occurrence of a specified with the string passed as last argument.  
**Syntax :-**

REPLACE ( string\_expression , string\_pattern , string\_replacement )

**For example :-**

Select Replace ('Abcabcabc', 'bc', 'xy')

This query will return the string **Axyaxyaxy**. In this example, Replace function replaces the occurrence of each **'bc'** string with **'xy'**.

**26. How to get @@ERROR and @@ROWCOUNT at the same time?**

If @@Rowcount is checked after Error checking statement then it will have 0 as the value of @@Recordcount as it would have been reset. And if @@Recordcount is checked before the error-checking statement then @@Error would get reset. To get @@error and @@rowcount at the same time do both in same statement and store them in local variable.

SELECT @RC = @@ROWCOUNT, [@ER](https://intellipaat.com/members/er/) = @@ERROR

**27. What is de-normalization in SQL database administration? Give examples**

De-normalization is used to optimize the readability and performance of the database by adding redundant data. It covers the inefficiencies in the relational database software.  
De-normalization logical data design tend to improve the query responses by creating rules in the database which are called as constraints.  
**Examples include the following :**

* Materialized views for implementation purpose such as :
* Storing the count of “many” objects in one-to-many relationship.
* Linking attribute of one relation with other relations.
* To improve the performance and scalability of web applications.

**28. Can you explain about buffer cash and log Cache in SQL Server?**

* **Buffer Cache :** Buffer cache is a memory pool in which data pages are read. The ideal performance of the buffer cache is indicated as: 95% indicates that pages that were found in the memory are 95% of time. Another 5% is need physical disk access.  
  If the value falls below 90%, it is the indication of more physical memory requirement on the server.
* **Log Caches :** Log cache is a memory pool used to read and write the log pages. A set of cache pages are available in each log cache. The synchronization is reduced between log and data buffers by managing log cache separately from the buffer cache.

**29. Describe how to use Linked Server.**

MS SQL Server supports the connection to different OLE DB on an ad hoc basis. This persistent connection is referred as Linked Server.  
**Following are the steps to use Linked Server for any OLE DB. You can refer this to use an MS-Excel workbook.**

1. Open SQL Server Management Studio in SQL Server.
2. Expand Server Objects in Object Explorer.
3. Right-click on Linked Servers. Click on New Linked Server.
4. Select General page in the left pane and
   * Type any name for the linked server in the first text box.
   * Select the Other Data Source option.
   * Click on Microsoft Jet 4.0 OLE DB Provider from the Provider list.
   * Type the Excel as the name of the OLE DB data source.
   * Type the full path and file name of the Excel file in Data Source box.
   * Type the Excel version no. (7.0, 8.0 etc) in the Provider String. Use Excel 8.0 for Excel 2000, Excel 2002 or Excel 97.
   * To create a linked server click on OK.

**30. How to find second highest salary of an Employee**

There are many ways to find second highest salary of Employees in SQ. You can either use SQL Join or Subquery to solve this problem.  
**Here is SQL query using Subquery :**

Select MAX(Salary) from Intellipaat\_emplyee WHERE Salary NOT IN ( select MAX(Salary) from Intellipaat\_employee.

**31. Explain how to send email from SQL database.**

SQL Server has a feature for sending mails. Stored procedures can also be used for sending mail on demand. With SQL Server 2005, MAPI client is not needed for sending mails.  
**The following is the process for sending emails from database.**

* Make sure that the SQL Server Mail account is configured correctly and enable Database Mail.
* Write a script to send an e-mail. The following is the script.
* USE [YourDB]
* EXEC msdb.dbo.sp\_send\_dbmail
* @recipients = 'xyz@intellipaat.com; abc@intellipaat.com;pqr@intellipaat.com’
* @body = ' A warm wish for your future endeavor',
* @subject = 'This mail was sent using Database Mail' ;

GO

**32. How to make remote connection in database?**

**The following is the process to make a remote connection in database :**

1. Use SQL Server Surface Area Configuration Tool for enabling the remote connection in database.
2. Click on Surface Area Configuration for Services and Connections.
3. Click on SQLEXPRESS/Database Engine/RemoteConnections.
4. Select the radio button: Local and Remote Connections and select ‘Using TCP/IP only’ under Local and Remote Connections.
5. Click on OK button / Apply button

**33. What is the purpose of OPENXML clause SQL server stored procedure?**

OPENXML parses the XML data in [SQL Server](https://intellipaat.com/sql-server-dba-video-tutorial/) in an efficient manner. It’s primary ability is to insert XML data to the RDB. It is also possible to query the data by using OpenXML. The path of the XML element needs to be specified by using ‘xpath’.  
**The following is a procedure for retrieving xml data:**

DECLARE @index int

DECLARE @xmlString varchar(8000)

SET @xmlString ='

abc

9343463943/PhoneNo>

xyz

9342673212

'

EXEC sp\_xml\_preparedocument @index OUTPUT, @xmlString

SELECT \* FROM OPENXML (@index, 'Persons/Person') WITH (id varchar(10), Name varchar(100) 'Name' , PhoneNo varchar(50) 'PhoneNo')

EXEC sp\_xml\_removedocument @index

The above code snippet results the following:

15201 abc 9343463943

15202 xyz 9342673212

**34. How to store pdf file in SQL Server?**

Create a column as type ‘blob’ in a table. Read the content of the file and save in ‘blob’ type column in a table.  
**Or**  
Store them in a folder and establish the pointer to link them in the database.

**35. Explain the use of keyword WITH ENCRYPTION. Create a Store Procedure with Encryption.**

It is a way to convert the original text of the stored procedure into encrypted form. The stored procedure gets obfuscated and the output of this is not visible to

CREATE PROCEDURE Abc

WITH ENCRYPTION

AS

<< SELECT statement>>

GO

WITH ENCRYPTION indicates that SQL Server will convert the original text of CREATE PROCEDURE statement to an encrypted format. Users that do not have no access to system tables or database files cannot retrieve the encrypted text. However, the text will be available to privileged users.

**Example:**

**CREATE PROCEDURE salary\_sum  
WITH ENCRYTION  
AS  
SELECT sum(salary)  
FROM employee  
WHERE emp\_dept LIKE Develop**

**36. What is lock escalation?**

Lock escalation is used to convert row locks and page locks into table locks thereby “escalating” the smaller or finer locks. This increases the system performance as each lock is nothing but a memory structure. Too many locks would mean more consumption of memory. Hence, escalation is used.  
Lock escalation from SQL Server 7.0 onwards is dynamically managed by SQL Server. It is the process of converting a lot of [low level locks into higher level locks](https://intellipaat.com/tutorial/oracle-plsql-tutorial/overview-of-plsql/).

**37. What is Failover clustering overview?**

Failover clustering is mainly used for data availability. **Typically, in a failover cluster, there are two machines.**

* One machine provides the basic services and the second is available to run the service when the primary system fails.
* The primary system is monitored periodically to check if it works. This monitoring may be performed by the failover computer or an independent system also called as cluster controller. In an event of failure of primary computer, the failover system takes control.

**38. What is Builtin/Administrator?**

The Builtin/Administrator account is basically used during some setup to join some machine in the domain. It should be disabled immediately thereafter. For any disaster recovery, the account will be automatically enabled. It should not be used for normal operations.

**39. What XML support does the SQL server extend?**

**SQL Server (server-side) supports 3 major elements :**

1. Creation of XML fragments: This is done from the relational data using FOR XML to the select query.
2. Ability to shred xml data to be stored in the database.
3. Finally, storing the xml data.

