CHAPTER NO. 3 INTERACTIVE SQL (26 Marks)

3.1 Introduction to SQL (10 Marks)

What is SQL?

- SQL is Structured Query Language, which is a computer language for storing, manipulating and retrieving data stored in a relational database.
- SQL is the standard language for Relational Database System. All the Relational Database Management Systems (RDMS) like MySQL, MS Access, Oracle, Sybase, Informix, Postgres and SQL Server use SQL as their standard database language.

Why SQL?

SQL is widely popular because it offers the following advantages:

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

Query Processing **(Q. Explain the steps used in query processing with suitable diagram.)**

- **Query processing** refers to the range of activities involved in extracting data from a database.
- It is a three step process that transforms a high-level query (of relational calculus/SQL) into an equivalent and more efficient lower-level query (of relational algebra).
- The steps involved processing a query appear in below Figure The basic steps are:

1. **Parsing and translation:** - Check syntax and verify relations. It translates the query into an equivalent relational algebra expression.
2. **Optimization**: - Generate an optimal evaluation plan (with lowest cost) for the query plan.
3. **Evaluation:** - The query-execution engine takes an (optimal) evaluation plan, executes that plan, and returns the answers to the query.

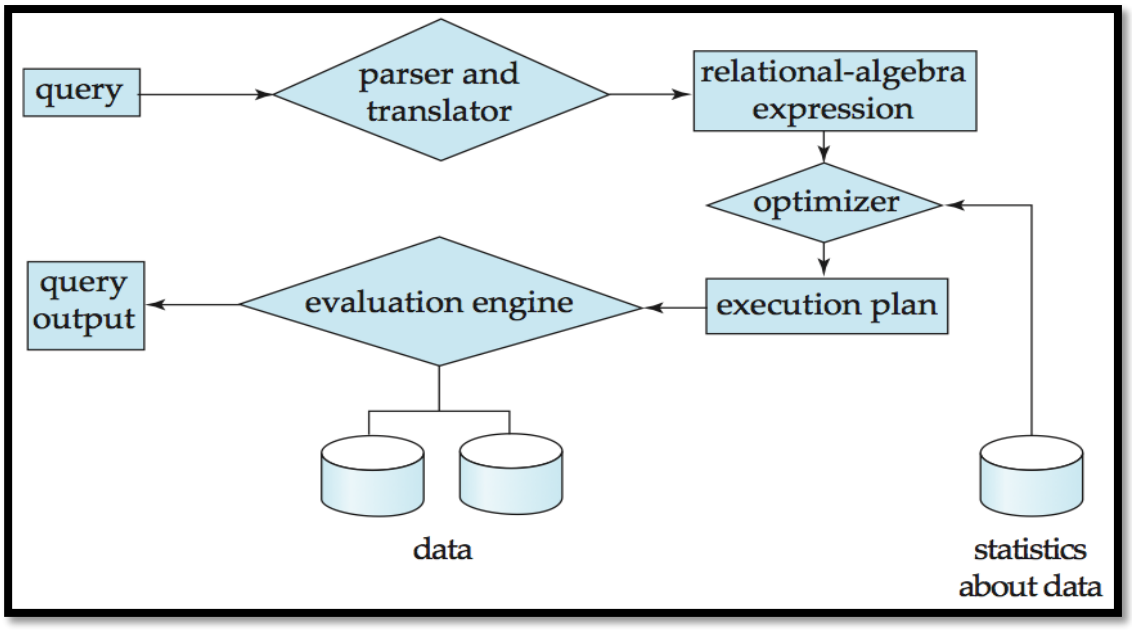


Fig. Steps in Query Processing

SQL - Data Types

DATA TYPE	Description
Char(Size)	Holds a fixed length string (can contain letters, numbers, and special characters). When you create a table with a CHAR column, you must specify a string length between 1 & 2000 bytes for the CHAR column width.
Varchar2(Size)	Holds a variable length string (can contain letters, numbers, and special characters). When you create a table with a VARCHAR2 column, you must specify a maximum string length between 1 & 4000 bytes for the VARCHAR2 column.
varchar(max)	The VARCHAR data type is synonymous with the VARCHAR2 data type. To avoid possible changes in behavior, always use the VARCHAR2 data type to store variable length character strings.
Date	The date data type stores point-in-time values (dates & times) in a table. Oracle database uses its own internal format to store dates. Date data is stored in fixed length fields of seven bytes each. Syntax:- DD-Month-YY for Time format HH:MI:SS

Number(P, S)	The number data type stores fixed & floating point numbers. Allows numbers from -10^38 +1 to 10^38 –1. The p parameter indicates the maximum total number of digits that can be stored (both to the left and to the right of the decimal point). P must be a value from 1 to 38. The s parameter indicates the maximum number of digits stored to the right of the decimal point. s must be a value from 0 to p. Default value is 0
LONG	Character data up to a length of 2GB. Only one long column is allowed per table.

DDL Commands:

1. **CREATE: -** The CREATE TABLE statement is used to create a new table in a database.
   - **Syntax: -** ***CREATE TABLE*** table_name *(*
     column1 Data_type,
     column2 Data_type,
     column3 Data_type…*);*
   - **Example:- CREATE TABLE** Persons (
     PersonID number(10),
     LastName varchar(20),
     FirstName varchar(20),
     Address varchar(20),
     City varchar(20));

2. **ALTER: -** The ALTER TABLE statement is used to add, delete, or modify columns in an existing table. The ALTER TABLE statement is also used to add and drop various constraints on an existing table.
   - To add a column in a table, use the following syntax:
     Syntax: - ALTER TABLE *table_name* **ADD** *column_name Data_type*;
     Example: - ALTER TABLE Persons **ADD** DateOfBirth date;
   - To delete a column in a table, use the following syntax
     Syntax: - ALTER TABLE *table_name* **DROP COLUMN** *column_name*;
     Example: - ALTER TABLE Persons **DROP COLUMN** DateOfBirth;

3. **DROP: -** The **DROP TABLE** statement is used to drop an existing table in a database.
   - **Syntax:- DROP TABLE** table_name**;**
   - **Example:- DROP TABLE** `Student;`

