Institute for Advanced Computing And Software Development (IACSD)
Akurdi, Pune

Data Collection and DBMS

Dr. D.Y. Patil Educational Complex, Sector 29, Behind Akurdi Railway Station,

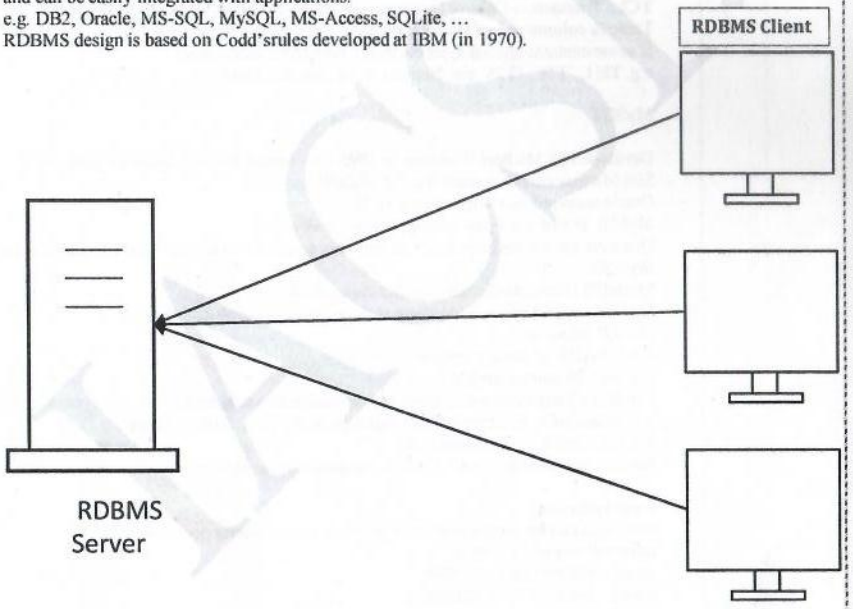
Nigdi Pradhikaran, Akurdi, Pune - 411044.

DBMS

Any enterprise application need to manage data.
In early days of software development, programmers store data into files and does operation on it.
However data is highly application specific.
Even today many software manage their data in custom formats e.g. Tally, Address,book, etc.
As data management became more common, DBMS systems were developed to handle the data. This enabled developers to focus on the business logic e.g. FoxPro,DBase, Excel, etc.
At least CRUD (Create, Retrieve, Update and Delete) operations are supported by all databases.
Traditional databases are file based, less secure, single-user, non- distributed, manage less amount of data (MB), complicated relationmanagement, file-locking and need number of lines of code to usein applications.

RDBMS

RDBMS is relational DBMS.
It organizes data into Tables, rows andcolumns. The tables are related to each other.
RDBMS follow table structure, more secure, multi-user, server-client architecture, server side processing, clustering support, manage huge data(TB), built-in relational capabilities, table-locking or row-locking and can be easily integrated with applications.
e.g. DB2, Oracle, MS-SQL, MySQL, MS-Access, SQLite, …
RDBMS design is based on Codd'srules developed at IBM (in 1970).



SQL

SQL is a standard language for storing, manipulating and retrieving data in databases.
Clients send SQL queries to RDBMS server and operations are performed accordingly.
Originally it was named as RQBE (Relational Query By Example).
SQL is ANSI standardised in 1987 and then revised multiple times adding newfeatures.
SQL is case insensitive.
There are five major categories:
DDL: Data Definition Language e.g. CREATE, ALTER, DROP, RENAME.

DML: Data Manipulation Language e.g. INSERT, UPDATE, DELETE.

DQL: Data Query Language e.g. SELECT.

DCL: Data Control Language e.g. CREATE USER, GRANT, REVOKE.

TCL: Transaction Control Language e.g. SAVEPOINT, COMMIT, ROLLBACK.
Table & column names allows alphabets, digits & few special symbols.
If name contains special symbols then it should be back-quotes.
e.g. Tbl1, `T1#`, `T2$` etc. Names can be max 30 chars long.

MySQL

Developed by Michael Widenius in 1995. It is named after his daughter name Myia.
Sun Microsystems acquired MySQL in 2008.
Oracle acquired Sun Microsystem in 2010.
MySQL is free and open-source database under GPL.
However some enterprise modules are close sourced and available only under commercial version of MySQL.
MariaDB is completely open-source clone of MySQL.
MySQL support multiple database storage and processing engines.
MySQL versions:
< 5.5: MyISAM storage engine
5.5: InnoDb storage engine
5.6: SQL Query optimizer improved, memcached style NoSQL
5.7: Windowing functions, JSON data type added for flexible schema
8.0: CTE, NoSQL document store.
MySQL is database of year 2019 (in database engine ranking).

Gettingstarted

root login can be used to perform CRUD as well as admin operations.

```
terminal> mysql –u root –p
mysql> SHOW DATABASES;
mysql> SELECT DATABASE();
mysql> USE mydb;
mysql> SHOW TABLES;
mysql> CREATE TABLE student(id INT, name VARCHAR(20), marks DOUBLE);
mysql> INSERT INTO student VALUES(1, 'Abc', 89.5);
mysql> SELECT * FROM student;
```


MySQL Data Types

String Data Types

Data type	Description
CHAR(size)	A FIXED length string (can contain letters, numbers, and special characters). The *size* parameter specifies the column length in characters - can be from 0 to 255. Default is 1
VARCHAR(size)	A VARIABLE length string (can contain letters, numbers, and special characters). The *size* parameter specifies the maximum string length in characters - can be from 0 to 65535
BINARY(size)	Equal to CHAR(), but stores binary byte strings. The *size* parameter specifies the column length in bytes. Default is 1
VARBINARY(size)	Equal to VARCHAR(), but stores binary byte strings. The *size* parameter specifies the maximum column length in bytes.
TINYBLOB	For BLOBs (Binary Large Objects). Max length: 255 bytes
TINYTEXT	Holds a string with a maximum length of 255 characters
TEXT(size)	Holds a string with a maximum length of 65,535 bytes
BLOB(size)	For BLOBs (Binary Large Objects). Holds up to 65,535 bytes of data
MEDIUMTEXT	Holds a string with a maximum length of 16,777,215 characters
MEDIUMBLOB	For BLOBs (Binary Large Objects). Holds up to 16,777,215 bytes of data

LONGTEXT	Holds a string with a maximum length of 4,294,967,295 characters
LONGBLOB	For BLOBs (Binary Large Objects). Holds up to 4,294,967,295 bytes of data
ENUM(val1, val2, val3, ...)	A string object that can have only one value, chosen from a list of possible values. You can list up to 65535 values in an ENUM list. If a value is inserted that is not in the list, a blank value will be inserted. The values are sorted in the order you enter them
SET(val1, val2, val3, ...)	A string object that can have 0 or more values, chosen from a list of possible values. You can list up to 64 values in a SET list

Numeric Data Types

Data type	Description
BIT(*size*)	A bit-value type. The number of bits per value is specified in *size*. The *size* parameter can hold a value from 1 to 64. The default value for *size* is 1.
TINYINT(*size*)	A very small integer. Signed range is from -128 to 127. Unsigned range is from 0 to 255. The *size* parameter specifies the maximum display width (which is 255)
BOOL	Zero is considered as false, nonzero values are considered as true.
BOOLEAN	Equal to BOOL
SMALLINT(*size*)	A small integer. Signed range is from -32768 to 32767. Unsigned range is from 0 to 65535. The *size* parameter specifies the maximum display width (which is 255)
MEDIUMINT(*size*)	A medium integer. Signed range is from -8388608 to 8388607. Unsigned range is from 0 to 16777215. The *size* parameter specifies the maximum display width (which

	is 255)
INT(*size*)	A medium integer. Signed range is from -2147483648 to 2147483647. Unsigned range is from 0 to 4294967295. The *size* parameter specifies the maximum display width (which is 255)
INTEGER(*size*)	Equal to INT(size)
BIGINT(*size*)	A large integer. Signed range is from -9223372036854775808 to 9223372036854775807. Unsigned range is from 0 to 18446744073709551615. The *size* parameter specifies the maximum display width (which is 255)
FLOAT(*size*, *d*)	A floating point number. The total number of digits is specified in *size*. The number of digits after the decimal point is specified in the *d* parameter. This syntax is deprecated in MySQL 8.0.17, and it will be removed in future MySQL versions
FLOAT(*p*)	A floating point number. MySQL uses the *p* value to determine whether to use FLOAT or DOUBLE for the resulting data type. If *p* is from 0 to 24, the data type becomes FLOAT(). If *p* is from 25 to 53, the data type becomes DOUBLE()
DOUBLE(*size*, *d*)	A normal-size floating point number. The total number of digits is specified in *size*. The number of digits after the decimal point is specified in the *d* parameter
DOUBLE PRECISION(*size*, *d*)	
DECIMAL(*size*, *d*)	An exact fixed-point number. The total number of digits is specified in *size*. The number of digits after the decimal point is specified in the *d* parameter. The maximum number for *size* is 65. The maximum number for *d* is 30. The default value for *size* is 10. The default value for *d* is 0.
DEC(*size*, *d*)	Equal to DECIMAL(size,d)

Date and Time Data Types

Data type	Description
DATE	A date. Format: YYYY-MM-DD. The supported range is from '1000-01-01' to '9999-12-31'
DATETIME(*fsp*)	A date and time combination. Format: YYYY-MM-DD hh:mm:ss. The supported range is from '1000-01-01 00:00:00' to '9999-12-31 23:59:59'. Adding DEFAULT and ON UPDATE in the column definition to get automatic initialization and updating to the current date and time
TIMESTAMP(*fsp*)	A timestamp. TIMESTAMP values are stored as the number of seconds since the Unix epoch ('1970-01-01 00:00:00' UTC). Format: YYYY-MM-DD hh:mm:ss. The supported range is from '1970-01-01 00:00:01' UTC to '2038-01-09 03:14:07' UTC. Automatic initialization and updating to the current date and time can be specified using DEFAULT CURRENT_TIMESTAMP and ON UPDATE CURRENT_TIMESTAMP in the column definition
TIME(*fsp*)	A time. Format: hh:mm:ss. The supported range is from '-838:59:59' to '838:59:59'
YEAR	A year in four-digit format. Values allowed in four-digit format: 1901 to 2155, and 0000. MySQL 8.0 does not support year in two-digit format.

SQL scripts

SQL script is multiple SQL queries written into a .sql file.
SQL scripts are mainly used while database backup and restore operations.
SQL scripts can be executed from terminal as:
terminal> mysql –u user –ppassword db < /path/to/sqlfile
SQL scripts can be executed from command line as:
mysql> SOURCE /path/to/sqlfile
Note that SOURCE is MySQL CLI client command.
It reads commands one by one from the script and execute them on server.

Basic Data Types ORACLE:

Datatype	Description
CHAR(n)	Stores fixed length string. Maximum length = 2000 bytesFor example: NAME CHAR(15)
VARCHAR2(n)	Stores variable length string. Maximum length = 4000 bytesFor example: DESCRIPTION VARCHAR2(100)
LONG(n)	Stores variable length string . Maximum length = 2 GIGA bytesFor example: SYNOPSIS LONG(5000)
NUMBER(p,s)	Stores numeric data . Range is 1E-129 to 9.99E125Max Number of significant digits = 38 For example: SALARY NUMBER(9,2)
DATE	Stores DATE. Range from January 1, 4712 BC to December 31, 9999 AD. Both DATE andTIME are stored. Requires 7 bytes. For example: HIREDATE DATE
RAW(n)	Stores data in binary format such as signature, photograph.Maximum size = 255 bytes
LONG RAW(n)	Same as RAW. Maximum size = 2 Gigabytes
TIMESTAMP	Stores the time to be stored as a date with fractional seconds. Extension to the DATA datatype There are some variations of the data type

Standard SQL statement groups

Groups	Statements	Description
DQL	SELECT	DATA QUERY LANGUAGE – It is used to get datafrom the database and impose ordering upon it.
DML	DELETE INSERT UPDATE MERGE	DATA MANIPULATION LANGUAGE – It is used tochange database data.
DDL	DROP TRUNCATE CREATE ALTER	DATA DEFINITION LANGUAGE – It is used to manipulate database structures and definitions.

TCL	COMMIT ROLLBACK SAVEPOINT	TCL statements are used to manage the transactions.
DCL (Rights)	REVOKE GRANT	They are used to remove and provide access rightsto database objects.

SQL SELECT Statement (Syntax is Same in MySql & Oracle)

The SELECT statement is the most commonly used command in Structured Query Language. It is used to access the records from one or more database tables and views. It also retrieves the selected data that follow the conditions we want.

By using this command, we can also access the particular record from the particular column of the table. The table which stores the record returned by the SELECT statement is called a result-set table.

Syntax of SELECT Statement in SQL

SELECT Column_Name_1, Column_Name_2, ..., Column_Name_N **FROM** Table_Name;

In this SELECT syntax, **Column_Name_1,Column_Name_2, ..., Column_Name_N** are the name of those columns in the table whose data we want to read.

If you want to access all rows from all fields of the table, use the following SQL SELECT syntax with * asterisk sign:

SELECT * FROM table_name;

Select specific columns / in arbitrary order.

SELECT c1, c2, c3 FROM table;
Column alias
SELECT c1 AS col1, c2 col2 FROM table;
Computed columns.
SELECT c1, c2, c3, expr1, expr2 FROM table;SELECT c1,
CASE WHEN condition1 THEN value1,
WHEN condition2 THEN value2,
...
ELSE valuen
END
FROM table;
Distinct values in column.
SELECT DISTINCT c1 FROM table;
SELECT DISTINCT c1, c2 FROM table;
Select limited rows.
SELECT * FROM table LIMIT n;
SELECT * FROM table LIMIT m, n;

The SQL LIKE Operator (Syntax is Same in MySql & Oracle)

The LIKE operator is used in a WHERE clause to search for a specified pattern in a column.

There are two wildcards often used in conjunction with the LIKE operator:

- The percent sign (%) represents zero, one, or multiple characters
- The underscore sign (_) represents one, single character

Note: MS Access uses an asterisk (*) instead of the percent sign (%), and a question mark (?) instead of the underscore (_).

The percent sign and the underscore can also be used in combinations!

LIKE Syntax

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

Tip: **You can also combine any number of conditions using AND or OR operators.**

Here are some examples showing different LIKE operators with '%' and '_' wildcards:

LIKE Operator	Description
WHERE CustomerName LIKE 'a%'	Finds any values that start with "a"
WHERE CustomerName LIKE '%a'	Finds any values that end with "a"
WHERE CustomerName LIKE '%or%'	Finds any values that have "or" in any position
WHERE CustomerName LIKE '_r%'	Finds any values that have "r" in the second position
WHERE CustomerName LIKE 'a_%'	Finds any values that start with "a" and are at least 2 characters in length

WHERE CustomerName LIKE 'a__%'	Finds any values that start with "a" and are at least 3 characters in length
WHERE ContactName LIKE 'a%o'	Finds any values that start with "a" and ends with "o"

SELECT – DQL – ORDER BY (Syntax is Same in MySql & Oracle)

The ORDER BY keyword is used to sort the result-set in ascending or descending order.