Client-side [XML support in SQL Server](https://intellipaat.com/tutorial/oracle-dba-tutorial/oracle-installation/) is in the form of SQLXML. **It can be described in terms of :**

* **XML Views :** providing bidirectional mapping between XML schemas and relational tables.
* **Creation of XML Templates :** allows creation of dynamic sections in XML.

SQL server can return XML document using FOR XML clause. XML documents can be added to SQL Server database and you can use the OPENXML clause to display the data from the document as a relational result set. SQL Server 2000 supports XPath queries.Get to know more about [SQL Techniques](https://intellipaat.com/blog/sql-optimization-techniques/) that can help you grow in your career.

[**« Previous**](https://intellipaat.com/interview-question/msbi-interview-questions/)

[SQL](https://dwbi.org/database/sql.html) is a language for accessing and manipulating database standardized by ANSI. To be successful with database-centric applications (which includes most of the applications Data Warehousing domain), one must be strong enough in SQL. In this article, we will learn more about SQL by breaking the subject in the form of several question-answer sessions commonly asked in Interviews.

**SET UP OF SAMPLE DATA FOR PRACTICING SQL**   
  
For the purpose of our demonstration, we will primarily use two database tables with just a few records - EMPLOYEE table and DEPT table. EMPLOYEE table will contain 10 records pertaining to 10 employees with funny sounding names of an imaginary organization and DEPT or Department table will contain 5 departments of that organization. [**Click here to download**](https://d2fvxlvmm9ujfq.cloudfront.net/db/sql-table-data.txt) the DDL/INSERT statements for this data if you want to practice the below SQLs in your personal computer

Contents of these tables are not same with Oracle emp and dept tables!!

## What is the difference between inner and outer join? Explain with example.

#### **Inner Join**

Inner join is the most common type of Join which is used to combine the rows from two tables and create a result set containing only such records that are present in both the tables based on the joining condition (predicate).

Inner join returns rows when there is at least one match in both tables

If none of the record matches between two tables, then INNER JOIN will return a NULL set. Below is an example of INNER JOIN and the resulting set.

SELECT dept.name DEPARTMENT, emp.name EMPLOYEE

FROM DEPT dept, EMPLOYEE emp

WHERE emp.dept\_id = dept.id

| **Department** | **Employee** |
| --- | --- |
| HR | Inno |
| HR | Privy |
| Engineering | Robo |
| Engineering | Hash |
| Engineering | Anno |
| Engineering | Darl |
| Marketing | Pete |
| Marketing | Meme |
| Sales | Tomiti |
| Sales | Bhuti |

#### **Outer Join**

Outer Join, on the other hand, will return matching rows from both tables as well as any unmatched rows from one or both the tables (based on whether it is single outer or full outer join respectively).

Outer Join can be full outer or single outer

Notice in our record set that there is no employee in the department 5 (Logistics). Because of this if we perform inner join, then Department 5 does not appear in the above result. However in the below query we perform an outer join (dept left outer join emp), and we can see this department.

SELECT dept.name DEPARTMENT, emp.name EMPLOYEE

FROM DEPT dept, EMPLOYEE emp

WHERE dept.id = emp.dept\_id (+)

| **Department** | **Employee** |
| --- | --- |
| HR | Inno |
| HR | Privy |
| Engineering | Robo |
| Engineering | Hash |
| Engineering | Anno |
| Engineering | Darl |
| Marketing | Pete |
| Marketing | Meme |
| Sales | Tomiti |
| Sales | Bhuti |
| Logistics |  |

The (+) sign on the emp side of the predicate indicates that emp is the outer table here. The above SQL can be alternatively written as below (will yield the same result as above):

SELECT dept.name DEPARTMENT, emp.name EMPLOYEE

FROM DEPT dept LEFT OUTER JOIN EMPLOYEE emp

ON dept.id = emp.dept\_id

## What is the difference between JOIN and UNION?

SQL JOIN allows us to “lookup” records on other table based on the given conditions between two tables. For example, if we have the department ID of each employee, then we can use this department ID of the employee table to join with the department ID of department table to lookup department names.

UNION operation allows us to add 2 similar data sets to create resulting data set that contains all the data from the source data sets. Union does not require any condition for joining. For example, if you have 2 employee tables with same structure, you can UNION them to create one result set that will contain all the employees from both of the tables.

SELECT \* FROM EMP1

UNION

SELECT \* FROM EMP2;

## What is the difference between UNION and UNION ALL?

UNION and UNION ALL both unify for add two structurally similar data sets, but UNION operation returns only the unique records from the resulting data set whereas UNION ALL will return all the rows, even if one or more rows are duplicated to each other.

In the following example, I am choosing exactly the same employee from the emp table and performing UNION and UNION ALL. Check the difference in the result.

SELECT \* FROM EMPLOYEE WHERE ID = 5

UNION ALL

SELECT \* FROM EMPLOYEE WHERE ID = 5

| **ID** | **MGR\_ID** | **DEPT\_ID** | **NAME** | **SAL** | **DOJ** |
| --- | --- | --- | --- | --- | --- |
| 5.0 | 2.0 | 2.0 | Anno | 80.0 | 01-Feb-2012 |
| 5.0 | 2.0 | 2.0 | Anno | 80.0 | 01-Feb-2012 |

SELECT \* FROM EMPLOYEE WHERE ID = 5

UNION

SELECT \* FROM EMPLOYEE WHERE ID = 5

| **ID** | **MGR\_ID** | **DEPT\_ID** | **NAME** | **SAL** | **DOJ** |
| --- | --- | --- | --- | --- | --- |
| 5.0 | 2.0 | 2.0 | Anno | 80.0 | 01-Feb-2012 |

## What is the difference between WHERE clause and HAVING clause?

WHERE and HAVING both filters out records based on one or more conditions. The difference is, WHERE clause can only be applied on a static non-aggregated column whereas we will need to use HAVING for aggregated columns.

To understand this, consider this example.   
Suppose we want to see only those departments where department ID is greater than 3. There is no aggregation operation and the condition needs to be applied on a static field. We will use WHERE clause here:

SELECT \* FROM DEPT WHERE ID > 3

| **ID** | **NAME** |
| --- | --- |
| 4 | Sales |
| 5 | Logistics |

Next, suppose we want to see only those Departments where Average salary is greater than 80. Here the condition is associated with a non-static aggregated information which is “average of salary”. We will need to use HAVING clause here:

SELECT dept.name DEPARTMENT, avg(emp.sal) AVG\_SAL

FROM DEPT dept, EMPLOYEE emp

WHERE dept.id = emp.dept\_id (+)

GROUP BY dept.name

HAVING AVG(emp.sal) > 80

| **DEPARTMENT** | **AVG\_SAL** |
| --- | --- |
| Engineering | 90 |

As you see above, there is only one department (Engineering) where average salary of employees is greater than 80.

## What is the difference among UNION, MINUS and INTERSECT?

UNION combines the results from 2 tables and eliminates duplicate records from the result set.

MINUS operator when used between 2 tables, gives us all the rows from the first table except the rows which are present in the second table.

INTERSECT operator returns us only the matching or common rows between 2 result sets.

To understand these operators, let’s see some examples. We will use two different queries to extract data from our emp table and then we will perform UNION, MINUS and INTERSECT operations on these two sets of data.