4. **TRUNCATE: -** The TRUNCATE TABLE statement is used to delete the data inside a table, but not the table itself.
   - **Syntax: - TRUNCATE TABLE** *table_name*;
   - **Example:- TRUNCATE TABLE** *Student*;
5. **RENAME:-** To change the name of the table
   - **Syntax:- RENAME** <old_table_name> **To** < new_table_name>;
   - **Example**:- **RENAME** employee **TO** my employee;
6. **DESC:-**
   - **Syntax:- DESC** table_name;
   - **Example:- DESC** Student;

DML Commands:

1. **INSERT: -** The **INSERT INTO** statement is used to insert new records in a table.
   - **Syntax: - INSERT INTO** *table_name* **VALUES** (values1, values2…….);
   - **Example:- INSERT INTO** *persons* **VALUES** ('10', 'Jain' , 'Vikas' , ' Kothrud' , 'Pune');
   - **Insert multiple values**
   - **INSERT INTO** *persons* **VALUES** (&PersonID, '& LastName', '& FirstName', '& Address','& City');
2. **UPDATE: -** The UPDATE statement is used to modify the existing records in a table.
   - **Syntax: - UPDATE** *table_name* **SET** *column1* = *value1*, *column2* = *value2*, **WHERE** *condition*;
   - **Example: - UPDATE** Persons **SET** Address = 'MG Road', City= 'Mumbai' **WHERE** PersonID = 1;
3. **DELETE: -** The **DELETE** statement is used to delete existing records in a table.
   - **Syntax: - DELETE FROM** *table_name* **WHERE** *condition*;
   - **Example:- DELETE FROM** Customers **WHERE** PersonID=2;
4. **SELECT: -** The SELECT statement is used to select data from a database.
   - **Syntax: - SELECT** *column1*, *column2...* **FROM** *table_name*; **SELECT** * **FROM** *table_name*;
   - **Example:-** SQL> **SELECT** PersonID, City **FROM** Persons; (Selects the "PersonID" and "City" columns from the "Persons" table)

     SQL> **SELECT** * **FROM** Customers; (Selects all the columns from the "Persons" table)

CALL Command

- Use the CALL statement to execute a routine (a standalone procedure or function) from within SQL.
- **Example:-** CALL my_procedure(3, 4)

SELECT DISTINCT

- Inside a table, a column often contains many duplicate values; and sometimes you only want to list the different (distinct) values.
- The distinct keyword is used to return only distinct (different) values.
- **Syntax: - SELECT DISTINCT** *column1, column2 ...FROM table_name*;
- **Example:- SELECT DISTINCT** `Country` **FROM** `Customers;`

"Customers" Table	
Sr. No.	**Country**
1	India
2	UK
3	India
4	Canada
5	UK

"Customers" Table	
Sr. No.	**Country**
1	India
2	UK
4	Canada

After DISTINCT keyword it produce the following output table.

WHERE Clause

- The WHERE clause is used to filter records.
- The WHERE clause is used to extract only those records that fulfill a specified condition.
- **Syntax: - SELECT** *column1, column2...* **FROM** *table_name* **WHERE** *condition*;
- **Example:- SELECT** * **FROM** `Customers` **WHERE** `CustomerID=1;`

3.2 SQL Operators (16 Marks)

3.2.1 Arithmetic Operators (Q. **List and explain any 4 arithmetic operators in SQL with example.)**

- Arithmetic operators are used to perform mathematical functions in SQL—the same as in most other languages. There are four conventional operators for mathematical functions:
  + (addition)
  - (subtraction)
  *** (multiplication)**
  / (division)

Operator	Description	Example
+ (addition)	Addition is performed through the use of the plus (+) symbol.	SELECT SALARY + BONUS FROM EMP;
- (subtraction)	Subtraction is performed using the minus (-) symbol.	SELECT SALARY - BONUS FROM EMP;
*** (multiplication)**	Multiplication is performed by using the asterisk (*) symbol.	SELECT SALARY * 10 FROM EMP;
/ (division)	Division is performed through the use of the —/‖ symbol.	SELECT SALARY / 10 FROM EMP;

3.2.2 Comparison Operators:-

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

<=	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.	(a <= b) is true.
!<	Checks if the value of left operand is not less than the value of right operand, if yes then condition becomes true.	(a !< b) is false.
!>	Checks if the value of left operand is not greater than the value of right operand, if yes then condition becomes true.	(a !> b) is true.

Example 1. SQL> SELECT * from customers where salary >15000;
SQL> Select *from customers where PersonID=15;

3.2.3 Logical Operators:

Operator	Description	Example
AND	The **AND** operator allows the existence of multiple conditions in an SQL statement's WHERE clause.	SQL> Select *from Persons Where FirstName = 'Vikas' **AND** LastName ='Jain';
OR	The **OR** operator is used to combine multiple conditions in an SQL statement's WHERE clause.	SQL> Select *from Persons Where FirstName = 'Vikas' **OR** LastName ='Jain';
NOT	The **NOT** operator reverses the meaning of the logical operator with which it is used. E.g.: NOT EXISTS, NOT BETWEEN, NOT IN, etc. **This is a negate operator.**	SQL> Select FirstName, LastName from Persons Where **NOT** Games='football';

3.2.4 Other Comparison Operators:-

Operator	Description	Example
LIKE	The LIKE operator is used to compare a value to similar values using wildcard operators.	SQL> Select FristName From Persons Where FirstName **LIKE** '_i%';
IN	The IN operator is used to compare a value is equal to any one of specified set of values.	SQL> Select FristName From Student Where subject **IN** ('Maths', 'Science');
BETWEEN …..AND	The BETWEEN operator is used to search for values that are within a set of values, given the minimum value AND the maximum value.	SQL> Select FristName From Student Where age **BETWEEN** 10 **AND** 15 ;
IS NULL	The NULL operator is used to compare a value with a NULL value.	SQL> Select FristName From Student Where games **IS NULL**;

3.2.5 SET Operator:-

- Set operators combine the results of two component queries into a single result. Queries containing set operators are called as compound queries. Set operators in SQL are represented with following special keywords as: **Union, Union all, intersection & minus**.

