**DATABASE:** An organized collection of data.

Most Oracle databases store files in a file system.

At the operating system level, Oracle Database stores database data in data files.

Every database must have at least one data file.

**TABLE SPACE** is a file or set or files that is used to store ORACLE data.

An ORACLE database is composed of the SYSTEM tablespace and possibly other tablespaces.

Each tablespace consists of one or more data files, which conform to the operating system in which Oracle Database is running.

A **database schema** is a logical container for data structures, called **schema** objects. Examples of schema objects are tables and indexes. Schema objects are created and manipulated with SQL.

Many other types of schema objects – Indexes, Partitions, Views, Sequence, Dimensions, Synonyms, PL/SQL subprograms and packages.

A database schema is owned by a database user and has the same name as the user name.

**RELATIONAL MODEL**

Was first described by E F Codd and represents all data in the database as simple row-column tables of data values.

An RDBMS is a database management system where all the data visible to the user is organized strictly as tables of data values. All database operations work on these tables.

Ex: - Oracle, Sybase, DB2

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

**PL/SQL**

PL/SQL is the procedural extension to SQL with design features of programming languages.

Data manipulation and query statements of SQL are included within procedural units of code.

**DDL** – Data Definition Language: CREATE, ALTER, DROP, RENAME, TRUNCATE

**DML** – Data Manipulation Language: INSERT, UPDATE, DELETE, MERGE

Transaction control - COMMIT, ROLLBACK, SAVEPOINT

**STORED PROCEDURE:** is a sequence of SQL statements that perform specific function/task.

A stored procedure (sometimes called a proc, sproc, StoPro, StoredProc, sp or SP) is actually stored in the database data dictionary.

**Data dictionary** is a data structure that stores metadata, i.e., (structured) data about data.

Centralized repository of information about data such as meaning, relationships to other data, origin, usage, and format.

The **PRIMARY KEY** is the column(s) used to uniquely identify each row of a table.

The **FOREIGN KEY** is a column in one table that matches the PRIMARY KEY of another table.

FOREIGN KEY always represents relationship.

Keys can be simple or composite.

A simple key comprises of a single attribute.

A COMPOSITE KEY is made up of two or more attributes.

The UNIQUE constraint uniquely identifies each record in a database table.

A **UNIQUE KEY** is one or more columns that must be unique for each row of the table.

The UNIQUE KEY column restricts entry of duplicate values but entry of NULL value is allowed.

In case of PRIMARY KEY, columns entry of duplicate as well as NULL value is restricted.

Note that you can have many UNIQUE constraints per table, but only one PRIMARY KEY constraint per table.

A **DATABASE TRIGGER** is a stored procedure associated with a table that automatically executes on one or more specified events (BEFORE or AFTER an INSERT, UPDATE or DELETE) affecting the table. Triggers can execute for the table as a whole or for each affected row in the table.

A PL/SQL or Java procedure that fires when a table or view is modified or when specific user or database actions occur. Procedures are explicitly run, whereas triggers are implicitly run.

**TABLE**: Basic unit of storage; composed of rows and columns

**VIEW**: Logically represents subsets of data from one or more tables

To restrict data access

To make complex queries easy

To present different views of the same data

A **CURSOR** can be viewed as a pointer to one row in a set of rows.

The cursor can only reference one row at a time, but can move to other rows of the result set as needed.

Types of Cursors – Implicit and Explicit

Implicit Cursor – Declared for all DML and PL/SQL Select statements

Explicit Cursor – declared and named by programmer

**SEQUENCE**: Generates primary key values

CREATE SEQUENCE "CARGO"."CSPS\_MAIL\_STORAGE\_ID\_SEQ" MINVALUE 1 MAXVALUE 999999999999 INCREMENT BY 1 START WITH 261 CACHE 20 NOORDER NOCYCLE ;

NEXTVAL returns the next available sequence value. It returns a unique value every time it is referenced, even for different users.

CURRVAL obtains the current sequence value.

NEXTVAL must be issued for that sequence before CURRVAL contains a value.