The ORDER BY keyword sorts the records in ascending order by default. To sort the records in descending order, use the DESC keyword.

ORDER BY Syntax

SELECT *column1*, *column2*, ...
FROM *table_name*
ORDER BY *column1*, *column2*, ... ASC|DESC;

In db rows are scattered on disk. Hence may not be fetched in a fixed order.
Select rows in asc order.
SELECT * FROM table ORDER BY c1;
SELECT * FROM table ORDER BY c2 ASC;
Select rows in desc order.
SELECT * FROM table ORDER BY c3 DESC;
Select rows sorted on multiple columns.
SELECT * FROM table ORDER BY c1, c2;
SELECT * FROM table ORDER BY c1 ASC, c2 DESC;
SELECT * FROM table ORDER BY c1 DESC, c2 DESC;

Select top or bottom n rows using 'LIMIT'.(works in mysql only)

SELECT * FROM table ORDER BY c1 ASC LIMIT n;
SELECT * FROM table ORDER BY c1 DESC LIMIT n;
SELECT * FROM table ORDER BY c1 ASC LIMIT m, n;

SELECT – DQL – WHERE (Syntax is Same in MySql & Oracle)

The WHERE clause is used to filter records.

It is used to extract only those records that fulfill a specified condition.

WHERE Syntax

SELECT *column1*, *column2*, ...
FROM *table_name*
WHERE *condition*;

Note: The WHERE clause is not only used in SELECT statements, it is also used in UPDATE, DELETE, etc.!

The following SQL statement selects all the customers from the country "Mexico", in the "Customers" table:

SELECT * FROM Customers
WHERE Country='Mexico';

The SQL CREATE TABLE Statement (Syntax is Same in MySql & Oracle)

The CREATE TABLE statement is used to create a new table in a database.

Syntax

CREATE TABLE *table_name* (
  column1 datatype,
  column2 datatype,
  column3 datatype,....);

The column parameters specify the names of the columns of the table.
The datatype parameter specifies the type of data the column can hold (e.g. varchar, integer, date, etc.).

SQL CREATE TABLE Example
The following example creates a table called "Persons" that contains five columns: PersonID, LastName, FirstName, Address, and City:

```
CREATE TABLE Persons (
  PersonID int,
  LastName varchar(255),
  FirstName varchar(255),
  Address varchar(255),
  City varchar(255)
);
```

The SQL INSERT INTO Statement (Syntax is Same in MySql & Oracle)

The INSERT INTO statement is used to insert new records in a table.

INSERT INTO Syntax

It is possible to write the INSERT INTO statement in two ways:

1. Specify both the column names and the values to be inserted:

INSERT INTO *table_name* (*column1*, *column2*, *column3*, ...)
VALUES (*value1*, *value2*, *value3*, ...);
2. If you are adding values for all the columns of the table, you do not need to specify the column names in the SQL query. However, make sure the order of the values is in the same order as the columns in the table. Here, the INSERT INTO syntax would be as follows:

INSERT INTO *table_name*
VALUES (*value1*, *value2*, *value3*, ...);

Example:

INSERT INTO Customers (CustomerName, ContactName, Address, City, PostalCode, Country)
VALUES ('Cardinal', 'Tom B. Erichsen', 'Skagen 21', 'Stavanger', '4006', 'Norway');
It is also possible to only insert data in specific columns.

The following SQL statement will insert a new record, but only insert data in the "CustomerName", "City", and "Country" columns (CustomerID will be updated automatically):

INSERT INTO Customers (CustomerName, City, Country)
VALUES ('Cardinal', 'Stavanger', 'Norway');

The SQL UPDATE Statement (Syntax is Same in MySql & Oracle)

The UPDATE statement is used to modify the existing records in a table.

UPDATE Syntax

UPDATE *table_name*
SET *column1 = value1, column2 = value2, ...*
WHERE *condition*;
Note: Be careful when updating records in a table! Notice the WHERE clause in the UPDATE statement. The WHERE clause specifies which record(s) that should be updated. If you omit the WHERE clause, all records in the table will be updated!

The SQL DELETE Statement (Syntax is Same in MySql & Oracle)

The DELETE statement is used to delete existing records in a table.

DELETE Syntax

DELETE FROM *table_name* WHERE *condition*;
Note: Be careful when deleting records in a table! Notice the WHERE clause in the DELETE statement. The WHERE clause specifies which record(s) should be deleted. If you omit the WHERE clause, all records in the table will be deleted!

The following SQL statement deletes the customer "Alfreds Futterkiste" from the "Customers" table:

DELETE FROM Customers WHERE CustomerName='Alfreds Futterkiste'

SQL functions

SQL Aggregate Functions (Syntax is Same in MySql & Oracle)

SQL aggregate functions return a single value, calculated from values in a column.

Useful aggregate functions:

- AVG() - Returns the average value

SELECT AVG(*column_name*)
FROM *table_name*

WHERE *condition*;

- COUNT() - Returns the number of rows

SELECT COUNT(*column_name*)
FROM *table_name*
WHERE *condition*;

- FIRST() - Returns the first value
- LAST() - Returns the last value
- MAX() - Returns the largest value

SELECT MIN(*column_name*)
FROM *table_name*
WHERE *condition*;

- MIN() - Returns the smallest value

SELECT MAX(*column_name*)
FROM *table_name*
WHERE *condition*;

- SUM() - Returns the sum

SELECT SUM(*column_name*)
FROM *table_name*
WHERE *condition*;

SQL Scalar functions

SQL scalar functions return a single value, based on the input value.

- UCASE() - Converts a field to upper case
- LCASE() - Converts a field to lower case
- MID() - Extract characters from a text field
- LEN() - Returns the length of a text field
- ROUND() - Rounds a numeric field to the number of decimals specified
- NOW() - Returns the current system date and time
- FORMAT() - Formats how a field is to be displayed

Date-Time and Information functions (Syntax is Same in MySql & Oracle)

SYSDATE()
The Oracle SYSDATE function allows you to easily output the current date. It shows the date and time of the database server.
An example of the SYSDATE function is:
SELECT SYSDATE
FROM dual;
Result:
10/SEP/22

Round()
The ROUND function allows you to round a date value to a format you specify.
This function is often used with numbers, but can also be used with dates.
If you use it with a date value, you can specify a DATE or TIMESTAMP value. You can specify any

format mask, but the default is the nearest day, and is returned as a DATE.
The syntax is:
ROUND (input_date, round_to)
If I wanted to round a date to the nearest month, I would use something like this:
SELECT
ROUND(SYSDATE, 'MM')
FROM dual;
Result:
01/SEP/22

TRUNC()
The TRUNC function, like the round function, works with numbers as well as dates.
It truncates or removes a part of the date to the format you specify.
The syntax is:
TRUNC (input_date, format_mask)
If you don't specify a format mask, then the function will truncate the value to the nearest day. This is helpful if you want to remove the time part of a date value.
For example, to show only the date part of today's date:
SELECT
TRUNC(SYSDATE)
FROM dual;
Result:
10/SEP/22
This shows only the date part of today.
Or, you can show only the YEAR:
SELECT
TRUNC(SYSDATE, 'YYYY')
FROM dual;
Result:
01/JAN/22

EXTRACT()

The EXTRACT function in Oracle extracts a specific part of a date from a date or interval value.
This means that it can get the month, or year, for example, from a DATE value. I think it's easier than using a conversion function such as TO_CHAR.
The EXTRACT function looks like this:
EXTRACT (date_component FROM expression)
The date_component is a keyword that represents the part of the date to extract, such as MONTH, DAY, YEAR, or HOUR.
The expression is a value or column that is a date or interval data type.
An example of this function is:
SELECT
SYSDATE,
EXTRACT(MONTH FROM SYSDATE) AS extract_month
FROM dual;

SYSDATE	EXTRACT_MONTH
10/SEP/22	9

This shows the month part of the SYSDATE.

TO_DATE()
The TO_DATE function allows you to convert a character value into a date value. It converts any character data type (CHAR, VARCHAR2, NCHAR, or NVARCHAR2) into a DATE type.
It's useful if you have a date in a particular format in a text format or text column, and you need it in a DATE format (for a function or to insert into a column, for example).
The syntax is:
TO_DATE (date_text [, format_mask] [, nls_date_language])
The date_text is the date you want to convert, which is in some kind of text or character format. You can optionally specify the format mask (which is the format that this date value was provided in), and the nls_date_langauge is used for dates in different languages or countries.
An example of this function is:
SELECT
TO_DATE('21-JAN-2022', 'DD-MON-YYYY')
FROM dual;
Result:
21/JAN/22
This shows the specified date (21 Jan 2022) converted to a date format. It might look the same in your IDE, but that's just how dates are displayed. The value started as a character value and is converted to a date value.
If your date is in a different format:
SELECT
TO_DATE('20220115_142309', 'YYYYMMDD_HH24MISS')
FROM dual;
Result:
15/JAN/22
This example shows a completely different format, but it can still be converted to a date format. We have specified the format mask here, which is the format that the first parameter is in.
In many Oracle functions that deal with dates, such as TO_CHAR, you can specify a "format mask". This is a parameter that lets you specify a certain combination of characters, which allows Oracle to translate into a specific format for a date.

TO_CHAR:-

It is used to convert a date from DATE value to a specified date format.
Syntax:
TO_CHAR(expression, date_format)
Parameters:
expression: It refers to the DATE or an INTERVAL value which needs to be converted. The expression can be of type DATE OR TIMESTAMP
date_format: It refers to the specified format in which we are going to convert the expression. It is optional parameter.
Example:
In this example we are going to convert the system date or current date into a string value in a format DD-MM-YYYY.
Code:

```
SELECT
TO_CHAR( sysdate, 'DD-MM-YYYY' )NEW_DATE
FROM
dual;
```

ADD_MONTHS():-

This function adds N months to a date and returns the same day N month after.

Syntax:

ADD_MONTHS(expression, N)

Parameters:

expression: It refers to the date value.

N: It represents the number of months.

Example:

To get the today system day date after 2 months using the ADD_MONTHS function.

Code:

```
SELECT
ADD_MONTHS( sysdate, 2 ) NEWDATE
FROM
dual;
```

Date Format Parameters:

You can use these characters to change the way that dates are formatted.

A full list of the date format parameters is shown here:

Year

Parameter	Explanation
YEAR	Year, spelled out in full words
YYYY	4-digit year
YYY	Last 3 digits of year
YY	Last 2 digits of year
Y	Last digit of year
IYY	Last 3 digits of ISO year
IY	Last 2 digits of ISO year
I	Last digit of ISO year
IYYY	4-digit year, which is based on the ISO standard
RRRR	This format accepts a 2-digit year, and returns a 4-digit year. If the provided value is between 0 and 49, it will return a year greater than or equal to 2000. If the provided value is between 50 and 99, it will return a year less than 2000

Month

Parameter	Explanation
Q	Quarter of year, from 1 to 4. JAN to MAR = 1
MM	Month, from 01 to 12. JAN = 01
MON	Abbreviated name of month.
MONTH	Name of month, padded with blanks to length of 9 characters.
RM	Roman numeral month, from I to XII. JAN = I.

Week

Parameter	Explanation
WW	Week of year, from 1 to 53. Week 1 starts on the first day of the year, and continues to the seventh day of the year.
W	Week of month, from 1 to 5. Week 1 starts on the first day of the month and ends on the seventh.
IW	Week of year, from 1 to 52 or 1 to 53, based on the ISO standard.

Day

Parameter	Explanation
D	Day of week, from 1 to 7.
DAY	Name of day.
DD	Day of month, from 1 to 31.
DDD	Day of year, from 1 to 366.
DY	Abbreviated name of day.
J	Julian day, which is the number of days since January 1, 4712 BC.

Time

Parameter	Explanation
HH	Hour of day, from 1 to 12.
HH12	Hour of day, from 1 to 12.
HH24	Hour of day, from 0 to 23.
MI	Minute, from 0 to 59
SS	Second, from 0 to 59
SSSSS	Seconds past midnight, from 0 to 86399.
FF	Fractional seconds. This uses a value from 1 to 9 after FF, to indicate the number of digits in the fractional seconds (e.g. FF7)

Indicators

Parameter	Explanation
AM, A.M., PM, or P.M.	Meridian indicator
AD or A.D	AD indicator

BC or B.C.	BC indicator
TZD	Daylight savings information
TZH	Time zone hour.
TZM	Time zone minute.
TZR	Time zone region.

GROUP BY Clause (Syntax is Same in MySql & Oracle)

In Oracle GROUP BY clause is used with SELECT statement to collect data from multiple records and group the results by one or more columns.

Syntax:

SELECT expression1, expression2, ... expression_n,
aggregate_function (aggregate_expression)
FROM tables
WHERE conditions
GROUP BY expression1, expression2, ... expression_n;

Parameters:

expression1, expression2, ... expression_n: It specifies the expressions that are not encapsulated within aggregate function. These expressions must be included in GROUP BY clause.

aggregate_function: It specifies the aggregate functions i.e. SUM, COUNT, MIN, MAX or AVG functions.

aggregate_expression: It specifies the column or expression on that the aggregate function is based on.

tables: It specifies the table from where you want to retrieve records.

conditions: It specifies the conditions that must be fulfilled for the record to be selected.

JOINS

Join is a query that is used to combine rows from two or more tables, views, or materialized views. It retrieves data from multiple tables and creates a new table.

Join Conditions

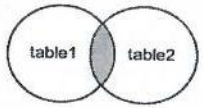
There may be at least one join condition either in the FROM clause or in the WHERE clause for joining two tables. It compares two columns from different tables and combines pair of rows, each containing one row from each table, for which join condition is true.

Types of Joins

- Inner Joins (Simple Join)
- Outer Joins
  - Left Outer Join (Left Join)
  - Right Outer Join (Right Join)
  - Full Outer Join (Full Join)
- Equijoins
- Self Joins
- Cross Joins (Cartesian Products)

INNER JOIN
Inner Join is the simplest and most common type of join. It is also known as simple join. It returns all rows from multiple tables where the join condition is met.

Syntax
SELECT columns
FROM table1
INNER JOIN table2
ON table1.**column** = table2.**column**;
Image representation of Inner Join



Example:

EDIT	SUPPLIER_ID	SUPPLIER_NAME	SUPPLIER_ADDRESS
	1	Bata shoes	Agra
	2	Kingfisher	Delhi
	3	VOLO	Lucknow

row(s) 1 - 3 of 3

EDIT	ORDER_NUMBER	SUPPLIER_ID	CITY
	101	1	Allahabad
	102	2	Kanpur

row(s) 1 - 3 of 3

```
SELECT suppliers.supplier_id, suppliers.supplier_name, order1.order_number
FROM suppliers
INNER JOIN order1
ON suppliers.supplier_id = order1.supplier_id;
```

Output:

SUPPLIER_ID	SUPPLIER_NAME	ORDER_NUMBER
1	Bata shoes	101
2	Kingfisher	102

2 rows returned in 0.03 seconds

Outer Join

- If a row does not satisfy a JOIN condition, then the row will not appear in thequery result.
- Syntax **(works only in oracle)**
  - ❖ The missing row(s) can be returned by using OUTER JOIN operator in the JOIN condition.
  - ❖ The operator is PLUS sign enclosed in parentheses (+), and is placed on the side of the join(table), which is deficient in information.
  - ❖ Table1.column = table2.column (+) means OUTER join is taken on table1.

