Chapter 1 Basic concepts of DBMS

Database :- It is an integrated collection of related file, along with details of the interpretation of the data contained therein.

→ An organization must have accurate and reliable data for effective decision making, for that reason we need to store them in a database.

DBMS :- A database management system is a s/w system that allows access to data contained in a database.

→ The objective of the DBMS is to provide a convinient and effective method of defining, storing and retrieving the information contained in the database.

→ The database system allows these users to access and manipulate the data contained in the database in a convinient and effective manner.

→ The DBMS exerts centralized control of the database, prevents unauthorized users from accessing the data and ensures the privacy of the data.

Data ABSTRACTION Data abstraction means showing only a portion of the database and not going to the background details.

→ Data abstraction is having different defn for different levels, it varies for users to users.

→ Data abstraction is concerned with hiding the particular portion of the database to some users.

→ Based on data abstraction the DBMS is divided into three levels. It is also known as three level architecture of DBMS. The three levels are external level, the conceptual level and the internal level.

→ The view at each of these levels is described by a scheme. A scheme is an outline or a plan that describes the records and relationships existing in the view. Scheme means a systematic plan for attaining some goal, is used interchangeably in the database with the word schema.

External or user view :- The external or user view is at the highest level of database abstraction where only those portions of the database of concern to a user or application program are included. Any no. of user views may exist for a given global or conceptual view.

→ It is described by external schema. The external schema contains the method of deriving the objects in the external view from the objects in the conceptual view.

Conceptual or global view :- At this level of database abstraction all the database entities and the relationships among them are included.

→ One conceptual view represents the entire database.

This conceptual view is defined by the conceptual schema.
→ It describes all the records and relationships included in the conceptual view and therefore in the database.
→ There is only one conceptual schema per database. This schema also contains the method of deriving the objects in the conceptual view from the objects in the internal view.

External view :- This view provides the lowest level of abstraction closest to the physical storage method used.
→ It indicates how the data will be stored and describes the data structures and access methods to be used by the database.

Logical view of an entity set employee

→ employee name
→ employee address

conceptual view

employee name : string
employee social security number : key
employee address : string
employee child : string

external view

name : string length 25
soc_sec_no : 9 decimal
department : string length 6

DATABASE USERS The users of a database system can be classified in the following groups depending upon the mode of interaction with the DBMS.

Naive users :- Users who need not to be aware of the presence of the database system or any other system supporting their usage are considered naive users.
→ A user of ATM falls in this category.

Online users There are users who may communicate with the database directly via an online terminal or indirectly via a user interface and application program.
→ These users are aware of the presence of the database system and may have acquired a certain amount interaction with the database through the application program.

Application programmers :- professional programmers who are responsible for developing application programs or user interfaces utilized by the naive and online users falls into this category. The application program could be written in a general purpose programming language such as C, COBOL, FORTRAN and include the commands required to manipulate the database.

Database Administrator :— centralized control of the database is exerted by a person or group of persons under the supervision of a high-level administrator. This person or group is referred to as the database administrator or DBA.

→ The DBA are the users who are most familiar with the database and are responsible for creating, modifying and maintaining its three levels.

→ The DBA is the custodian of the data and controls the database structure.

→ The DBA guides the three levels of the database and in consultation with the overall user community, sets up the def^n of the global view and the DBA also specifies the external view of the various users and applications and is responsible for the def^n of internal level.

→ The DBA is also responsible for defining procedures to recover the database from failures due to human, natural or hardware causes with minimal loss of data.

Data definition language :— DBMS provide a facility known as DDL, which can be used to define the conceptual scheme and also give some details about how to implement this scheme.

→ This def^n includes all the entity sets and their associated attributes as well as relationship among the entity sets. The def^n also maintained any constraints regarding the attributes.

→ These def^n which can be described as metadata about the data in the database are expressed in the DDL of the DBMS and maintained in a compiled form known as data dictionary, directory or system catalog.

→ The DBMS maintains the information on the file structure, the method used to efficiently access the relevant data. The application program could use a subset of the conceptual data def^n language or a separate language.

Data dictionary :— The data dictionary contains the meta data. The meta data contains information regarding the structure and usage of data contained in the database.

→ The term system catalog also describes this metadata.

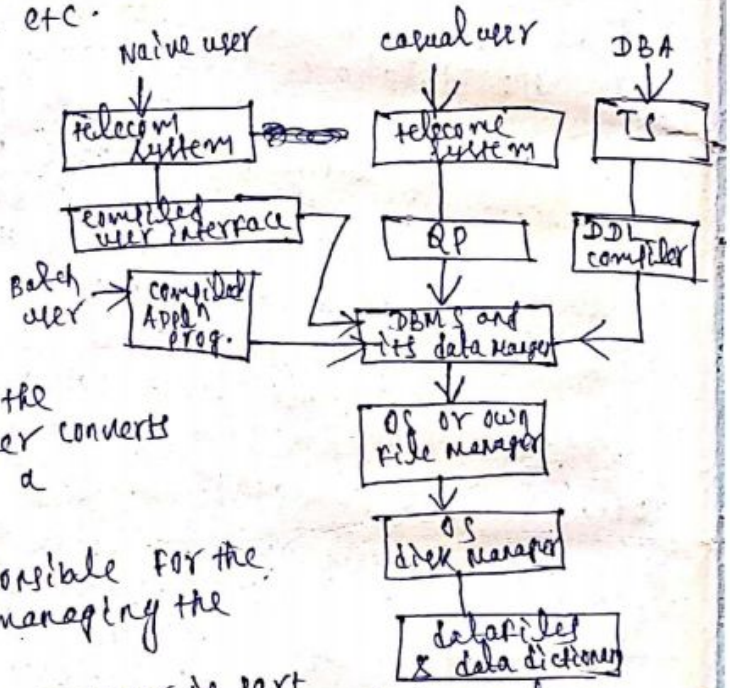
→ The data dictionary, which is a database itself, documents the data.