**INDEXES**

An index:

* Is a schema object
* Is used by the Oracle server to **speed up the retrieval of rows** by using a pointer
* Can reduce disk I/O by using a rapid path access method to locate data quickly
* Is independent of the table it indexes
* Is used and maintained automatically by the Oracle server

Indexes allow the database application to find data fast; without reading the whole table.

An index can be created in a table to find data more quickly and efficiently.

The users cannot see the indexes; they are just used to speed up searches/queries.

Using indexes to **improve query performance**

In general, my indexing strategy would be something like this

* Pick a good clustering key - a good clustering key is narrow, unique, stable, ever-increasing
* For any column that is being used as a foreign key into another table - add an index.

Create an index in columns that are common in WHERE, ORDER BY and GROUP BY clauses

Add one index at a time and again.

An index can be made up of more than one column. Such an index is called a concatenated index.

A **clustered index** is a special type of index that reorders the way records in the table are physically stored. Therefore table can have only one clustered index.

A **non-clustered index** is a special type of index in which the logical order of the index does not match the physical stored order of the rows on disk.

Delete/Truncate/Drop

DELETE

1. DELETE is a DML Command.
2. DELETE statement is executed using a row lock, each row in the table is locked for deletion.
3. We can specify filters in where clause
4. It deletes specified data if where condition exists.
5. Delete activates a trigger because the operation are logged individually.
6. Slower than truncate because, it keeps logs.
7. Rollback is possible.

TRUNCATE

1. TRUNCATE is a DDL command.
2. Removes all rows from a table.
3. Does not require a WHERE clause, not allowed here.
4. Identity columns are re-seeded on this operation, if no seed was defined then the default value 1 is used.
5. No triggers are fired on this operation because it does not log individual rows.
6. It deallocates data pages instead of rows in transaction logs, thus is faster than DELETE.
7. Thus it also requires less number of locks.
8. TRUNCATE is not possible when a table is reference by a Foreign Key or tables used in replication or with Indexed views.
9. This is a DDL command as its resets identity columns, deallocates data pages and empty them for use of other objects in the database.

DROP Table

1. All the rows in the table are deleted and the table structure is removed from the database.
2. Once a table is dropped we cannot get it back.
3. All the relationships with other tables will no longer be valid.
4. The integrity constraints will be dropped, grant or access privileges on the table will also be dropped.
5. But, if a table is truncated, the table structure remains the same.

**NORMALIZATION:** is a step by step decomposition of complex records into simple records.

Normalization reduces redundancy using the principle of non-loss decomposition.

Non-loss decomposition is the reduction of table into smaller tables without loss of information.

Normalization results in formation of tables that satisfy certain specified constrains and represents certain normal forms.

**Normal forms** are table structures with minimum redundancy.

1st NF

2nd NF

3rd NF

Boyce-Codd Normal Form

Normalization helps simplify the structure of tables.

The intentional introduction of redundancy to a table to improve performance is called *de-normalization.*

**JOINS**

Join is a query which retrieves related columns or rows from multiple tables.

**INNER JOIN**: Returns all rows when there is at least one match in BOTH tables

Example:

SELECT column\_name(s)

FROM table1

INNER JOIN table2

ON table1.column\_name=table2.column\_name;

**OUTER JOINS**

**LEFT JOIN**: returns all rows from the left table, even if there are no matches in the right table.

Example:

SELECT column\_name(s)

FROM table1

LEFT JOIN table2

ON table1.column\_name=table2.column\_name;

OR

SELECT column\_name(s)

FROM table1

LEFT OUTER JOIN table2

ON table1.column\_name=table2.column\_name;

**RIGHT JOIN**: returns all rows from the right table, even if there are no matches in the left table.

Example:

SELECT column\_name(s)

FROM table1

RIGHT JOIN table2

ON table1.column\_name=table2.column\_name;

OR

SELECT column\_name(s)

FROM table1

RIGHT OUTER JOIN table2

ON table1.column\_name=table2.column\_name;



**FULL JOIN**: returns rows when there is a match in one of the tables.

SELECT column\_name(s)

FROM table1

FULL OUTER JOIN table2

ON table1.column\_name=table2.column\_name;

**SELF JOIN**: is used to join a table to itself as if the table were two tables, temporarily renaming at least one table in the SQL statement.

