**DATABASE MANAGEMENT SYSTEM**

**(DBMS)**

**DBMS: -**

It is a software application that interact with the user, application, and the database itself. Its facilities the creation, maintenance and use of database.

**Advantage of using DBMS: -**

1. Data Centralization: - All data is stored in a central repository making it easier to manage and access.
2. Data Consistency: - The DBMS enforces rules to maintain data integrity.
3. Improve Data Security: - Access control and encryption mechanism.
4. Data sharing: - Multiple users can access and share data simultaneously.
5. Efficient data retriever: - Optimize quarries and indexing allows for fast efficient data retriever.

**Database: -** A structured collection of data that is organized and stored electronically. It is responsible to manage and retrieve information efficiently.

**Table: -** It is fundamental structure image in rational database that organize data into rows (taffel, /record) and column (attribute / field)

**Row: -** A single entry or instance of data in a table.

**Column: -** A specific piece of data within a table which has same characteristics.

**Primary Key: -** A unique identifier for each row within a table. It ensures each row should be distinct and identifier.

**Foreign Key: -** A field that links to the primary key in another table and responsible to establish a relationship between two tables.

**Index: -** The data structure that enhances data retrieval speed.

**Query: -** A request for data retrieval and manipulation, database usually written in SQL

**SQL (Structure Quarry Language): -** A domain specific language use to manage the database. It allows user to do operation like incretion, deletion, updating and retrieval of data.

**Normalization: -** The process of organizing database to minimize redundancy (Repetition) and dependency by splitting large table into small one related table.

**Transection: -** A set of database operation are treated as a single unit.

**Backup and Restore: -** The process of creating copies of database to prevent data loss incase of system failure or any other disaster.

**Schema: -** Logical grouping that defines the structure relationship and constraints of data stored in database.

**Data dictionary: -** A repository that stores metadata about a data in database. It includes relationship and

**Data warehousing: -** The process of collecting and managing large volume of structure and unstructured form of data from various sources of analyzing and reporting.

**TYPE OF DBMS: -**

1. RDBMS: - (Relational Database management system)
   1. Organized data into structured tabulated format.
   2. Enforces data integrity through Constance.

c. It uses SQL for data manipulation.

E.g., MySQL, MSSQL

1. NO SQL management system: -
   1. Design for unstructured, semi structured or highly dynamic data.
   2. It is more flexible in data modeling.

TYPES: -

1. Document store: - data store as document.

E.g., mongo DB, CouchDB.

1. Key value pair storage: -

e.g., Redis, Amazon Dynamo DB.

1. Graph database: - It focuses relationship between data elements

e.g., neoforge, Amazon Neptune.

1. Object Oriented Database Management System: -

It stores data in form of objects that allowing complex data modelling supports OOPS concept like inheritance and encapsulation

E.g., Ver cent

1. In Memory Database Management System: -

Stores and manages data in the ram rather than on disc for faster data retriever usually it is useful for research work in data base.

E.g., Mem SQL, SAP HANA.

1. Distributed Database Management System: -

Manages data distributed across multiple servers so that it achieves high availability, scalability and fault Torance

E.g., Amazon Aurora, Apache Casandra, Google big table.

**DATABASE DESIGNING**

**Models**

1. Conceptual Model: -
2. Logical Model
3. Physical Model

|  |  |  |  |
| --- | --- | --- | --- |
| Feature | Conceptual Model | Logical Model | Physical Model |
| Entity Name | Tick with solid fill | Tick with solid fill |  |
| Entity Relation | Tick with solid fill | Tick with solid fill |  |
| Attribute |  | Tick with solid fill |  |
| Primary key |  |  | Tick with solid fill |
| Foreign key |  |  | Tick with solid fill |
| Table |  |  | Tick with solid fill |
| Data type |  |  | Tick with solid fill |
| Column |  |  | Tick with solid fill |

**ER DIAGRAM (ENTITY RELATIONSHIP DIAGRAM): -**

It is introduced by Dr. Peter Chen in 1976.

It is a non-technical design method works on conceptual level based on the perception of real world.

It provides standard and logical way of visualizing the database.

It is easy to convert into relation database.

**Entity: -**

An entity is the object of real world that can be distinguishable from other objects based on values of their attribute.

