Database : A database is a collection of related data. By data, we mean known facts that can be recorded and that have implicit meaning. A database has the following implicit properties

1. A database is a logically coherent collection of data with some inherent meaning.
2. A database is designed, built, and populated with data for a specific purpose. It has an intended group of users and some preconceived application in which these users are interested.
3. A database represents some aspect of the real world.

A database can be of any size and of varying complexity. A database may be generated and maintained manually or by machine.

A database management system (DBMS) is a collection of programs that enables users to create and maintain a database. A DBMS is hence a general-purpose software system that facilitates the processes of defining, constructing and manipulating databases for various applications. The database and software are together called a database system.

Use of DBMS:-

1. Controlling Redundancy: - Redundancy means storing of same information in multiple time. It leads to several problems like performing single logical update many times, wastage of memory space and inconsistency. To avoid all such problem we should have a database design that stores each logical data item in only one place in the database.

2. Sharing of data :- The DBMS must include concurrency control software to ensure that several users trying to update the same data do so in a controlled manner so that the result of the updates is correct.

3. Restricting Unauthorized Access: - When multiple users share a database, it is likely that some users are not authorized to access all information in the database. In addition some user may be permitted only to retrieve data, whereas others are allowed to both retrieve and update. Hence the type of access operation can also be controlled. A DBMS should provide a security and authorization subsystem, which is used by the DBA to create account and specify account restrictions.

5. Providing Multiple Interfaces:- Because many types of users, with varying technical knowledge, use a database, a DBMS should provide a variety of user interfaces. The types of interfaces include query languages for casual users, programming language interfaces for application programmers forms for parametric users, menu-driven interfaces for native users and natural language interfaces.

6. Representing Complex Relationships among Data: - A database may include a variety of data that are interrelated in many ways. A DBMS must have the capability to represent a variety of complex relationships among the data as well as to retrieve and update related data in easy and efficient manners.

7. Enforcing Integrity Constraints: - Most database applications will have certain integrity constraints like data type of data item, uniqueness constraint of data item, relationship between records of different file etc. These integrity constraints must be specified during the database design.

8. Provide Backup and Recovery: - A DBMS must provide facilities for recovering from hardware or software failures. The backup and recovery subsystem of the DBMS is responsible for recovery.

---

3. Degree of a Relationship Type: The degree of a relationship type is the number of participating entity types. A relationship type of degree two is called binary and of degree three is called ternary.

4. Constraints on Relationship Types: - There are two types of constraints on relationship types.

a) Cardinality Ratio: - The cardinality ratio constraints specify the number of relationship instances that an entity can participate in. Different ratios are 1:1 1:N and M: N

b) Participation constraint: - Participation constraint specifies whether the existence of an entity on its being related to another entity via the relationship type. There are two types of participation constraints, total and partial.

Relational Algebra: - Relational algebra is a collection of operations on relation. Each operation takes one or more relations as its operand and produces another relation as its result. These operations are used to select tuples from individual relations and to combine related tuples from several relations for the purpose of specifying a query, a retrieval request, on the database. Different types of operation in relational algebra are

1. SELECT operation: - The algebraic SELECT operator yields a horizontal subset of a fiven relation that is subset of tuples for which a specified predicates is specified. The predicate is expressed as a Boolean combination of terms, each term being a simple comparison that can be established as tru or false for a given tuple by inspecting that tuple in isolation.

The SELECT operator is unary that is it is applied on a single relation. Hence SELECT cannot be used to select tuples from more than one relation. The number of tuples in the resulting relation is always less that or equal to the number of tuples in the original relation.

2. PROJECT operation: - The PROJECT operator yields a vertical subset of a given relation, that is that subset obtained by selecting specified attribute in a specified left to right order and then eliminating

duplicate tuples within the attributes selected. The number of tuples in a relation resulting from a PROJECT operation will be les than or equal to the number of tuples in the original relation.

3. JOIN:- The JOIN operation is used to combine related tuples from two relations into single tuples. The result of the JOIN operation is a relation Q with m+n attributes that is m attributes from first relation and n attributes from another relation. The tuples in the relation resulting from a JOIN operation are those, which will satisfy the condition given in the join operation. Different types of JOIN operation are :

a) Theta join: - A join condition is of the form

<Condition> AND <condition> AND……. AND <condition> where each condition is of the form $A_i \theta B_j$. A_i is an attribute of R. B_j is an attribute of S, A_i and B_j has the same domain and θ is one of the comparison operators { =, <, <=, >, >=, #}. A join operation with such a general join condition is called a theta join.

b) Equi Join: - The most common join operation involves join condition with equality comparisons only. Such a join where only comparison operator used is equal sign is called an Equijoin. The relation produced by Equijoin, contain one or more pairs of attributes that have identical values in every tuple because the equality join condition is specified on these two attributes.