**CARTESIAN JOIN**: returns the Cartesian product of the sets of records from the two or more joined tables.

**UNION**

Combine the result of two or more SELECT statements.

Selects only distinct values by default.

To allow duplicate values, use the ALL keyword with UNION.

SELECT column\_name(s) FROM table1 UNION SELECT column\_name(s) FROM table2;

SELECT column\_name(s) FROM table1 UNION ALL SELECT column\_name(s) FROM table2;

**TRANSACTION**

A collection of DML statements that form a logical unit of work.

When you want to add, update, or delete data in the database, you execute a DML statement.

A transaction should be

Atomic: Either it all works or it all fails (and rolls back)

Consistent: Whether it works (commits) or fails (rolls back) the **data** should be consistent.

Isolation: enables transactions to operate independently of and transparent to each other.

Durability: ensures that the result or effect of a committed transaction persists in case of a system failure.

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

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

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

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

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

A FUNCTION is always returns a value using the return statement.

A PROCEDURE may return one or more values through parameters or may not return at all.

A **database driver** is a program for which implements a protocol (ODBC, JDBC) for connecting to a database.

It is an Adaptor which connects a generic interface to a specific vendors implementation, just like printer drivers etc.

A **database dialect** is a configuration setting for platform independent software (JPA, Hibernate, etc) which allows such software to translate its generic SQL statements into vendor specific DDL, DML.

**BLOB** (Binary Large OBject) is a collection of binary data stored as a single entity in a database management system.

BLOB could be an image (jpeg, gif), audio or other multimedia objects.

**CLOB** (character large object) is a collection of character data in a database management system.

CLOB could be some large text file (txt, csv, xml, etc.).

CLOBs usually have very high size limits, on the order of 2GB or more

SQL has many built-in functions for performing calculations on data.

**SQL Aggregate Functions**

* AVG() - Returns the average value
* COUNT() - Returns the number of rows
* FIRST() - Returns the first value
* LAST() - Returns the last value
* MAX() - Returns the largest value
* MIN() - Returns the smallest value
* SUM() - Returns the sum

Aggregate functions often need an added GROUP BY statement.

The **GROUP BY** statement is used in conjunction with the aggregate functions to group the result-set by one or more columns.

The **ORDER BY** keyword is used to **sort** the result-set.

**SQL Constraints**

SQL constraints are used to specify rules for the data in a table.

* NOT NULL - Indicates that a column cannot store NULL value
* UNIQUE - Ensures that each row for a column must have a unique value
* PRIMARY KEY - A combination of a NOT NULL and UNIQUE. Ensures that a column (or combination of two or more columns) have an unique identity which helps to find a particular record in a table more easily and quickly
* FOREIGN KEY - Ensure the referential integrity of the data in one table to match values in another table
* CHECK - Ensures that the value in a column meets a specific condition
* DEFAULT - Specifies a default value when specified none for this column

Constraints can be specified when the table is created (inside the CREATE TABLE statement) or after the table is created (inside the ALTER TABLE statement).

**QUERIES**

SECOND HIGHEST SALARY

Select Max(Salary) from Employee where salary < (Select Max(Salary) from employee)

Select Salary from Employee where salary < (Select Max(Salary) from employee)

//Selects the maximum and second maximum element from table

Select max(salary) from employee where salary < (select max(salary) from employee) UNION select max(salary) from employee;

NUMBER OF PEOPLE IN EACH DEPARTMENT

select count(e.deptid) as NoOfPeopleInEachDept

from employee e, department d

where e.DeptID = d.deptno

group by e.deptid;

OR

select count(employee.deptid) as NoOfPeopleInEachDept

from employee inner join department

on employee.DeptID = department.deptno

group by employee.deptid;