#### **UNION**

SELECT \* FROM EMPLOYEE WHERE ID = 5

UNION

SELECT \* FROM EMPLOYEE WHERE ID = 6

| **ID** | **MGR\_ID** | **DEPT\_ID** | **NAME** | **SAL** | **DOJ** |
| --- | --- | --- | --- | --- | --- |
| 5 | 2 | 2.0 | Anno | 80.0 | 01-Feb-2012 |
| 6 | 2 | 2.0 | Darl | 80.0 | 11-Feb-2012 |

#### **MINUS**

SELECT \* FROM EMPLOYEE

MINUS

SELECT \* FROM EMPLOYEE WHERE ID > 2

| **ID** | **MGR\_ID** | **DEPT\_ID** | **NAME** | **SAL** | **DOJ** |
| --- | --- | --- | --- | --- | --- |
| 1 |  | 2 | Hash | 100.0 | 01-Jan-2012 |
| 2 | 1 | 2 | Robo | 100.0 | 01-Jan-2012 |

#### **INTERSECT**

SELECT \* FROM EMPLOYEE WHERE ID IN (2, 3, 5)

INTERSECT

SELECT \* FROM EMPLOYEE WHERE ID IN (1, 2, 4, 5)

| **ID** | **MGR\_ID** | **DEPT\_ID** | **NAME** | **SAL** | **DOJ** |
| --- | --- | --- | --- | --- | --- |
| 5 | 2 | 2 | Anno | 80.0 | 01-Feb-2012 |
| 2 | 1 | 2 | Robo | 100.0 | 01-Jan-2012 |

## What is Self Join and why is it required?

Self Join is the act of joining one table with itself.

Self Join is often very useful to convert a hierarchical structure into a flat structure

In our employee table example above, we have kept the manager ID of each employee in the same row as that of the employee. This is an example of how a hierarchy (in this case employee-manager hierarchy) is stored in the RDBMS table. Now, suppose if we need to print out the names of the manager of each employee right beside the employee, we can use self join. See the example below:

SELECT e.name EMPLOYEE, m.name MANAGER

FROM EMPLOYEE e, EMPLOYEE m

WHERE e.mgr\_id = m.id (+)

| **EMPLOYEE** | **MANAGER** |
| --- | --- |
| Pete | Hash |
| Darl | Hash |
| Inno | Hash |
| Robo | Hash |
| Tomiti | Robo |
| Anno | Robo |
| Privy | Robo |
| Meme | Pete |
| Bhuti | Tomiti |
| Hash |  |

The only reason we have performed a left outer join here (instead of INNER JOIN) is we have one employee in this table without a manager (employee ID = 1). If we perform inner join, this employee will not show-up.

## How can we transpose a table using SQL (changing rows to column or vice-versa) ?

The usual way to do it in SQL is to use CASE statement or DECODE statement.

## How to generate row number in SQL Without ROWNUM

Generating a row number – that is a running sequence of numbers for each row is not easy using plain SQL. In fact, the method I am going to show below is not very generic either. This method only works if there is at least one unique column in the table. This method will also work if there is no single unique column, but collection of columns that is unique. Anyway, here is the query:

SELECT name, sal, (SELECT COUNT(\*) FROM EMPLOYEE i WHERE o.name >= i.name) row\_num

FROM EMPLOYEE o

order by row\_num

| **NAME** | **SAL** | **ROW\_NUM** |
| --- | --- | --- |
| Anno | 80 | 1 |
| Bhuti | 60 | 2 |
| Darl | 80 | 3 |
| Hash | 100 | 4 |
| Inno | 50 | 5 |
| Meme | 60 | 6 |
| Pete | 70 | 7 |
| Privy | 50 | 8 |
| Robo | 100 | 9 |
| Tomiti | 70 | 10 |

The column that is used in the row number generation logic is called “sort key”. Here sort key is “name” column. For this technique to work, the sort key needs to be unique. We have chosen the column “name” because this column happened to be unique in our Employee table. If it was not unique but some other collection of columns was, then we could have used those columns as our sort key (by concatenating those columns to form a single sort key).

Also notice how the rows are sorted in the result set. We have done an explicit sorting on the row\_num column, which gives us all the row numbers in the sorted order. But notice that name column is also sorted (which is probably the reason why this column is referred as sort-key). If you want to change the order of the sorting from ascending to descending, you will need to change “>=” sign to “<=” in the query.

As I said before, this method is not very generic. This is why many databases already implement other methods to achieve this. For example, in Oracle database, every SQL result set contains a hidden column called ROWNUM. We can just explicitly select ROWNUM to get sequence numbers.

## How to select first 5 records from a table?

This question, often asked in many interviews, does not make any sense to me. The problem here is how do you define which record is first and which is second. Which record is retrieved first from the database is not deterministic. It depends on many uncontrollable factors such as how database works at that moment of execution etc. So the question should really be – “how to select any 5 records from the table?” But whatever it is, here is the solution:

In Oracle,

SELECT \*

FROM EMP

WHERE ROWNUM <= 5;

In SQL Server,

SELECT TOP 5 \* FROM EMP;

**Generic solution**,

I believe a generic solution can be devised for this problem if and only if there exists at least one distinct column in the table. For example, in our EMP table ID is distinct. We can use that distinct column in the below way to come up with a generic solution of this question that does not require database specific functions such as ROWNUM, TOP etc.

SELECT name

FROM EMPLOYEE o

WHERE (SELECT count(\*) FROM EMPLOYEE i WHERE i.name < o.name) < 5

| **name** |
| --- |
| Inno |
| Anno |
| Darl |
| Meme |
| Bhuti |

I have taken “name” column in the above example since “name” is happened to be unique in this table. I could very well take ID column as well.

In this example, if the chosen column was not distinct, we would have got more than 5 records returned in our output.

Do you have a better solution to this problem? If yes, post your solution in the comment.

## What is the difference between ROWNUM pseudo column and ROW\_NUMBER() function?

ROWNUM is a pseudo column present in Oracle database returned result set prior to ORDER BY being evaluated. So ORDER BY ROWNUM does not work.

ROW\_NUMBER() is an analytical function which is used in conjunction to OVER() clause wherein we can specify ORDER BY and also PARTITION BY columns.

Suppose if you want to generate the row numbers in the order of ascending employee salaries for example, ROWNUM will not work. But you may use ROW\_NUMBER() OVER() like shown below:

SELECT name, sal, row\_number() over(order by sal desc) rownum\_by\_sal

FROM EMPLOYEE o

| **name** | **Sal** | **ROWNUM\_BY\_SAL** |
| --- | --- | --- |
| Hash | 100 | 1 |
| Robo | 100 | 2 |
| Anno | 80 | 3 |
| Darl | 80 | 4 |
| Tomiti | 70 | 5 |
| Pete | 70 | 6 |
| Bhuti | 60 | 7 |
| Meme | 60 | 8 |
| Inno | 50 | 9 |
| Privy | 50 | 10 |

### **What are the differences among ROWNUM, RANK and DENSE\_RANK?**

ROW\_NUMBER assigns contiguous, unique numbers from 1.. N to a result set.