1. Union:-

- The UNION operator is used to combine the result-set of two or more SELECT statements.
  i. Each SELECT statement within UNION must have the same number of columns
  ii. The columns must also have similar data types
  iii. The columns in each SELECT statement must also be in the same order

- **Syntax:-** **SELECT** *column_name(s)* **FROM** *table1*
  UNION
  SELECT *column_name(s)* **FROM** *table2*;
- **Example 1**: - **SELECT** * **FROM** `Sales1` **UNION** **SELECT** * **FROM** `Sales2` ;

Sales1	
Person	**Amount**
Joe	1000
Alex	2000
Bob	5000

Sales2	
Person	**Amount**
Joe	2000
Alex	2000
Zach	35000

Output Table	
Person	**Amount**
Joe	1000
Alex	2000
Bob	5000
Joe	2000
Zach	35000

- **Example 2.:-** **SELECT** `City` **FROM** `Customers`
  UNION
  SELECT `City` **FROM** `Suppliers` **ORDER BY** `City`;
- Consider Two table like "Customers" & "Suppliers"

Customers			
C_ID	**C_Name**	**City**	**Country**
1	Alex	Pune	India
2	Bob	Mumbai	India
3	Zach	Berlin	Germany

Suppliers			
S_ID	**S_Name**	**City**	**Country**
1	Alice	Shaman	China
2	Bob	Mumbai	India
3	Joe	Berlin	Germany

Output
Country
China
Germany
India

2. **Union All: -** The following SQL statement selects all values (duplicate values also) from Table.

- **Syntax:- SELECT** *column_name(s)* **FROM** *table1*
  UNION ALL
  SELECT *column_name(s)* **FROM** *table2*;
- **Example 1 : - SELECT** * **FROM** `Sales1` **UNION ALL SELECT** * **FROM** `Sales2` ;

Sales1	
Person	**Amount**
Joe	1000
Alex	2000
Bob	5000

Sales2	
Person	**Amount**
Joe	2000
Alex	2000
Zach	35000

Output Table	
Person	**Amount**
Joe	1000
Alex	2000
Bob	5000
Joe	2000
Alex	2000
Zach	35000

3. **Intersect: -** The SQL Intersect operator takes the results of two queries & returns only rows that appear in both result sets.

- **Syntax:- SELECT** *column_name(s)* **FROM** *table1*
  INTERSECT
  SELECT *column_name(s)* **FROM** *table2*;
- **Example 1 : - SELECT** * **FROM** `Sales1`
  INTERSECT
  SELECT * **FROM** `Sales2`;

Sales1	
Person	**Amount**
Joe	1000
Alex	2000
Bob	5000

Sales2	
Person	**Amount**
Joe	2000
Alex	2000
Zach	35000

Output Table	
Person	**Amount**
Alex	2000

4. **Minus: -** The SQL MINUS query returns all rows in the first SQL SELECT statement that are not returned in the second SQL SELECT statement.

- **Syntax:- SELECT** *column_name(s)* **FROM** *table1*

MINUS
SELECT *column_name(s)* **FROM** *table2*;

- **Example 1**: - **SELECT** * **FROM** Sales1
  MINUS
  SELECT * **FROM** Sales2;

Sales1	
Person	**Amount**
Joe	1000
Alex	2000
Bob	5000

Sales2	
Person	**Amount**
Joe	2000
Alex	2000
Zach	35000

Output Table	
Sales1 - Sales2	
Person	**Amount**
Joe	1000
Bob	5000

Output Table	
Sales2 – Sales1	
Person	**Amount**
Joe	2000
Zach	35000

3.2.6 SQL (Oracle) Functions

- **Two Types of SQL Functions**
  1. **Single-row functions**
     a. Manipulate data items
     b. Accept arguments and return one value
     c. Act on each row returned
     d. Return one result per row
     e. May modify the data type
     f. Can be nested

     ***Syntax: - function_name* (*column* | *expression*, [*arg1, arg2*...])**

  2. **Multiple-row or Group functions**

- **Single-row functions :-** There are four types of single row functions are:
  1. **String Functions**

FUNCTION	USE/DEFINITION
INITCAP	Capitalizes the first letter of a string of characters
INSTR	Searches a character string for a character string subset and returns the start position and/or occurrence of the substring
LOWER	Returns a character value that is all lower case
UPPER	Returns a character value that is all upper case
LTRIM	Trims specified characters from the left end of a string
RTRIM	Trims specified characters from the right end of a string

LPAD	It is used for formatting from left side by using any character.
RPAD	It is used for formatting from right side by using any character.
LENGTH	Returns a numeric value equivalent to the number of characters in a string of characters
SUBSTR	Returns a string of specified *length* from a larger character string beginning at a specified character *position*

Consider following "Person" table

LastName	FirstName	Address	City
Sharma	Rajesh	GM Road	Pune
Keri	John	CAMP	Pune
Kari	Johnson	RTO Road	Banglore

1. INITCAP: - It display the initial letter as capital.

- **Syntax: -** INITCAP ('String')
- **Example:-** SQL> Select Initcap(city) from Person;
- **Result Table:-**

City
Pune
Pune
Banglore

2. LOWER & UPPER:- Returns a character value that is all lower/ Upper case

- **Syntax: - Lower ('String')**
  Upper ('String')
- **Example:-** SQL> Select lower(city) from Person;
  SQL> Select upper (city) from Person;
- **Result Table:-**

City
pune
pune
banglore

City
PUNE
PUNE
BANGLORE

3. Ltrim & Rtrim: - It trims or cuts the character from left or right.

- **Syntax: - Ltrim('String','trim text')**
  Rtrim ('String', 'trim text')

- **Example:-** SQL> Select Ltrim('Pune', 'P') from Person;
  SQL> Select Rtrim ('Pune', 'e') from Person;
- **Result Table:-**

City
une
une
banglore

City
pun
pun
banglore

4. Translate: - It is used to translate the given character from input string.

- **Syntax: - Translate ('main string' , 'string to replaced', ' string to be replaced by')**
- **Example:-** SQL> Select translate ('Banglore', 'B' , ' M') from Person;
- **Result Table:-**