    The (+) sign must be kept on the side of the join that is deficient in information

    Depending on the position of the outer join (+), it can be denoted as Left Outeror Right outer Join

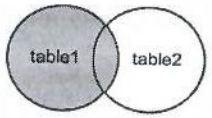
Left Outer Join (works in oracle and Mysql)

Left Outer Join returns all rows from the left (first) table specified in the ON condition and only those rows from the right (second) table where the join condition is met.

Syntax

1. **SELECT** columns
2. **FROM** table1
3. LEFT [OUTER] JOIN table2
4. **ON** table1.**column** = table2.**column**;

Image representation of left outer join



Example:

Consider above supplier and order table

```
SELECT suppliers.supplier_id, suppliers.supplier_name, order1.order_number
FROM suppliers
LEFT OUTER JOIN order1
ON suppliers.supplier_id = order1.supplier_id;
```

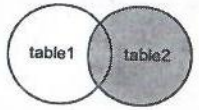

Right Outer Join

The Right Outer Join returns all rows from the right-hand table specified in the ON condition and only those rows from the other table where the join condition is met.

Syntax

```
SELECT columns
FROM table1
RIGHT [OUTER] JOIN table2
ON table1.column = table2.column;
```

Image representation of Right Outer Join



Example:

Consider above supplier and order table

```
SELECT order1.order_number, order1.city, suppliers.supplier_name
FROM suppliers
RIGHT OUTER JOIN order1
ON suppliers.supplier_id = order1.supplier_id;
```

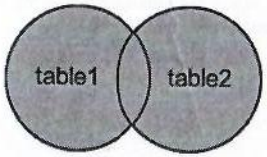
Full Outer Join

The Full Outer Join returns all rows from the left hand table and right hand table. It places NULL where the join condition is not met.

Syntax

```
SELECT columns
FROM table1
FULL [OUTER] JOIN table2
ON table1.column = table2.column;
```

Image representation of Full Outer Join



Example:

Consider above supplier and order table

```
SELECT suppliers.supplier_id, suppliers.supplier_name, order1.order_number
FROM suppliers
FULL OUTER JOIN order1
ON suppliers.supplier_id = order1.supplier_id;
```

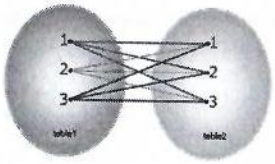
Cross Join:-

The CROSS JOIN specifies that all rows from first table join with all of the rows of second table. If there are "x" rows in table1 and "y" rows in table2 then the cross join result set have x*y rows. It normally happens when no matching join columns are specified.
In simple words you can say that if two tables in a join query have no join condition, then the Oracle returns their Cartesian product.

Syntax

```
SELECT *
FROM table1
CROSS JOIN table2;
```

Image representation of cross join



EQUI JOIN

Oracle Equi join returns the matching column values of the associated tables. It uses a comparison operator in the WHERE clause to refer equality.

Syntax

```
1. SELECT column_list
2. FROM table1, table2....
3. WHERE table1.column_name =
4. table2.column_name;
```

Equijoin also can be performed by using JOIN keyword followed by ON keyword and then specifying names of the columns along with their associated tables to check equality.

SELF JOIN

Self Join is a specific type of Join. In Self Join, a table is joined with itself (Unary relationship). A self join simply specifies that each rows of a table is combined with itself and every other row of the table.

Syntax

```
SELECT a.column_name, b.column_name...
FROM table1 a, table1 b
WHERE a.common_filed = b.common_field;
```

Sub queries (Syntax is Same in MySql & Oracle)

Sub-query is query within query. Typically it work with SELECT statements.Output of inner query is used as input to outer query.
If no optimization is enabled, for each row of outer query result, sub-query isexecuted once. This reduce performance of sub-query.

Single row sub-query

Sub-query returns single row.
Usually it is compared in outer query using relational operators.

Multi-row sub-query

Sub-query returns multiple rows.
Usually it is compared in outer query using operators like IN, ANY or ALL.
IN operator compare for equality with results from sub-queries (at least one result should match)..
ANY operator compares with the results from sub-queries (at least one result should match).
ALL operator compares with the results from sub-queries (all results should match).

Correlated sub-query

If number of results from sub-query are reduced, query performance will increase.
This can be done by adding criteria (WHERE clause) in sub-query based on outer query row.
Typically correlated sub-query use IN, ALL, ANY and EXISTS operators.
Sub queries with UPDATE and DELETE are not supported in all RDBMS.
In MySQL, Sub-queries in UPDATE/DELETE is allowed, but sub-query should not SELECT from the same table, on which UPDATE/DELETE operation is in progress.

Syntax:

```
SELECT column_name
FROM table_name
WHERE column_name expression operator
  ( SELECT COLUMN_NAME from TABLE_NAME  WHERE ... );
```

Example 1: To display name of students from "Mechanics" department.

```
SELECT student_code, student_name FROM student_master
WHERE dept_code = (SELECT dept_code
FROM department_master WHERE dept_name = 'Mechanics');
```

Example 2: To display all staff details of who earn salary least salary

```
SELECT staff_name, staff_code, staff_sal FROM staff_master
WHERE staff_sal = (SELECT MIN(staff_sal)
FROM staff_master) ;
```

The SQL ANY Operator
returns a boolean value as a result
returns TRUE if ANY of the subquery values meet the condition
ANY means that the condition will be true if the operation is true for any of the values in the range.

ANY Syntax
SELECT *column_name(s)*
FROM *table_name*
WHERE *column_name operator* ANY
(SELECT *column_name*
FROM *table_name*
WHERE *condition*);

Note: The *operator* must be a standard comparison operator (=, <>, !=, >, >=, <, or <=).

Example 1: To display staff details who earn salary greater than average salary earned in dept 10

```
SELECT staff_code,staff_sal FROM staff_master
WHERE staff_sal > ANY(SELECT AVG(staff_sal)
FROM staff_master WHERE dept_code=10);
```

The SQL ALL Operator

returns a boolean value as a result
returns TRUE if ALL of the subquery values meet the condition
is used with SELECT, WHERE and HAVING statements
ALL means that the condition will be true only if the operation is true for all values in the range.

ALL Syntax With SELECT
SELECT ALL *column_name(s)*
FROM *table_name*
WHERE *condition*;

ALL Syntax With WHERE or HAVING
SELECT *column_name(s)*
FROM *table_name*
WHERE *column_name operator* ALL
(SELECT *column_name*
FROM *table_name*
WHERE *condition*);
Note: The *operator* must be a standard comparison operator (=, <>, !=, >, >=, <, or <=).

VIEWS:

RDBMS view represents view (projection) of the data.View is based on SELECT statement.
Typically it is restricted view of the data (limited rows or columns) from one or moretables (joins and/or sub-queries) or summary of the data (grouping).Data of view is not stored on server hard-disk; but its SELECT statement is stored incompiled form. It speed up execution of view.

Views are of two types: Simple view and Complex view
Usually if view contains computed columns, group by, joins or sub-queries, then the views are said to be complex. DML operations are not supported on these views.DML operations on view affects underlying table.View can be created with CHECK OPTION to ensure that DML operations can beperformed only the data visible in that view.
Views can be differentiated with: SHOW FULL TABLES.
Views can be dropped with DROP VIEW statement.
View can be based on another view.
Applications of views
Security: Providing limited access to the data.
Hide source code of the table.
Simplifies complex queries.
In SQL, a view is a virtual table based on the result-set of an SQL statement.A view contains rows and columns, just like a real table. The fields in a view are fields from one or more real tables in the database. You can add SQL statements and functions to a view and present the data as if the data were coming from one single table.A view is created with the CREATE VIEW statement.

CREATE VIEW Syntax
CREATE VIEW *view_name* AS
SELECT *column1*, *column2*, ...
FROM *table_name*
WHERE *condition*;
Note: A view always shows up-to-date data! The database engine recreates the view, every time a user queries it.

SQL CREATE VIEW Examples

```
CREATE VIEW [Brazil Customers] AS
SELECT CustomerName, ContactName
FROM Customers
WHERE Country = 'Brazil';
```

SQL Updating a View

A view can be updated with the CREATE OR REPLACE VIEW statement.

SQL CREATE OR REPLACE VIEW Syntax

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

The following SQL adds the "City" column to the "Brazil Customers" view:

CREATE OR REPLACE VIEW [Brazil Customers] AS
SELECT CustomerName, ContactName, City
FROM Customers
WHERE Country = 'Brazil';

SQL Dropping a View

A view is deleted with the DROP VIEW statement.

SQL DROP VIEW Syntax
DROP VIEW *view_name*;
The following SQL drops the "Brazil Customers" view:
DROP VIEW [Brazil Customers];

INDEX

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 discusses a certain topic, you first refer to the index, which lists all the topics alphabetically and are then referred to one or more specific page numbers.

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

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 an ascending or descending order.

Indexes can also be unique, like the **UNIQUE** constraint, in that the index prevents duplicate entries in the column or combination of columns on which there is an index.

The CREATE INDEX Command

CREATE [UNIQUE] INDEX index_name ON table_name(col_name1 [ASC|DESC],col_name2,.....)

Single-Column Indexes

A single-column index 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. Its 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 the performance may either slow down or improve.

DROP INDEX index_name;
You can check the INDEX Constraint chapter to see some 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 updates 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.

CONSTRAINTS: (Oracle)

Constraints are restrictions imposed on columns.

Superkey:

An attribute, or set of attributes, that uniquely identifies each entityoccurrence.

If we add additional attributes to a primary key, the resulting combination would still uniquely identify an instance of the entity set. Such augmentedkeys are called superkey.

Candidate Key:

A superkey that contains only the minimum number of attributes necessaryfor unique identification of each entity occurrence. Ex. BranchNo in entity Branch.

The minimal super key is also referred as candidate key.

Primary Key: The candidate key that is selected to uniquely identify eachoccurrence of an entity type.E.g: National Insurance Number.

Alternate keys: The candidate keys that are not selected as the primary key ofthe entity.

Composite Key: A candidate key that consist of two or more attributes.

Few constraints can be applied at either column level or table level. Few constraints can be applied on both.
Optionally constraint names can be mentioned while creating the constraint. If notgiven, it is auto-generated.
Each DML operation check the constraints before manipulating the values. If anyconstraint is violated, error is raised.

NOT NULL
NULL values are not allowed.
Can be applied at column level only.
CREATE TABLE table(c1 TYPE NOT NULL, …);
UNIQUE
Duplicate values are not allowed.
NULL values are allowed.
Not applicable for TEXT and BLOB.

UNIQUE
can be applied on one or more columns.Internally creates unique index on the column (fast searching).
A table can have one or more unique keys.Can be applied at column level or table level.
CREATE TABLE table(c1 TYPE UNIQUE, …);
CREATE TABLE table(c1 TYPE, …, UNIQUE(c1));
CREATE TABLE table(c1 TYPE, …, CONSTRAINT constraint_name UNIQUE(c1));

PRIMARY KEY
Column or set of columns that uniquely identifies a row.Only one primary key is allowed for a table.
Primary key column cannot have duplicate or NULL values.Internally index is created on PK column.
TEXT/BLOB cannot be primary key.If no obvious choice available for PK, composite or surrogate PK can be created.
Creating PK for a table is a good practice.
PK can be created at table level or column level.
CREATE TABLE table(c1 TYPE PRIMARY KEY, …);
CREATE TABLE table(c1 TYPE, …, PRIMARY KEY(c1));
CREATE TABLE table(c1 TYPE, …, CONSTRAINT constraint_name PRIMARY KEY(c1));
CREATE TABLE table(c1 TYPE, c2 TYPE, …, PRIMARY KEY(c1, c2));

Rules for Primary key

Following are the rules for the primary key:

1. The primary key column value must be unique.
2. Each table can contain only one primary key.
3. The primary key column cannot be null or empty.
4. MySQL does not allow us to insert a new row with the existing primary key.
5. It is recommended to use INT or BIGINT data type for the primary key column.

We can create a primary key in two ways:

- CREATE TABLE Statement
- ALTER TABLE Statement

Primary Key Using CREATE TABLE Statement

In this section, we are going to see how a primary key is created using the CREATE TABLE statement.

Syntax
The following are the syntax used to create a primary key in MySQL.
If we want to create only one primary key column into the table, use the below syntax:

```
CREATE TABLE table_name(
  col1 datatype PRIMARY KEY,
  col2 datatype,.....);
```

If we want to create more than one primary key column into the table, use the below syntax:

```
CREATE TABLE table_name
( col1 col_definition,
 col2 col_definition, .....
 CONSTRAINT [constraint_name]
  PRIMARY KEY (column_name(s)) );
```

Parameter Explanation

Parameter Name	Descriptions
Table_name	It is the name of the table that we are going to create.
Col1, col2	It is the column names that contain in the table.
Constraint_name	It is the name of the primary key.
Column_name(s)	It is the column name(s) that is going to be a primary key.

FOREIGN KEY

The FOREIGN KEY constraint is used to prevent actions that would destroy links between tables.A FOREIGN KEY is a field (or collection of fields) in one table, that refers to the PRIMARY KEY in another table.The table with the foreign key is called the child table, and the table with the primary key is called the referenced or parent table.
Look at the following two tables:

Persons Table

PersonID	LastName	FirstName	Age
1	Hansen	Ola	30
2	Svendson	Tove	23
3	Pettersen	Kari	20

Orders Table

OrderID	OrderNumber	PersonID
1	77895	3
2	44678	3
3	22456	2
4	24562	1

Notice that the "PersonID" column in the "Orders" table points to the "PersonID" column in the "Persons" table.
The "PersonID" column in the "Persons" table is the PRIMARY KEY in the "Persons" table.
The "PersonID" column in the "Orders" table is a FOREIGN KEY in the "Orders" table.
The FOREIGN KEY constraint prevents invalid data from being inserted into the foreign key column, because it has to be one of the values contained in the parent table.

