RDBMS – Important questions with Answer

1. List out DML commands and explain any one with example.

- DML (Data Manipulation Language) commands in DBMS are used to manage data within database objects. The primary DML commands are:

1. **SELECT**: Retrieves data from the database.
2. **INSERT**: Adds new data into a database table.
3. **UPDATE**: Modifies existing data in a database table.
4. **DELETE**: Removes data from a database table.

INSERT Command

- The INSERT command is used to add new rows to a table.
- Here is the general syntax:

```
INSERT INTO table_name (column1, column2, column3, ...)
VALUES (value1, value2, value3, ...);
```

Example

```
INSERT INTO employees (id, name, position, salary) VALUES (1, 'abc', 'Software Engineer', 75000.00);
```

2. List out DDL commands and explain with example

- DDL (Data Definition Language) commands in DBMS are used to define and manage all the database objects such as tables, indexes, and schemas. The primary DDL commands are:

1. **CREATE**: Creates a new database object, such as a table, view, or index.
2. **ALTER**: Modifies an existing database object.
3. **DROP**: Deletes an existing database object.
4. **TRUNCATE**: Removes all records from a table, including all spaces allocated for the records.

CREATE Command

- The CREATE command is used to create a new database object.
- Here is the general syntax for creating a table:

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

Example

- Suppose we want to create a table named students to store student information. The table should include the following columns: id, name, age, and grade.

```
CREATE TABLE students
( id number, name VARCHAR(100),
age INT, grade CHAR(1) );
```

3. What is Index? Explain Composite INDEX with an example.

- An **index** in a database is a structure that improves data retrieval speed.

Composite Index

- A composite index is created on multiple columns of a table, enhancing performance for queries that involve those columns.

Example

```
CREATE INDEX idx_lastname_department ON employees (last_name, department);
```

4. Explain synonyms. How to create and destroy it?

- In a database management system, a **synonym** is an alias or an alternate name for a database object such as a table, view, sequence, procedure, or another synonym.
- Synonyms are used to simplify SQL statements for database users by providing a more user-friendly name and to hide the underlying object details from users.

Creating a Synonym

- To create a synonym, you use the CREATE SYNONYM statement.
- The basic syntax is:
  CREATE SYNONYM synonym_name FOR object_name;

Example

- Suppose you have a table named employees and you want to create a synonym called emp:

```
CREATE SYNONYM emp FOR employees;
```

Dropping a Synonym

- To remove a synonym, you use the DROP SYNONYM statement.
- The basic syntax is:
  DROP SYNONYM synonym_name;

Example

```
DROP PUBLIC SYNONYM emp;
```

5. What is sequence? How to create it? How to destroy it?

- In a database management system, a **sequence** is a database object that generates a sequence of unique numbers. These numbers are often used for auto-generating primary key values.

Creating a Sequence

- To create a sequence, you use the CREATE SEQUENCE statement.
- Here is the basic syntax:

```
CREATE SEQUENCE sequence_name
START WITH start_value
```

```
INCREMENT BY increment_value
[MINVALUE min_value]
[MAXVALUE max_value]
[CYCLE | NOCYCLE]
[CACHE cache_size | NOCACHE];
```

- sequence_name: The name of the sequence.
- START WITH start_value: The starting value of the sequence.
- INCREMENT BY increment_value: The value by which the sequence is incremented.
- MINVALUE min_value: The minimum value of the sequence (optional).
- MAXVALUE max_value: The maximum value of the sequence (optional).
- CYCLE | NOCYCLE: Whether the sequence should restart from the beginning when the maximum value is reached (CYCLE) or not (NOCYCLE).
- CACHE cache_size | NOCACHE: Specifies how many sequence numbers to cache for performance (optional).

Example

- Creating a sequence that starts at 1, increments by 1, and caches 10 values:

```
CREATE SEQUENCE employee_seq
START WITH 1
INCREMENT BY 1
CACHE 10;
```

Dropping a Sequence:

```
DROP SEQUENCE sequence_name;
```

6. Define VIEW, give types of it and how to create it? Also give advantages.

- A **VIEW** in a database is a virtual table that is based on the result-set of an SQL query.
- It contains rows and columns just like a real table and can be used in queries just like a real table.