RANK does not assign unique numbers—nor does it assign contiguous numbers. If two records tie for second place, no record will be assigned the 3rd rank as no one came in third, according to RANK. See below:

SELECT name, sal, rank() over(order by sal desc) rank\_by\_sal

FROM EMPLOYEE o

| **name** | **Sal** | **RANK\_BY\_SAL** |
| --- | --- | --- |
| Hash | 100 | 1 |
| Robo | 100 | 1 |
| Anno | 80 | 3 |
| Darl | 80 | 3 |
| Tomiti | 70 | 5 |
| Pete | 70 | 5 |
| Bhuti | 60 | 7 |
| Meme | 60 | 7 |
| Inno | 50 | 9 |
| Privy | 50 | 9 |

DENSE\_RANK, like RANK, does not assign unique numbers, but it does assign contiguous numbers. Even though two records tied for second place, there is a third-place record. See below:

SELECT name, sal, dense\_rank() over(order by sal desc) dense\_rank\_by\_sal

FROM EMPLOYEE o

| **name** | **Sal** | **DENSE\_RANK\_BY\_SAL** |
| --- | --- | --- |
| Hash | 100 | 1 |
| Robo | 100 | 1 |
| Anno | 80 | 2 |
| Darl | 80 | 2 |
| Tomiti | 70 | 3 |
| Pete | 70 | 3 |
| Bhuti | 60 | 4 |
| Meme | 60 | 4 |
| Inno | 50 | 5 |
| Privy | 50 | 5 |

## Advanced SQL Interview Questions and Answers

Here are some complex SQL interview problems that are for people who are looking for more advanced and challenging questions, along with the answers and complete explanations. Try to figure out the answer to the questions yourself before reading the answers.

***Suppose we have 2 tables called Orders and Salesperson shown below:***

|  |  |
| --- | --- |
| Salesperson | Orders |
| |  |  |  |  | | --- | --- | --- | --- | | ID | Name | Age | Salary | | 1 | Abe | 61 | 140000 | | 2 | Bob | 34 | 44000 | | 5 | Chris | 34 | 40000 | | 7 | Dan | 41 | 52000 | | 8 | Ken | 57 | 115000 | | 11 | Joe | 38 | 38000 | | |  |  |  |  |  | | --- | --- | --- | --- | --- | | Number | order\_date | cust\_id | salesperson\_id | Amount | | 10 | 8/2/96 | 4 | 2 | 540 | | 20 | 1/30/99 | 4 | 8 | 1800 | | 30 | 7/14/95 | 9 | 1 | 460 | | 40 | 1/29/98 | 7 | 2 | 2400 | | 50 | 2/3/98 | 6 | 7 | 600 | | 60 | 3/2/98 | 6 | 7 | 720 | | 70 | 5/6/98 | 9 | 7 | 150 | |

***Now suppose that we want to write SQL that must conform to the SQL standard.***

***We want to retrieve the names of all salespeople that have more than 1 order from the tables above. You can assume that each salesperson only has one ID.***

***If that is the case, then what (if anything) is wrong with the following SQL?:***

SELECT Name

FROM Orders, Salesperson

WHERE Orders.salesperson\_id = Salesperson.ID

GROUP BY salesperson\_id

HAVING COUNT( salesperson\_id ) >1

## The answer and explanation to advanced SQL question 1

There is definitely something wrong with the SQL above, and it is probably something that most beginner SQL programmers may not notice. The problem is that the SQL Standard says that we can not select a column that is not part of the group by clause unless it is also contained within an aggregate function. If we try to run the SQL above in SQL Server, we would get an error that looks like this:

Column 'Name' is invalid in the select list because it is

not contained in either an aggregate function or

the GROUP BY clause.

|  |
| --- |
|  |

You might be confused now, so let’s explain what that error means in plain English and through some simple examples. The most important thing you should take out of this discussion is understanding exactly why we get that error, and how to avoid it. There is a good reason for the error – read on to understand why.

You can see in the bad SQL above that the “Name” column is clearly ***not*** also a part of the group by statement, nor is it contained within an aggregate function (like SUM, MAX, etc).

As the error above suggests, we can fix the error by either wrapping the Name column inside an aggregate function or adding it to the Group By clause.

So if we want to write SQL that complies with the standard, then we could write something like this by adding the Name column to the Group By:

SELECT Name

FROM Orders, Salesperson

WHERE Orders.salesperson\_id = Salesperson.ID

GROUP BY salesperson\_id, Name

-- we added the name column to the group by, and now it works!

HAVING COUNT( salesperson\_id ) >1

The SQL above will run just fine without giving any error.

We could also fix the problem by putting the Name column in any aggregate function, and then simply make that a part of our select statement. So, we could just write this SQL instead, and it would be perfectly legal according to the SQL standard. We chose to use the MAX aggregate function, but any other aggregate would work just fine as well:

SELECT MAX(Name) --put name in an aggregate function

FROM Orders, Salesperson

WHERE Orders.salesperson\_id = Salesperson.ID

GROUP BY salesperson\_id

HAVING COUNT( salesperson\_id ) >1

Adding the Name column to the group by, or wrapping the Name column in an aggregate will certainly fix the error – but it’s very important to note that both of those things will change the data that is returned to a state that you may not want.

## Why does the selected column have to be in the group by clause or part of an aggregate function?

|  |
| --- |
|  |

So, now you understand how to fix the error – but do you understand why it is a problem in the first place? Well, you should – because **that is the most important thing to understand!** So, let’s explain some more about ***why*** SQL gives that error shown above .

First off, let’s talk a little bit more about aggregate functions. You probably know what aggregate functions in SQL are – we used one in the example above. In case you forgot, aggregate functions are used to perform a mathematical function on the values inside a given column, which is passed into the aggregate function. Here are some of the commonly used aggregate functions:

AVG() - Returns the average value

COUNT() - Returns the number of rows

FIRST() - Returns the first value

LAST() - Returns the last value

MAX() - Returns the largest value

MIN() - Returns the smallest value

SUM() - Returns the sum

To illustrate why the SQL standard says that a selected column has to be in the group by clause or part of an aggregate function, let’s use another example. Suppose we have some tables called Starbucks\_Stores and Starbucks\_Employees. In case you don’t already know, Starbucks is a popular coffee shop/cafe in the USA:

|  |  |
| --- | --- |
| Starbucks\_Employees | Starbucks\_Stores |
| |  |  |  |  |  | | --- | --- | --- | --- | --- | | ID | Name | Age | HourlyRate | StoreID | | 1 | Abe | 61 | 14 | 10 | | 2 | Bob | 34 | 10 | 30 | | 5 | Chris | 34 | 9 | 40 | | 7 | Dan | 41 | 11 | 50 | | 8 | Ken | 57 | 11 | 60 | | 11 | Joe | 38 | 13 | 70 | | |  |  | | --- | --- | | store\_id | city | | 10 | San Francisco | | 20 | Los Angeles | | 30 | San Francisco | | 40 | Los Angeles | | 50 | San Francisco | | 60 | New York | | 70 | San Francisco | |

Now, given the tables above let’s say that we write some SQL like this:

SELECT count(\*) as num\_employees, HourlyRate

FROM Starbucks\_Employees JOIN Starbucks\_Stores

ON Starbucks\_Employees.StoreID = Starbucks\_Stores.store\_id

GROUP BY city

It looks like the SQL above would just return the number of Starbucks employees in each city, along with the HourlyRate – because it will group the employees based on whatever city they work in (thanks to the “group by city” statement).

[Subscribe to our newsletter for more free interview questions.](http://www.programmerinterview.com/index.php/popup/)

## The problem with selecting a non-aggregate column that is not in the group by

But **the real question** here is what exactly would be returned for the HourlyRate in the SQL above? Would it return every employee’s hourly rate separated by commas? Since we group by city, will it return the highest hourly rate for each city? Will it return the hourly rate as a distinct list, so those 2 guys making 11 dollars an hour will have the 11 returned only once?