City
pune
pune
Manglore

5. Substr: - It returns the substring from specified position.

- **Syntax: - Substr ('main string' , position, character to be replaced')**
- **Example:-** SQL> Select Substr (city, 1, 3) from Person;
  - **Result Table:-**

City(Substr)
pun
pun
Man

6. LPAD & RPAD: - It is used for formatting from left & right side by using any character.

- **Syntax: - Lpad ('main string' , length, character to be padded')**
  Rpad ('main string', length, character to be padded')
- **Example:-** SQL> Select Lpad (city, 10, ' * ') from Person;
  SQL> Select Rpad (city, 10, '& ') from Person;
- **Result table:-**

City
******Pune
******Pune
**Banglore

City
Pune &&&&&&
Pune &&&&&&
Banglore &&

7. Concatenation: - It is used to concatenate combining two strings.

- **Syntax: - String || String**

- **Example:- SQL> Select ('the first name is' || firstname || ' and city is ' || city) from Person;**
- **Result Table:-**

the first name is **rajesh** and city is **pune**
the first name is **john** and city is **pune**
the first name is **johnson** and city is **banglore**

8. Length: - It is used to calculate the length of given string.

- **Syntax: - length ('string')**
- **Example:-** SQL> Select length (city) from Person;
- **Result Table:-**

City(Length)
4
4
8

2. Numeric Function

Function	Description	Syntax	Example	Output
ABS (absolute)	It returns the absolute value of 'n'.	ABS(-15.36)	SQL> Select ABS(-15.36) from dual;	15.36
Power	It returns m raised to the nth power. Nth must be an integer else an error is returned.	Power (m, n)	SQL> select power (3,2) "raised" from dual;	6
Round (X,Y)	Round the value of number x to y decimal places.	round(X, [Y])	SQL> select round (140.234, 2) from dual;	140.23
TRUNC	Truncates the value of number x to y decimal places.	trunk (no, [decimal_places])	SQL> select trunk (125.815,1) from dual;	125.8
SQRT	Returns square root of n.	sqrt(n)	select sqrt(25) from dual;	5
EXP(e)	Returns e raised to the nth power e	exp(n)	select exp (5) from dual;	148.41315
GREATET	Returns a greatest value in a list of expressions.	Greatest (expr1,expr2,expr3…exprn)	select greatest(4,5,17) from dual;	17
LEAST	Returns the least value in a list of expressions.	least(expr1,expr2,…..,exprn);	select least(4,5,17) from dual;	4
MOD	Returns the remainder of a first number divided by second number passed a parameter.	mod(m, n)	select mod(15,7) from dual;	1

FLOOR	Return a largest integer value that is equal to less than a number.	floor(n)	select floor(24.8) ”flr1”, floor(13.15)”flr2” from dual;	24 13
CEIL	Return the smallest integer value that is greater than or equal to a number.	ceil(n)	select ceil(24.8)”ceil”, ceil(13.15)”ceil2” from dual;	25 14
Log(X)	Function will return the natural algorithm of X.	log(X)	select log(45) from dual;	3.806662

3. Date & Time Functions

- Oracle stores dates in an internal numeric format: century, year, month, day, hours, minutes, seconds.
- The default date format is DD-MON-YY.
- SYSDATE is a function returning date and time.
- DUAL is a dummy table used to view SYSDATE.

Function	Description	Return Value
months_between(d1,d2) Where, d1 and d2 are dates	Used to find number of months between d1 and d2. **Example: Select months_between (‘05-Sep-1996‘, ‘05-Dec-1996‘) from dual;**	**3**
add_months (d, n) Where, d is date, n no of months to be added	Returns date after adding the number of months specified with the function. **Example: Select add_months(‘16-Sep-17’,3) from dual;**	**16-Dec-17**
Next_day (d, char) Where d is date, char- day of week	Returns the date of the first weekday named char that is after the date named by date. **Example: Select next_day (‘01-Sep-2017‘, ‘Wednesday‘) from dual;**	**06-Sep-17**
Last_day (d) Where, d is date	Returns the last day of the month that contains date d. **Example: Select last_day (‘1-Aug-17’) from dual**	**31-Aug-17**

Round(date, [fmt]) Where, fmt format model Month Day Year	Returns date rounded to the unit specified by the format model fmt. **Example: Select ('25-JUL-95', 'MONTH ') from dual;** **SQL> Select ('25-JUL-95', 'MONTH ') from dual;**	**01-AUG-95** **01-JAN-96**
Trunc(date ([fmt]) Where, fmt format model Month Day Year	Returns date with the time portion of the day truncated to the unit specified by the format model fmt. **Example: Select trunc('25-JUL-95', 'MONTH') from dual;** **SQL>Select trunc ('25-JUL-95', 'YEAR') from dual;**	**01-JUL-95** **01-JAN-95**

4. Conversion Functions

- TO_CHAR and TO_DATE Functions
- The functions are used to format output and to convert data from one data type to another.

Function	Description	Return Value
To_Char (X, [Y])	Converts numeric & date values to a character string value. **Example: - Select To_char (sysdate, 'Day, Month YYYY') from dual.**	**Monday, Sep 2017**
To_DATE (X, [date_format])	Converts a valid numeric & character values to date value. **Example:- Select To_Date ('January 15, 2017) from dual;**	**15-Jan-17**

2. Group Function/Aggregate Functions

Q. What is an aggregate function?

- An aggregate function summarizes the results of an expression over a number of rows, returning a single value.
- The general syntax for most of the aggregate functions is as follows:
- Syntax:- Aggregate_function ([DISTINCT | ALL] expression)
- Some of the commonly used aggregate functions are :
- **SUM**
- **COUNT**
- **AVG**
- **MIN**
- **MAX**
- Consider following table:-

Sno	Fname	Salary	Position
SL100	John	30000.00	Manager
SL101	Susan	24000.00	Manager
SL102	David	12000.00	Project Manager
SL103	Ann	12000.00	Project Manager
SL104	Mary	9000.00	Project Manager

