DBMS ASSIGNMENT - 1.

2334A05R7
K. Keerthika.

UNIT - 1 :

1. Explain the structure of DBMS with a neat diagram?

A Database Management System (DBMS) consists of multiple components that interact to provide an efficient way of storing, managing and retrieving data. The structure of a DBMS can be divided into following main components.

1. Database Users:
   - End Users: People who interact with the database using applications.
   - Database Administrators (DBA): Manage & maintain the database.
   - Application programmers: Develop application that interact with the DBMS
   - System Analysts: Design the database structure & Queries.
2. DBMS Interfaces:
   - Provides interaction between users & database
   Ex: Query language (SQL), forms, reports, APIs
3. Query Processor:
   - Converts user Queries into a form that the DBMS can process.
   - Includes
     - DDL Interpreter: Processes Data Definition Language commands
     - DML compiler: Converts Data Manipulation language Queries into executable code.

- Query Optimizer: Optimizes queries for efficient execution.

4. Storage Manager

→ Manages data storage on physical devices

→ Authorization & Integrity Manager: Ensures Security and integrity.

→ Transaction Manager: Handles transactions & concurrency control.

→ File Manager: Manages storage allocation and file structures.

→ Buffer Manager: Manges data in memory for Quick access

5. Database Engine.

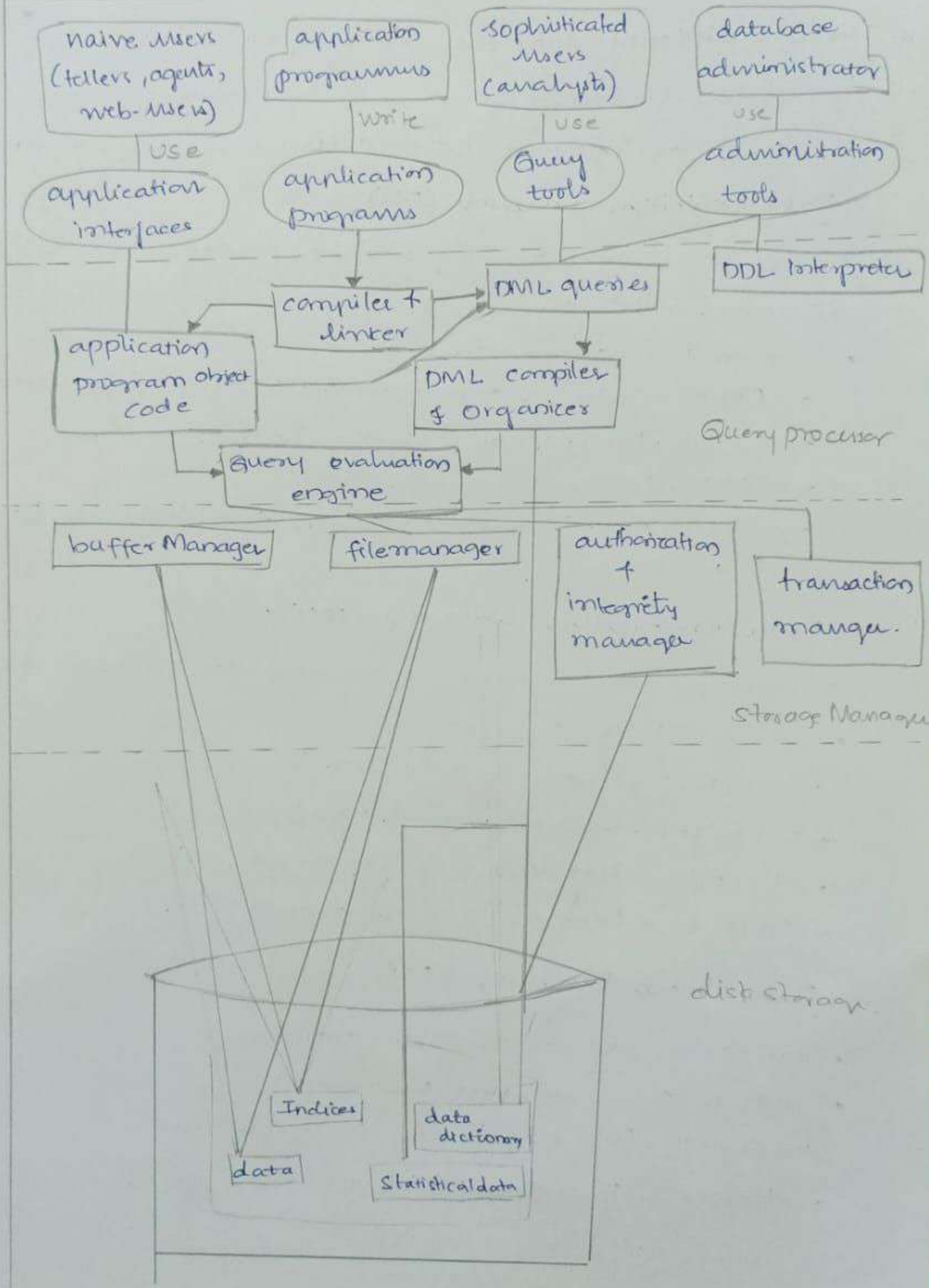
- Executes Queries and updates data in database
- Ensures ACID properties (Atomicity, Consistency, Isolation, Durability)