SQL FOREIGN KEY on CREATE TABLE

The following SQL creates a FOREIGN KEY on the "PersonID" column when the "Orders" table is created:

```
CREATE TABLE Orders (
  OrderID int NOT NULL,
  OrderNumber int NOT NULL,
  PersonID int,
  PRIMARY KEY (OrderID),
  FOREIGN KEY (PersonID) REFERENCES Persons(PersonID));
```

DDL – ALTER statement
Oracle Syntax:

ALTER statement is used to do modification into table, view, function, procedure,ALTER TABLE is used to change table structure.
Add new column(s) into the table.
ALTER TABLE table ADD col TYPE;
ALTER TABLE table ADD c1 TYPE, c2 TYPE;

Modify column of the table.
ALTER TABLE table MODIFY col NEW_TYPE;
Rename column.
ALTER TABLE table_name RENAME COLUMN old_col TO new_col;
Drop a column
ALTER TABLE DROP COLUMN col;
Rename table
ALTER TABLE table RENAME TO new_table;

Mysql Syntax:

1) ADD a column in the table
Syntax:

ALTER TABLE table_name
ADD new_column_name column_definition
[**FIRST** | **AFTER** column_name];

Parameters
table_name: It specifies the name of the table that you want to modify.

new_column_name: It specifies the name of the new column that you want to add to the table.
column_definition: It specifies the data type and definition of the column (NULL or NOT NULL, etc).
FIRST | AFTER column_name: It is optional. It tells MySQL where in the table to create the column. If this parameter is not specified, the new column will be added to the end of the table.
Example:
In this example, we add a new column "cus_age" in the existing table "cus_tbl".
Use the following query to do this:

ALTER TABLE cus_tbl
ADD cus_age **varchar**(40) NOT NULL;

2) Add multiple columns in the table

Syntax:

ALTER TABLE table_name
ADD new_column_name column_definition
[**FIRST** | **AFTER** column_name],
ADD new_column_name column_definition
[**FIRST** | **AFTER** column_name],

Example:
In this example, we add two new columns "cus_address", and cus_salary in the existing table "cus_tbl". cus_address is added after cus_surname column and cus_salary is added after cus_age column

Use the following query to do this:

ALTER TABLE cus_tbl
ADD cus_address **varchar**(100) NOT NULL

AFTER cus_surname,
ADD cus_salary **int**(100) NOT NULL
AFTER cus_age ;

3) MODIFY column in the table

The MODIFY command is used to change the column definition of the table.

Syntax:

ALTER TABLE table_name
MODIFY column_name column_definition
[**FIRST** | **AFTER** column_name];

Example:

In this example, we modify the column cus_surname to be a data type of varchar(50) and force the column to allow NULL values.
Use the following query to do this:
ALTER TABLE cus_tbl
MODIFY cus_surname **varchar**(50) NULL;

Window functions ORACLE :

Window functions applies aggregate and ranking functions over a particular window (set of rows). OVER clause is used with window functions to define that window. OVER clause does two things :

- Partitions rows into form set of rows. (PARTITION BY clause is used)
- Orders rows within those partitions into a particular order. (ORDER BY clause is used)

Note: If partitions aren't done, then ORDER BY orders all rows of table.

Aggregate Window Function :

Various aggregate functions such as SUM(), COUNT(), AVERAGE(), MAX(), MIN() applied over a particular window (set of rows) are called aggregate window functions.

Consider the following **employee** table :

Name	Age	Department	Salary
Ramesh	**20**	**Finance**	**50, 000**
Deep	25	Sales	30, 000
Suresh	22	Finance	50000
Ram	28	Finance	20, 000
Pradeep	22	Sales	20, 000

Example –
Find average salary of employees for each department and order employees within a department by age.

```
SELECT Name, Age, Department, Salary,
 AVERAGE(Salary) OVER( PARTITION BY Department) AS Avg_Salary
FROM employee
```

This outputs the following:

Name	Age	Department	Salary	Avg_Salary
Ramesh	20	Finance	50,000	40,000
Suresh	22	Finance	50000	40,000
Ram	28	Finance	20,000	40,000
Pradeep	22	Sales	20,000	25,000
Deep	25	Sales	30,000	25,000

Notice how all the average salaries in a particular window have the same value.

Let's consider another case:

```
SELECT Name, Age, Department, Salary,
 AVERAGE(Salary) OVER( PARTITION BY Department ORDER BY Age) AS Avg_Salary
FROM employee
```

Here we also order the records within the partition as per age values and hence the average values change as per the sorted order.
The output of above query will be :

Name	Age	Department	Salary	Avg_Salary
Ramesh	20	Finance	50, 000	50, 000
Suresh	22	Finance	50000	50, 000
Ram	28	Finance	20, 000	40, 000
Pradeep	22	Sales	20, 000	20, 000
Deep	25	Sales	30, 000	25, 000

Hence, we should be careful while adding order by clauses to window functions with aggregates.

Ranking Window Functions :
Ranking functions are, RANK(), DENSE_RANK(), ROW_NUMBER()

- **RANK() –**
  As the name suggests, the rank function assigns rank to all the rows within every partition. Rank is assigned such that rank 1 given to the first row and rows having same value are assigned same rank. For the next rank after two same rank values, one rank value will be skipped.
- **DENSE_RANK() –**
  It assigns rank to each row within partition. Just like rank function first row is assigned rank 1 and rows having same value have same rank. The difference between RANK() and DENSE_RANK() is that in DENSE_RANK(), for the next rank after two same rank, consecutive integer is used, no rank is skipped.

- **ROW_NUMBER() –**
  It assigns consecutive integers to all the rows within partition. Within a partition, no two rows can have same row number.

Note –
ORDER BY() should be specified compulsorily while using rank window functions.

Calculate row no., rank, dense rank of employees is employee table according to salary within each department.

```
SELECT
ROW_NUMBER() OVER (PARTITION BY Department ORDER BY Salary DESC)
AS emp_row_no, Name, Department, Salary,
RANK() OVER(PARTITION BY Department
ORDER BY Salary DESC) AS emp_rank,
DENSE_RANK() OVER(PARTITION BY Department
        ORDER BY Salary DESC)
        AS emp_dense_rank,
FROM employee
```

Functions:

FIRST_VALUE() :
It is an analytic function as the name suggests is used to provide the value of the first row in an ordered set of rows.
Example:
In this example we are going to look into the lowest age based on city in the table employee.
Code:

```
select employee_id ,age,city,
FIRST_VALUE(age)
OVER(PARTITION BY city
ORDER BY employee_id)FIRST_ from employee;
```

LAST_VALUE() :
It is also an analytical function which is used to get the value of the last row in an ordered set of rows.
Example:
In this example we will try to get the highest age based on the city in the employees table.
Code:

```
select employee_id ,age,city,
LAST_VALUE(age)
OVER(PARTITION BY city
ORDER BY employee_id
RANGE BETWEEN UNBOUNDED PRECEDING AND
UNBOUNDED FOLLOWING)HIGHEST_AGE
from employee;
```

The clause 'RANGE BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING'

means that the window frame starts at the first row and ends in the last row of the result set.

LEAD() :
It is a type of analytic function that allows us to access a following row from the current row based on an offset value without using self join.
Example:
In this example we are going to get the following age of the employees from the city of Delhi.

Code:

```
SELECT city,age,
LEAD(age) OVER (ORDER BY city) following_employee_age
FROM
employee
WHERE
city = 'Delhi',
```

LAG() :
It is a type of analytic function that allows us to access a prior row from the current row based on an offset value without using self join.
Example:

Code:

```
SELECT city,age,
LAG(age) OVER (ORDER BY city) following_employee_age FROM employee
WHERE
city = 'Delhi',
```

What is a Stored Procedure? (ORACLE syntax)

A stored procedure is a prepared SQL code that you can save, so the code can be reused over and over again.

So if you have an SQL query that you write over and over again, save it as a stored procedure, and then just call it to execute it.

You can also pass parameters to a stored procedure, so that the stored procedure can act based on the parameter value(s) that is passed.

Stored Procedure Syntax

```
CREATE [OR REPLACE ] PROCEDURE procedure_name (parameter_list)
IS
[declaration statements]
BEGIN
[execution statements]
END;
```

Execute a Stored Procedure

EXECUTE procedure_name(arguments);

OR

EXEC procedure_name(arguments);

Stored Procedure Example

The following procedure accepts a customer id and prints out the customer's contact information including first name, last name, and email:

```
CREATE OR REPLACE PROCEDURE print_contact(in_customer_id NUMBER)
IS
 r_contact contacts%ROWTYPE;
BEGIN
 SELECT *
 INTO r_contact
 FROM contacts
 WHERE customer_id = p_customer_id;
 dbms_output.put_line( r_contact.first_name || ' ' ||
 r_contact.last_name || '<' || r_contact.email ||'>' );

END;
```

Variable declaration

```
VARIABLE varname DATATYPE;
```

Stored Functions (ORACLE syntax)

A function is a subprogram that is used to return a single value. You must declare and define a function before invoking it. It can be declared and defined at a same time or can be declared first and defined later in the same block. CREATE [OR REPLACE] FUNCTION function_name

```
  [ (parameter [,parameter]) ]
RETURN return_datatype
IS | AS
 [declaration_section]
BEGIN
  executable_section
[EXCEPTION
  exception_section]
END [function_name];
```


Example of function

```
create or replace function adder(n1 in number, n2 in number)
return number
is
n3 number(8);
begin
n3 :=n1+n2;
return n3;
end;
/
```

Now write another program to **call the function**.

```
DECLARE
  n3 number(2);
BEGIN
  n3 := adder(11,22);
  dbms_output.put_line('Addition is: ' || n3);
END;
/
```

Output:

```
Addition is: 33
```

Triggers: (ORACLE syntax)

In Oracle, you can define procedures that are implicitly executed when an INSERT, UPDATE or DELETE statement is issued against the associated table. These procedures are called database triggers.

Use the `CREATE TRIGGER` statement to create and enable a **database trigger**, which is:

- A stored PL/SQL block associated with a table, a schema, or the database or
- An anonymous PL/SQL block or a call to a procedure implemented in PL/SQL or Java

Oracle Database automatically executes a trigger when specified conditions occur.

When you create a trigger, the database enables it automatically. You can subsequently disable and enable a trigger with the `DISABLE` and `ENABLE` clause of the `ALTER TRIGGER` or `ALTER TABLE` statement.

Example –
Suppose, we are adding a tuple to the 'Donors' table that is some person has donated blood. So, we can design a trigger that will automatically add the value of donated blood to the 'Blood_record' table.
Types of Triggers –

1. **BEFORE INSERT:** activated before data is inserted into the table.
2. **BEFORE UPDATE:** activated before data in the table is modified.
3. **BEFORE DELETE:** activated before data is deleted/removed from the table.

Examples showing implementation of Triggers:

This statement specifies that Oracle will fire this trigger **BEFORE** the INSERT/UPDATE or DELETE operation is executed.

```
CREATE [ OR REPLACE ] TRIGGER trigger_name
BEFORE INSERT or UPDATE or DELETE
 ON table_name
  [ FOR EACH ROW ]
DECLARE
  -- variable declarations
BEGIN
  -- trigger code
EXCEPTION
  WHEN ...
  -- exception handling
END;
```

Example:

```
CREATE OR REPLACE TRIGGER  "SUPPLIERS_T1"
BEFORE
insert or update or delete on "SUPPLIERS"
for each row
begin
when the person performs insert/update/delete operations into the table.
end;
/
ALTER TRIGGER  "SUPPLIERS_T1" ENABLE
/
```

1. **AFTER INSERT** activated after data is inserted into the table.
2. **AFTER UPDATE:** activated after data in the table is modified.
3. **AFTER DELETE:** activated after data is deleted/removed from the table.

This statement specifies that Oracle will fire this trigger **AFTER** the INSERT/UPDATE or DELETE operation is executed.

```
CREATE [ OR REPLACE ] TRIGGER trigger_name
AFTER INSERT or UPDATE or DELETE
 ON table_name

  [ FOR EACH ROW ]
DECLARE
  -- variable declarations

BEGIN

  -- trigger code

END;
```


Example:

```
CREATE OR REPLACE TRIGGER  "SUPPLIERS_T2"

AFTER

insert or update or delete on "SUPPLIERS"

for each row

begin

when the person performs insert/update/delete operations into the table.

end;

/

ALTER TRIGGER  "SUPPLIERS_T2" ENABLE

/
```

How to drop Trigger?

syntax

DROP TRIGGER trigger_name;

Example:

```
DROP TRIGGER SUPPLIERS_T1;
```

How to disable trigger?

```
ALTER TRIGGER trigger_name DISABLE;
```

Normalization

A large database defined as a single relation may result in data duplication. This repetition of data may result in:
Making relations very large.
It isn't easy to maintain and update data as it would involve searching many records in relation.
Wastage and poor utilization of disk space and resources.
The likelihood of errors and inconsistencies increases.
So to handle these problems, we should analyze and decompose the relations with redundant data into smaller, simpler, and well-structured relations that are satisfy desirable properties. Normalization is a process of decomposing the relations into relations with fewer attributes.

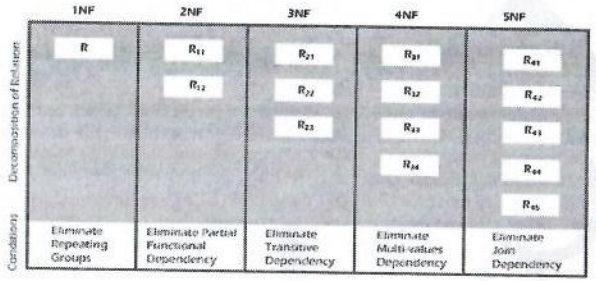
What is Normalization?

Normalization is the process of organizing the data in the database.
Normalization is used to minimize the redundancy from a relation or set of relations. It is also used to eliminate undesirable characteristics like Insertion, Update, and Deletion Anomalies.
Normalization divides the larger table into smaller and links them using relationships.
The normal form is used to reduce redundancy from the database table.

Data modification anomalies can be categorized into three types:
Insertion Anomaly: Insertion Anomaly refers to when one cannot insert a new tuple into a relationship due to lack of data.
Deletion Anomaly: The delete anomaly refers to the situation where the deletion of data results in the unintended loss of some other important data.
Updatation Anomaly: The update anomaly is when an update of a single data value requires multiple rows of data to be updated.

Types of Normal Forms:
Normalization works through a series of stages called Normal forms. The normal forms apply to individual relations. The relation is said to be in particular normal form if it satisfies constraints.

Following are the various types of Normal forms:



Normal Form	Description
1NF	A relation is in 1NF if it contains an atomic value.
2NF	A relation will be in 2NF if it is in 1NF and all non-key attributes are fully functional dependent on the primary key.
3NF	A relation will be in 3NF if it is in 2NF and no transition dependency exists.
BCNF	A stronger definition of 3NF is known as Boyce Codd's normal form.
4NF	A relation will be in 4NF if it is in Boyce Codd's normal form and has no multi-valued dependency.
5NF	A relation is in 5NF. If it is in 4NF and does not contain any join dependency, joining should be lossless.

Advantages of Normalization

- Normalization helps to minimize data redundancy.
- Greater overall database organization.
- Data consistency within the database.
- Much more flexible database design.
- Enforces the concept of relational integrity.

First Normal Form (1NF)

- A relation will be 1NF if it contains an atomic value.
- It states that an attribute of a table cannot hold multiple values. It must hold only single-valued attribute.
- First normal form disallows the multi-valued attribute, composite attribute, and their combinations.

Example: Relation EMPLOYEE is not in 1NF because of multi-valued attribute EMP_PHONE.