The problem here is that we do not know what will be returned **because we are *not* specific enough with what we are asking for in the SQL!** If what we are asking for is not specific enough, then the SQL processor will not know what to return.

This is why almost all database implementations return an error when the SQL above is run (with the notable exception of MySQL) – and this is why the SQL does not conform to the Standard. In SQL Server running the SQL above will return the same error that we showed earlier.

Let’s explain even further in case the problem with that SQL is not crystal clear. The order of operations in which things will happen with the SQL above is:

1. The 2 tables are joined on the condition that the

Starbucks\_Employees.StoreID column value is equal to the

Starbucks\_Stores.store\_id column values.

2. Groups are then created for each city - which means that

each distinct city will have it's own "group". So, there will

be a total of 3 groups one each for San Francisco, New York,

and Los Angeles.

3. The data we are interested in is selected from each group

that is created in step 2.

Because we end up with different groups based on the city, when we select a count(\*), that will find the total count of rows in each and every group. But, the problem is that when we select HourlyRate, there will be multiple values for the HourlyRate within each group. For example, for the group created by the city of San Francisco there will be 4 different values for the HourlyRate – 14, 10, 11, and 13.

|  |
| --- |
|  |

So the question is which value of the HourlyRate should be selected from each group? Well, it could be any one of those values – which is why that SQL results in an error. This is because what we are asking for is NOT specific enough – hopefully this is crystal clear now to you.

If the same HourlyRate were part of an aggregate function like MAX then it would simply return the highest HourlyRate within each group. And that is why having an aggregate function would fix the SQL error – because only one value will be selected from any given group.

So, this SQL is perfectly fine because we are more specific in what we ask for – but this SQL would only work for you if you actually want the highest HourlyRate for each city:

SELECT count(\*) as num\_employees, MAX(HourlyRate)

FROM Starbucks\_Employees JOIN Starbucks\_Stores

ON Starbucks\_Employees.StoreID = Starbucks\_Stores.store\_id

GROUP BY city

## Fix the error by adding column to the group clause

Another way to fix the error is to simply add the HourlyRate column to the group by clause. This also means that having the HourlyRate column wrapped in aggregate function is no longer necessary. So you could write some SQL like this and it would fix the error:

SELECT count(\*) as num\_employees, HourlyRate

FROM Starbucks\_Employees JOIN Starbucks\_Stores

ON Starbucks\_Employees.StoreID = Starbucks\_Stores.store\_id

GROUP BY city, HourlyRate

This would then create groups based on the unique ***combination*** of the values in the HourlyRate and City columns. This means that there will be a different group for each HourlyRate and City combination – so $11, San Francisco and $11, Los Angeles will be 2 different groups. If you need to read up more on this topic then you can go here: [Group By With Multiple Columns](http://www.programmerinterview.com/index.php/database-sql/sql-group-by-with-multiple-columns/)

With the SQL above, each group will only have one value for the HourlyRate, which also means that there will be no ambiguity or confusion when selecting the HourlyRate since there is only possible value to select. It is now very clear that one and only one HourlyRate value can be returned for each group.

## Adding the column to the group by clause fixes the error but will alter the data that is returned

But, one very important thing to note is that even though adding the column to the group by will fix the error, *it will also change the groups that are created*. This means that the data returned will be completely different from what was returned before. So, the count(\*) function will no longer return the count of employees in a given city, and will instead return the number of rows in each group created by the unique combination of the HourlyRate and city columns.

## MySQL – selecting non-aggregate columns not in the group by

One very important thing that you should know is that MySQL actually **allows** you to have non-aggregated columns in the select list even if they are not a part of the group by clause (a quick side note: a non-aggregated column is simply a column that is *not* wrapped within an aggregate function). What this means is that you will not receive an error if you try to run any of the “bad” SQL above in MySQL. The reason it is allowed in MySQL is because MySQL assumes that you know what you are doing – and it does actually make sense in some scenarios. For instance, let’s refer back to the SQL that we started with:

SELECT Name

FROM Orders, Salesperson

WHERE Orders.salesperson\_id = Salesperson.ID

GROUP BY salesperson\_id

HAVING COUNT( salesperson\_id ) >1

The reason the original SQL code (presented above) works just fine in MySQL is because there is a 1 to 1 mapping of salesperson name to ID – meaning that for every unique salesperson ID there is only one possible name. Another way of saying that is that each salesperson can only have one name. So when we create groups (which is done in the “GROUP BY salesperson\_id”) based on the salesperson ID, each group will only have one and only one name.

This SQL will also run just fine in MySQL without returning an error:

SELECT count(\*) as num\_employees, HourlyRate

FROM Starbucks\_Employees JOIN Starbucks\_Stores

ON Starbucks\_Employees.StoreID = Starbucks\_Stores.store\_id

GROUP BY city

But, even though the code above will not return an error, the HourlyRate that is returned by MySQL will be some arbitrary (random) value within each group. This is because when we create each group based on the city, each group can have different values for the HourlyRate.

In other words, there is no one to one mapping between the HourlyRate and the city like we had before with the salesperson ID and the name. So, because we are not being specific as to which HourlyRate we want, MySQL will return an arbitrary value . For instance, in the group created by the city of San Francisco, MySQL could return the HourlyRate for any employee who works in San Francisco – whether it is 14, 10, 11, or 13 we don’t really know since it is arbitrary/random in MySQL.

That concludes part 1 of our more difficult and complex SQL questions. Click on next to check out the next question that’s a part of our advanced SQL interview questions list.

## Database Job Interview Questions and Answers

Once you’ve landed the job interview, that’s when the fun begins. As with any other tech job interview, if a technology is listed on your resume, you must be prepared to answer questions about it (especially if that technology is featured prominently in the job description).

### A) Data Modeling Questions

Before getting to the database-specific questions, a prospective candidate must be able to handle the task of taking a set of requirements from a conceptual data model to a physical one. Here is where performance considerations are addressed, tables are designed, and triggers are addressed.

Some questions seen here are related to a candidate’s specific background, like what tools have been used to model data? Other questions might be more related to data modeling fundamentals. Here are some questions with their answers for common questions on the fundamentals.

**1. What is cardinality?**

Thinking mathematically, it is the number of elements in a set. Thinking in the database world, cardinality has to do with the counts in a relationship, one-to-one, one-to-many, or many-to-many.

**2. Describe the differences in the first through fifth normalization forms.**

Database candidates should be familiar with most if not all of these without needing to lookup definitions. Some of the other normalization forms are less commonly known/used, but could theoretically be asked. Knowing the differences between second and third is probably a good idea.

From Wikipedia, here are their definitions:

* First: The domain of each attribute contains only atomic values, and the value of each attribute contains only a single value from that domain.
* Second: No non-prime attribute in the table is functionally dependent on a proper subset of any candidate key.
* Third: Every non-prime attribute is non-transitively dependent on every candidate key in the table. The attributes that do not contribute to the description of the primary key are removed from the table. In other words, no transitive dependency is allowed.
* Fourth: Every non-trivial multivalued dependency in the table is a dependency on a superkey.
* Fifth: Every non-trivial join dependency in the table is implied by the superkeys of the table.