Function	Description	Output
Avg(column)	Returns: The average of the values in a specified column. Syntax:- SELECT AVG(*column_name*) FROM *table_name* WHERE *condition*; **Example:-** **SQL>Select avg (salary) from staff;** **SQL> Select Avg(salary) from staff where position= 'manager';**	 **17400** **27000**
SUM(column)	Returns: The sum of the values in a specified column. Syntax:- SELECT SUM(column_name) FROM table_name WHERE condition; **Example:-** **SQL>Select sum (salary) from staff;** **SQL> Select sum(salary) from staff where position= 'manager';**	 **87000** **54000**

MAX(column)	Returns: MAX () returns the largest value of a column. Syntax:- SELECT max(*column_name*) FROM *table_name*; **Example:-** **SQL>Select max (salary) from staff;**	**30000**
MIN(column)	Returns: MIN () returns the smallest value of a column. Syntax:- SELECT min(column_name) FROM table_name; **Example:-** **SQL>Select min (salary) from staff;**	**9000**
Count(column)	Returns: The number of values in the specified column. Syntax:- SELECT COUNT(column_name) FROM table_name WHERE condition; **Example:- SQL>Select count (Sno) from staff;** **SQL>SELECT COUNT(Sno) FROM Staff WHERE position = 'Manager';**	**5** **2**

3.2.7 Use of GROUP BY Clause

- **1. GROUP BY**: It groups the data from the SELECT table and produce a single summary row for each group
- When Using GROUP BY:
  1. Each item in the SELECT list must be single valued per group.
  2. SELECT clause may only contain: Columns names, Aggregate function, Constants, An expression involving combinations of the above.
- **Syntax:- SELECT** column_name, sum(column_name) **FROM** table **GROUP BY** column_name;
- **Consider following Table:-**

"Sales" Table

C_Code	Company	Turnover(Cr)
B1	Atlas	9500
B2	Infosys	4500
B3	Wipro	7100
B1	Atlas	2500
B2	Infosys	3500

- **Example:- SELECT** c_code, company, sum(turnover) **FROM** sales **GROUP BY** company;

C_Code	Company	Turnover(Cr)
B1	Atlas	12000
B2	Infosys	8000
B3	Wipro	7100

Resultant Table

2. Having Clause

- Having clause was added to SQL because the where keyword could not be used against aggregate functions (like sum, avg), & without having clause would be impossible to test for result conditions.
- **Syntax:-**
  SELECT column, sum (column) **FROM** table **GROUP BY** column **HAVING** sum (column) condition value;
- **Example:-**

```
SELECT c_code, company, sum (turnover)
FROM sales GROUP BY company
HAVING sum (amount)>10000;
```

C_Code	Company	Turnover(Cr)
B1	Atlas	12000

Resultant Table

3. Order BY Clause

- The ORDER BY clause is used in a SELECT statement to sort results either in ascending or descending order.
- Oracle sorts query results in ascending order by default.
- Syntax:-
  SELECT column-list
  FROM table_name [WHERE condition]
  [**ORDER BY** column1 [, column2... column] [DESC/ASC]];
- **For Example:** If you want to sort the staff table by salary of the staff. The SQL query would be.
- SQL> **SELECT** name, salary
  FROM staff
  ORDER BY salary;

Fname	Salary
Mary	9000.00
David	12000.00
Ann	12000.00
Susan	24000.00
John	30000.00

Resultant Table

- SQL> **SELECT** name, salary
  FROM Staff
  ORDER BY name, salary DESC;

Fname	Salary
John	30000.00
Susan	24000.00
David	12000.00
Ann	12000.00
Mary	9000.00

Resultant Table

3.2.8 DCL (Data Control Language) Commands

- Data Control Language provides database administrators with the ability to grant users database permissions, revoke permissions previously granted.
- Two types of DCL commands are:

1. Grant and
2. Revoke.

1. Grant:-

- To allow specified users to perform specified tasks
- SQL GRANT is a command used to provide access or privileges on the database objects to the users.

Syntax:-

GRANT privilege_name
ON object_name
TO {user_name |PUBLIC |role_name}
[WITH GRANT OPTION];

Where,

- ***Privilege_name*** is the access right or privilege granted to the user. Some of the access rights are ALL, EXECUTE, SELECT, ALTER, DELETE, INSERT, and UPDATE.
- ***Object_name*** is the name of a database object like TABLE, VIEW, STORED PROC and SEQUENCE.
- ***User_name*** is the name of the user to whom an access right is being granted.
- ***PUBLIC*** is used to grant access rights to all users.
- ***ROLES*** are a set of privileges grouped together.
- ***WITH GRANT OPTION*** - allows a user to grant access rights to other users.

Example:-

- a). GRANT SELECT **ON** employee TO **user1 (**This command grants a SELECT permission on employee table to user1.)
- b). Grant all privileges on student table to user Prakash.
  SQL> **GRANT** All **ON** student **TO** Prakash

2. Revoke:-

- The revoke command removes user access rights or privileges to the database objects.

Syntax:-

REVOKE privilege_name
ON object_name
FROM {user_name |PUBLIC |role_name}

Example:-

a) **REVOKE** SELECT **ON** employee **FROM** user1; (This command will revoke a SELECT privilege on employee table) from user1.
b) Revoke delete privilege on student table from Prakash.
SQL>REVOKE Delete **ON** student **FROM** Prakash.

3.2.9 TCL (Transaction Control Language) Commands

- A Transaction begins with the first executable SQL statement after a Commit, Rollback or Connection made to the Oracle engine.
- All changes made to an Oracle table data via a transaction are made or undo at one instance.
- SQL Transaction Control Language commands are used for managing changes affecting the data.
- These commands are COMMIT, ROLLBACK and SAVEPOINT.