Type of entity: -

1. Tangible entity: - The entity which are physically exist in real world.

E.g., car, bike.

1. Intangible entity: - The entity which exist logically not physically.

E.g., Account

Entity set is the table but entity is the object in entity set.

**Entity set (Schema): -**

It is the group of same types of entities that shares same properties or attributes.

It is defined in rectangular box and its attributes in ellipse form.

**Attribute: -**

Types of attributes: -

1. Simple attributes: - Attributes which are not breakable.

e.g., student id, phone no.

1. Composite Attributes: - Attributes which are breakable.

e.g., Name (First name, middle, last)

1. Single Value Attributes: - There will only single value for each attribute.

E.g., Father name.

1. Multi Value Attributes: - Having multiple value of single attribute. Denoted with double ellipse.

E.g., Phone number.

1. Stored Attributes: - The attributes which are stored in table.

E.g., Date of birth.

1. Derived Attributes: - the attributes which are derived from stored attributes. Denoted with dotted lines

E.g., Age

**Relationship: -** It is an association between two or more entities of same or different entity set. It is represented by rhombus (diamond) shape.

Minimum 1 is required i.e., unary, 2 is binary.

COMPONENTS OF RELATIONSHIP: -

1. Degree: - It defines the number of entities set associated in particular relationship.
2. Cardinality Ratio (Participation Constraint): - It

Employees

Company

One to many

Employer

1 N

One to one

Marry

Wife

Husband

Participation Constraint

Employees

Project

(3,15) (0,2)

Strong Entity Set

Family

Employee

Strong entity set: - Define by single line.

Weak Entity set: - Define by double line.

**Redundancy: -** Repetition of data.

**Data Anomaly: -** To check anomaly or unwanted change due to Insertion, deletion and updation anomaly.

Student

|  |  |  |  |
| --- | --- | --- | --- |
| **Id** | **Name** | **Age** | **Course** |
| 101 | Ramesh | 23 | Java |
| 102 | Suresh | 25 | Python |
| 103 | Dinesh | 24 | CPP |
| 104 | Ganesh | 25 | * (Insertion Anomaly) |

Grade

|  |  |  |
| --- | --- | --- |
| **Sid** | **Subject** | **Grade** |
| 101 | Java | A |
| 102 | Python | A+ |
| 103 | CPP | B+ |

**FUNCTIONAL DEPENDENCY: -** One column functionally depended on another column

|  |  |  |  |
| --- | --- | --- | --- |
| **Student Id** | **Student Name** | **Course Id** | **Course Name** |
| 101 | Ramesh | 1 | Java |
| 102 | Suresh | 2 | Python |

**First normal form: -** A attribute should be atomic.

|  |  |  |
| --- | --- | --- |
| **Student Id** | **Student Name** | **Course Name** |
| 101 | Ramesh | Java, Python |
| 102 | Suresh | CPP, Python, Java |

|  |  |  |
| --- | --- | --- |
| **Student Id** | **Student Name** | **Course Name** |
| 101 | Ramesh | Java |
| 101 | Ramesh | Python |
| 102 | Suresh | CPP |
| 102 | Suresh | Python |
| 102 | Suresh | Java |

**Second Normal Form: -**

* It must follow First normal form.
* All non-key attributes are fully functionally dependent on the entire primary key.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Order Id** | **Product Id** | **Product Name** | **Quantity** | **Customer Id** | **Customer Name** | **Price** |
| 1 | 101 | Laptop | 2 | 25 | Ramesh | 1k |
| 1 | 102 | Mouse | 3 | 25 | Ramesh | 800 |
| 2 | 103 | Keyboard | 1 | 26 | Suresh | 250 |

Sales

Partial Dependency

|  |  |  |
| --- | --- | --- |
| **Ord id** | **Product id** | **Quantity** |
| 1 | 101 | 2 |
| 1 | 102 | 3 |
| 2 | 103 | 1 |

Fully Dependency

|  |  |  |
| --- | --- | --- |
| **Product id** | **Product Name** | **Price** |
| 101 | Laptop | 4500 |
| 102 | Mouse | 200 |
| 103 | Keyboard | 50 |