→ Each database user can consult the data dictionary to learn what each piece of data and the various synonyms of the data fields mean.

→ In an integrated system (i.e., in a system where the data dictionary is part of the DBMS) the data dictionary stores information concerning the external, conceptual and internal levels of the database.

→ It contains the source of each data field value, the frequency of its use etc.

Structure of DBMS



DDL compiler :- The DDL compiler converts the data defn statements into a set of tables. These tables contain meta data.

Data manager :- It is the central s/w component of the DBMS. The data manager converts operations in user queries to a physical file system.

File manager :- It is responsible for the structure of the files and managing the file space.

Disk manager :- The disk manager is part of the OS of the host computer and all physical i/p and o/p operations are performed by it.

Query processor :- The database user retrieves data by formulating a query in the data manipulation language. The QP is used to interpret the user's query and convert it into an efficient series of operations in a form capable of being sent to the data manager for execution.

TS :- Online users of a computer system, communicate with the computer by sending or receiving messages over communication lines. These messages are routed via an independent s/w system called telecommunication system.

Data files :- Data files contains the data portion of the database.

Data dictionary :-

Advantages of DBMS

- centralized control
- data independency
- control redundancy
- enhanced data quality
- security enforcement possible

Disadvantages

- problems associated with centralization
- cost of h/w & s/w
- complexity of backup and recovery

chapter-2 Data Models

Data independence :- There are 2 types of data independence are there that is logical data independence and physical data independence

Logical data independence :- Logical data independence indicates that the conceptual schema can be changed without affecting the existing external schemas. The change would be absorbed by the mapping between the external and conceptual levels.

→ Logical data independence also insulates application programs from operations such as combining two records into one or splitting an existing record into two or more records.

→ Logical data independence is achieved by providing the external level or user view of the database.

→ The application programs or users see the database as described by their respective external views.

→ The DBMS provides a mapping from this view to the conceptual view.

Physical data independence :- physical data independence indicates that the physical storage structures or devices used for storing the data could be changed without necessitating a change in the conceptual view or any of the external views.

→ The change would be absorbed by the mapping between the conceptual and internal levels.

→ physical data independence is achieved by the presence of the internal level of the database and the mapping or transformation from the conceptual level of the database to the internal level.

→ The physical data independence criterion requires that the conceptual level does not specify storage structures or the access methods used to retrieve the data from the physical storage medium.

→ making the conceptual schema physically data independent means that the external schema, which is defined on the conceptual schema, is in turn physically data independent.

Entity-Relationship Model :-

Entity :- An entity is a thing or object in the real world that is distinguishable from all other objects.

→ Entity can have concrete existence or constitute ideas or concepts. The e.g. of entity are building, room, chair, transaction, course, machine, employee.

Entity set or entity type :- An entity set is a set of entities of the same type that share the same properties or attributes.

→ The set of all persons who are customers at a given bank, can be defined as the entity set customer.

→ Similarly the entity set loan might represent the set of all loans awarded by a particular bank.

Attributes :- Attributes are the descriptive properties possessed by each member of an entity set. The designation of an attribute for an entity set expresses that the database stores similar information concerning each entity in the entity set however each entity may have its own value for each attribute.

customer

customer id	customer name	customer city
x001	YZ	BBSR
x002	NM	BGH
x003	MM	BBSR

→ Each entity has a value for each of its attributes.
→ For each attribute, there is a set of permitted values called the domain, or value set.
→ A database hence includes a collection of entity sets, each of which contains any no. of entities of the same type. For e.g. we can have a bank database that consist of 2 entity sets customer and loan.
* The entity-relationship data models considers the real world as consisting of basic objects, called entities and the relationship among these objects.

Attribute types :-

Simple attribute :- The attribute which can't be divided into subparts is called simple attribute.

composite attribute :- It could be divided into subparts. For e.g. an attribute name could be structured as a composite attribute consisting of first-name, middle-initial and last-name.
→ Address attribute could be treated as composite attribute. Its componet attribute street can be further divided into street-number, street-name etc and can also be treated as a composite attribute.

Single-valued attribute :- The attribute that have a single value for a particular entity is called so. For e.g. the loan-number attribute for a specific loan entity refers to only one loan number.

Multivalued Attribute :- There may be instances where an attribute has a set of values for a specific entity. Consider an employee entity set with the attribute phone-number. An employee may have zero, one or several phone numbers and different employee may have different numbers of phones. This type of attribute is called multivalued-

Derived attributes :- The value for this type of attribute can be derived from the values of other related attributes or entities.

For e.g. Suppose an entity set customer has 2 attributes age, and date-of-birth. Here age can be treated as the derived attribute because we can calculate the age by knowing the value of date-of-birth attribute and the current date. Here the date-of-birth is known as base attribute or a stored attribute.

* An attribute takes null value when an entity does not have a value for it. The null value means the value may indicate "not applicable" — that is that the value does not exist for the entity.

Relationships :- The relationship set is used in data modeling to represent an association between entity sets.

→ Here Employee & department are two entity sets and they are associated with each other through the relationship set works-for.

→ There could be a no. of entity sets involved in a relationship set and the same entity set could be involved in a number of different relationship sets.