1. COMMIT:-

- Commit command is used to permanently save any transaction into database.
- The COMMIT command is used to save changes invoked by a transaction to the database.
- The COMMIT command saves all transactions to the database since the last COMMIT command.
- The **syntax** for COMMIT command is as follows:
- **SQL> COMMIT;**

2. ROLLBACK:-

- This command restores the database to last committed state.
- The ROLLBACK command is 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.
- We can either rollback the entire transaction or till a particular save point transaction can be rolled back.
- The **syntax** for ROLLBACK is: **ROLLBACK TO SAVEPOINT_NAME**; OR **ROLLBACK;**
- **Example:- ROLLBACK TO sv1;** OR **ROLLBACK;**

3. SAVEPOINT:-

- Savepoint command is used to temporarily save a transaction so that you can rollback to that point whenever necessary.
- **Syntax:**
  SAVEPOINT <Save_Point_Name>
- **Example:-**
  SAVEPOINT A;

Example of Commit, Savepoint and Rollback

1. Following is the Student table,

ID	NAME
1	abhi
2	adam
4	alex

2. Let's use some SQL queries on the above table and see the results.

```
SQL> INSERT into student values (5,'Rahul');
SQL> commit;
SQL> UPDATE student set name='abhijit' where id='5';
SQL> savepoint A;
SQL> INSERT into student values (6,'Chris');
SQL> savepoint B;
SQL> INSERT into student values (7,'Bravo');
SQL> savepoint C;
SQL> SELECT * from Student;
```

3. The resultant table will look like,

ID	NAME
1	abhi
2	adam
4	alex
5	abhijit
6	chris
7	bravo

4. Now rollback to savepoint B

```
SQL> rollback to B;
SQL> SELECT * from Student;
```

5. The resultant table will look like

ID	NAME
1	abhi
2	adam
4	alex
5	abhijit
6	chris

6. Now rollback to savepoint B

SQL> rollback to A;
SQL> SELECT * from Student;

7. The resultant table will look like

ID	NAME
1	abhi
2	adam
4	alex
5	abhijit

3.2.10 Transaction Concept

- A transaction is a *unit* of program execution that accesses and possibly updates various data items.
- E.g., transaction to transfer $50 from account A to account B:
    1. read(A)
    2. $A := A - 50$
    3. write(A)
    4. read(B)
    5. $B := B + 50$
    6. write(B)
- Two main issues to deal with:
    - Failures of various kinds, such as hardware failures and system crashes
    - Concurrent execution of multiple transactions

Transaction Properties (ACID)

- To ensure integrity of the data, the database system must maintain the following properties of transaction.
  1. Atomicity
  2. Consistency
  3. Isolation
  4. Durability

1. Atomicity (all or nothing)

- Atomicity means either all operations will take place property and reflect in the database or none of them will be reflected.

2. Consistency (no violation of integrity constraints)

- Consistency keeps the database consistent. It helps in reducing complications of executing multiple transactions at a time and preserves the consistency of the database.

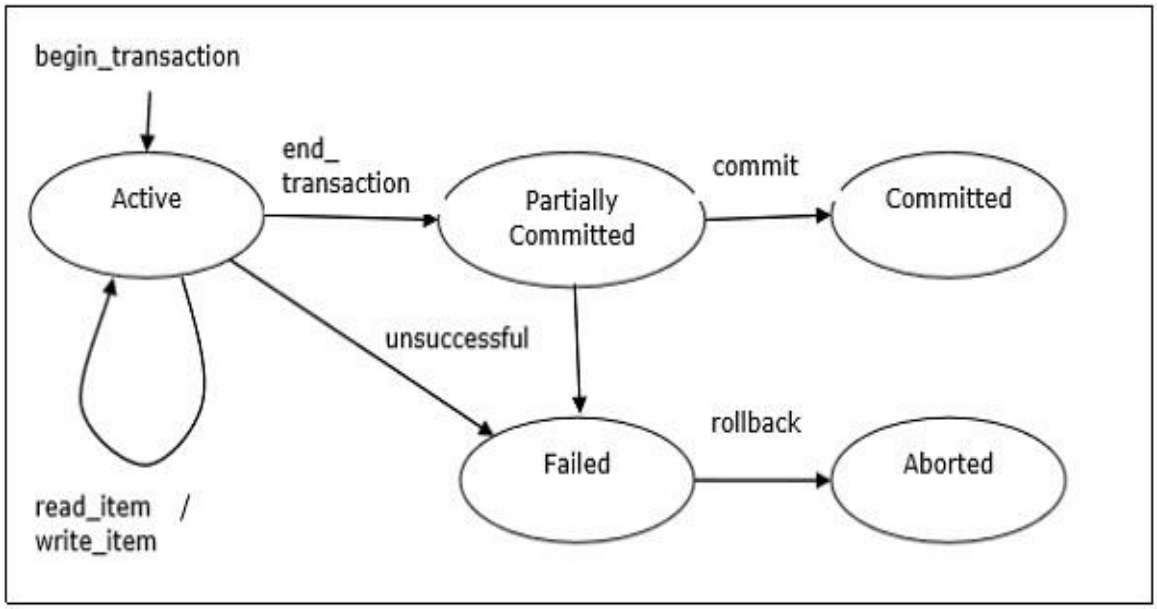
3. Isolation (concurrent changes invisible -> serializable)

- It is necessary to maintain isolation for the transactions. This means one transaction should not be aware of another transaction getting executed. Also their intermediate result should be kept hidden.

4. Durability (committed updates persist)

- After a transaction completes successfully, the changes it has made to the database persist, even if there are system failures such as a crash.

State of a transaction



1. Active – The initial state; the transaction stays in this state while it is executing.

2. Partially committed – The transaction ends after execution of final statement ("commit requested").

3. Failed -- after the discovery that normal execution can no longer proceed.

4. Aborted – after the transaction has been rolled back and the database restored to its state prior to the start of the transaction.

Two options after it has been aborted:

1. Restart the transaction
   - can be done only if no internal logical error
2. Kill the transaction

5. Committed – after successful completion.

Concurrent Executions

- Multiple transactions are allowed to run concurrently in the system. Advantages are:
  - Increased processor and disk utilization, leading to better transaction *throughput*
    - E.g. one transaction can be using the CPU while another is reading from or writing to the disk
  - Reduced average response time for transactions: short transactions need not wait behind long ones.
- The transaction processing system allows multiple transactions to run concurrently.
- This causes several complication with the consistency of the data.
- To avoid the complications it is better to run the transactions serially.