Types of Views

- There are two main types: **Read-Only Views** and **Updatable Views**

Creating a View

```
CREATE [ OR REPLACE ] VIEW viewName
As SELECT … ….
[ WITH READ ONLY ];
```

- This statement creates a view based on query specified in **SELECT** statement.
- **OR REPLACE**option re-creates the view if it is already existing maintaining the privileges granted to view that is given by viewName.
- **WITH READ ONLY** option creates **read-only views**. If this option is not provided then **by defaultupdatable views** are created.

Example:

```
CREATE VIEW Acc_vvn
AS SELECT * FROM Account
WHERE bname = 'vvn';
```

Advantages of Views

1. **Simplified Querying**: Views simplify complex queries by encapsulating them into a single table-like structure.
2. **Security**: Views can be used to provide specific users access to certain data without giving them direct access to the underlying tables.
3. **Data Abstraction**: Views provide a level of abstraction by hiding the complexity of the underlying tables and their relationships.
4. **Consistency**: Views can present a consistent, unchanged interface to the data even if the underlying tables are altered.
5. **Reuse**: Views allow for the reuse of common queries, making it easier to maintain and update complex logic in one place.

7. Differentiate Grant v/s Revoke.

	GRANT	**REVOKE**
1	GRANT command is used to give access privileges to the users or other rights or opportunities for the database.	The REVOKE command does just opposite to the GRANT command. It withdraws user privileges on database objects.
2	It authorizes access preferences to users.	It withdraws access preferences to users.
3	In the GRANT command, you need to define the permissions for each user.	In the REVOKE command, if the access for one user is withdrawn, then all the permissions provided by that particular person to others will also be removed.
4	GRANT is easy to execute.	REVOKE is hard to perform.

8. Explain any four Character Functions.

LOWER():

- This function converts all characters in a string to lowercase.
- SELECT LOWER('Hello World');
- **Output**: 'hello world'

UPPER():

- This function converts all characters in a string to uppercase.
- SELECT UPPER('Hello World');
- **Output**: 'HELLO WORLD'

LENGTH():

- Depending on the SQL dialect, this function returns the length of a string (number of characters).
- SELECT LENGTH('Hello World');
- **Output**: 11 (length of 'Hello World')

SUBSTRING():

- This function extracts a substring from a string. It allows you to specify the starting position and optionally the length of the substring to extract.
- SELECT SUBSTRING('Hello World', 7, 5);
- **Output**: 'World'

9. List different operators. Explain range searching (between) operators withexample.

- Arithmetic operator
- Comparison operator
- Logical operator
- Bitwise Operators
- Compound Operators

Range searching operator

- The BETWEEN operator in SQL is used to filter the result set within a certain range.
- It selects values within a given range.
- The range can be numbers, text, or dates.
- The BETWEEN operator includes both the start and end values.

Syntax:

```
SELECT column_name(s)
FROM table_name
WHERE column_name BETWEEN value1 AND value2;
```

Example

```
SELECT *
FROM Employees
WHERE Salary BETWEEN 30000 AND 50000;
```

10. Explain group-by, having and order by clause

GROUP BY Clause:

- The GROUP BY clause is used to group rows that have the same values in specified columns into summary rows, like "find the total salary by department."
- It is often used with aggregate functions (COUNT, MAX, MIN, SUM, AVG) to perform operations on each group of rows.

Syntax:

```
SELECT column_name(s), aggregate_function(column_name)
FROM table_name
WHERE condition GROUP BY column_name(s);
```

Example:

```
SELECT Department, COUNT(*)
FROM Employees
GROUP BY Department;
```

HAVING Clause:

The HAVING clause is used to filter groups of rows created by the GROUP BY clause. It is similar to the WHERE clause but is used for groups, not individual rows.

Syntax:

```
SELECT column_name(s), aggregate_function(column_name)
FROM table_name
WHERE condition
GROUP BY column_name(s)
HAVING condition;
```

Example:

```
SELECT Department, COUNT(*)
FROM Employees
GROUP BY Department
HAVING COUNT(*) > 10;
```

ORDER BY Clause:

- The ORDER BY clause is used to sort the result set by one or more columns.
- By default, it sorts the records in ascending order.
- You can also specify descending order.