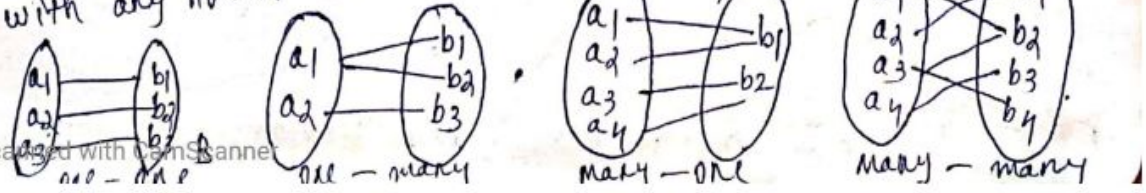
Mapping constraints :- Mapping constraints express the number of entities to which another entity can be associated via a relationship set.

One to one :- An entity in A is associated with at most one entity in B, and an entity in B is associated with at most one entity in A.

One to many An entity in A is associated with any number of entities in B. An entity in B, however can be associated with at most one entity in A.

Many to one :- An entity in A is associated with at most one entity in B. An entity in B, however, can be associated with any number of entities in A.

Many to many :- An entity in A is associated with any number of entities in B, and an entity in B is associated with any no. of entities in A.



Keys The keys are the attributes values those could uniquely identify an entity.

→ Keys also help uniquely identify relationships.

Superkey A superkey is a set of one or more attributes that, taken collectively, allow us to identify uniquely an entity in the entity set.

→ For e.g. the customer-id attribute of the entity set customer is sufficient to distinguish one customer entity from another.

→ We may have also the customer-name & customer-id collectively as superkey because customer name only can't be able to uniquely identify the customer entity.

candidate keys :- candidate keys are the subset of superkeys. Suppose cus-name & cus-street collectively form the super key then they may be treated as candidate keys. But cus-id & cus-name can't be treated as candidate key because cus-id alone is candidate key.

Primary key :- primary key is a candidate key that is chosen by the database designer as the principal means of identifying entities within an entity set.

→ The primary key should be chosen such that its attributes are never or very ~~rather changed~~ rarely changed.

Entity-relationship diagram / E-R diagram :- An E-R diagram can express the overall logical structure of a database graphically. E-R diagrams are simple and clear. The components of this diagram are :-

Rectangles :- which represents the entity sets

Ellipses :- which represent attributes

Diamonds :- which represents relationship sets.

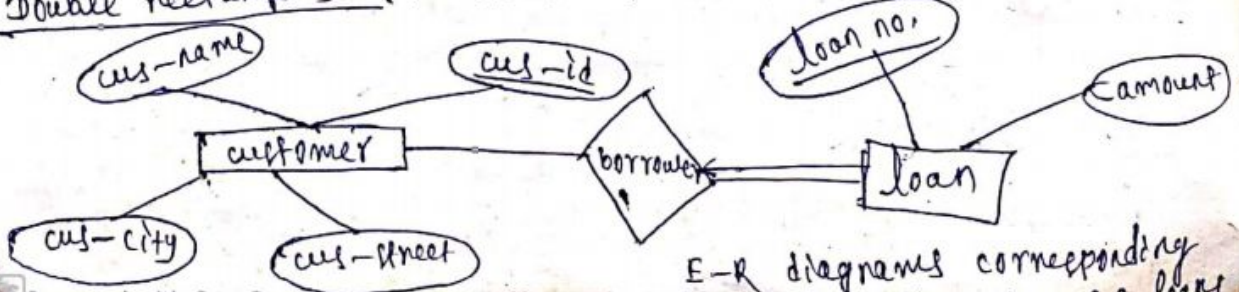
Lines :- which links attributes to entity sets and entity sets to relationship sets.

Double ellipses :- which represent multivalued attributes.

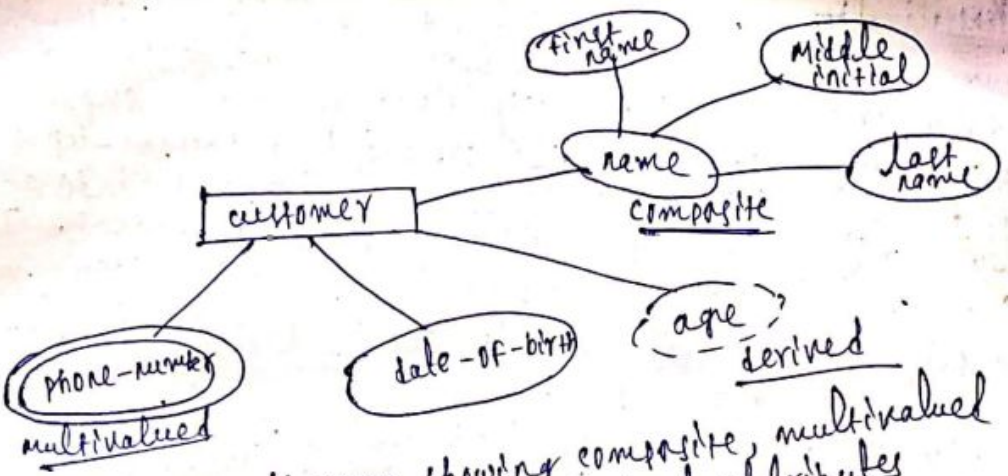
Dashed ellipse :- which denote derived attributes

Double lines :- which indicate total participation of an entity in a relationship set.

Double rectangles :- which represent weak entity set.