EMP_ID	EMP_NAME	EMP_PHONE	EMP_STATE
14	John	7272826385, 9064738238	UP
20	Harry	8574783832	Bihar
12	Sam	7390372389, 8589830302	Punjab

The decomposition of the EMPLOYEE table into 1NF has been shown below:

EMP_ID	EMP_NAME	EMP_PHONE	EMP_STATE
14	John	7272826385	UP
14	John	9064738238	UP
20	Harry	8574783832	Bihar
12	Sam	7390372389	Punjab
12	Sam	8589830302	Punjab

Second Normal Form (2NF)

In the 2NF, relational must be in 1NF.In the second normal form, all non-key attributes are fully functional dependent on the primary key
Example: Let's assume, a school can store the data of teachers and the subjects they teach. In a school, a teacher can teach more than one subject.

TEACHER table

TEACHER_ID	SUBJECT	TEACHER_AGE
25	Chemistry	30
25	Biology	30

47	English	35
83	Math	38
83	Computer	38

In the given table, non-prime attribute TEACHER_AGE is dependent on TEACHER_ID which is a proper subset of a candidate key. That's why it violates the rule for 2NF.
To convert the given table into 2NF, we decompose it into two tables:

TEACHER_DETAIL table:

TEACHER_ID	TEACHER_AGE
25	30
47	35
83	38

TEACHER_SUBJECT table:

TEACHER_ID	SUBJECT
25	Chemistry
25	Biology
47	English
83	Math
83	Computer

Third Normal Form (3NF)

- A relation will be in 3NF if it is in 2NF and not contain any transitive partial dependency.
- 3NF is used to reduce the data duplication. It is also used to achieve the data integrity.
- If there is no transitive dependency for non-prime attributes, then the relation must be in third normal form.A relation is in third normal form if it holds atleast one of the following conditions for every non-trivial function dependency $X \rightarrow Y$.
- X is a super key.
- Y is a prime attribute, i.e., each element of Y is part of some candidate key.

Example:EMPLOYEE_DETAIL table:

EMP_ID	EMP_NAME	EMP_ZIP	EMP_STATE	EMP_CITY
222	Harry	201010	UP	Noida
333	Stephan	02228	US	Boston
444	Lan	60007	US	Chicago
555	Katharine	06389	UK	Norwich
666	John	462007	MP	Bhopal

Super key in the table above:
{EMP_ID}, {EMP_ID, EMP_NAME}, {EMP_ID, EMP_NAME, EMP_ZIP}....so on
Candidate key: {EMP_ID}
Non-prime attributes: In the given table, all attributes except EMP_ID are non-prime.
Here, EMP_STATE & EMP_CITY dependent on EMP_ZIP and EMP_ZIP dependent on EMP_ID. The non-prime attributes (EMP_STATE, EMP_CITY) transitively dependent on super key(EMP_ID). It violates the rule of third normal form.
That's why we need to move the EMP_CITY and EMP_STATE to the new <EMPLOYEE_ZIP> table, with EMP_ZIP as a Primary key.

EMPLOYEE table:

EMP_ID	EMP_NAME	EMP_ZIP
222	Harry	201010
333	Stephan	02228
444	Lan	60007
555	Katharine	06389
666	John	462007

EMPLOYEE_ZIP table:

EMP_ZIP	EMP_STATE	EMP_CITY
201010	UP	Noida
02228	US	Boston
60007	US	Chicago
06389	UK	Norwich
462007	MP	Bhopal

Boyce Codd normal form (BCNF)
BCNF is the advance version of 3NF. It is stricter than 3NF.
A table is in BCNF if every functional dependency X → Y, X is the super key of the table.
For BCNF, the table should be in 3NF, and for every FD, LHS is super key.
Example: Let's assume there is a company where employees work in more than one department.

EMPLOYEE table:

EMP_ID	EMP_COUNTRY	EMP_DEPT	DEPT_TYPE	EMP_DEPT_NO
264	India	Designing	D394	283
264	India	Testing	D394	300
364	UK	Stores	D283	232
364	UK	Developing	D283	549

In the above table Functional dependencies are as follows:
EMP_ID → EMP_COUNTRY
EMP_DEPT → {DEPT_TYPE, EMP_DEPT_NO}
Candidate key: {EMP-ID, EMP-DEPT}
The table is not in BCNF because neither EMP_DEPT nor EMP_ID alone are keys.
To convert the given table into BCNF, we decompose it into three tables:

EMP_COUNTRY table:

EMP_ID	EMP_COUNTRY
264	India
264	India

EMP_DEPT table:

EMP_DEPT	DEPT_TYPE	EMP_DEPT_NO
Designing	D394	283
Testing	D394	300
Stores	D283	232
Developing	D283	549

EMP_DEPT_MAPPING table:

EMP_ID	EMP_DEPT
D394	283
D394	300
D283	232
D283	549

Functional dependencies:
EMP_ID → EMP_COUNTRY
EMP_DEPT → {DEPT_TYPE, EMP_DEPT_NO}
Candidate keys:
For the first table: EMP_ID
For the second table: EMP_DEPT
For the third table: {EMP_ID, EMP_DEPT}
Now, this is in BCNF because left side part of both the functional dependencies is a key.

Dnormalization

Normalization will yield a structure that is non-redundant.
Having too many inter-related tables will lead to complex and inefficient queries.
To ensure better performance of analytical queries, few rules of normalization can becompromised.This process is de-normalization.

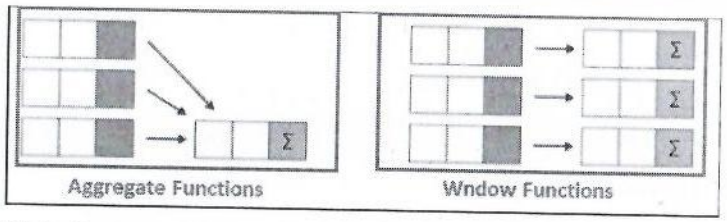
SQL Server Window Functions

We are all well-known for the regular aggregate function that performs calculations on the table and works with a GROUP BY clause. However, only a small percentage of SQL users use Window functions, and these are functions that work on a group of rows and display a single aggregated value for every row. This article will discuss in detail the window functions in SQL Server.

What is window functions?

Window functions are used to **perform a calculation on an aggregate value** based on a set of rows and return multiple rows for each group. The window word represents the group of rows on which the function will be operated. This function performs a calculation in the same way that the aggregate functions would perform. Unlike aggregate functions that operate on the entire table, Window functions do not give a result to be combined into a single row. It means that window functions work on a group of rows and return a total value for each row. As a result, each row retains its distinct identity.

The below pictorial representations explain the difference of aggregate function and window function in SQL Server:



Window Functions Types

1. **Aggregate Window Functions:** These functions operated on multiple rows and Examples of such functions are SUM(), MAX(), MIN(), AVG(), COUNT(), etc.
2. **Ranking Window Functions:** These functions ranks each row of a partition in a table. Example of such functions are RANK(), DENSE_RANK(), ROW_NUMBER(), NTILE(), etc.
3. **Value Window Functions:** These functions are locally represented by a power series. Example of such functions are LAG(), LEAD(), FIRST_VALUE(), LAST_VALUE(), etc.

Syntax

The following are the basic syntax for using a window function:

```
window_function_name([ALL] expression)
OVER (
  [partition_defintion]
  [order_definition]
)
```

Parameter Explanation

Let us understand the arguments used in the above syntax:

window_function: It indicates the name of your window function.

ALL: It is an optional keyword that is used to count all values along with duplicates. We cannot use the DISTINCT keyword in window functions.

Expression: It is the name of the column or expression on which window functions is operated. In other words, it is the column name for which we calculate an aggregated value.

OVER: It specifies the window clauses for aggregate functions. It mainly contains two expressions partition by and order by, and it always has an opening and closing parenthesis even there is no expression.

PARTITION BY: This clause divides the rows into partitions, and then a window function is operated on each partition. Here we need to provide columns for the partition after the PARTITION BY clause. If we want to specify more than one column, we must separate them by a comma operator. SQL Server will group the entire table when this clause is not specified, and values will be aggregated accordingly.

ORDER BY: It is used to specify the order of rows within each partition. When this clause is not defined, SQL Server will use the ORDER BY for the entire table.

PARTITION BY: This clause divides the rows into partitions, and then a window function is operated on each partition. Here we need to provide columns for the partition after the PARTITION BY clause. If we want to specify more than one column, we must separate them by a comma operator. SQL Server will group the entire table when this clause is not specified, and values will be aggregated accordingly.

ORDER BY: It is used to specify the order of rows within each partition. When this clause is not defined, SQL Server will use the ORDER BY for the entire table.

What is Data Warehousing?

Data warehousing is the process of constructing and using a data warehouse. A data warehouse is constructed by integrating data from multiple heterogeneous sources that support analytical reporting, structured and/or ad hoc queries, and decision making. Data warehousing involves data cleaning, data integration, and data consolidations.

Functions of Data Warehouse Tools and Utilities

The following are the functions of data warehouse tools and utilities –

- **Data Extraction** – Involves gathering data from multiple heterogeneous sources.
- **Data Cleaning** – Involves finding and correcting the errors in data.
- **Data Transformation** – Involves converting the data from legacy format to warehouse format.
- **Data Loading** – Involves sorting, summarizing, consolidating, checking integrity, and building indices and partitions.
- **Refreshing** – Involves updating from data sources to warehouse.

What is NoSQL Database

Databases can be divided in 3 types:

1. RDBMS (Relational Database Management System)
2. OLAP (Online Analytical Processing)
3. NoSQL (recently developed database)

NoSQL Database

NoSQL Database is used to refer a non-SQL or non relational database.
It provides a mechanism for storage and retrieval of data other than tabular relations model used in relational databases. NoSQL database doesn't use tables for storing data. It is generally used to store big data and real-time web applications.

History behind the creation of NoSQL Databases

In the early 1970, Flat File Systems are used. Data were stored in flat files and the biggest problems with flat files are each company implement their own flat files and there are no standards. It is very difficult to store data in the files, retrieve data from files because there is no standard way to store data.Then the relational database was created by E.F. Codd and these databases answered the question of having no standard way to store data. But later relational database also get a problem that it could not handle big data, due to this problem there was a need of database which can handle every types of problems then NoSQL database was developed.

Advantages of NoSQL

1. It supports query language.
2. It provides fast performance.
3. It provides horizontal scalability.

Difference between NoSQL and RDBMS

Basis of Comparison	NoSQL	RDBMS
Definition	Non-relational databases, often known as distributed databases, are another name for NoSQL databases.	RDBMS, which stands for Relational Database Management Systems, is the most common name for SQL databases.
Query	No declarative query language	SQL stands for Structured Query Language.
Scalability	NoSQL databases are horizontally scalable	RDBMS databases are vertically scalable
Design	NoSQL combines multiple database technologies. These databases were created in response to the application's requirements.	Traditional RDBMS systems use SQL syntax and queries to get insights from data. Different OLAP systems use them.
Speed	NoSQL databases use denormalization to optimise themselves. One record stores all the query data. This simplifies finding matched records, which speeds up queries.	Relational database models contain data in different tables; when running a query, you must integrate the information and set table-spanning restrictions. Because of so many tables, the database's query time is slow.

What is MongoDB

MongoDB is an open-source document database that provides high performance, high availability, and automatic scaling.

In simple words, you can say that - Mongo DB is a document-oriented database. It is an open source product, developed and supported by a company named 10gen.

MongoDB is available under General Public license for free, and it is also available under Commercial license from the manufacturer.

The manufacturing company 10gen has defined MongoDB as:

"MongoDB is a scalable, open source, high performance, document-oriented database." - 10gen

MongoDB was designed to work with commodity servers. Now it is used by the company of all sizes, across all industry.

History of MongoDB

The initial development of MongoDB began in 2007 when the company was building a platform as a service similar to window azure.
MongoDB was developed by a NewYork based organization named 10gen which is now known as MongoDB Inc. It was initially developed as a PAAS (Platform as a Service). Later in 2009, it is introduced in the market as an open source database server that was maintained and supported by MongoDB Inc.

Purpose of Building MongoDB
It may be a very genuine question that - "what was the need of MongoDB although there were many databases in action?
All the modern applications require big data, fast features development, flexible deployment, and the older database systems not competent enough, so the MongoDB was needed.
The primary purpose of building MongoDB is:

- Scalability
- Performance
- High Availability
- Scaling from single server deployments to large, complex multi-site architectures.
- Key points of MongoDB
- Develop Faster
- Deploy Easier
- Scale Bigger

Example of Document-Oriented Database
MongoDB is a document-oriented database. It is a key feature of MongoDB. It offers a document-oriented storage. It is very simple you can program it easily.
MongoDB stores data as documents, so it is known as document-oriented database.

FirstName = "John",
Address = "Detroit",
Spouse = [{Name: "Angela"}].
FirstName ="John",
Address = "Wick"

Features of MongoDB
These are some important features of MongoDB:
1. Support ad hoc queries

In MongoDB, you can search by field, range query and it also supports regular expression searches.

2. Indexing

You can index any field in a document.

3. Replication

MongoDB supports Master Slave replication.

A master can perform Reads and Writes and a Slave copies data from the master and can only be used for reads or back up (not writes)

4. Duplication of data

MongoDB can run over multiple servers. The data is duplicated to keep the system up and also keep its running condition in case of hardware failure.

5. Load balancing

It has an automatic load balancing configuration because of data placed in shards.

It also supports:

- JSON data model with dynamic schemas
- Auto-sharding for horizontal scalability
- Built in replication for high availability
- Now a day many companies using MongoDB to create new types of applications, improve performance and availability.

SQL vs Document Databases

SQL databases are considered relational databases. They store related data in separate tables. When data is needed, it is queried from multiple tables to join the data back together.

MongoDB is a document database which is often referred to as a non-relational database. This does not mean that relational data cannot be stored in document databases. It means that relational data is stored differently. A better way to refer to it is as a non-tabular database.

MongoDB stores data in flexible documents. Instead of having multiple tables you can simply keep all of your related data together. This makes reading your data very fast.

You can still have multiple groups of data too. In MongoDB, instead of tables these are called collections.

MongoDB Query API Uses

You can use the MongoDB Query API to perform:

- Adhoc queries with mongosh, Compass, VS Code, or a MongoDB driver for the programming language you use.
- Data transformations using aggregation pipelines.
- Document join support to combine data from different collections.
- Graph and geospatial queries.
- Full-text search.
- Indexing to improve MongoDB query performance.
- Time series analysis

Create Database using mongosh
After connecting to your database using mongosh, you can see which database you are using by typing db in your terminal.
If you have used the connection string provided from the MongoDB Atlas dashboard, you should be connected to the myFirstDatabase database.
Show all databases
To see all available databases, in your terminal type show dbs.
Notice that myFirstDatabase is not listed. This is because the database is empty. An empty database is

essentially non-existant.
Change or Create a Database
You can change or create a new database by typing use then the name of the database.
Example
Create a new database called "blog":
use blog
Create Collection using mongosh
There are 2 ways to create a collection.
Method 1
You can create a collection using the createCollection() database method.
Example
db.createCollection("posts")
Method 2
You can also create a collection during the insert process.
Example
We are here assuming object is a valid JavaScript object containing post data:
db.posts.insertOne(object)
This will create the "posts" collection if it does not already exist.