1. Serial schedule

- The sequence of execution of transaction is known as schedule.
- **Example: - consider the transaction T1 & T2 that transfer funds from account A to account B.**
- **Schedule1:** Suppose the current value of A is A=500 & current value of B is B=1000. the two transactions are executing in the sequence T1 followed by T2.

(Table 1. serial schedule1)

Task T1	Task T2	Process
READ(A);		A=500
A:=A-50;		A=500-50=450
WRITE(A);		A=450
READ(B);		B=1000
B:=B+50;		B=1000+50
WRITE(B);		B=1050
		A+B= 450+1050= 1500 is preserved
	READ(A);	A=450
	A:=A-100;	A=450-100=350
	WRITE(A);	A=350
	READ(B);	B=1050
	B:=B+100;	B=1050+100=1150
	WRITE(B);	B=1150
		A+B= 350+1150= 1500 is preserved

Table. 1 Serial schedule1 T1 followed by T2
after executions the values of A & B are 350 & 1150 respectively

- **Schedule2:** Suppose we change the sequence transactions T2 followed by T1 the addition of A+B should be preserved. (fig. 2 serial schedule2)

Task T1	Task T2	Process
	READ(A);	A=500
	A:=A-100;	A=500-100=400
	WRITE(A);	A=400
	READ(B);	B=1000
	B:=B+100;	B=1000+100
	WRITE(B);	B=1100
		A+B= 400+1100= 1500 is preserved
READ(A);		A=400
A:=A-50;		A=400-50=350
WRITE(A);		A=350
READ(B);		B=1100
B:=B+50;		B=1100+50=1150
WRITE(B);		B=1150
		A+B= 350+1150= 1500 is preserved

Table. 2 Serial schedule2 T2 followed by T1
after changing the sequence the addition will be same.
So serial schedule 1 is equal to serial schedul2

2. Concurrent Schedule

- When the multiple transactions are executing concurrently in the system the schedule is said to be concurrent.
- It two transactions are executing concurrently then operating system will execute first transaction then perform a context switch & will execute second transaction & will again switch back to first.
- When multiple transactions are executing the CPU time is shared among all the transactions.

Task T1	Task T2	Process T1	Process T2
READ(A);		A=500	
A:=A-50;		A=500-50=450	
WRITE(A);		A=450	
	READ(A);		A=450
	A:=A-100;		A=450-100
	WRITE(A);		A=350

READ(B);		B=1000	
B:=B+50;		B=1000+50	
WRITE(B);		B=1050	
	READ(B);		B=1050
	B:=B+100;		B=1050+100
	WRITE(B);		1150

Table. 3 Concurrent schedule
after executions the values of A & B are 350 & 1150 respectively
output of this schedule is equivalent to the serial schedule.

3.2.11 Serializability

- It is sequence of actions (read, write, commit or abort) from set of transactions.
- Basic Assumption – Each transaction preserves database consistency.
- Thus, serial execution of a set of transactions preserves database consistency.
- A (possibly concurrent) schedule is serializable if it is equivalent to a serial schedule. Different forms of schedule equivalence give rise to the notions of:
  1. Conflict serializability
  2. View serializability

1. Conflict Serializability:-

- A *conflict* occurs when one transaction in a schedule WRITEs a data item which another transaction also uses (READs or WRITEs).
- Let l_i and $l\ j$ be two Instructions of transactions *Ti* and *T j* respectively. Instructions l_i and l_j conflict if and only if there exists some item *Q* accessed by both l_i and l_j, and at least one of these instructions wrote *Q*.
  1. l_i = read(*Q*), l_j = read(*Q*). l_i and l_j don't conflict.
  2. l_i = read(*Q*), l_j = write(*Q*). They conflict.
  3. l_i = write(*Q*), l_j = read(*Q*). They conflict
  4. l_i = write(*Q*), l_j = write(*Q*). They conflict
- Intuitively, a conflict between l_i and l_j forces a (logical) temporal order between them.
- If l_i and l_j are consecutive in a schedule and they do not conflict, their results would remain the same even if they had been interchanged in the schedule.
- If a schedule *S* can be transformed into a schedule *S1* by a series of swaps of non-conflicting instructions, we say that *S* and *S1* are conflict equivalent.
- We say that a schedule *S* is conflict serializable if it is conflict equivalent to a serial schedule.

2. **View Serializability**

   - Let *S1* and *S2* be two schedules with the same set of transactions. *S1* and *S2* are view equivalent if the following three conditions are met, for each data item *Q,*

   1. If in schedule S1, transaction T_i reads the initial value of *Q*, then in schedule *S2* also transaction T_i must read the initial value of *Q*.
   2. If in schedule S1 transaction T_i executes read(*Q*), and that value was produced by transaction T_j (if any), then in schedule *S2* also transaction T_i must read the value of *Q* that was produced by the operation of transaction T_j .
   3. The transaction (if any) that performs the final write(*Q*) operation in schedule *S1* must also perform the final write(*Q*) operation in schedule *S2.*

   As can be seen, view equivalence is also based purely on reads and writes alone.