Syntax:

```
SELECT column_name(s)
FROM table_name
WHERE condition
ORDER BY column_name(s) [ASC|DESC];
```

Example:

```
SELECT * FROM Employees
ORDER BY FirstName DESC;
```

11. What is PL/SQL? Explain the Generic Block of PL/SQL.

- PL/SQL (Procedural Language/SQL) is Oracle Corporation's procedural extension for SQL and the Oracle relational database.

- PL/SQL is a powerful language that combines the data manipulation capabilities of SQL with the procedural features of traditional programming languages.
- It enables users to write complex scripts for data manipulation, including loops, conditional statements, and exception handling.

Generic Block of PL/SQL

- A PL/SQL block is the basic unit of a PL/SQL program.
- It is a piece of code that is executed as a single unit and can be nested within other blocks.
- The generic structure of a PL/SQL block consists of the following sections:

1. **Declaration Section**: This is where variables, constants, cursors, and other elements are declared. This section is optional.
2. **Executable Section**: This is the core part of the block where the procedural and SQL statements are written. It is mandatory.
3. **Exception Handling Section**: This section handles exceptions (runtime errors) that occur during the execution of the block. This section is optional.

12. Write a PL/SQL block to find maximum number out of three numbers.

```
DECLARE
   num1 NUMBER := 15;  -- First number
   num2 NUMBER := 30;  -- Second number
   num3 NUMBER := 25;  -- Third number
   max_num NUMBER;     -- Variable to store the maximum number

BEGIN
   IF num1 >= num2 AND num1 >= num3 THEN
      max_num := num1;
   ELSIF num2 >= num1 AND num2 >= num3 THEN
      max_num := num2;
   ELSE
      max_num := num3;
   END IF;

   DBMS_OUTPUT.PUT_LINE('The maximum number is: ' || max_num);

END;
```

13. Write a PL/SQL block to find factorial of given number.

```
DECLARE

   num NUMBER := 5;
   factorial NUMBER := 1;
   i NUMBER;
```

```
BEGIN
   IF num < 0 THEN
      DBMS_OUTPUT.PUT_LINE('Factorial is not defined for negative numbers.');
   ELSE
      FOR i IN 1..num LOOP
         factorial := factorial * i;
      END LOOP;

      DBMS_OUTPUT.PUT_LINE('Factorial of ' || num || ' is: ' || factorial);
   END IF;
END;
```

14. What is Normalization? Explain advantages and disadvantages of it.

Definition: Normalization is a database design technique that organizes tables and fields to minimize redundancy and dependency, ensuring data integrity and efficient management.

Advantages:

1. **Reduces Redundancy**: Minimizes duplicate data, saving storage space.
2. **Improves Data Integrity**: Reduces anomalies like insert, update, and delete anomalies.
3. **Simplifies Maintenance**: Easier to modify and update the database structure.
4. **Facilitates Queries**: Simplifies complex queries and enhances performance.
5. **Supports Scalability**: Handles data growth and changes effectively.

Disadvantages:

1. **Increased Complexity**: More intricate database schema design.
2. **More Joins Required**: Retrieving data may involve multiple table joins.
3. **Performance Overhead**: Join operations can impact query performance.
4. **Design Overhead**: Requires careful planning and understanding of relationships.
5. **Storage Overhead**: Additional indexes and disk space may be needed.

15. Explain First Normal Form with Example.

- A relation **R** is in **first normal form** (1NF) if and only if all **domains** contain **atomic** values only.

Example: Customer table

Cid	Name	Address	Contact_No
		Society, City	
C01	Aarav	Nana Bazar, Anand	9879898798,7877855416
C02	Kahaan	C.G. Road, Ahmedabad	9825098254
C03	Atharv	M.G. Road, Rajkot	9898787898

- Above relation has four attributes **Cid, Name, Address, Contact_no.** Here **Address** is **composite** attribute which is further divided in to sub attributes as **Society** and **City**. Another attribute **Contact_no** is **multi valued attribute** which can store more than one values. So

- above **relation is not in 1NF**.

Problem:

- Suppose we want to find all customers for some particular city then it is difficult to retrieve. Reason is city name is combined with society name and stored whole as address.