c) Natural join: - It is basically an equijoin but it eliminates the duplicate attribute in the result. It is denoted by *.

d) Outer join: - Generally Join operation select all the tuples from the two relation which will satisfy the join condition. But outer join is used to keep all tuples in R or S or both in the result, whether or not they have matching tuples in the other relation. Different types of outer joins are

i) Left outer join: - The left outer join operation keeps every tuples in the first or left relation R in R S. If no matching tuple is found in S, then the attributes of S in the result are filled with null values.

ii) Right outer join:- Right outer join( ) keeps every tuple in the second or right relations S in the result of R S.

iii) Full outer join:- The full outer join( ) keeps all tuples in both the left and right relations when no matching tuples are found, assigning them with null values as needed.

e) SET operation:- Set operations are the standard mathematical operation on sets. They apply to the relational model because a relation is defined to be a set of tuples and they are used whenever we process the tuples in two relation as sets. Set operations are binary that is they are applied to two sets. The two relation on which these operations are applied, must be union compatib le. Union compatible means the two relation must have the same type of tuples. There are three set operations :

i) UNION: - The union of two relations A and B, A UNION B is the set of all tuples t belonging to either A or B (or both). Duplicate tuples are eliminated

ii) INTERSECTION: - The intersection of two relations A and B, A INTERSECTION B, is the set of all tuples t belonging to both A and B.

iii) DIFFERENCE: - The difference between two relations A and B, A MINUS B, is the set of all tuples t belonging to A and not to B

iv) Cartesian Product: - Cartesian product is a binary set operation but the relations on which it is applied do not have to be union compatible. This operation is used to combine tuples from two relations so that related tuples can be identified. In general, the result of R(A1,A2,…….An) X (B1, B2…….Bm) is the relation Q with (n+m) attributes Q(A1, A2…….An, B1, B2…….Bm) in that order. The resulting relation Q has one tuple for each combination of tuples one from R and one from S. The Cartesian product creates tuples with the combined attributes of two relations. We can select only related tuples from the two relations by specifying an appropriate selection condition.

Structured Query Language (SQL): - SQL is a comprehensive database language; it has statements for data definition, query, and update. Hence, it is both a DDL and DML. It is a very powerful language in the sense that most of the operations in RDBMS can be performed using SQL. An important feature of SQL is that it is a non-procedural language. In a non-procedural language, we have to describe what to do rather than how to do.

Advantage of SQL: -

Data types in SQL: -

Data Type	Specification	Description
Char	CHAR (size)	Char data, size if specified in number
Varchar	VARCHAR(size)	Same as above
Date	DATE	Used for specifying dates
Number	NUMBER	Used to specify numeric data, NUMBER(size) or NUMBER(size, dec) number with digits equal to specified size.
Integer	INTEGER	Same as number but without decimal digit
Raw	RAW(size)	Raw binary data
Long raw	LONG RAW	Raw large binary data
Blob, Clob		Large binary and character data

Bfile		External files

Different SQL commands are: -

1. CREATE TABLE command: - This command is used to specify a new relation by giving it a name and specifying each of its attributes. Each attribute is given a name, a data type to specify its domain of values, and some constraints on the attribute. The syntax is

```
CREATE TABLE tablename (attributename   datatype   constraint,
                        attributename   datatype   constraint,
                        - - - - - - - - - - - - ) ;
```

for example we can create a student table as follows

```
create table student (  Roll     number(3)   primary key,
                        name     varchar(30)  not null,
                        class    varchar (10) );
```

2. DROP TABLE command: - We can delete the table or relation and its definition using the DROP TABLE comand

```
DROP TABLE tablename ;
```

We can drop the student table as DROP TABLE student ;

3.ALTER TABLE command: - To add attributes to an existing relation, we can use ALTER TABLE command.

```
ALTER TABLE tablename ADD attributename datatype ;
```

We can add an extra attribute RESULT to the student table as follows:

```
ALTER TABLE student ADD result varchar(10) ;
```

4. INSERT command: - INSERT command is used to add a single record or tuple to a relation. If we want to insert record consisting of values of all attributes then no need to specify the attribute name, otherwise specify the name of those attributes for which we want to insert values.

```
INSERT INTO tablename VALUES( value1, value2, ……. Value n)
        Or
```

INSERT INTO tablename(attribute1, attribute2,…….. attribute n) VALUES(value1, value2,……value n)

For example add a new record in student table

INSERT INTO student VALUES(1, “AAAA”, “B.Sc. 1st Year”, “1st class”);

5. DELETE command: - The DELETE command removes tuples from a relation. It includes a where-clause, to select the tuples to be deleted. Tuples are deleted from only one table at a time. A missing where-clause specifies that all tuples in the relation are to be deleted.

```
DELETE FROM tablename
        Or
DELETE FROM tablename WHERE expression;
```