3.2.12 Inner join and Outer join (Q. Explain Inner join and Outer join with example.)

- INNER Join: This is a simple JOIN in which the result is based on matched data as per the condition specified in the query.
- Inner Join Syntax :
  SELECT column_name_list **FROM** table_name1
  INNER JOIN
  table_name2 ON table_name1.column_name = table_name2.column_name;
- Inner Join Example :
  SELECT * **FROM** emp **INNER JOIN** dept **ON** emp.id = dept.id;

- Outer Join is based on both matched and unmatched data. Outer Joins subdivide further into,
  - Left Outer Join
  - Right Outer Join
  - Full Outer Join

1. **Left Outer Join**
   The left outer join returns a result table with the matched data of two tables then remaining rows of the left table and null for the right table's column.
   - **Left Outer Join syntax :**
     SELECT column-name-list **FROM** table-name1 **LEFT OUTER JOIN** table-name2 **ON** table-name1.column-name = table-name2.column-name;

- Left Outer Join Example:

  SELECT * FROM emp LEFT OUTER JOIN dept ON (emp.id=dept.id);

2. **Right Outer Join**
   - The right outer join returns a result table with the matched data of two tables then remaining rows of the right table and null for the left table's columns.
   - **Right Outer Join Syntax:**
     SELECT column-name-list **FROM** table-name1 **RIGHT OUTER JOIN** table-name2
     ON table-name1.column-name = table-name2.column-name;
   - **Right Outer Join Example:**
     SELECT * **FROM** emp **RIGHT OUTER JOIN** dept **ON** (emp.id=dept.id)
3. **Full Outer Join**
   - The full outer join returns a result table with the matched data of two table then remaining rows of both left table and then the right table.
   - **Full Outer Join Syntax :**
     SELECT column-name-list **FROM** table-name1 **FULL OUTER JOIN** table-name2
     ON table-name1.column-name = table-name2.column-name;
   - **Full Outer Join Example:**
     SELECT empname, sal **FROM** emp **FULL OUTER JOIN** dept **ON** emp.id = dept.id;

Example: - (On Select Query)

1. **Consider the following database: Employee (emp_id, emp_name, emp_city, emp_addr, emp_dept, join_date) Solve the following query:**

i) Display the names of employees in capital letters.

ii) Display the emp_id of employee who live in city Pune and Mumbai.

iii) Display the details of employees whose joining date is after '01-Apr.-1997'.

iv) Display the total number of employees whose dept.no.is '10'. *(Each correct query - 1 mark)*

Ans:- i) Select upper(emp_name) from Employee;
ii) Select emp_id from Employee where emp_city = 'Pune'or emp_city = ‗Mumbai';
iii) Select * from Employee where join_date>‗01-Apr-1997';
iv) Select count (emp_id) from Employee where emp_dept = 10;

2. **Consider the structure for book table as Book_master {book_id, book_name, subcode, author, no_of_copies, price}. Write SQL queries for the following:**

i) Display total no. of book for subject 'DBM'.

ii) Get authorwise list of all books.

iii) Display all books whose prices are between Rs. 200 and Rs.500

Ans: - i. Select sum (no_of_copies) from Book_master where book_name='DBM';

ii. Select author, book_name From Book_master Order by author;

iii. Select book_id From Book_master Where price between 200 and 500;

OR

iii. Select book_id From Book_master Where price >= 200 and price <= 500;

SUMMER-16 (38 Marks)

3. Draw the state diagram of transaction.
4. Explain the steps used in query processing with suitable diagram. *(Diagram - 2 marks; Explanation - 2 marks)*
5. Explain ACID properties of transaction. *(Four ACID properties - 1 mark each)*
6. Describe Commit and Rollback with syntax. *(For each command explanation - 1 mark; syntax - 1 mark)*
7. Consider the following database:

Employee (emp_id, emp_name, emp_city, emp_addr, emp_dept, join_date) Solve the following query:

i) Display the names of employees in capital letters.

ii) Display the emp_id of employee who live in city Pune and Mumbai.

iii) Display the details of employees whose joining date is after '01-Apr.-1997'.

iv) Display the total number of employees whose dept.no.is '10'. *(Each correct query - 1 mark)*

8. Describe Grant and Revoke commands. *(Description of Grant - 2 marks; Revoke - 2 marks)*
9. Describe string function, date and time function. *(Any two String Function - 2 marks; any two Date and Time Function - 2 marks)*
10. Explain with example group by and having clause. *(For each clause - Explanation - 1 mark, syntax/example - 1 mark)*
11. List and explain any 4 arithmetic operators in SQL with example. *(For each - 1 mark)*
12. Consider the structure for book table as Book_master {book_id, book_name, subcode, author, no_of_copies, price}. Write SQL queries for the following:

i) Display total no. of book for subject 'DBM'.

ii) Get authorwise list of all books.

iii) Display all books whose prices are between Rs. 200 and Rs.500

WINTER– 16

1. List four DDL commands.
2. List DCL commands any four.

3. Explain DELETE and DROP Command with syntax and example.
4. Consider the following database Employee (emp-id, emp-name, emp-city, empaddr, emp-dept, join-date)
   i) Display the emp-id of employee who live in city Pune or Nagpur.
   ii) Display the details of employee whose joining date is after 02-July-2007.
   iii) Change employee name 'Ajit' to 'Aarav'.
   iv) Display the total number of employees whose dept is '50'.
5. Give the syntax and example of CREATE and RENAME Commands.
6. Explain ALTER command with any two options.
7. Describe ACID properties of transaction.
8. Explain any four string functions with example.
9. Consider the following database schema: EMP (Empno, Ename, job, mgr, joindate, salary, comm., deptno). Write the SQL queries for the following:
   i) Write a query to find list of employees whose salary is not less 5000.
   ii) Write a query to find list of employees whose job title is either "Manager" or "Analyst".
   iii) Change the location of deptno 40 to Pune from Chandrapur.
   iv) Display the Ename and salary of employees who earn more than Rs. 50,000 and are in deptno 10 or 30.
10. Give the use of grant and revoke command with syntax and example.
11. Explain Inner join and Outer join with example.

WINTER – 15

1. Explain the use of truncate statement. Give example.
2. Consider the structure of student (Name, Mark, Age, Place, Phone, Birthdate). Write SQL quries for the following:
   i. To list name of student who do not have phone number
   ii. To list students from Mumbai and Pune.
   iii. To change mark of 'Ajay' to 88 instead of 80.
   iv. To list the students whose age is more than 12. (Correct query – 1 Mark each)
3. Describe how to delete the data from table with an example. (Description - 2 Marks; Syntax - 1 Mark; Example - 1 Mark)
4. Explain the set operator with help of example. (Each operator - 1 Mark)
5. Explain on delete cascade clause with suitable example. (Explanation - 2 Marks, Example – 2 Marks)
6. Explain any four date function with example. (Any four functions -1 Mark Each)
7. Explain ACID properties of transaction. (Each property – 1 Mark)