Solution:

- Insert separate attribute for each sub attribute of **composite** attribute.
- Determine maximum allowable values for **multi-valued** attribute.
- Insert separate attribute for **multi valued** attribute and insert only one value on one attribute and other in other attribute.
- So, above table can be created as follows:

Cid	Name	Society	City	Contact_No1	Contact_No2
C01	Aarav	Nana Bazar	Anand	9879898798	7877855416
C02	Kahaan	C.G. Road	Ahmedabad	9825098254	
C03	Pari	M.G. Road	Rajkot	9898787898	

16. Explain check and not null constraint.

CHECK Constraint:

- **Definition**: The CHECK constraint is used to enforce domain integrity by defining a condition that each row in a table must satisfy. It restricts the values that can be inserted into a column.

Syntax:

```
column_name datatype CONSTRAINT constraint_name CHECK (condition)
```

Example:

```
CREATE TABLE Employees
(
  EmployeeID INT PRIMARY KEY,
  Age INT CHECK (Age >= 18),
  Salary DECIMAL(10,2) CHECK (Salary > 0)
);
```

NOT NULL Constraint:

- **Definition**: The NOT NULL constraint ensures that a column cannot contain NULL values. It requires every row to have a value for that column.

Syntax:

```
column_name datatype NOT NULL
```

Example:

```
CREATE TABLE Customers
(
   CustomerID INT,
   Name VARCHAR(100) NOT NULL,
   Email VARCHAR(255)
);
```

17. What is a constraint? Explain primary key constraint with example.

- A constraint in SQL is a rule defined on a column or a set of columns in a table that enforces data integrity.
- Constraints ensure that the data in the database conforms to specified rules or conditions, thereby maintaining the accuracy, consistency, and reliability of the data.

Primary Key Constraint:

- **Definition**: A primary key constraint is a type of constraint that uniquely identifies each record (row) in a table.
- It ensures that each value in a column (or a combination of columns) is unique and not null.
- Additionally, a primary key constraint automatically creates a unique index on the column(s) defined as the primary key.

Syntax:

```
CREATE TABLE table_name
(
   column1 datatype PRIMARY KEY,
   column2 datatype,
   ...
);
```

Example:

```
CREATE TABLE Students
(
   StudentID INT PRIMARY KEY,
   Name VARCHAR(100),
   City VARCHAR(50)
);
```

18. Explain SET operators with example.

- SET operators in SQL are used to combine the results of two or more SELECT statements.
- There are three main SET operators: UNION, INTERSECT, and MINUS.
- These operators allow you to perform operations like combining, intersecting, or subtracting data sets from different queries.

1. UNION Operator:

- The UNION operator is used to combine the results of two or more SELECT statements into a single result set.
- It removes duplicate rows from the final result set by default (use UNION ALL to include duplicates).

Syntax:

```
SELECT column1, column2, ...FROM table1
UNION
SELECT column1, column2, ...FROM table2;
```

Example:

```
SELECT Name FROM Students
UNION
SELECT Name FROM Teachers;
```

2. INTERSECT Operator:

- The INTERSECT operator returns only the rows that are common between the result sets of two SELECT statements.
- It effectively performs an intersection operation.

Syntax:

```
SELECT column1, column2, ...FROM table1
INTERSECT
SELECT column1, column2, ...FROM table2;
```

Example:

```
SELECT CourseName FROM StudentCourse
INTERSECT
SELECT CourseName FROM TeacherCourse;
```

3. MINUS Operator:

- In SQL, the MINUS operator is used to retrieve rows from the first query result set that are not present in the second query result set.
- It effectively performs a set difference operation between two SELECT statements.

Syntax:

```
SELECT column1, column2, ...FROM table1
MINUS
SELECT column1, column2, ...FROM table2;
```

Example:

```
SELECT StudentID, Name FROM Student
MINUS
SELECT TeacherID, Name FROM Teacher;
```

19. Explain outer join with example.

- An outer join in SQL is used to retrieve matching rows from two or more tables, along with unmatched rows from one or both tables.
- It allows you to combine rows from tables even if there is no matching row in one of the tables.