Insert Documents
There are 2 methods to insert documents into a MongoDB database.
insertOne()
To insert a single document, use the insertOne() method.
This method inserts a single object into the database.

```
db.posts.insertOne({
  title: "Post Title 1",
  body: "Body of post.",
  category: "News",
  likes: 1,
  tags: ["news", "events"],
  date: Date()
})
```

insertMany()
To insert multiple documents at once, use the insertMany() method.
This method inserts an array of objects into the database.
Example

```
db.posts.insertMany([
  {
    title: "Post Title 2",
    body: "Body of post.",
    category: "Event",
    likes: 2,
    tags: ["news", "events"],
    date: Date()
  },
  {
    title: "Post Title 3",
    body: "Body of post.",
    category: "Technology",
    likes: 3,
    tags: ["news", "events"],
    date: Date()
```

```
  },
  {
   title: "Post Title 4",
   body: "Body of post.",
   category: "Event",
   likes: 4,
   tags: ["news", "events"],
   date: Date()
  }
])
```

Find Data

There are 2 methods to find and select data from a MongoDB collection, find() and findOne().

find()

To select data from a collection in MongoDB, we can use the find() method.

This method accepts a query object. If left empty, all documents will be returned.

Example

```
db.posts.find()
```

findOne()

To select only one document, we can use the findOne() method.

This method accepts a query object. If left empty, it will return the first document it finds.

Note: This method only returns the first match it finds.

Example

```
db.posts.findOne()
```

Querying Data

To query, or filter, data we can include a query in our find() or findOne() methods.

Example

```
db.posts.find( {category: "News"} )
```

Projection

Both find methods accept a second parameter called projection.

This parameter is an object that describes which fields to include in the results.

Note: This parameter is optional. If omitted, all fields will be included in the results.

Example

This example will only display the title and date fields in the results.

```
db.posts.find({}, {title: 1, date: 1})
```

Update Document

To update an existing document we can use the updateOne() or updateMany() methods.

The first parameter is a query object to define which document or documents should be updated.

The second parameter is an object defining the updated data.

updateOne()

The updateOne() method will update the first document that is found matching the provided query.

Let's see what the "like" count for the post with the title of "Post Title 1":

Example

```
db.posts find( { title: "Post Title 1" } )
```

Now let's update the "likes" on this post to 2. To do this, we need to use the $set operator.

```
db.posts.updateOne( { title: "Post Title 1" }, { $set: { likes: 2 } } )
```

Check the document again and you'll see that the "like" have been updated.

Example

```
db.posts.find( { title: "Post Title 1" } )
```


Insert if not found

If you would like to insert the document if it is not found, you can use the upsert option.

Example

Update the document, but if not found insert it:

```
db.posts.updateOne(
 { title: "Post Title 5" },
 {
  $set:
   {
    title: "Post Title 5",
    body: "Body of post.",
    category: "Event",
    likes: 5,
    tags: ["news", "events"],
    date: Date()
   }
 },
 { upsert: true }
)
```

updateMany()

The updateMany() method will update all documents that match the provided query.

Example

Update likes on all documents by 1. For this we will use the $inc (increment) operator:

```
db.posts.updateMany({}, { $inc: { likes: 1 } })
```

Now check the likes in all of the documents and you will see that they have all been incremented by 1.

Delete Documents

We can delete documents by using the methods deleteOne() or deleteMany().

These methods accept a query object. The matching documents will be deleted.

deleteOne()

The deleteOne() method will delete the first document that matches the query provided.

Example

```
db.posts.deleteOne({ title: "Post Title 5" })
```

deleteMany()

The deleteMany() method will delete all documents that match the query provided.

Example

```
db.posts.deleteMany({ category: "Technology" })
```

Aggregation Pipelines

Aggregation operations allow you to group, sort, perform calculations, analyze data, and much more.

Aggregation pipelines can have one or more "stages". The order of these stages are important. Each stage acts upon the results of the previous stage.

Example

```
db.posts.aggregate([
 // Stage 1: Only find documents that have more than 1 like
 {
  $match: { likes: { $gt: 1 } }
 },
 // Stage 2: Group documents by category and sum each categories likes
 {
  $group: { _id: "$category", totalLikes: { $sum: "$likes" } }
 }])
```

Aggregation $group
This aggregation stage groups documents by the unique _id expression provided.
Don't confuse this _id expression with the _id ObjectId provided to each document.
Example
In this example, we are using the "sample_airbnb" database loaded from our sample data in the Intro to Aggregations section.

```
db.listingsAndReviews.aggregate(
   [ { $group : { _id : "$property_type" } } ]
)
```

Aggregation $limit
This aggregation stage limits the number of documents passed to the next stage.
Example
In this example, we are using the "sample_mflix" database loaded from our sample data in the Intro to Aggregations section.

```
db.movies.aggregate([ { $limit: 1 } ])
```

Aggregation $project
This aggregation stage passes only the specified fields along to the next aggregation stage.
This is the same projection that is used with the find() method.
Example
In this example, we are using the "sample_restaurants" database loaded from our sample data in the Intro to Aggregations section.

```
db.restaurants.aggregate([
  {
    $project: {
      "name": 1,
      "cuisine": 1,
      "address": 1 }},
{
   $limit: 5 }])
```

This will return the documents but only include the specified fields.
Notice that the _id field is also included. This field is always included unless specifically excluded.
We use a 1 to include a field and 0 to exclude a field.
Note: You cannot use both 0 and 1 in the same object. The only exception is the _id field. You should either specify the fields you would like to include or the fields you would like to exclude.

Aggregation $sort
This aggregation stage groups sorts all documents in the specified sort order.
Remember that the order of your stages matters. Each stage only acts upon the documents that previous stages provide.
Example
In this example, we are using the "sample_airbnb" database loaded from our sample data in the Intro to Aggregations section.

```
db.listingsAndReviews.aggregate([
  {  $sort: { "accommodates": -1 }
  },{$project: {
     "name": 1,
     "accommodates": 1}},
  {
    $limit: 5
  }
])
```


This will return the documents sorted in descending order by the accommodates field.
The sort order can be chosen by using 1 or -1. 1 is ascending and -1 is descending.

Aggregation $match
This aggregation stage behaves like a find. It will filter documents that match the query provided.
Using $match early in the pipeline can improve performance since it limits the number of documents the next stages must process.
Example
In this example, we are using the "sample_airbnb" database loaded from our sample data in the Intro to Aggregations section.

```
db.listingsAndReviews.aggregate([
  { $match : { property_type : "House" } },
  { $limit: 2 },
  { $project: {
    "name": 1,
    "bedrooms": 1,
    "price": 1
  }}])
```

This will only return documents that have the property_type of "House".
Aggregation $addFields
This aggregation stage adds new fields to documents.
Example
In this example, we are using the "sample_restaurants" database loaded from our sample data in the Intro to Aggregations section.

```
db.restaurants.aggregate([
  {$addFields: {avgGrade: { $avg: "$grades.score" }}}, {$project: { "name": 1,"avgGrade": 1 } },{$limit: 5}])
```

This will return the documents along with a new field, avgGrade, which will contain the average of each restaurants grades.score.
Aggregation $count
This aggregation stage counts the total amount of documents passed from the previous stage.

Example
In this example, we are using the "sample_restaurants" database loaded from our sample data in the Intro to Aggregations section.

```
db.restaurants.aggregate([
  {$match: { "cuisine": "Chinese" }},{ $count: "totalChinese" }])
```

This will return the number of documents at the $count stage as a field called "totalChinese".
Aggregation $lookup
This aggregation stage performs a left outer join to a collection in the same database.
There are four required fields:

- from: The collection to use for lookup in the same database
- localField: The field in the primary collection that can be used as a unique identifier in the from collection.
- foreignField: The field in the from collection that can be used as a unique identifier in the primary collection.
- as: The name of the new field that will contain the matching documents from the from collection.

Example
In this example, we are using the "sample_mflix" database loaded from our sample data in the Intro to

Aggregations section.

```
db.comments.aggregate([
  {$lookup: {from: "movies", localField: "movie_id",foreignField: "_id",as: "movie_details", },}, {
$limit: 1 }])
```

This will return the movie data along with each comment.

Aggregation $out

This aggregation stage writes the returned documents from the aggregation pipeline to a collection. The $out stage must be the last stage of the aggregation pipeline.

Example

In this example, we are using the "sample_airbnb" database loaded from our sample data in the Intro to Aggregations section.

```
db.listingsAndReviews.aggregate([
  {
   $group: {
     _id: "$property_type",
    properties: {
     $push: {
      name: "$name",
      accommodates: "$accommodates",
      price: "$price", }, },}, },{ $out: "properties_by_type" },])
```

The first stage will group properties by the property_type and include the name, accommodates, and price fields for each. The $out stage will create a new collection called properties_by_type in the current database and write the resulting documents into that collection.

Data Model XML

The XML data model is described in terms of sequences and items, atomic values, and nodes. Sequences and items. The XPath data model is based on the notion of a sequence. The value of an XPath expression is always a sequence. A sequence is an ordered collection of zero or more items.

Key Difference between OLTP and OLAP

- Online Analytical Processing (OLAP) is a category of software tools that analyze data stored in a database, whereas Online transaction processing (OLTP) supports transaction-oriented applications in a 3-tier architecture.
- OLAP creates a single platform for all types of business analysis needs which includes planning, budgeting, forecasting, and analysis, while OLTP is useful for administering day-to-day transactions of an organization.
- OLAP is characterized by a large volume of data, while OLTP is characterized by large numbers of short online transactions.
- In OLAP, a data warehouse is created uniquely so that it can integrate different data sources for building a consolidated database, whereas OLTP uses traditional DBMS.

What is OLAP?

Online Analytical Processing, a category of software tools which provide analysis of data for business decisions. OLAP systems allow users to analyze database information from multiple database systems at one time.

The primary objective is data analysis and not data processing.

What is OLTP?

Online transaction processing shortly known as OLTP supports transaction-oriented applications in a 3-tier architecture. OLTP administers day to day transaction of an organization.

The primary objective is data processing and not data analysis

Example of OLAP

Any Datawarehouse system is an OLAP system. Uses of OLAP are as follows

- A company might compare their mobile phone sales in September with sales in October, then compare those results with another location which may be stored in a sperate database.
- Amazon analyzes purchases by its customers to come up with a personalized homepage with products which likely interest to their customer.

Example of OLTP system

An example of OLTP system is ATM center. Assume that a couple has a joint account with a bank. One day both simultaneously reach different ATM centers at precisely the same time and want to withdraw total amount present in their bank account.

However, the person that completes authentication process first will be able to get money. In this case, OLTP system makes sure that withdrawn amount will be never more than the amount present in the bank. The key to note here is that OLTP systems are optimized for **transactional superiority instead data analysis.**

Other examples of OLTP applications are:

What is Apache Cassandra?

Documentation Link : https://cassandra.apache.org/doc/latest/cassandra/cql/index.html

Apache Cassandra is an open source, distributed and decentralized/distributed storage system (database), for managing very large amounts of structured data spread out across the world. It provides highly available service with no single point of failure.

Listed below are some of the notable points of Apache Cassandra –

It is scalable, fault-tolerant, and consistent.

It is a column-oriented database.

Its distribution design is based on Amazon's Dynamo and its data model on Google's Bigtable.

Created at Facebook, it differs sharply from relational database management systems.

Cassandra implements a Dynamo-style replication model with no single point of failure, but adds a more powerful "column family" data model.

Cassandra is being used by some of the biggest companies such as Facebook, Twitter, Cisco, Rackspace, ebay, Twitter, Netflix, and more.

Features of Cassandra

Cassandra has become so popular because of its outstanding technical features. Given below are some of the features of Cassandra:

Elastic scalability – Cassandra is highly scalable; it allows to add more hardware to accommodate more customers and more data as per requirement.

Always on architecture – Cassandra has no single point of failure and it is continuously available for business-critical applications that cannot afford a failure.

Fast linear-scale performance – Cassandra is linearly scalable, i.e., it increases your throughput as you increase the number of nodes in the cluster. Therefore it maintains a quick response time.

Flexible data storage – Cassandra accommodates all possible data formats including: structured, semi-structured, and unstructured. It can dynamically accommodate changes to your data structures according to your need.

Easy data distribution – Cassandra provides the flexibility to distribute data where you need by replicating data across multiple data centers.

Transaction support – Cassandra supports properties like Atomicity, Consistency, Isolation, and Durability (ACID).

Fast writes – Cassandra was designed to run on cheap commodity hardware. It performs blazingly fast writes and can store hundreds of terabytes of data, without sacrificing the read efficiency.

History of Cassandra

Cassandra was developed at Facebook for inbox search.

It was open-sourced by Facebook in July 2008.

Cassandra was accepted into Apache Incubator in March 2009.

It was made an Apache top-level project since February 2010.

CREATE FILES in Cassandra

The Cassandra Query Language (CQL) is very similar to SQL but suited for the JOINless structure of Cassandra.

Create a file named data.cql and paste the following CQL script in it. This script will create a keyspace, the layer at which Cassandra replicates its data, a table to hold the data, and insert some data into that table:

Syntax:

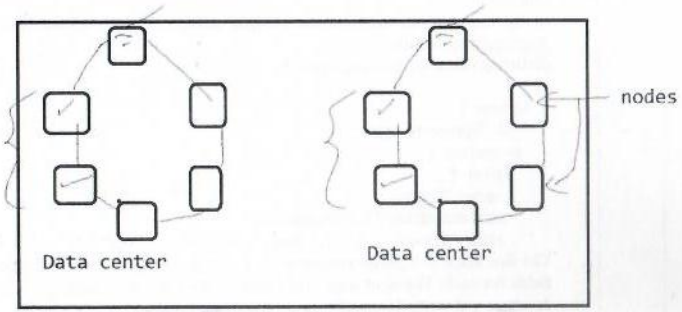
```
-- Create a keyspace
CREATE KEYSPACE IF NOT EXISTS store WITH REPLICATION = { 'class' : 'SimpleStrategy',
'replication_factor' : '1' };

-- Create a table
CREATE TABLE IF NOT EXISTS store.shopping_cart (
userid text PRIMARY KEY,
item_count int,
last_update_timestamp timestamp);

-- Insert some data
INSERT INTO store.shopping_cart(userid, item_count, last_update_timestamp)
VALUES ('9876', 2, toTimeStamp(now()));

INSERT INTO store.shopping_cart(userid, item_count, last_update_timestamp)
VALUES ('1234', 5, toTimeStamp(now()));
```

READ SOME DATA in Cassandra

```
SELECT * FROM store.shopping_cart;
```


Features of cassandra

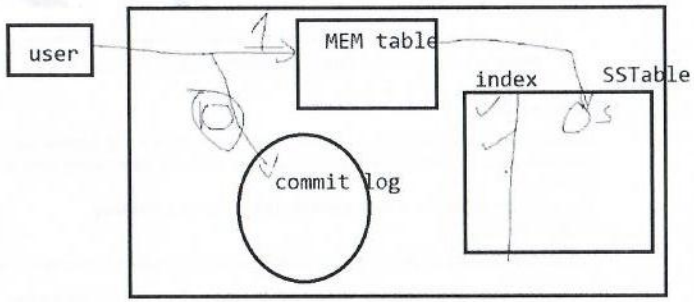
- Highly scalable
- Highly available --→ fault tolerant
- Cassandra support CAP theorem
- It is structured as well as unstructured database.
- It stores the data in column-oriented manner.