E-R diagrams corresponding to customers & loans



E-R diagram showing composite, multivalued and derived attributes

weak entity sets :- An entity set may not have sufficient attributes to form a primary key. Such an entity set is known as weak entity set and the entity set that has a primary key is termed a strong entity set.

→ For e.g. consider the entity set payment, which has 3 attributes payment-no, payment-date and payment-amount. payment numbers are typically sequential numbers, starting from 1, generated separately for each loan. But payments for different loans may share the same payment number. Thus, this entity set does not have a primary key, it is a weak entity set. Here the payment number allows certain level of distinction and this type of attribute is called discriminator of the weak entity set.

Extended E-R features :-

Specialization :- An entity set may include subgroupings of entities that are distinct in some way from other entities in the set. The E-R model provides a means for representing these entity groupings.

→ consider an entity set person, with attributes name, street and city. A person may be further classified into

* customer * employee

* Here customer having attributes that is same as person as well as some additional attributes such as customer-id. whereas employee entities may be described further by the attributes employee-id and salary. The process of designating subgrouping within an entity set is called **specialization**.

→ The specialization of person allows us to distinguish among persons according to whether they are employees or customers.

Generalization The refinement from an initial entity set into successive levels of entity subgroupings represents a top-down design process in which distinctions are made explicit.

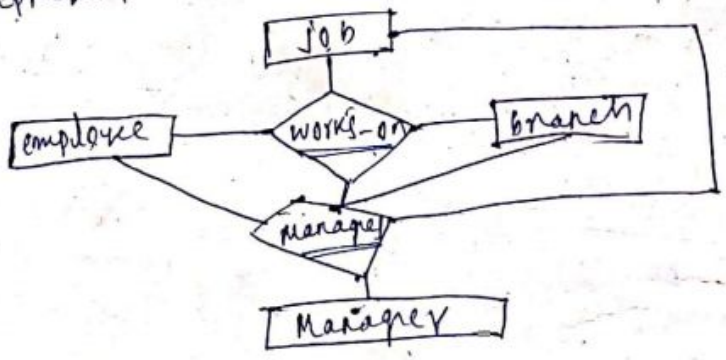
→ The design process may also proceed in a bottom-up manner, in which multiple entity sets are synthesized into a higher level entity set on the basis of common features. This commonality can be expressed by generalization.

→ For e.g. the customer entity set is having attributes name, street, city and customer-id and an employee entity set with the attributes name, street, city, employee-id and salary. There are similarities between the entity set customer and entity set employee. These similarities can be expressed by using generalization features.

→ Generalization is used to emphasize the similarities among lower-level entity sets and to hide the differences.

Aggregation :- Aggregation is an abstraction through which relationships are treated as higher level entities.

→ In this e.g. the relationship set works-on is treated as a higher level entity set. We can now create a binary relationship manages between works-on and manager to represent who manages the task.



The Network data Model :- The network data model represents data for an entity set by a logical record type.

→ The data for an instance of the entity set is represented by a record occurrence of the record type.

→ consider the entity set client which is having attributes client-no, name and address.

CLIENT | client-no | name | address |

This record type can be defined like this

```
type CLIENT = record
                client-no : integer;
                name : string;
                address : string;
              end
```

The occurrence of client record types are :-

1	X	BGH
2	Y	etc
3	Z	BBSR

Record :- Records provide the basic unit of access in the database such as client, employee.

Data item :- Data item is the smallest unit of named data. An occurrence of a data item is a representation of a value.

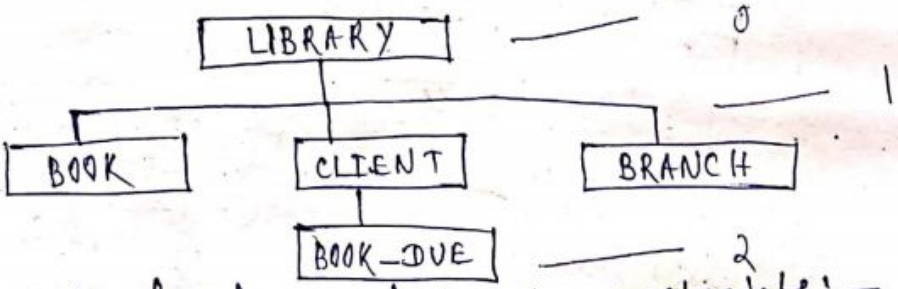
Data Aggregates :- A record could contain a named collection of data-items called data aggregates.

Keys :- A record key is a group of data-items or a single data-item used to identify a record or a group of records.

Hierarchical data model

The hierarchical data model uses the tree concept to represent data and the relationship among data.

→ The nodes of the tree are the record types representing the entity sets and are connected by pointers or links.
→ The relationship between the entities is represented by the structure of the resulting ordered tree.
→ A pointer or link as in the ~~network~~ Hierarchical data model represents a relationship between exactly two records.
→ The Hierarchical data model restricts each record type to only one parent record type.
→ A parent record type can have any number of children record types.
→ Two record types ~~can have~~ in a hierarchical tree can have at most one relationship between them and this relationship is that of one-to-one or one-to-many.



The hierarchical data model has the constraints:—
→ Each hierarchical tree can have only one root record type and this record type does not have a parent record type.
→ The root can have any no. of child record types, each of which can itself be a root of a hierarchical (sub) tree.
→ Each child record type can have only one parent record type thus a many-to-many relationship cannot be directly expressed between two record types.
→ Data in a parent record applies to all its children records.
→ Each occurrence of a record type can have any number of occurrences of each of its child record types.
→ A child record occurrence must have a parent record occurrence; deleting a parent record occurrence requires deleting all its child record occurrences.
→ A hierarchical tree can have any no. of record occurrences for each record type at each level of the hierarchical tree.
→ The diagram representing hierarchical data model is known as tree structure diagrams, defn trees or hierarchical defn trees.