**3. When might someone denormalize their data?**

Typically done for performance reasons, to reduce the number of table joins. This is not a good idea in a transactional environment as there are inherent data integrity risks or performance risks due to excessive locking to maintain data integrity.

Questions related to the Unified Modeling Language (UML) or Entity-Relationship Diagrams (ERDs) may also be asked here.

**4. What are the elements of an ERD?**

The three elements include the entities for which someone is seeking information, the attributes of those entities, and the relationships between the entities.

### B) SQL Job Interview Questions

After getting some foundations out of the way, an interviewer can switch over to questions specific to SQL. The difficulties here will be varied. An introductory candidate might be asked only the introductory questions. A more seasoned candidate would be asked those as well as the more advanced questions.

Introductory questions are typically related to syntax and basic SQL commands that are typically common across database vendors.

**1) Which SQL command is used to add a row?**

INSERT

**2) Write the command to remove all employees named John from the EMPLOYEE table.**

DELETE from EMPLOYEE WHERE firstName = ‘John’

**3) What are the differences between primary and foreign keys?**

The primary key is the column or set of columns used to uniquely identify the items in a table. A foreign key is used to uniquely identify the items in a different table, allowing join operations to happen.

**4) What does SQL stand for?**  
Structured Query Language

### C) Advanced SQL Job Interview Questions

Advanced SQL questions would involve more advanced SQL operations,  but could also get into the details of Oracle or SQL Server specific constructs. If asked a question about a technology a candidate isn’t familiar with, they should not be afraid to respond as such. There are many database systems out there, and most people will not be familiar with the ins and out of every one of them.

**1) What is the difference between an inner and outer join?**

An inner join involves joining two tables where a common id/key exists in both. An outer join is the joining of two tables, but where there is no match in the second (or first).

**2) How do you maintain database integrity where deletions from one table will automatically cause deletions in another table?**

You create a trigger that will automatically delete elements in the second table when elements from the first table are removed.

**3) What port does SQL server run on?**

1433 is the standard port for SQL server.

**4) What is the SQL CASE statement used for and give an example?**

It allows you to embed an if-else like clause in the SELECT clause.

SELECT Employee\_Name, CASE Location

WHEN 'Boston' THEN Bonus \* 2

WHEN 'Austin' THEN Bonus \* ,5

ELSE Bonus

END

"New Bonus"

FROM Employee;

**5) Table problem solving**

Candidates are almost guaranteed to be asked a series of questions related to a deriving data from a provided set of tables. Typically, this might be something along the lines of finding the average grades for the students who took a particular class or perhaps related to finding the top salesperson for a company.

Here’s a sample problem  based on the following tables:

Salesperson

ID Name Salary Commission Rate Hire Date

1 John 100000 6 4/1/2006

2 Amy 120000 5 5/1/2010

3 Mark 65000 12 12/25/2008

4 Pam 25000 25 1/1/2005

5 Alex 50000 10 2/3/2007

Customer

ID Name City

1 Red Boston

2 Orange New York

3 Yellow Boston

4 Green Austin

Order

ID Date Cust\_ID Sales\_ID Amount

1 1/1/2014 3 4 100000

2 2/1/2014 4 5 5000

3 3/1/2014 1 1 50000

4 4/1/2014 1 4 25000

### Sample questions based on the above tables:

***1. Given the tables above, write a query that will calculate the total commission by salesperson.***

SELECT o.amount \* s.commission / 100 from salesperson s, order o where o.sales\_id = s.id

**2. Name all salespersons who did not sell to company Red.**

SELECT s.name from salesperson s

WHERE s.ID NOT IN (

SELECT o.sales\_id FROM orders o, customer c

WHERE o.cust\_id = c.ID

AND c.Name = 'RED')

### D) More Database Job Interview Questions — Beyond SQL

The world used to revolve around SQL-based databases. Times have changed and the world did not go to an object-based database world. It is good for the database job candidate to have a working knowledge of what the alternatives are all about.

These are minimums here on necessary knowledge. The more advanced a job, the more depth is needed.

1) **What is Hibernate?**

Hibernate is an object-relational mapping library that takes Java objects and maps them into relational database tables. It provides its own query language (Hibernate Query Language / HQL) that fills in where SQL falls short when dealing with objects. The latest version as of June 2014 is 4.3.5.

2) **What are Hadoop and Hive?**

Hadoop is an Apache project for dealing with large data sets, basically providing a file system with libraries for large scale data processing like map-reduce. Hive provides the layer on top of Hadoop for query and analysis. The query language is calle HiveQL and does not support transactions.

3) **What is NoSQL?**

It stands for Not Only SQL and provides an alternative to relational databases. Instead of tabular data stores, they use graph stores, key-value stores, document databases, and wide-column stores. It is popular in the agile development world as developers don’t have to finalize the data model before storing information.

On the more in-depth side, consider a question like:

4) **What are the risks of storing a hibernate-managed object in a cache? And, how do you overcome the problems?**

With an answer of something along the lines of the following; The primary problem here is the object will outlive the session it came from. Lazily loaded properties won’t get loaded if needed later. To overcome the problem, cache just the object’s id and class, then have retrieve the object in the current session context.

### E) More Database / SQL Interview Tips and Advice

Even in highly technical job interviews, it’s not all about  the technical questions. Candidates need to prepare to give examples from past job experiences, sharing stories that  allow the potential employer to get a deeper understanding of existing skillset.

You must also be ready to eloquently answer the necessary “fit” questions, which generally center around your career goals, your strengths and weaknesses, and your soft skills.

This is where the dreaded “tell me about yourself” question usually comes into play.  The interviewers want to feel confident that you have the right skills, but also that your will be a productive and pleasant coworker.

This is also where it helps to be able to demonstrate your knowledge of the company and its needs. More and more company details are becoming public these days and if a candidate can conduct some basic research and then ask a smart question about an active development the company, it will show initiative and interest on the candidate’s part. If an employer is on the fence between two candidates, that initiative and interest could push them over the edge.

Dear readers, these **SQL Interview Questions** have been designed specially to get you acquainted with the nature of questions you may encounter during your interview for the subject of **SQL**. As per my experience good interviewers hardly plan to ask any particular question during your interview, normally questions start with some basic concept of the subject and later they continue based on further discussion and what you answer:

What is the difference between SQL and MySQL or SQL Server?

SQL or Structured Query Language is a language; language that communicates with a relational database thus providing ways of manipulating and creating databases. MySQL and Microsoft’s SQL Server both are relational database management systems that use SQL as their standard relational database language.

What is the difference between SQL and PL/SQL?

PL/SQL is a dialect of SQL that adds procedural features of programming languages in SQL. It was developed by Oracle Corporation in the early 90's to enhance the capabilities of SQL.

What are various DDL commands in SQL? Give brief description of their purposes.

Following are various DDL or Data Definition Language commands in SQL −

* **CREATE −** it creates a new table, a view of a table, or other object in database.
* **ALTER −** it modifies an existing database object, such as a table.
* **DROP −** it deletes an entire table, a view of a table or other object in the database.

What are various DML commands in SQL? Give brief description of their purposes.

Following are various DML or Data Manipulation Language commands in SQL −

* **SELECT −** it retrieves certain records from one or more tables.
* **INSERT −** it creates a record.
* **UPDATE −** it modifies records.
* **DELETE −** it deletes records.

What are various DCL commands in SQL? Give brief description of their purposes.

Following are various DCL or Data Control Language commands in SQL −

* **GRANT −** it gives a privilege to user.
* **REVOKE −** it takes back privileges granted from user.