**Third Normal Form: -**

|  |  |  |  |
| --- | --- | --- | --- |
| **Student Id** | **Course Id** | **Professor** | **Professor Specialization** |
| 1 | 101 | Kaur Mam | CPP |
| 2 | 102 | Sarvat Mam | Java |

|  |  |  |
| --- | --- | --- |
| Student id | Course Id | Professor |
| 1 | 101 | Kaur mam |
| 2 | 102 | Sarvat mam |

|  |  |
| --- | --- |
| Professor | Specialization |
| Kaur mam | CPP |
| Sarvat mam | Java |

**DDL: -** Data Definition Language (Create, Alter, Drop)

**DML: -** Data Manipulation Language (insert, delete, update, select)

1. Insert into table Name (col1, col2, col3….)

Value (val1, val2, val3…...) (101, “Ramesh”, “Manager”);

1. Update table tablename set name = “Suresh” where id=101;
2. Delete from tablename where id=101;

**DATA TYPE: -**

Numeric: - int, big int, tiny int, mid int,

String: - varchar (capacity), char (capacity), varchar2(capacity)

|  |
| --- |
| Devashish |

|  |
| --- |
| Devashish (Extra space) |

Date: - date, time, date-time

Decimal: - decimal (n, m)

n = number before decimal. m = number after decimal point

BLOB (Binary Long Object)

CLOB (Character Long Object)

**CONSTRAINTS: -**

1. **Primary Key: -**

* It should not be null.
* It is unique
* Can be create with single or multiple

1. **Foreign Key: -**

* To integrate two tables.

1. **Check: -**

* To check data is important or not.

1. **Unique: -**

* Only one in the table.
* Allows null value.
* Data should be identical.

1. **Not Null: -**

* Mandatory data.

1. **Default: -**

* Use to set default value.

**AGGREGATE FUNCTION: -**

1. Count (\*)
2. Max (column)
3. Min ()
4. Avg ()
5. Sum ()

**CLAUSES: -** To get specific data from table.

1. **Where: -** comparison

E.g., >, <, =

1. **In: -** specific value

E.g., in (value)

1. **Between: -** from one to another

E.g., 1 – 10

1. **And: -** give two or more condition

E.g., this and that

1. **Or: -** either one of

E.g., 1 or 5.

1. **Order by: -** Use for sorting (by default ascending for descending type desc).
2. **Like: -** select data having particular value.

E.g., D%( Devashish)

1. **Having: -** only work with group by or followed group by.
2. **Group by: -**  Use to show the sum of data in group having same age.
3. **Distinct: -** it will hide similar data from particular id in the table.

**RELATIONAL ALGEBRA: -**

1. Select: - σ
2. Projection: -π Unary Operator
3. Union: - U



1. Intersection: - Ո Binary Operator



1. Difference “minus”: - ▬
2. Cartesian (Gross) Product: -X

|  |  |  |  |
| --- | --- | --- | --- |
| id | name | course | city |
| 1 | A | x | l |
| 2 | B | y | m |
| 3 | C | z | n |
| 4 | D | p | o |
| 5 | E | q | p |

1. **Selection (σ): -**

* The select operation selects tuple that satisfy a given predicate(condition).

1. **Projection (π) : -**

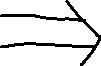
* This operation shows the list of those attributes that we wish to appear in result rest of the attributes are eliminated from the result.

E.g., Π name, city (student) 🡪 It will give name and city column.

Π name, city ( σ id=2 ׀׀ id=3 (student))

1. **Union (U): -**

* the union operation contains all the tuples that are either in R or either in S or in both.
* It eliminates the duplicate tuples.



R S

E.g., Π name (emp) U Π name (Depa)

1. **Difference (▬): -**



Suppose there a two relation R and S the minus operation provides all tuples that are in R but not in S.

Π name (R) - Π name (S) 🡪 A, B,C

1. **Cartesian (Gross) Product (X) : -**

E.g., R X S

The Cartesian product is use to combine each row in one table with each row in another table.

**JOINS:-**

1. **Inner join: -**

When we apply this join on two tables then a common tuple is produced as a relationship.

1. **Left join**
2. **Right join**

**TRANSECTION CONTROL LANGUAGE (TCL): -**

1. **Commit: -**

To save the updates.

1. **Rollback: -**

To undo any updates.

1. **Save point: -**

To save particular step of updating or insertion.

It is use to apply on update, delete, insert commands only.