6. Database (Physical Storage)

→ The actual location where data is stored.

→ Includes tables, indexes, metadata & logs.

DBMS Structure Diagram:


naive users (tellers, agents, web-users)
application programmers
sophisticated users (analysts)
database administrator
use
write
use
use
application interfaces
application programs
query tools
administration tools
DDL interpreter
compiler & linker
DML queries
application program object code
DML compiler & organizer
Query processor
query evaluation engine
buffer Manager
file manager
authorization & integrity manager
transaction manager
Storage Manager
disk storage
Indices
data dictionary
data
Statistical data

2. Discuss different data base languages?

Database languages are used do creati, manage & manipulat. databases. The different types of database languages includes:

1, Data Definition language (DDL)

- Used to define & modify database structures such as tables, Schemas, & indexes.
- Common DDL commands.
  - → CREATE - Creates a new database object (table, view, index)
  - → ALTER - Modifies an existing database structure
  - → DROP - Deletes a database object.
  - → TRUNCATE - Removes all records from a table but keeps the structure.

2, Data Manipulation language (DML):

- Used for data retrieval & manipulation within the database
- Common DML commands:
  - → SELECT - Retrieves data from the database
  - → INSERT - Adds new records to a table
  - → UPDATE - Modifies existing records.
  - → DELETE - Removes records from a table.

3, Data Control Language (DCL)

- Manages user access and permissions
- Common DCL commands:
  - → GRANT - Provides specific privileges to users
  - → Revoke - Removes privileges from users.

4. Transaction Control Language (TCL)

- Manages database transactions to ensure consistency & integrity.
- Common TCL commands:
  - → COMMIT : Saves changes permanently
  - → ROLLBack : Reverts changes if an error occurs
  - → SAVEPOINT : Creates a point in a transaction to which you can later roll back.

UNIT-2

3. Discuss ER-model with an example?

Entity-Relationship (ER) Model

The Entity-Relationship (ER) model is high-level conceptual data model used for designing & representing the structure of a database. It provides a graphical way of describing the data & its relationships in a system.

Components of ER Model

1, Entities:

→ Objects or things in the real world that have attributes

→ Represented as rectangles in the ER diagram.

Ex: Student, Teacher, Course.

2, Attributes:

→ Properties or characteristics of an entity

→ Represented as Ovals in the ER diagram.

Ex; student entity may have attributes like Student-ID, Name, Age.

3. Entity sets:

→ collection of similar entities

→ Ex; A group of students forms a student entity set.

4. Relationships:

→ Associates between entities

→ Represented as diamonds in the ER diagram

→ Ex; A student enrolls in a course.