Relational Database

Structure of Relational databases :-
A relational database consists of a collection of tables, each of which is assigned a unique name.

→ consider the account table. Acc-no, branch-name and balances are the attributes. For each attribute there is a set of permitted values known as domain of that attribute.

Account

acc no.	branch name	balance
A-1	Bgh	4000
A-2	ctc	5000
A-3	Bgh	6000
A-4	ctc	7000

→ D_1 represent the set of all account no. D_1 D_2 D_3
D_2 " " " " branch name.
D_3 " " " " balances.

→ Any row of a table consists of certain tuples.
In general, a table of n attributes must be a subset of $D_1 \times D_2 \times D_3 \times D_4 \times \dots \times D_{n-1} \times D_n$

→ The def^n of table corresponds to a relation in mathematics.

The Relational Algebra :- The relational algebra is a procedural query language. It consists of a set of operations that take one or two relations as input and produce a new relation as their result.

Basic operations :- The relational algebric operations can be divided into set-oriented operations and relational-oriented operations. The set-oriented operations are the union, intersection, cartesian product. The other type of operations include select, project and join.

Union (U) operation :- The union operation requires that the operand relations be union compatible.

→ 2 relations are union compatible if they have same no. of attributes and the attributes should be identical.

→ If we assume that P(P) and Q(Q) are two union compatible relation, then the union of P(P) and Q(Q) is represented by resultant relation R.

$$R = P \cup Q = \{ t \mid t \in P \vee t \in Q \} \text{ and}$$
$$\max(|P|, |Q|) \leq |R| \leq |P| + |Q|$$

P:

id	name
101	X
103	Y
104	M
107	Z
110	Y

Q:

id	name
103	Y
104	M
106	A
110	Y

R:

id	name
101	X
103	Y
104	M
106	A
107	Z
110	Y

P∪Q

Difference (−) :- The difference operation should be union compatible. The difference operation removes common tuples from the first relation.

$R = P - Q$ such that

$R = \{ t \mid t \in P \wedge t \notin Q \}$ and

$0 \le |R| \le |P|$

R: P−Q

id	name
101	X
107	Z

Intersection (∧) :- The intersection operation should also be union compatible. The intersection operation selects the common tuples from the two relations.

$R = P \wedge Q$ where $R = \{ t \mid t \in P \wedge t \in Q \}$ and

$0 \le |R| \le \min(|P|, |Q|)$

$R = P \wedge Q$

id	name
103	Y
104	M
110	Y

Cartesian product :- The cartesian product of two relations is the concatenation of tuples belonging to the two relations. A new resultant relation scheme is created consisting of all possible combination of the tuples.

$R = P \times Q$ $\quad R = \{ t_1 \| t_2 \mid t_1 \in P \wedge t_2 \in Q \}$

P:

id	name
101	X
102	Y
103	Z
104	A
105	B

Q:

S
J_1
J_2

R = P×Q:

id	name	S
101	X	J_1
101	X	J_2
102	Y	J_1
102	Y	J_2
103	Z	J_1
103	Z	J_2
104	A	J_1
104	A	J_2
105	B	J_1
105	B	J_2

Additional Relational Algebric operations! :-

Projection (π) :- The projection of a relation is defined as a projection of all its tuples over some set of attributes. It yields a vertical subset of the relation.

→ The projection operation is used to either reduce the no. of attributes in the resultant relation or to reorder attributes.

→ In the first case the arity or degree of the relation is reduced. Here the cardinality may also be reduced due to the deletion of duplicate tuples in the projected relation

PERSONNEL

id	name
1	X
2	Y
3	Z
4	A
5	X

$\Pi_{name}(PERSONNEL)$

name
X
Y
Z
A

CUSTOMER

id	name	address
1	A	BGH
2	B	BGH
3	C	CTC
4	D	CTC

$\Pi_{name, address}(CUSTOMER)$

name	address
A	BGH
B	BGH
C	CTC
D	CTC

Selection (σ) :- The projection operation yields a vertical subset of a relation. The selection operation, however, yields a horizontal subset of a given relation i.e., the action is defined over the complete set of attributes names but only a subset of the tuples are included in the result.

→ To have a tuple included in the result relation, the specified selection condition or predicates must be satisfied by it. The selection operation, represented by the symbol σ and it is sometimes known as restriction operation.

→ E.g. $\sigma_{id<5}(PERSONNEL) \Rightarrow$

id	name
1	X
2	Y
3	Z
4	A

→ Find those customers who live in BGH.

$$\Pi_{name}(\sigma_{address = "BGH"}(customer))$$

CUSTOMER

id	name	address
1	A	BGH
2	B	BGH
3	C	CTC
4	D	CTC

⇒

id	name	address
1	A	BGH
2	B	BGH

⇓

name
A
B

✓

Join (⋈) :- The join operator, as the name suggests, allows the combining of two relations to form a single new relation.

→ The tuples from the operand relations that participate in the operation, and contribute to the result are related.

→ The join operation allows the processing of relationships existing between the operand relations.

→ The natural join is a binary operation that allows us to combine certain selections and a cartesian product into one operation.

EMP:

id	Name
1	X
2	Y
3	Z

SALARY

id	salary
1	1000
2	2000
3	3000

EMP ⋈ SALARY

id	name	salary
1	X	1000
2	Y	2000
3	Z	3000

→ The join require the domain compatibility feature. It means the domains of A_i and B_i be compatible, and for this reason relation schemas P and Q have attributes defined on common domains.