Example DELETE FROM student

Or DELETE FROM student WHERE roll = 2;

UPDATE command: - The UPDATE command is used to modify attribute values of one or more selected tuples. Where-clause can be used to specify the tuples to be modified from a single relation. The syntax is

```
UPDATE tablename
SET attributename= value
WHERE expression ;
```

UPDATE student SET result =”1st class” WHERE roll = 3 ;

Build-in-Functions: -

a) COUNT:- The COUNT function returns the number of tuples or values specified in a query.
b) SUM:- The SUM function return summation of the specified attribute’s value
c) MAX:- The MAX function return maximum value of the specified attribute

d) MIN:- The MIN function return minimum value of the specified attribute
e) AVG:- The AVG function return average value of the specified attribute

Integrity Constraints: - Intetrity constraints are specified on a database schema and are expected to hold on every database instance of that schema. There are three types of integrity constraints :

i) Key Constraint: - Key constraints specify the candidate keys of each relation schema. Candidate key values must be unique for every tuple in any relation instance of that relation schema.
ii) Entity Integrity Constraint: - The entity integrity constraint states that no primary key value can be null
iii) Referential Integrity Constraint: - It is a constraint that is specified between two relations and is used to maintain the consistency among tuples of the two relations. Informally, the referential integrity constraint states that a tuple in one relation that refers to another relation must refer to an existing tuple in that relation.

Functional Dependency(FD): - A functional dependency is a constraint between two sets of attributes from the database. A functional dependency, denoted by X $\longrightarrow$ Y, between two sets of attributes X and Y that are subsets of R specifies a constraint on the possible tuples that can form a relation instance r of R. The constraint states that for any two tuples t1 and t2 in such that t1[X] = t2[X], we must also have t1[Y] = t2[Y]. This means that the values of the Y component of a tuple in r depend on, or are determined by the values of the X component or, alternatively, the values of the X component of a tuple uniquely determine the values of the Y component. We also say that there is a functional dependency from X to Y or that Y is functionally dependent on X.

Entity Relationship Model (ER Model): - At the present time, the ER model is used mainly during the process of database design.

1. Entities: - The basic object that the ER model represents is an entity, which is a “thing” in the real world with an independent existence. An entity may be an object with a physical existence – a particular person, car etc, or it may be an object with a conceptual existence like a company, a job or a university course etc.

2. Weak Entity: - Some entity may not have any key attributes of their own. This implies that we may not be able to distinguish between some entities because the combinations of values of their attributes can be identical. Such entity is called weak entity. Weak entity is identified by being related to specific entities from another entity type in combination with some of their attribute values. Weak entity always has a total participation constraint with respect to its identifying relationship. Weak Entity type has a partial key, which is the set of attributes that can uniquely identify weak entities related to the same owner entity.
3. Attribute: - Each entity has particular properties called attributes that describe it. For example a student entity may be described by RollNo, Name, Class, Address etc. Different types of attributes are

   a) Composite Attribute: - An attribute, which is composed of more basic attributes, is called composed attribute. For example Address attribute of a student.

   b) Atomic Attribute: - The attributes that are not divisible are called simple or atomic attributes. For example RollNo attribute of a student.

   c) Single-valued Attribute: - Most attributes have a single value for a particular entity, such attributes are called single-valued attribute. For example Date_of_Birth attribute of a person.

   d) Multivalued Attribute: - The attribute, which has a set of values for the same entity is called multivalued attribute. A multivalued attribute may have lower and upper bounds on the number of values for an individual entity. For example Subject attribute of a student.

   e) Derived Attribute: - In some cases two or more attribute values are related. The value of one attribute has to be calculated from the value of another attribute. Such type of attribute is called derived attribute. For example Age attribute of a person can be calculated from current date and his date of birth.

f) Key Attribute: - An entity type usually has an attribute whose values are distinct for each individual entity. Such an attribute is called a key attribute and its values can be used to identify each entity uniquely. Sometimes several attributes together can form a key, meaning that the combination of the attribute values must be distinct for each individual entity. Some entity types have more than one key attribute. In this case, each of the keys is called a candidate key. When a relation schema has several candidate keys, the choice of one to become primary key is arbitrary, however, it is usually better to choose a primary key with a single attribute or a small number of attributes.