5. Cardinality:

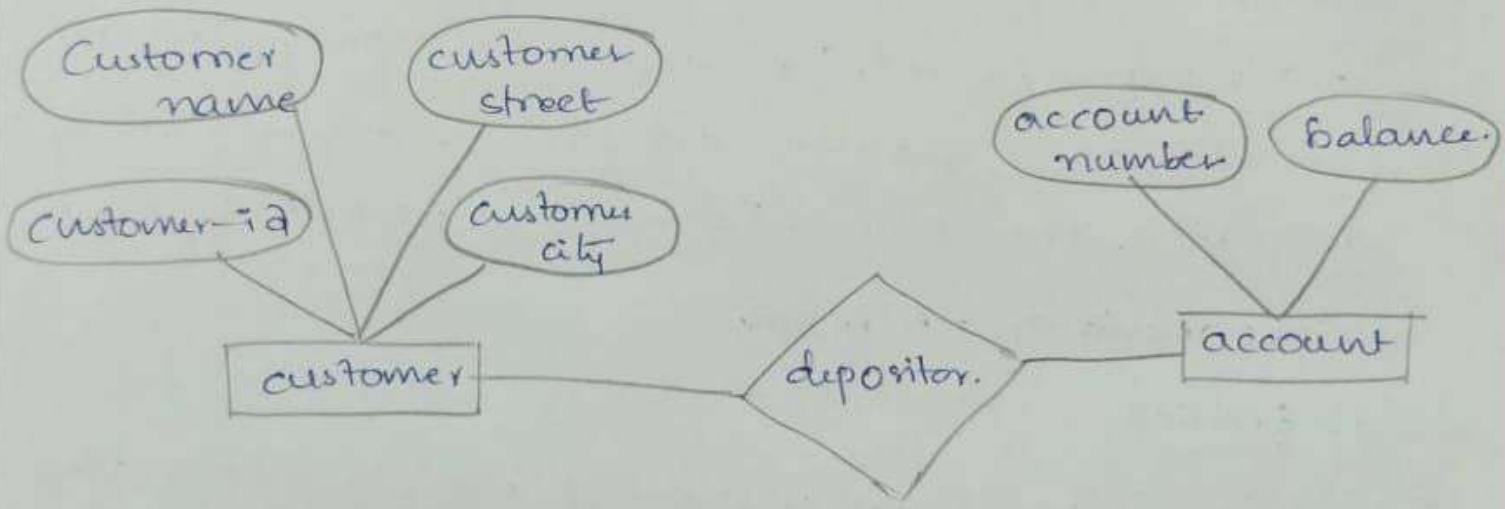
→ Defines how many entities can be related to another entity

→ Types:

1, One to one (1:1)

2, One to many (1:M)

3, Many to Many (M:N)



4. Describe different integrity constraints with examples?

Integrity constraints are rules applied to ensure the accuracy & consistency of data in a database. These constraints prevents invalid data entry & help maintain database integrity.

1. UNIQUE constraint.

- Ensures that all values in a column are distinct.
- A column with a UNIQUE constraint allows only one NULL value.
- Multiple UNIQUE constraints can exist in a table, but only one PRIMARY KEY is allowed.

Ex:
```
CREATE TABLE student(
    sid NUMBER(5),
    name VARCHAR2(10) UNIQUE,   // should have unique names
    login VARCHAR2(20) NOTNULL,
    age NUMBER(2),
    GPA NUMBER(2,1)
);
```

2, NOT NULL constraint

- Ensures that a column cannot have NULL values.
- By default, columns can hold NULL values unless specified otherwise.

Ex:
```
CREATE TABLE student(
    sid NUMBER(5),
    name VARCHAR(10),
    login VARCHAR(20) NOT NULL,   //must have value
);
```

3, PRIMARY KEY constraint

- Uniquely identifies each record in a table.
- A table can have only one PRIMARY KEY, which must be unique & cannot be NULL.

Ex; CREATE TABLE Students (
Sid NUMBER (5) PRIMARY KEY,
name VARCHAR(10),
);

→ The sid column acts as a unique identifier for each student.

4. FOREIGN KEY Constraint.

→ A foreign key in one table refers to a primary key in another table.

→ Ensures referential integrity by allowing only values present in the referenced table.

Ex; CREATE TABLE Enrolled (
Sid VARCHAR (20) PRIMARY KEY,
grade VARCHAR(2),
Sid NUMBER(5),
FOREIGN KEY (sid) REFERNCIES Students (sid)
);

Here sid in the enrolled table must match an existing sid in the students table.

5. CHECK Constraint

→ Ensures that values in a column meet a specified condi

→ Used to restrict values within a defined range

Ex; CREATE TABLE Students(
Sid NUMBER(5) PRIMARY KEY,
name VARCHAR (10),
age NUMBER(2) CHECK (age > 16),
);

the age column must contain values greater than 16.