**DATA CONTROL LANGUAGE (DCL): -**

1. **Grant: -**

To give permission

1. **Revoke: -**

Get back given permission

**SUB QURRIES: -**

showView is used which helps to convert long quarry into short.

**INDEX: -**

It is a data structure which is responsible to enhance performing for searching or retrieval of information on particular column.

**Types: -**

1. **Primary**

Primary

Cluster

1. **Secondary** Ordered
2. **Cluster**

Secondary

Secondary

unordered

Key Non-Key

**ACID PROPERTIES: -**

**A =** Atomicity: - if any error occurs rollback, it and if completed without error it is committed

**B =** Consistency: - It must fulfill all 3 conditions

**C =** Isolation: - logically isolated but not physically.

**D =** Durability: - Until and unless you change it, it will not change

**Atomicity: -**

This property says all quarries of transection should complete their execution or no quarry should able to make the change.

**Isolation: -**

Ideally it is expected from transection that it should execute completely in logical isolation

**Durability: -**

This property ensures the consistent state of transection for long duration till the authorize person make changes to its database.

**Consistency: -**

We can call our transection consistent because of complication of transection our database again achieves consistent state.

**TEMPORARY TABLE: -**

Session ends table automatically gets deleted.

**DATABASE ENGINE: -**

1. InnoDB: -

* transection support
* Row level locking
* Foreign key support
* Crash recovery
* High reliability

1. MyISAM: -

* Table level locking
* No transection supports
* No foreign key supports
* Good for read heavy application

**DATABASE PROGRAMMING: -**

Oracle 🡪 PL/ SQL (Procedural language)

MySQL 🡪 PL/PGSQL (Procedural language/ Postgrad SQL)

**STORED PROCEDURE: -**

A stored procedure is a precompiled, stored set SQL statements and procedural logic that perform a specific task or set of tasks within a DBMS.

It allows to group multiple SQL statement into a single unit so that modularity, Maintainability, efficiency enhances.

**Benefits: -**

* Modularity and code organization
* Reuse and code sharing
* Improved performance
* Enhance security
* It reduces network traffic
* It supports error handling and logging.

**CONDITIONAL STATEMENT: -**

**If Condition Syntax: -**

if( 10 < 20, “10 is smaller”, “20 is smaller”);

**If else Condition Syntax: -**

if condition then

-- statement of condition true;

Else

-- statement of condition false;

End if;

**Switch case Syntax: -**

case

When condition then result

When condition then result 1

else

end;

**Elseif Condition Syntax: -**

if condition then

-- statement of condition true;

Elseif

-- statement of condition false;

else -- statement;

End if;

**LOOPS: -**

**While loop Syntax: -**

While condition do

--------------

-------------

Increment\decrement

End while;

**Loop Syntax: -**

Label: Loop

if condition then

-------;

end if;

leave label; // break

else iterate label; // continue

end loop

**Repeat Syntax: -**

Repeat

statement (x = 0)

until

condition (x < 10)

end repeat;

**STORED FUNCTION: -**

* select char(22,44,88);
* select concat("Devashish"," ","Patil");
* select lower("DEVASHISH");
* select upper("deva");
* select trim("Deva");
* select substr("Devashish",1,4);
* select instr("My name is Deva", "a");
* select length("Devashish Patil");

**NUMERIC FUNCTION: -**

* select mod(10,2);
* select power(10,2);
* select round(193.25,1);
* select truncate(193.222,2);

**DATE FUNCTION: -**

* select curdate();
* select now();
* select month(curdate());
* select year(curdate());
* select day(curdate());
* select month('2023-12-23 3:45:00');
* select sysdate();

**CURSOR: -**

A cursor is responsible to handle a reserve set inside a stored procedure it allows us to iterate a set of rows return by a select quarry and process it individually

MySQL cursor is Read-only, Non-scorable and Asensitive

Read-only: -

We cannot update data int the table through cursor.

Non-Scorable: -

We can only fetch record in the order determine by the select statement. We cannot fetch record in reverse order.

Asensitive: -

It points to the actual data where as insensitive cursor uses a temporary copy of data.