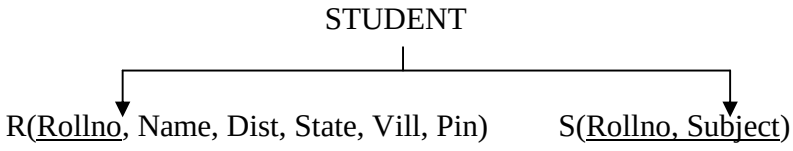
Normal forms: - Normalization of data can be looked on as a process during which unsatisfactory relation schemas are decomposed b breaking up their attributes into smaller relation schemas that posses desirable properties.

i. First Normal Form (1NF): - The first normal form is defined to disallow multivalued attributes, composite attributes and their combinations. The only attribute values permitted by 1NF are single atomic values. For example

STUDENT (Rollno, Name, Subject, Address).

In the STUDENT relation Subject attribute is multivalued attribute and Address is composite attribute. So this relation is not under 1NF. We decompose it according to the 1NF.

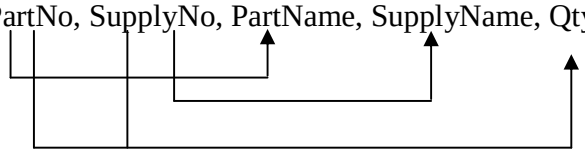
R (Rollno, Name, Dist, State, Vill, Pin) S (Rollno, Subject)



ii. Second Normal Form(2NF): - The second normal form is based on the concept of a full functional dependency. A functional dependency X ⟶ Y is a full functional dependency if removal of any attribute A from X means that the dependency does not hold any more, that is, for any attribute A ∈ X. A functional dependency X⟶ Y is a partial dependency if there is some attribute A ∈ X that can be removed from X and the dependency will still hold.

A relation schema R is in 2NF if every nonprime attributes A in R is fully functionally dependent on the primary key of R. For example:

PARTS (PartNo, SupplyNo, PartName, SupplyName, Qty).



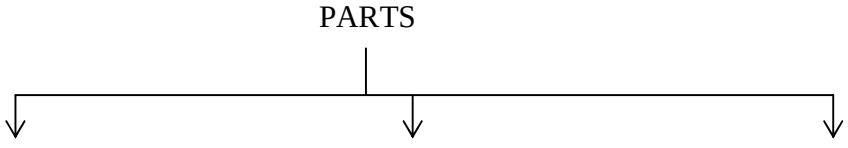
F1: PartNo ⟶ PartName

F2: SupplyNo ⟶ SupplyName

F3: (PartNo, SupplyNo) ⟶ Qty

In this PARTS table F1, F2, F3 functional dependency hold. Now if we remove PartNo attribute from PARTS then the functional dependencies F1 and F3 will not hold. Again if we remove SupplyNo attribute from PARTS then the functional dependencies F2 and F3 will not hold. That means if we remove any attribute either PartNo or SupplyNo the functional dependency F3 certainly will not hold. So F3 is the full functional dependency and F1 & F2 are partial functional dependency. If the table has partial functional dependency then that table is not under 2NF. So we decompose it into three tables as follows

R (PartNo, PartName) S (SupplyNo, SupplyName) T(PartNo, SupplyNo, Qty)

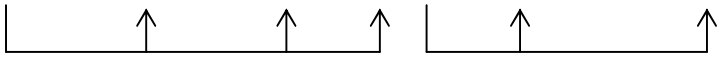


R(PartNo PartName) S(SupplyNo, SupplyName) T(PartNo, SupplyNo, Qty)

iii) Third Normal Form (3NF): - The third normal form is based on the concept of a transitive dependency. A functional dependency X → Y in a relation schema R is a transitive dependency if there is a set of attributes Z that is not a subset of any key of R, and both X → Z and Z → Y hold.

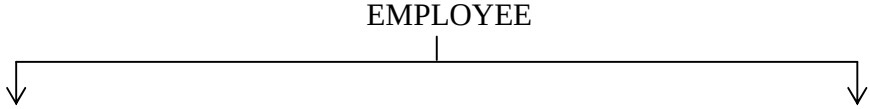
A relation is in 3NF if it is in 2NF and no nonprime attribute of R is transitively dependent on the primary key. For example

EMPLOYEE (EmpCode, EmpName, Salary DeptCode DeptName DeptLocation)



In this relation EmpCode can determine the value of EmpName, Salary and DeptCode. So these three attributes are depending on EmpCode. On the other hand DeptCode can determine the value of the attributes DeptName and DeptLocation. So both the attributes are depending on DeptCode but DeptCode is not a primary key. That means some nonprime attribute (DeptName, DeptLocation) are depending on another nonprime attribute (DeptCode) and that non prime attribute is depending on primary key (EmpCode). This relation is transitive dependency, so it is not under 3NF. We decompose it as follows:

R (EmpCode EmpName Salary DeptCode) S (DeptCode DeptName DeptLocation)



R (EmpCode Empname Salary DeptCode) S (DeptCode DeptName DeptLocation)

iv. Boyce Code Normal Form (BCNF): - A relation is in BCNF if whenever a functional dependency X → A holds in R the X is super key of R.