6. DEFAULT Constraint:

- Assigns a default value to a column if no value is provided

Ex: CREATE TABLE students (

```
    sid NUMBER(5) PRIMARY KEY,
    name VARCHAR2(10) DEFAULT 'Jones',
    login VARCHAR(20),
);
```

→ If no name is provided, 'Jones' will be inserted by default.

7. Enforcing Integrity Constraints.

SQL providing mechanisms to enforce constraints to ensure data integrity.

→ Preventing Duplicate Entries (PRIMARY KEY Violation)

```
INSERT INTO students VALUES (53688, 'Mike', 'mike@ee', (1, 3.4));
```

- This insertion fails if sid 53688 already exists.

→ Handling Foreign Key Deletions (Referential Integrity)

```
CREATE TABLE Enrolled (
    cid VARCHAR(20) PRIMARY KEY,
    grade VARCHAR(2),
    sid NUMBER(5),
    FOREIGN KEY(sid) REFERENCES students(sid)
    ON DELETE CASCADE
);
```

→ If a student is deleted their enrollments will also be deleted (CASCADE).

UNIT-3

5. What is a nested Query? Explain with an Example?

A nested query, also known as a subquery, is a SQL query that is embedded inside another query. The inner query is executed first, & its result is used by the outer query.

Types of Nested Queries:

1, Single-row subquery - Returns only one value.

2, Multi-row Subquery - Returns multiple values.

3, Correlated Subquery - The inner query depends on the outer query.

Single-row Nested Query:

// Find names of students who have higst GPA.

```
SELECT name
FROM Students
WHERE gpa = (SELECT MAX(gPA) FROM Students);
```

→ inner query finds highest GPA & Outer query fetches the students name(s) with that GPA.

Multi-row Nested Query:

// Find students who are enrolled in 'Math 101'

```
SELECT name
FROM Students
WHERE sid IN (SELECT sid FROM Enrolled WHERE
                                    cid = 'Math 101');
```

→ Inner Query Retrieves all students IDs from Enrolled in 'Math 101'. & Outer query fetches names of Students using the retrieved sid values.

Correlated Nested Query:

Find students who have a GPA greater than the average GPA of students in their Department.

```
SELECT name, dept
FROM students S1
WHERE GPA > ( SELECT AVG (gpa) FROM students S2
    WHERE S1.dept = S2.dept);
```

→ Inner Query calculates the average GPA for each department.

→ Outer Query: selects students whose GPA is above their department's average.

---

6. Discuss different types of joins? (Inner, full Outer, left Outer, Right Outer).

TYPES OF JOINS IN SQL

Joins are used to combine data from multiple tables based on a related column.

1. INNER JOIN:

- Returns only matching records from both tables.
- If a record has no match in either table, it is excluded.

Ex:
```
SELECT students.sid, students.name, Enrolled.sid
    FROM students
    INNER JOIN Enrolled ON students.sid = Enrolled.sid;
```

→ Retrieves only students who are enrolled in a course.

2. LEFT OUTER JOIN (LEFT JOIN)

→ Returns all records from the left table + matching records from the right table.

→ If no match is found, NULL values are returned for columns from the right table.

Ex: SELECT students.sid, students.name, Enrolled.sid
FROM students
LEFT JOIN Enrolled ON students.sid = Enrolled.sid;

→ show all students, even if they are not enrolled in any course.

3. RIGHT OUTER JOIN (RIGHT JOIN)

→ Returns all records from the right table + matching records from the left table.

→ If no match is found, NULL values are returned for columns from the left table.

Ex: SELECT Students.sid, Students.name, Enrolled.sid
FROM Students
RIGHT JOIN Enrolled ON students.sid = Enrolled.sid;

→ Displays all enrollments, even if some students are missing in the students table.

4. FULL OUTER JOIN:

→ Returns all records from both (sides) tables.

→ If no match is found, NULL values appear in missing columns.

Ex: SELECT students.sid, students.name, Enrolled.sid from students
FULL OUTER JOIN Enrolled ON students.sid = Enrolled.sid;

→ Displays all students + all enrollments with NULL values where no match exists.