Can you sort a column using a column alias?

Yes. A column alias could be used in the ORDER BY clause.

Is a NULL value same as zero or a blank space? If not then what is the difference?

A NULL value is not same as zero or a blank space. A NULL value is a value which is ‘unavailable, unassigned, unknown or not applicable’. Whereas, zero is a number and blank space is a character.

Say True or False. Give explanation if False.

If a column value taking part in an arithmetic expression is NULL, then the result obtained would be NULLM.

True.

If a table contains duplicate rows, does a query result display the duplicate values by default? How can you eliminate duplicate rows from a query result?

A query result displays all rows including the duplicate rows. To eliminate duplicate rows in the result, the DISTINCT keyword is used in the SELECT clause.

What is the purpose of the condition operators BETWEEN and IN?

The BETWEEN operator displays rows based on a range of values. The IN condition operator checks for values contained in a specific set of values.

How do you search for a value in a database table when you don’t have the exact value to search for?

In such cases, the LIKE condition operator is used to select rows that match a character pattern. This is also called ‘wildcard’ search.

What is the default ordering of data using the ORDER BY clause? How could it be changed?

The default sorting order is ascending. It can be changed using the DESC keyword, after the column name in the ORDER BY clause.

What are the specific uses of SQL functions?

SQL functions have the following uses −

* Performing calculations on data
* Modifying individual data items
* Manipulating the output
* Formatting dates and numbers
* Converting data types

What are the case manipulation functions of SQL?

LOWER, UPPER, INITCAP

Which function returns the remainder in a division operation?

The MOD function returns the remainder in a division operation.

What is the purpose of the NVL function?

The NVL function converts a NULL value to an actual value.

What is the difference between the NVL and the NVL2 functions?

The NVL(exp1, exp2) function converts the source expression (or value) exp1 to the target expression (or value) exp2, if exp1 contains NULL. The return value has the same data type as that of exp1.

The NVL2(exp1, exp2, exp3) function checks the first expression exp1, if it is not null then, the second expression exp2 is returned. If the first expression exp1 is null, then the third expression exp3 is returned.

What is the use of the NULLIF function?

The NULLIF function compares two expressions. If they are equal, the function returns null. If they are not equal, the first expression is returned.

Discuss the syntax and use of the COALESCE function?

The COALESCE function has the expression COALESCE(exp1, exp2, …. expn)

It returns the first non-null expression given in the parameter list.

Which expressions or functions allow you to implement conditional processing in a SQL statement?

There are two ways to implement conditional processing or IF-THEN-ELSE logic in a SQL statement.

* Using CASE expression
* Using the DECODE function

You want to display a result query from joining two tables with 20 and 10 rows respectively. Erroneously you forget to write the WHERE clause. What would be the result?

The result would be the Cartesian product of two tables with 20 x 10 = 200 rows.

What is the difference between cross joins and natural joins?

The cross join produces the cross product or Cartesian product of two tables. The natural join is based on all the columns having same name and data types in both the tables.

What is the purpose of the group functions in SQL? Give some examples of group functions.

Group functions in SQL work on sets of rows and returns one result per group. Examples of group functions are AVG, COUNT, MAX, MIN, STDDEV, SUM, VARIANCE.

Say True or False. Give explanation if False.

By default the group functions consider only distinct values in the set.

By default, group functions consider all values including the duplicate values.

Say True or False. Give explanation if False.

The DISTINCT keyword allows a function consider only non-duplicate values.

True.

Say True or False. Give explanation if False.

All group functions ignore null values.

True.

Say True or False. Give explanation if False.

COUNT(\*) returns the number of columns in a table.

False. COUNT(\*) returns the number of rows in a table.

What’s wrong in the following query?

SELECT subject\_code, count(name)

FROM students;

It doesn’t have a GROUP BY clause. The subject\_code should be in the GROUP BY clause.

SELECT subject\_code, count(name)

FROM students

GROUP BY subject\_code;

What’s wrong in the following query?

SELECT subject\_code, AVG (marks)

FROM students

WHERE AVG(marks) > 75

GROUP BY subject\_code;

The WHERE clause cannot be used to restrict groups. The HAVING clause should be used.

SELECT subject\_code, AVG (marks)

FROM students

HAVING AVG(marks) > 75

GROUP BY subject\_code;

Say True or False. Give explanation if False.

Group functions cannot be nested.

False. Group functions can be nested to a depth of two.

What do you understand by a subquery? When is it used?

A subquery is a SELECT statement embedded in a clause of another SELECT statement. It is used when the inner query, or the subquery returns a value that is used by the outer query. It is very useful in selecting some rows in a table with a condition that depends on some data which is contained in the same table.

Say True or False. Give explanation if False.

A single row subquery returns only one row from the outer SELECT statement

False. A single row subquery returns only one row from the inner SELECT statement.

Say True or False. Give explanation if False.

A multiple row subquery returns more than one row from the inner SELECT statement.

True.

Say True or False. Give explanation if False.

Multiple column subqueries return more than one column from the inner SELECT statement.

True.

What’s wrong in the following query?

SELECT student\_code, name

FROM students

WHERE marks =

(SELECT MAX(marks)

FROM students

GROUP BY subject\_code);

Here a single row operator = is used with a multiple row subquery.

What are the various multiple row comparison operators in SQL?

IN, ANY, ALL.

What is the pupose of DML statements in SQL?

The DML statements are used to add new rows to a table, update or modify data in existing rows, or remove existing rows from a table.

Which statement is used to add a new row in a database table?

The INSERT INTO statement.

Say True or False. Give explanation if False.

While inserting new rows in a table you must list values in the default order of the columns.

True.

How do you insert null values in a column while inserting data?

Null values can be inserted into a table by one of the following ways −

* Implicitly by omitting the column from the column list.
* Explicitly by specifying the NULL keyword in the VALUES clause.

Say True or False. Give explanation if False.

INSERT statement does not allow copying rows from one table to another.

False. INSERT statement allows to add rows to a table copying rows from an existing table.

How do you copy rows from one table to another?

The INSERT statement can be used to add rows to a table by copying from another table. In this case, a subquery is used in the place of the VALUES clause.

What happens if you omit the WHERE clause in the UPDATE statement?

All the rows in the table are modified.

Can you modify the rows in a table based on values from another table? Explain.

Yes. Use of subqueries in UPDATE statements allow you to update rows in a table based on values from another table.

Say True or False. Give explanation if False.

The DELETE statement is used to delete a table from the database.

False. The DELETE statement is used for removing existing rows from a table.

What happens if you omit the WHERE clause in a delete statement?

All the rows in the table are deleted.

Can you remove rows from a table based on values from another table? Explain.

Yes, subqueries can be used to remove rows from a table based on values from another table.

Say True or False. Give explanation if False.

Attempting to delete a record with a value attached to an integrity constraint, returns an error.

True.

Say True or False. Give explanation if False.

You can use a subquery in an INSERT statement.

True.

What is the purpose of the MERGE statement in SQL?

The MERGE statement allows conditional update or insertion of data into a database table. It performs an UPDATE if the rows exists, or an INSERT if the row does not exist.

Say True or False. Give explanation if False.

A DDL statement or a DCL statement is automatically committed.

True.

What is the difference between VARCHAR2 AND CHAR datatypes?

VARCHAR2 represents variable length character data, whereas CHAR represents fixed length character data.