Components in Cassandra

- Data center-→ collection of nodes
- Node-→ it is a machine which stored the data
- Cluster-→collection of data centers
- Mem table --→ it is table which resides in RAM. After making entry in the commit log the data will be written in memtable, there can be more than one memtable
- SSTAble (sorted string table) - It is a disk file, in which data will be storedwhen mem table will reach to threshold value.

Write operation performed



Read operation

1. Direct read
2. Digest read
3. Read repair request

Replication factor--→Number of nodes in the data centre, on which the copy should be made, iscalled as replication factor

Replica placement strategy

1. Simple strategy-→
   a. If we are working with the single data centre., then we will use this strategy
2. Network topology
   a. If we are working with 2 data canters then we use network strategy, replication factor will be decided based on number of nodes in one data center, it will keep on replication data till it reaches to next data center.
   b. Replication happens in clockwise direction.

Cassandra does not support joins, group by, aggregate functions.

In Cassandra to save the data we need to create keyspace, Keyspace are similar to databaseData types

CQL Type	Constants	Description
ascii	Strings	US-ascii character string
bigint	Integers	64-bit signed long
blob	blobs	Arbitrary bytes in hexadecimal
boolean	Booleans	True or False
counter	Integers	Distributed counter values 64 bit
decimal	Integers, Floats	Variable precision decimal
double	Integers, Floats	64-bit floating point
float	Integers, Floats	32-bit floating point
frozen	Tuples, collections, user defined types	stores cassandra types
inet	Strings	IP address in ipv4 or ipv6 format
int	Integers	32 bit signed integer

list	Duplicate values are allowed , represented in []	Collection of elements
map	Keys are unique, represented in {}	JSON style collection of elements
set	Only unique values are allowed,{}	Collection of elements
text	strings	UTF-8 encoded strings
timestamp	Integers, Strings	ID generated with date plus time
timeuuid	uuids	Type 1 uuid
tuple	Read only and fixed size	A group of 2,3 fields
uuid	uuids	Standard uuid
varchar	strings	UTF-8 encoded string
varint	Integers	Arbitrary precision integer

CQL type compatibility
CQL data types have strict requirements for conversion compatibility. Thefollowing table shows the allowed alterations for data types:

Data type may be altered to:	Data type
ascii, bigint, boolean, decimal, double, float, inet, int, timestamp, timeuuid, uuid, varchar, varint	blob
int	varint
text	varchar
timeuuid	uuid
varchar	text

Clustering columns have even stricter requirements, because clustering columns mandate the order in which data is written to disk. The following table shows the allow alterations for data types used in clustering columns:

Data type may be altered to:	Data type
int	varint
text	varchar
varchar	text

To create key space

CREATE KEYSPACE iacsd0923 WITH replication = {'class': 'SimpleStrategy', 'replication_factor': '1'};

CREATE KEYSPACE iacsd0923 WITH replication = {'class': 'NetworTopologyStartegy', 'DC1':1,'DC2':3} and durable_writes=false;

By default durable_writes=true; which means that write in the commit log;

To change the keyspace

Use iacsd0923;

To create table

Create table customer(cno int primary key,cname text, mobile text);

to insert data into customer data

insert into customer(cno,cname,mobile) values

(1,'Rajesh','333333');insert into customer(cno,cname,mobile)

values (2,'Rutuja','333333');

to see the table data

select * from customer;

1. To create an employee table, with empid, emp_salary as partition keyAnd emp_firstname,emp_dob as cluster key

```
Create table employee(Empid int,
emp_salary int,
emp_lastname varchar,emp_firstname text,
emp_dob date , emp_deptno int,
emp_comm float,
primary key((empid,emp_salary),emp_firstname,emp_dob)
);
```

To see the structure of the tableDesc
employee

To alter the table
Alter table employee add manager varchar

To modify the data
Alter table employee alter manager text

To drop the column
Alter table employee drop manager;

To delete the table Drop

table employee

Truncate the table
Truncate table employee;

Retrieval of data
Select * from employee;

To insert datainsert
into
employee(empid,emp_salary,emp_lastname,emp_firstname,emp_dob,emp_deptno,
emp_comm) values(12,1200, 'Khadilkar','Kishori','2000-11-11',10,345);

insert into
employee(empid,emp_salary,emp_lastname,emp_firstname,emp_dob,emp_deptno,
emp_comm) values(12,1200, 'vaze', 'Amruta','2000-11-11',10,345);

insert into employee(empid,
emp_salary,emp_lastname,emp_firstname,emp_dob,emp_deptno,
... emp_comm) values(13,2000, 'Khadilkar','Rajan','2015-11-11',10,345);

insert into
employee(empid,emp_salary,emp_lastname,emp_firstname,emp_dob,emp_deptno,
... emp_comm) values(14,1500, 'kulkarni','Revati','2012-11-11',20,500);

Rules for where clause

1. have to use all partition key columns with = operatorYou Select * from employee where empid=13; ---------- error
   Select * from employee where empid=13 and emp_salary=2000;----------right
2. Use cluster columns in the same sequence in which it is created.
   Select * from employee where empid=13 and emp_salary=2000 and emp_dob='2000-11-11';---------- wrong, because we are skipping emp_firstname

Select * from employee where empid=13 and emp_salary=2000 and emp_dob='2000-11-11' and emp_firstname='Rajan';---------- right

Select * from employee where empid=13 and emp_salary=2000 and emp_firstname='Rajan';---------- right

Select * from employee where empid=13 and emp_salary=2000 and emp_firstname='Rajan' and emp_dob='2000-11-11';---------- right

3. You cannot use = operator only on non primary key columns without partition key and ifyou want to use it then add allow filtering

Select * from employee where emp_comm=345 ---------- error

Select * from employee where emp_comm=345 ALLOW FILTERING ---right

4. You cannot use = operator on only cluster key columns without partition key
   Select * from employee where emp_firstname='Rajan' and emp_dob='2000-11-11';----- wrong

   Select * from employee where empid=13 and emp_salary=2000 and emp_firstname='Rajan' and emp_dob='2000-11-11';---------- right

   Select * from employee where emp_firstname='Rajan' and emp_dob='2000-11-11' allow filtering----------right

5. In operator is allowed on all the columns of partition key but it slows the performance Select * from employee where empid in (12,13) and emp_salary in (1200,2000);
   right
   Select * from employee where empid in (12,13) ; ----------wrong
   Select * from employee where emp_salary in (1200,1110);---------- wrong

1. **> , <, <=,>= operators are not allowed on partition key**

   Select * from emp where empid = 1100 and emp_salary = 1200

2. **> , <, <=,>= operators can be used only on cluster key columns, and partition keycolumns**

   Select * from emp

   Where empid=100 and emp_salary=1002 and emp_firstname="xxxx" and emp_dob='1999-12-11'and emp_comm>1; wrong

   Select * from employee

   Where empid=12 and emp_salary=1110 and emp_firstname='kishori' and emp_dob>'1998-12-11';

3. Order by can used only on cluster key but where clause should have equality conditionbased on partition key;

   select * from employee where empid=13 and emp_salary=1110 order by emp_firstname desc,emp_dob desc;

   select * from employee where empid=13 and emp_salary=1110 order by emp_firstname

   desc; select * from employee where empid=13 and emp_salary=1110 order by emp_dob

   desc; ---------- wrong

   Create table customer(cno int primary key,

   cname text, mobile text);

to update data

update customer set

mobile='333333'

where cno=1;

to delete the data

delete from customer where cno=1;

delete mobile from customer where cno=1;

Data type

1.Set -

a. stores unique values
b. Represent in {}
c. Mutable
d. No indexing is possible

2. **List**
3. **Map**
4. **Tuple**

Create table

student(Id int

primary key,

Name text,

Courses set<text>);

1. Insert into student(id,name,courses) values(123,'Rajas',{'Python','Perl','PHP'});

2. To overwrite existing

coursesUpdate student

Set courses={'a','b','c','d'}

Where id=123

3. To add in existing

coursesUpdate student

Set

courses=courses+{'a','b','c','d'}

Where id=131sdfj

1.To delete from the set

Update student

Set courses=courses-

{python} Where id=131sdfj

1. **To remove all elements from the set**
2. Update student

Set courses={}

Where id=131sdfj

Delete courses from student where id=131sdfj

3. List
   a. Uses []
   b. Duplicates are allowed
   c. Ordered collection, indexing is possible
   d. Mutable

Create table emp11(

Id int primary

key,Name text,

Companies list<text>)

1. To insert data

   Insert into employee(id,name,companies) values(11,'Rajan',['cognizant','cabgemin','Tech-M'])

2. Update list add at the beginning
   Update employee set companies=['Wipro']+companies where id=1

3. Update list add at the end
   Update employee set companies=companies+['Wipro'] where id=1

4. Update list at particular location, value will be overwritten
   Update emp11 set companies[2]= 'Wipro' where id=1

5. to remove the element from particular position delete
   companies[2] from employee where id=1;

4. Map ------
   a. Store data in key value format
   b. Keys should be unique
   c. Data is stored in {}
   d. Values can bw retrieved by using keys.

Create table empdata(id int, name text, companies map<text,int>)

1. Insert into empdata(id,name,companies) values(111,'xxx',{'cognizant':5,'igate':3})

2. Update companies to add new key value pair
   Update empdata set companies=companies+{'acceture':5}

3. Update value of a particular key in maps
   Update empdata set companies['igate']=5 where id=111;

4. Delete a particular key value pair
   Delete companies['igate'] from employee where id=111;

5. Delete multiple keys also

Update set companies=companies-{'cognizant','igate'} where id=111;

Tuple

1. Duplicates are allowed
2. Uses()
3. Ordered collection,Indexing is possible
4. Tuples are immutable

Create table studdata(id int primary key,name text, marks

tuple<int,text,int>);insert into

studdata(id,name,marks)values(1,'a',(1,'a',1))

to update marks for id 1

update studdata set marks=(1,'xx',2);

Create table trainee(id int primary key,name text, marks

tuple<int,text,tuple<int,int,int>>);insert into trainee(id,name,marks)

values(11,'Revati',(90,'A',(20,40,80)))

delete marks from trainee where id=11;

update trainee set marks=(10,'a',(12,13,15)) where id=11;

----------create user defined data type

Create type address(street text,zipcode text,city

text)To create a table friend

Create table friend(fid int primary key, name text,loc address);

Insert into friend(fid,name,loc) values(12,'Ashwini',{street:'kothrud', zipcode:'234456', city:'Pune'})

Use FROZEN keyword when in user defined data type part of the type is not allowed to change butentire value can be overwritten

Create table friend(fid int, name text,loc FROZEN<address>);

Create table friend(fid int, name text,loc LIST<FROZEN<address>>);

To alter the data type

Alter type address add country

text;To rename field of a type

Alter type address rename country to cont;

Batch operations in

cassandaraBegin batch

Insert into empdata(id,name,companies)

values(111,'xxx',{'cognizant':5,'igate':3});Insert into

empdata(id,name,companies) values(112,yyy',{'TechM':5,'igate':3});

Delete companies from empdata where

id=123;Apply batch;

Indexes in Cassandra

Create index idxname on empdata(name)

To add data in json format

INSERT INTO Registration JSON'

{

"Emp_id": 2000,

"current_address": { "Emp_id":1,"h_no" : "A 210", "city" :

"delhi", "state" : "DL",

"pin_code"

:"201307","country"

:"india"},

"Emp_Name": "Ashish Rana"}' ;

PL/SQL (Extra)

1. What are the features of PL/SQL?

- PL/SQL provides the feature of decision making, looping, and branching by making use of its procedural nature.
- Multiple queries can be processed in one block by making use of a single command using PL/SQL.
- The PL/SQL code can be reused by applications as they can be grouped and stored in databases as PL/SQL units like functions, procedures, packages, triggers, and types.
- PL/SQL supports exception handling by making use of an exception handling block.
- Along with exception handling, PL/SQL also supports error checking and validation of data before data manipulation.

- Applications developed using PL/SQL are portable across computer hardware or operating system where there is an Oracle engine.

2. What do you understand by PL/SQL table?

- PL/SQL tables are nothing but objects of type tables that are modeled as database tables. They are a way to provide arrays that are nothing but temporary tables in memory for faster processing.
- These tables are useful for moving bulk data thereby simplifying the process.

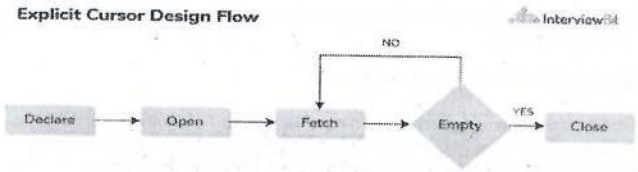
3. Explain the basic structure followed in PL/SQL?

- The basic structure of PL/SQL follows the BLOCK structure. Each PL/SQL code comprises SQL and PL/SQL statement that constitutes a PL/SQL block.
- Each PL/SQL block consists of 3 sections:
  - The optional Declaration Section
  - The mandatory Execution Section
  - The optional Exception handling Section

[DECLARE]
--declaration statements (optional)
BEGIN
--execution statements
[EXCEPTION]
--exception handling statements (optional)
END;

4. What is a PL/SQL cursor?

- A PL/SQL cursor is nothing but a pointer to an area of memory having SQL statements and the information of statement processing. This memory area is called a context area. This special area makes use of a special feature called cursor for the purpose of retrieving and processing more than one row.
- In short, the cursor selects multiple rows from the database and these selected rows are individually processed within a program.
- There are two types of cursors:
  - **Implicit Cursor**:
    - Oracle automatically creates a cursor while running any of the commands - SELECT INTO, INSERT, DELETE or UPDATE implicitly.
    - The execution cycle of these cursors is internally handled by Oracle and returns the information and status of the cursor by making use of the cursor attributes- ROWCOUNT, ISOPEN, FOUND, NOTFOUND.
  - **Explicit Cursor:**
    - This cursor is a SELECT statement that was declared explicitly in the declaration block.
    - The programmer has to control the execution cycle of these cursors starting from OPEN to FETCH and close.
    - The execution cycle while executing the SQL statement is defined by Oracle along with associating a cursor with it.
  - **Explicit Cursor Execution Cycle:**
    - Due to the flexibility of defining our own execution cycle, explicit cursors are used in many instances. The following diagram represents the execution flow of an explicit cursor:



Cursor Declaration:
The first step to use an explicit cursor is its declaration.
Declaration can be done in a package or a block.
Syntax: CURSOR cursor_name IS query; where cursor_name is the name of the cursor, the query is the query to fetch data from any table.
Open Cursor:
Before the process of fetching rows from cursor, the cursor has to be opened.
Syntax to open a cursor: OPEN cursor_name;
When the cursor is opened, the query and the bind variables are parsed by Oracle and the SQL statements are executed.
The execution plan is determined by Oracle and the result set is determined after associating the cursor parameters and host variables and post these, the cursor is set to point at the first row of the result set.
Fetch from cursor:
FETCH statement is used to place the content of the current row into variables.
Syntax: FETCH cursor_name INTO variable_list;
In order to get all the rows of a result set, each row needs to be fetched.
Close Cursor:
Once all the rows are fetched, the cursor needs to be closed using the CLOSE statement.
Syntax: CLOSE cursor_name;
The instructions tell Oracle to release the memory allocated to the cursor.
Cursors declared in procedures or anonymous blocks are by default closed post their execution.
Cursors declared in packages need to be closed explicitly as the scope is global.
Closing a cursor that is not opened will result in INVALID_CURSOR exception.