**Steps for Creation and Manipulation of cursor: -**

1. Declare the cursor

**NOTE: -** Declaration of cursor must be after declaration of variable in stored procedure.

1. Open the cursor
2. Fetch record by the cursor
3. Close the cursor

**TRIGGER: -**

A trigger is a special type of stored procedure i.e., associated with particular table and is activated or triggered automatically in responsible to certain event on the table such as insert, update, delete operations.

Events: -

An event is an action that occurs on a table.

Such as

* Insert
* Update
* Delete

**Types of triggers: -**

1. Before trigger: -

It executed before triggering the event. It can be use to modify data before its actual inserted, updated or deleted.

1. After trigger: -

It executed after triggering the event. It can be used to perform action after the data has been inserted, updated or deleted.

Trigger Timing: -

For each row all row level trigger will be fire for each row affected by triggering event.

New and old keyword are used in trigger timing.

**There are special references available in trigger: -**

**New**: - the new value that are been inserted or updated

**Old: -** the old value before an update or delete operation

Real life scenario: -

1. Enforcing business rules
2. Auditing
3. Data validation

**Syntax: -**

Create trigger TriggerName

Before/After eventname (insert/update/delete) on tablename

For each row

Begin

-----

End

**Exception Handling: -**

**E.g.,**

CREATE DEFINER=`root`@`localhost` PROCEDURE `divide`(in a int)

BEGIN

declare c int;

declare exit handler for 1365

begin

select "Can not divide any number by 0" as message;

end;

set c = a/0;

END

**NoSQL DATABASE: - (Not only SQL)**

**Intro: -**

NoSQL Database is a type of DBMS i.e., design to handle and store large volume of unstructured and semi-structured data.

|  |  |
| --- | --- |
| **Semi-structured data** | **Unstructured data** |
| JSON  XML | Text, image |

It uses flexible data models that can adopt to changes in data structure and are capable of scaling horizontally to handle growing amount of data.

Mago DB is a part of NoSQL. It is documentary type of database.

Two types of scaling: -

1. Vertical Scaling: - Scale up. Upgradation of hardware
2. Horizontal Scaling: - Scale down. Adding parallel system or server.

Features of NoSQL: -

1. Dynamic schema.
2. Horizontal scaling.
3. Document based database.

**E.g.,** Mago DB.

1. Key value-based database.

**E.g.,** REDIS, DynamoDB.

1. Column based database.

**E.g.,** Kasandra.

1. Graph based database.
2. It is distributed and highly availability.
3. Flexibility.
4. Performance.

**Structured data Vs Semi-structured data Vs Unstructured data: -**

|  |  |  |
| --- | --- | --- |
| **Structured data** | **Semi-structured data** | **Unstructured data** |
| SQL | JSON  XML | Text, image |

Categories of NoSQL: -

1. Document based database.

This database stored data as semi-structure database such as JSON and XML and can be query using document-oriented query language.

**E.g.,** Mongo DB, Couch DB, CloudEnd.

1. Key value-based database.

This database stored data as a key value pair and optimized for simple and fast read-write operation.

**E.g.,** REDIS, DynamoDB, Memcached, Coherence.

1. Column based database.

This database stores data as column families which are sets of columns that are treated as a single entity. They are optimized for fast and efficient query of large amount of data.

**E.g.,** Kasandra, H Base, Big Table, Accumulo.

1. Graph based database.

This database saves data as node and edges.

**E.g.,** Amazon Neptune, Neoforge.

**INTRODUCTION TO MONGODB: -**

Mongo DB is an open source, non-relational, document-oriented database system it was develop by Mongo DB and is a part of NoSQL data family. IT stores tata in flexible JSON like format known as BSON (Binary JSON) It is well suited for scenario where data doesn’t fit neatly into traditional relational base table and benefits from horizontal scaling to handle large data set and high traffic load.

Features: -

1. Rich query language.
2. Secure.
3. High Availability.
4. Scalable
5. Community and Vendor support

|  |  |
| --- | --- |
| **MySQL** | **NoSQL** |
| 1. SQL is rational database | 1. Non-relational database |
| 1. Fixed schema design and structure | 1. Dynamic schema design and structure |
| 1. Handle complex query | 1. Handle large volume of data |
| 1. Vertical scalable | 1. Horizontal scalable |
| 1. It follows ACID property | 1. It follows CAP property   **C: -** Consistency  **A: -**Availability  **P: -** Patrion tolerance |