Say True or False. Give explanation if False.

A DROP TABLE statement can be rolled back.

False. A DROP TABLE statement cannot be rolled back.

Which SQL statement is used to add, modify or drop columns in a database table?

The ALTER TABLE statement.

What is a view? Why should you use a view?

A view is a logical snapshot based on a table or another view. It is used for −

* Restricting access to data;
* Making complex queries simple;
* Ensuring data independency;
* Providing different views of same data.

Say True or False. Give explanation if False.

A view doesn’t have data of its own.

**Question 1: SQL Query to find second highest salary of Employee**

Answer: There are many ways to find second highest salary of Employee in SQL, you can either use SQL Join or Subquery to solve this problem. Here is SQL query using Subquery:

select MAX(Salary) from Employee WHERE Salary NOT IN (select MAX(Salary) from Employee );

See [How to find second highest salary in SQL](http://javarevisited.blogspot.com/2012/12/how-to-find-second-highest-or-maximum-salary-sql.html) for more ways to solve this problem.

**Question 2: SQL Query to find Max Salary from each department.**

Answer: You can find the maximum salary for each department by grouping all records by DeptId and then using MAX() function to calculate maximum salary in each group or each department.

SELECT DeptID, MAX(Salary) FROM Employee  GROUP BY DeptID.

These questions become more interesting if Interviewer will ask you to print department name instead of department id, in that case, you need to join Employee table with Department using foreign key DeptID, make sure you do LEFT or RIGHT OUTER JOIN to include departments without any employee as well.  Here is the query

SELECT DeptName, MAX(Salary) FROM Employee e RIGHT JOIN Department d ON e.DeptId = d.DeptID GROUP BY DeptName;

In this query, we have used RIGHT OUTER JOIN because we need the name of the department from Department table which is on the right side of JOIN clause, even if there is no reference of dept\_id on Employee table.  **Question 3: Write SQL Query to display the current date.**

Answer: SQL has built-in function called GetDate() which returns the current timestamp. This will work in Microsoft SQL Server, other vendors like Oracle and MySQL also has equivalent functions.

SELECT GetDate();

**Question 4: Write an SQL Query to check whether date passed to Query is the date of given format or not**.

Answer: SQL has IsDate() function which is used to check passed value is a date or not of specified format, it returns 1(true) or 0(false) accordingly. Remember ISDATE() is an MSSQL function and it may not work on Oracle, MySQL or any other database but there would be something similar.

SELECT  ISDATE('1/08/13') AS "MM/DD/YY";

It will return 0 because passed date is not in correct format.

**Question 5: Write an SQL Query to print the name of the distinct employee whose DOB is between 01/01/1960 to 31/12/1975.**

Answer: This SQL query is tricky, but you can use BETWEEN clause to get all records whose date fall between two dates.

SELECT DISTINCT EmpName FROM Employees WHERE DOB  BETWEEN ‘01/01/1960’ AND ‘31/12/1975’;

**Question 6: Write an SQL Query find number of employees according to gender  whose DOB is between 01/01/1960 to 31/12/1975.**

Answer : 

SELECT COUNT(\*), sex from Employees WHERE DOB BETWEEN '01/01/1960' AND '31/12/1975' GROUP BY sex;

**Question 7: Write an SQL Query to find an employee whose Salary is equal or greater than 10000**.

Answer : 

SELECT EmpName FROM Employees WHERE Salary>=10000;

**Question 8: Write an SQL Query to find name of employee whose name Start with ‘M’**

Answer : 

SELECT \* FROM Employees WHERE EmpName like 'M%';

**Question 9: find all Employee records containing the word "Joe", regardless of whether it was stored as JOE, Joe, or joe.**

Answer :

SELECT \* from Employees WHERE UPPER(EmpName) like '%JOE%';

**Question 10: Write an SQL Query to find  the year from date.**

Answer:  Here is how you can find Year from a Date in SQL Server 2008 

SELECT YEAR(GETDATE()) as "Year";

**Question 11: Write SQL Query to find duplicate rows in a database? and then write SQL query to delete them?**  
Answer: You can use the following query to select distinct records:

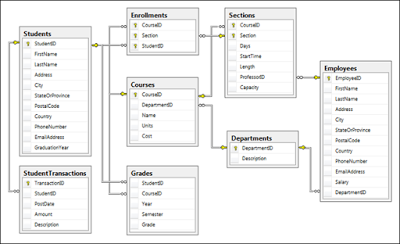
SELECT \* FROM emp a WHERE rowid = (SELECT MAX(rowid) FROM EMP b WHERE a.empno=b.empno)

to Delete:

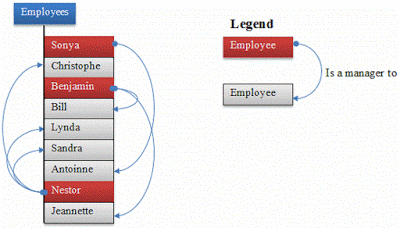
DELETE FROM emp a WHERE rowid != (SELECT MAX(rowid) FROM emp b WHERE a.empno=b.empno);

**Question 12: There is a table which contains two column Student and Marks, you need to find all the students, whose marks are greater than average marks i.e. list of above average students.**  
Answer: This query can be written using subquery as shown below:

SELECT student, marks from table where marks > SELECT AVG(marks) from table)

[](http://1.bp.blogspot.com/-EBP7clmjL1Q/VXrjOw_yTVI/AAAAAAAADAs/n_pQ6j7vkZw/s1600/SQL+Schema+Interview+Questions.png)

**Question 13: How do you find all employees which are also manager? .**  
You have given a standard employee table with an additional column mgr\_id, which contains employee id of the manager.

[](http://1.bp.blogspot.com/-A8OVkcRMrsM/VXrj9FBPKvI/AAAAAAAADA0/XZvHQHDVaNQ/s1600/Employee+Manager+query.gif)

Answer: You need to know about self-join to solve this problem. In Self Join, you can join two instances of the same table to find out additional details as shown below

SELECT e.name, m.name FROM Employee e, Employee m WHERE e.mgr\_id = m.emp\_id;

this will show employee name and manager name in two column e.g.  
  
name  manager\_name  
John   David  
  
One follow-up is to modify this query to include employees which don't have a manager. To solve that, instead of using the inner join, just use left outer join, this will also include employees without managers.  
  
  
  
**Question 14: You have a composite index of three columns, and you only provide the value of two columns in WHERE clause of a select query? Will Index be used for this operation?** For example if Index is on EmpId, EmpFirstName, and EmpSecondName and you write query like

SELECT \* FROM Employee WHERE EmpId=2 and EmpFirstName='Radhe'

If the given two columns are secondary index column then the index will not invoke, but if the given 2 columns contain the primary index(first column while creating index) then the index will invoke. In this case, Index will be used because EmpId and EmpFirstName are primary columns.

Hope this article will help you to take a quick practice whenever you are going to attend any interview and not have much time to go into the deep of each query, but if you have good time to prepare then I suggest you to read and solve SQL queries from **Joe Celko's**[**SQL Puzzles and Answers**](http://www.amazon.com/Puzzles-Answers-Edition-Kaufmann-Management/dp/0123735963?tag=javamysqlanta-20)**, Second edition**, one of the best book for SQL query lovers and enthusiastic.

Read more: <http://www.java67.com/2013/04/10-frequently-asked-sql-query-interview-questions-answers-database.html#ixzz4aLEhN8Za>