5. What is the use of WHERE CURRENT OF in cursors?

We use this clause while referencing the current row from an explicit cursor. This clause allows applying updates and deletion of the row currently under consideration without explicitly referencing the row ID.

Syntax:
UPDATE table_name SET field=new_value WHERE CURRENT OF cursor_name

6. How can a name be assigned to an unnamed PL/SQL Exception Block?

This can be done by using Pragma called **EXCEPTION_INIT**.
This gives the flexibility to the programmer to instruct the compiler to provide custom error messages based on the business logic by overriding the pre-defined messages during the compilation time.
Syntax:

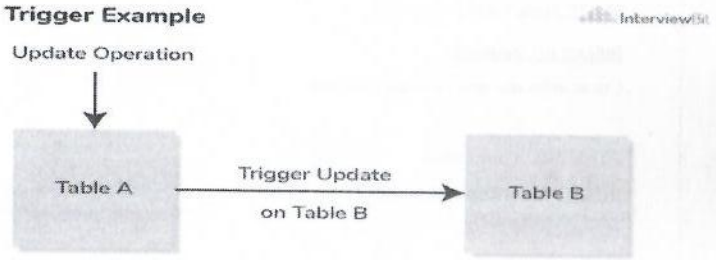
```
DECLARE
 exception_name EXCEPTION;
 PRAGMA EXCEPTION_INIT (exception_name, error_code);
```



```
BEGIN
// PL/SQL Logic
EXCEPTION
 WHEN exception_name THEN
   // Steps to handle exception
END;
```

7. What is a Trigger? Name some instances when "Triggers" can be used.

As the name indicates, '**Trigger**' means to 'activate' something. In the case of PL/SQL, a trigger is a stored procedure that specifies what action has to be taken by the database when an event related to the database is performed.



Syntax:

```
TRIGGER trigger_name
 trigger_event
 [ restrictions ]
BEGIN
 actions_of_trigger;
END;
```

In the above syntax, if the trigger_name the trigger is in the enabled state, the trigger_event causes the database to fire actions_of_trigger if the restrictions are TRUE or unavailable.

They are mainly used in the following scenarios:
In order to maintain complex integrity constraints.
For the purpose of auditing any table information.
Whenever changes are done to a table, if we need to signal other actions upon completion of the change, then we use triggers.
In order to enforce complex rules of business.
It can also be used to prevent invalid transactions.
You can refer https://docs.oracle.com/database/121/TDDDG/tdddg_triggers.htm for more information regarding triggers.

8. When does a DECLARE block become mandatory?

This statement is used by anonymous blocks of PL/SQL such as non-stored and stand-alone procedures. When they are being used, the statement should come first in the stand-alone file.

9. How do you write comments in a PL/SQL code?

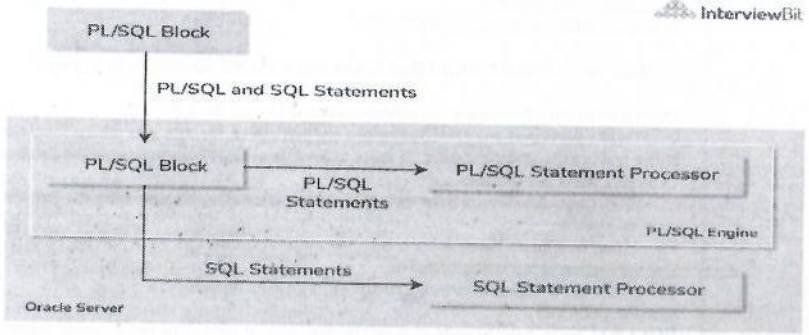
Comments are those sentences that have no effect on the functionality and are used for the purpose of enhancing the readability of the code. They are of two types:
Single Line Comment: This can be created by using the symbol -- and writing what we want to mention as a comment next to it.
Multi-Line comment: These are the comments that can be specified over multiple lines and the syntax goes like /* comment information */
Example:

```
SET SERVEROUTPUT ON;
DECLARE

-- Hi There! I am a single line comment.
var_name varchar2(40) := 'I love PL/SQL' ;
BEGIN
/*
Hi! I am a multi line
comment. I span across
multiple lines
*/
dbms_output.put_line(var_name);
END;
/
Output:
I love PL/SQL
```

11. Can you explain the PL/SQL execution architecture?

The PL/SQL engine does the process of compilation and execution of the PL/SQL blocks and programs and can only work if it is installed on an Oracle server or any application tool that supports Oracle such as Oracle Forms.

PL/SQL is one of the parts of Oracle RDBMS, and it is important to know that most of the Oracle applications are developed using the client-server architecture. The Oracle database forms the server-side and requests to the database form a part of the client-side.
So based on the above fact and the fact that PL/SQL is not a standalone programming language, we must realize that the PL/SQL engine can reside in either the client environment or the server environment. This makes it easy to move PL/SQL modules and sub-programs between server-side and client-side applications.
Based on the architecture shown below, we can understand that PL/SQL engine plays an important role in the process and execute the PL/SQL statements and whenever it encounters the SQL statements, they are sent to the SQL Statement Processor.



Case 1: PL/SQL engine is on the server: In this case, the whole PL/SQL block gets passed to the PL/SQL engine present on the Oracle server which is then processed and the response is sent.
Case 2: PL/SQL engine is on the client: Here the engine lies within the Oracle Developer tools and the processing of the PL/SQL statements is done on the client-side.
In case, there are any SQL statements in the PL/SQL block, then they are sent to the Oracle server for SQL processing.
When there are no SQL statements, then the whole block processing occurs at the client-side.

14. Differentiate between SQL and PL/SQL.

SQL	PL/SQL
)L is a natural language meant for the interactive)cessing of data in the database.	/SQL is a procedural extension of SQL.
cision-making and looping are not allowed in SQL.	/SQL supports all features of procedural iguage such as conditional and looping tements.
l SQL statements are executed at a time by the database ver which is why it becomes a time-consuming)cess.	/SQL statements are executed one block at a ie thereby reducing the network traffic.
ere is no error handling mechanism in SQL.	is supports an error handling mechanism.

15. What is the importance of %TYPE and %ROWTYPE data types in PL/SQL?

%TYPE: This declaration is used for the purpose of anchoring by providing the data type of any variable, column, or constant. It is useful during the declaration of a variable that has the same data type as that of its table **column**.
Consider the example of declaring a variable named ib_employeeid which has the data type and its size same as that of the column employeeid in table ib_employee.
The syntax would be : ib_employeeid ib_employee.employeeid%TYPE;
%ROWTYPE: This is used for declaring a variable that has the same data type and size as that of a row in the table. The row of a table is called a record and its fields would have the same data types and names as the columns defined in the table.

For example: In order to declare a record named ib_emprecord for storing an entire row in a table called ib_employee, the syntax is:
ib_emprecord ib_employee%ROWTYPE;

21. What are COMMIT, ROLLBACK and SAVEPOINT statements in PL/SQL?

These are the three transaction specifications that are available in PL/SQL.
COMMIT: Whenever any DML operations are performed, the data gets manipulated only in the database buffer and not the actual database. In order to save these DML transactions to the database, there is a need to COMMIT these transactions.
COMMIT transaction action does saving of all the outstanding changes since the last commit and the below steps take place:
The release of affected rows.
The transaction is marked as complete.
The details of the transaction would be stored in the data dictionary.
Syntax: COMMIT;
ROLLBACK: In order to undo or erase the changes that were done in the current transaction, the changes need to be rolled back. ROLLBACK statement erases all the changes since the last COMMIT.
Syntax: ROLLBACK;
SAVEPOINT: This statement gives the name and defines a point in the current transaction process where any changes occurring before that SAVEPOINT would be preserved whereas all the changes after that point would be released.
Syntax: SAVEPOINT <savepoint_name>;

22. How can you debug your PL/SQL code?

We can use DBMS_OUTPUT and DBMS_DEBUG statements for debugging our code:
DBMS_OUTPUT prints the output to the standard console.
DBMS_DEBUG prints the output to the log file.

23. What is the difference between a mutating table and a constraining table?

A table that is being modified by the usage of the DML statement currently is known as a mutating table. It can also be a table that has triggers defined on it.
A table used for reading for the purpose of referential integrity constraint is called a constraining table.

24. In what cursor attributes the outcomes of DML statement execution are saved?

The outcomes of the execution of the DML statement is saved in the following 4 cursor attributes:
SQL%FOUND: This returns TRUE if at least one row has been processed.
SQL%NOTFOUND: This returns TRUE if no rows were processed.
SQL%ISOPEN: This checks whether the cursor is open or not and returns TRUE if open.
SQL%ROWCOUNT: This returns the number of rows processed by the DML statement.

25. Is it possible to declare column which has the number data type and its scale larger than the precision? For example defining columns like: column name NUMBER (10,100), column name NUMBER (10,-84)

Yes, these type of declarations are possible.
Number (9, 12) indicates that there are 12 digits after decimal point. But since the maximum precision is 9, the rest are 0 padded like 0.000999999999.

Number (9, -12) indicates there are 21 digits before the decimal point and out of that there are 9 possible digits and the rest are 0 padded like 999999999000000000000.0

PL/SQL Programs

26. Write a PL/SQL program using WHILE loop for calculating the average of the numbers entered by user. Stop the entry of numbers whenever the user enters the number 0.

```
DECLARE
  n NUMBER;
  average NUMBER :=0 ;
  sum NUMBER :=0 ;
  count NUMBER :=0 ;
BEGIN
  -- Take input from user
  n := &input_number;
  WHILE(n<>0)
    LOOP
      -- Increment count to find total elements
      count := count+1;
      -- Sum of elements entered
      sum := sum+n;
      -- Take input from user
      n := &input_number;
    END LOOP;
  average := sum/count;
  DBMS_OUTPUT.PUT_LINE('Average of entered numbers is '||average);
END;
```

27. Write a PL/SQL procedure for selecting some records from the database using some parameters as filters.

Consider that we are fetching details of employees from ib_employee table where salary is a parameter for filter.

```
CREATE PROCEDURE get_employee_details @salary nvarchar(30)

AS
BEGIN
  SELECT * FROM ib_employee WHERE salary = @salary;
END;
```

28. Write a PL/SQL code to count the number of Sundays between the two inputted dates.

```
--declare 2 dates of type Date
DECLARE
  start_date Date;
  end_date Date;
  sundays_count Number:=0;
BEGIN
  -- input 2 dates
  start_date:='&input_start_date';
  end_date:='&input_end_date';
```

```
/*
Returns the date of the first day after the mentioned date
and matching the day specified in second parameter.
*/
start_date:=NEXT_DAY(start_date-1, 'SUNDAY');
--check the condition of dates by using while loop.
while(start_date<=end_date)
LOOP
   sundays_count:=sundays_count+1;
   start_date:=start_date+7;
END LOOP;
dbms_output.put_line('Total number of Sundays between the two dates:'||sundays_count);
END;
/
```

Input:

```
start_date = '01-SEP-19'
end_date = '29-SEP-19;
```

Output:

Total number of Sundays between the two dates: 5

29. Write PL/SQL code block to increment the employee's salary by 1000 whose employee_id is 102

```
DECLARE
 employee_salary  NUMBER(8,2);

 PROCEDURE update_salary (
  emp       NUMBER,
  salary IN OUT NUMBER
 ) IS
 BEGIN
  salary := salary + 1000;
 END;

BEGIN
 SELECT salary INTO employee_salary
 FROM ib_employee
 WHERE employee_id = 102;

 DBMS_OUTPUT.PUT_LINE
  ('Before update_salary procedure, salary is: ' || employee_salary);

 update_salary (100, employee_salary);

 DBMS_OUTPUT.PUT_LINE
  ('After update_salary procedure, salary is: ' || employee_salary);
END;
/
```

Result:

```
Before update_salary procedure, salary is: 17000
After update_salary procedure, salary is: 18000
```


30. Write a PL/SQL code to find whether a given string is palindrome or not.

```
DECLARE
-- Declared variables string, letter, reverse_string where string is the original string.
string VARCHAR2(10) := 'abccba';
letter VARCHAR2(20);
reverse_string VARCHAR2(10);
BEGIN
FOR i IN REVERSE 1..LENGTH(string) LOOP
 letter := SUBSTR(string, i, 1);
 -- concatenate letter to reverse_string variable
 reverse_string := reverse_string ||''||letter;
END LOOP;
IF reverse_string = string THEN
     dbms_output.Put_line(reverse_string||''||' is palindrome');
ELSE
     dbms_output.Put_line(reverse_string ||'' ||' is not palindrome');
END IF;
END;
```

31. Write PL/SQL program to convert each digit of a given number into its corresponding word format.

```
DECLARE
-- declare necessary variables
  -- num represents the given number
  -- number_to_word represents the word format of the number
  -- str, len and digit are the intermediate variables used for program execution
num   INTEGER;
number_to_word VARCHAR2(100);
digit_str   VARCHAR2(100);
len   INTEGER;
digit   INTEGER;
BEGIN
  num := 123456;
  len := LENGTH(num);
  dbms_output.PUT_LINE('Input: ' ||num);
  -- Iterate through the number one by one
  FOR i IN 1..len LOOP
    digit := SUBSTR(num, i, 1);
    -- Using DECODE, get the str representation of the digit
    SELECT Decode(digit, 0, 'Zero ',
                  1, 'One ',
                  2, 'Two ',
                  3, 'Three ',
                  4, 'Four ',
                  5, 'Five ',
                  6, 'Six ',
                  7, 'Seven ',
                  8, 'Eight ',
                  9, 'Nine ')
```

```
    INTO digit_str
    FROM dual;
    -- Append the str representation of digit to final result.
    number_to_word := number_to_word || digit_str;
  END LOOP;
  dbms_output.PUT_LINE('Output: ' ||number_to_word);
END;
```

Input: 12345
Output: One Two Three Four Five

32. Write PL/SQL program to find the sum of digits of a number.

```
DECLARE
--Declare variables num, sum_of_digits and remainder of datatype Integer
num  INTEGER;
sum_of_digits INTEGER;
remainder  INTEGER;
BEGIN
num := 123456;
sum_of_digits := 0;
-- Find the sum of digits until original number doesnt become null
WHILE num <> 0 LOOP
 remainder := MOD(num, 10);
 sum_of_digits := sum_of_digits + remainder;
 num := TRUNC(num / 10);
END LOOP;
dbms_output.PUT_LINE('Sum of digits is '|| sum_of_digits);
END;
```