Types of Outer Joins:

There are three types of outer joins:

1. **Left Outer Join (LEFT JOIN)**: Retrieves all rows from the left table, and the matched rows from the right table. If there is no match, NULL values are returned for the columns from the right table.
2. **Right Outer Join (RIGHT JOIN)**: Retrieves all rows from the right table, and the matched rows from the left table. If there is no match, NULL values are returned for the columns from the left table.
3. **Full Outer Join (FULL JOIN)**: Retrieves all rows from both tables. If there is no match, NULL values are returned for the columns from the opposite table.

Syntax:

1. **Left Outer Join (LEFT JOIN):**

```
SELECT column1, column2, ...
FROM table1
LEFT JOIN table2 ON table1.column_name = table2.column_name;
```

2. **Right Outer Join (RIGHT JOIN):**

```
SELECT column1, column2, ...
FROM table1
RIGHT JOIN table2 ON table1.column_name = table2.column_name;
```

3. **Full Outer Join (FULL JOIN):**

```
SELECT column1, column2, ...
FROM table1
FULL JOIN table2 ON table1.column_name = table2.column_name;
```

Example:

```
SELECT Students.StudentID, Students.Name, Grades.Grade
FROM Students
LEFT JOIN Grades ON Students.StudentID = Grades.StudentID;
```

20. What is trigger? How to create it Explain with its types.

- Triggers can be classified based on two different criteria:

1) Based on number of times trigger action is executed.
   a) Row Trigger
   b) Statement Trigger

2) Based on timing when trigger action is executed.
   a) Before Trigger
   b) After Trigger

1) Based on number of times trigger action is executed:

Row Trigger	Statement Trigger
Fired **each time** the table is affected by the triggering statement.	Fired only **once.**
Example: If an **UPDATE** statement updates	**Example:** If an **UPDATE** statement updates
multiple rows of a table, a row trigger is fired **once for each row** affected by the	**multiple rows** of a table, statement trigger is fired only **once.**
UPDATE statement If no rows are affected by the triggering statement, a trigger will **not be executed**.	Trigger will be **executed once**, if no rows are affected by the triggering statement.

2) Based on timing when trigger action is executed:

Before Trigger	After Trigger
Trigger is executed before the triggering statement.	Trigger is executed after the triggering statement.
Used to determines whether the triggering statement should be allowed to execute or not.	Used when there is a need for a triggering statement to complete execution before trigger.

- These types are used in combination, provides total four types of triggers:

1) Before Statement: Execute trigger once before triggering statement.
2) Before Row: Execute trigger multiple times before triggering statement.

3) **After Statement:** Execute trigger once after triggering statement.
4) **After Row:** Execute trigger multiple times after triggering statement.

Example:

```
CREATE OR REPLACE TRIGGER balNegative
            BEFORE  INSERT
            ON Account FOR EACH ROW
 BEGIN
            IF          :NEW.Balance < 0      THEN
            dbms_output.put_line ('Balance is negative..');
            END IF ;
END ;
/
```

21. List out pre-defined Exceptions and explain any one with example.

Exception	**Raised When…**
INVALID_NUMBER	TO_NUMBER function failed in converting string to number.
NO_DATA_FOUND	SELECT … INTO statement couldn't find data.
ZERO_DIVIDE	Divide by zero error occurred.
TOO_MANY_ROWS	SELECT … INTO statement found more than one record.
LOGIN_DENIED	Invalid username or password found while logging.
NOT_LOGGED_ON	Statements tried to execute without logging.
INVALID_CURSOR	A cursor is attempted to use which is not open.
PROGRAM_ERROR	PL/SQL found internal problem.
DUP_VAL_ON_INDEX	Duplicate value found in column defined as unique or primary key.
VALUE_ERROR	Error occurred during conversion of data.
OTHERS	Stands for all other exceptions.

Example:

```
DECLARE
    -- declare required variable
        no   Account.Acc_No%TYPE;
        bal   Account.Balance%TYPE;
        branch   Account.B_Name%TYPE;

BEGIN
    --read an account number, balance and branch name for new record
        no := &no;

        bal  := &bal;
        branch := &branch;
    --insert record into Account table

        INSERT INTO Account VALUES (no, bal, branch);
        --commit and display message confirming insertion

        COMMIT;
        dbms_output.put_line('Record inserted successfully.');
EXCEPTION
    --handle named exception

        WHEN  DUP_VAL_ON_INDEX      THEN
          dbms_output.put_line('Duplicate value found for primary
        key.');
END;
/
```