→ Therefore, join attributes have common domains in the relation schemes P and Q. Consequently, only one set of the join attributes on these common domain needs to be preserved in the result relation. This is achieved by taking a projection after the join operation, thereby eliminating the duplicate attributes. ⋯→

$$R = P \underset{B}{\bowtie} Q$$

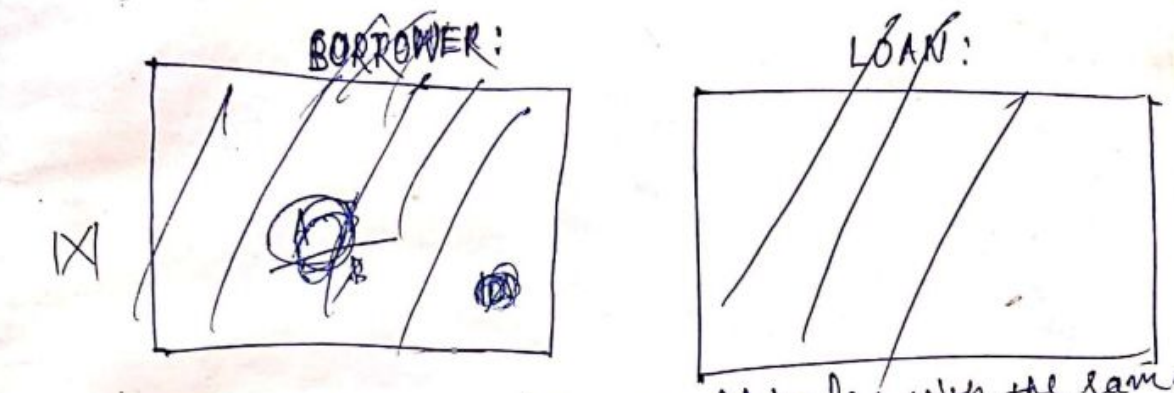
$$R = \{ t \mid t_1 \| t_2 \wedge t_1 \in P \wedge t_2 \in Q \wedge B \}$$

where B is the selection predicate consisting of the terms of the form:

$$(t_1[A_i] \; \theta_i \; t_2[B_i]) \text{ for } i = 1, 2 \dots n$$

✓ $\theta_i \in \{ =, \neq, <, \leq, >, \geq \}$ is known as comparison operator and A_i and B_i are some domain compatible attributes.

Equi join :- In the equi join the comparison operator (θ_i) $(i = 1, 2 \ldots n)$ is always the equality operator.



→ If the relations P and Q have attributes with the same domains but different names, then renaming or projection may be specified.

Division (÷)

P(P):

A	B
a_1	b_1
a_1	b_2
a_2	b_1
a_3	b_1
a_5	b_1
a_5	b_2

Q(Q):

B
b_1
b_2

$R = P \div Q$

A
a_1
a_5

Example

EMP:

EMP NO.	Name
101	A
103	B
104	C
106	D
107	E

ASSIGNED-TO

Proj. No	Emp No.
C45	101
C54	103
C43	104
C31	106
C53	107

Project

Proj. No.	Proj. Name	Chief Architect
C31	DB	106
C53	DB	107
C54	OS	103
C45	OS	101

Prob :- "Get the employee number of employees working on project 'C53'".

$\Pi_{\text{emp no.}}\left(\sigma_{\text{project no.} = \text{'C53'}}(\text{ASSIGNED-TO})\right)$

Ans :-

Emp no.
107

~~Prob Get details~~

Ans :-

EMP. No	Name
107	E

Prob : Get details of employees (both name and number) working on project 'C53'.

Ans :- EMPLOYEE ⋈ $\Pi_{\text{emp no.}}\left(\sigma_{\text{project no.} = \text{'C53'}}(\text{ASSIGNED-TO})\right)$

chapter-5 Structured Query language

Query language :- A query language is a language in which a user requests information from the database. These languages are usually on a level higher than that of a standard programming language.

→ Query languages can be of two types and they are namely procedural and non procedural language.

→ In a procedural language, the user instructs the system to perform a sequence of operations on the database to compute the desired result. In a non-procedural language the user describes the desired information without giving a specific procedure for obtaining that information.

SQL :- Structured query language originated with the System R project in 1974 at IBM's San Jose Research center. The System R project, concluded in 1979, was followed by the release of a no. of commercial relational DBMS products from IBM. The first of these SQL/DS for IBM's mid-range computers.

→ SQL (the original version was called SEQUEL and a predecessor of SEQUEL was named SQUARE) was the data definition and manipulation language for System R.

→ SQL has emerged as the standard query language for relational DBMSs, and most of the commercial relational DBMS use SQL or a variant of SQL.

→ SQL is both the data definition and data manipulation language of a no. of relational database systems like IBM prototype System R, IBM's DB2 and ORACLE and many others.

→ SQL is based on tuple calculus and relational algebra.

Data definition : SQL :- Data definition in SQL is via the create statement. The statement can be used to create a table, index or view (i.e. a virtual table based on existing tables). To create a table, the create statement specifies the name of the table and the names and data types of each column of the table.

Its format is :

create table <relation> (<attribute list>)

where the <attribute list> is specified as:

<attribute list> = <attribute name> (<data type>) [not null] [, <attribute list>] ;

→ The datatypes supported by SQL depend on the particular implementation.

Relational Database

Structure of Relational databases :-
A relational database consists of a collection of tables, each of which is assigned a unique name.

→ consider the account table. Acc-no, branch-name and balances are the attributes. For each attribute there is a set of permitted values known as domain of that attribute.

Account

acc no.	branch name	balance
A-1	Bgh	4000
A-2	ctc	5000
A-3	Bgh	6000
A-4	ctc	7000

→ D_1 represent the set of all account no. D_1 D_2 D_3
D_2 " " " " " branch name.
D_3 " " " " " balances.

→ Any row of a table consists of certain tuples.
In general, a table of n attributes must be a subset of $D_1 \times D_2 \times D_3 \times D_4 \times \dots \times D_{n-1} \times D_n$

→ The def^n of table corresponds to a relation in mathematics.

The Relational Algebra :- The relational algebra is a procedural query language. It consists of a set of operations that take one or two relations as input and produce a new relation as their result.

Basic operations :- The relational algebric operations can be divided into set-oriented operations and relational-oriented operations. The set-oriented operations are the union, intersection, cartesian product. The other type of operations include select, project and join.

Union (U) operation :- The union operation requires that the operand relations be union compatible.

→ 2 relations are union compatible if they have same no. of attributes and the attributes should be identical.

→ If we assume that P(P) and Q(Q) are two union compatible relation, then the union of P(P) and Q(Q) is represented by resultant relation R.

$$R = P \cup Q = \{ t \mid t \in P \vee t \in Q \} \text{ and}$$
$$\max(|P|, |Q|) \leq |R| \leq |P| + |Q|$$

P:

id	name
101	X
103	Y
104	M
107	Z
110	Y

Q:

id	name
103	Y
104	M
106	A
110	Y

R:

id	name
101	X
103	Y
104	M
106	A
107	Z
110	Y

P∪Q

Difference (−) :- The difference operation should be union compatible. The difference operation removes common tuples from the first relation.

$R = P - Q$ such that

$R = \{ t \mid t \in P \wedge t \notin Q \}$ and

$0 \leq |R| \leq |P|$

R: P−Q

id	name
101	X
107	Z

Intersection (∧) :- The intersection operation should also be union compatible. The intersection operation selects the common tuples from the two relations.

$R = P \wedge Q$ where $R = \{ t \mid t \in P \wedge t \in Q \}$ and

$0 \leq |R| \leq \min(|P|, |Q|)$

$R = P \wedge Q$

id	name
103	Y
104	M
110	Y

Cartesian product :- The cartesian product of two relations is the concatenation of tuples belonging to the two relations. A new resultant relation scheme is created consisting of all possible combination of the tuples.

$R = P \times Q$ $\quad R = \{ t_1 \| t_2 \mid t_1 \in P \wedge t_2 \in Q \}$

P:

id	name
101	X
102	Y
103	Z
104	A
105	B

Q:

S
J_1
J_2

R = P×Q:

id	name	S
101	X	J_1
101	X	J_2
102	Y	J_1
102	Y	J_2
103	Z	J_1
103	Z	J_2
104	A	J_1
104	A	J_2
105	B	J_1
105	B	J_2

Additional Relational Algebric operations! :-

Projection (π) :- The projection of a relation is defined as a projection of all its tuples over some set of attributes. It yields a vertical subset of the relation.

→ The projection operation is used to either reduce the no. of attributes in the resultant relation or to reorder attributes.

→ In the first case the arity or degree of the relation is reduced. Here the cardinality may also be reduced due to the deletion of duplicate tuples in the projected relation

PERSONNEL

id	name
1	X
2	Y
3	Z
4	A
5	X

$\Pi_{name}(PERSONNEL)$

name
X
Y
Z
A

CUSTOMER

id	name	address
1	A	BGH
2	B	BGH
3	C	CTC
4	D	CTC

$\Pi_{name, address}(CUSTOMER)$

name	address
A	BGH
B	BGH
C	CTC
D	CTC

Selection (σ) :- The projection operation yields a vertical subset of a relation. The selection operation however, yields a horizontal subset of a given relation i.e., the action is defined over the complete set of attributes names but only a subset of the tuples are included in the result.

→ To have a tuple included in the result relation, the specified selection condition or predicates must be satisfied by it. The selection operation, represented by the symbol σ and it is sometimes known as restriction operation

→ E.g. $\sigma_{id<5}(PERSONNEL) \Rightarrow$

id	name
1	X
2	Y
3	Z
4	A

→ Find those customers who live in BGH.

$$\Pi_{name}(\sigma_{address = "BGH"}(customer))$$

CUSTOMER

id	name	address
1	A	BGH
2	B	BGH
3	C	CTC
4	D	CTC

⇒

id	name	address
1	A	BGH
2	B	BGH

⇓

name
A
B

✓

Join (⋈) :- The join operator, as the name suggests, allows the combining of two relations to form a single new relation.

→ The tuples from the operand relations that participate in the operation, and contribute to the result are related.

→ The join operation allows the processing of relationships existing between the operand relations.

→ The natural join is a binary operation that allows us to combine certain selections and a cartesian product into one operation.

EMP:

id	Name
1	X
2	Y
3	Z

SALARY

id	salary
1	1000
2	2000
3	3000

EMP ⋈ SALARY

id	name	salary
1	X	1000
2	Y	2000
3	Z	3000

→ The join require the domain compatibility feature. It means the domains of A_i and B_i be compatible, and for this reason relation schemas P and Q have attributes defined on common domains.

→ Therefore, join attributes have common domains in the relation schemes P and Q. Consequently, only one set of the join attributes on these common domain needs to be preserved in the result relation. This is achieved by taking a projection after the join operation, thereby eliminating the duplicate attributes. ⋯→

$$R = P \underset{B}{\bowtie} Q$$

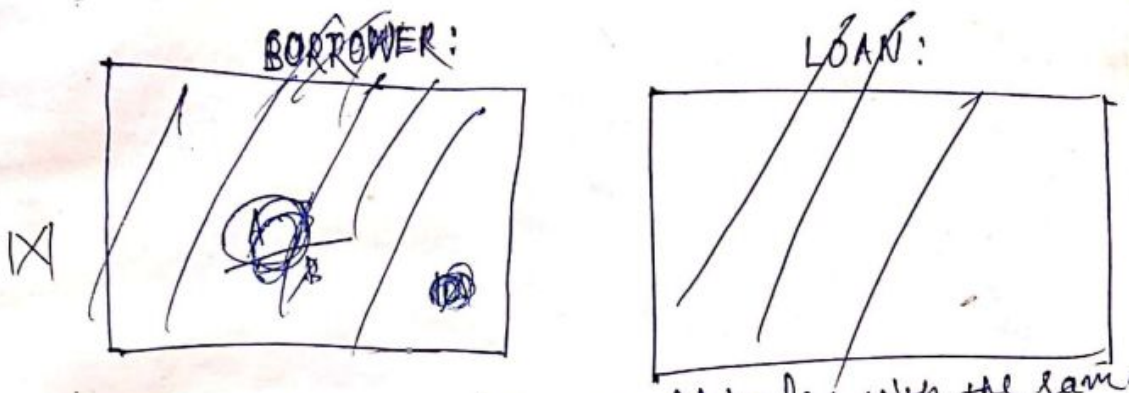
$$R = \{ t \mid t_1 \| t_2 \wedge t_1 \in P \wedge t_2 \in Q \wedge B \}$$

where B is the selection predicate consisting of the terms of the form:

$$(t_1[A_i] \; \theta_i \; t_2[B_i]) \text{ for } i = 1, 2 \dots n$$

$\theta_i \in \{ =, \neq, <, \leq, >, \geq \}$ is known as comparison operator and A_i and B_i are some domain compatible attributes.

Equi join :- In the equi join the comparison operator (θ_i) $(i = 1, 2 \ldots n)$ is always the equality operator.



→ If the relations P and Q have attributes with the same domains but different names, then renaming or projection may be specified.

Division (÷)

P(P):

A	B
a_1	b_1
a_1	b_2
a_2	b_1
a_3	b_1
a_4	b_1
a_5	b_2

Q(Q):

B
b_1
b_2

$R = P \div Q$

A
a_1
a_5

Example

EMP:

EMP NO.	Name
101	A
103	B
104	C
106	D
107	E

ASSIGNED-TO

Proj. NO	Emp NO.
C45	101
C54	103
C43	104
C31	106
C53	107

Project

Proj. NO.	Proj. Name	Chief Architect
C31	DB	106
C53	DB	107
C54	OS	103
C45	OS	101

Prob :- "Get the employee number of employees working on project 'C53'".

$\Pi_{\text{emp no.}}\left(\sigma_{\text{project no.} = \text{'C53'}}(\text{ASSIGNED-TO})\right)$

Ans :-

Emp no.
107

~~Prob Get details~~

Ans :-

EMP. No	Name
107	E

Prob :- Get details of employees (both name and number) working on project 'C53'.

Ans :- EMPLOYEE ⋈ $\Pi_{\text{emp no.}}\left(\sigma_{\text{project no.} = \text{'C53'}}(\text{ASSIGNED-TO})\right)$

chapter-5 Structured Query language

Query language :- A query language is a language in which a user requests information from the database. These languages are usually on a level higher than that of a standard programming language.

→ Query languages can be of two types and they are namely procedural and non procedural language.

→ In a procedural language, the user instructs the system to perform a sequence of operations on the database to compute the desired result. In a non-procedural language the user describes the desired information without giving a specific procedure for obtaining that information.

SQL :- Structured query language originated with the System R project in 1974 at IBM's San Jose Research center. The System R project, concluded in 1979, was followed by the release of a no. of commercial relational DBMS products from IBM. The first of these SQL/DS for IBM's mid-range computers.

→ SQL (the original version was called SEQUEL and a predecessor of SEQUEL was named SQUARE) was the data definition and manipulation language for System R.

→ SQL has emerged as the standard query language for relational DBMSs, and most of the commercial relational DBMS use SQL or a variant of SQL.

→ SQL is both the data definition and data manipulation language of a no. of relational database systems like IBM prototype System R, IBM's DB2 and ORACLE and many others.

→ SQL is based on tuple calculus and relational algebra.

Data definition : SQL :- Data definition in SQL is via the create statement. The statement can be used to create a table, index or view (i.e. a virtual table based on existing tables). To create a table, the create statement specifies the name of the table and the names and data types of each column of the table.

Its format is :

create table <relation> (<attribute list>)

where the <attribute list> is specified as:

<attribute list> = <attribute name> (<data type>) [not null] [, <attribute list>] ;

→ The datatypes supported by SQL depend on the particular implementation.

→ The data types included are: integer, decimal, real and character strings, both of fixed size and varying length.

The data types in SQL are:-

char(n): A fixed-length character string with user-specified length n. character, can be used instead.

→ varchar(n): A variable-length character string with user-specified maximum length n. The full form is character varying.

→ int: An integer datatype.

→ smallint: A small integer is a subset of int type.

→ numeric(p,d): A fixed-point number with user specified precision. The number consists of p digits (plus a sign) and d of the p digits are to the right of the decimal point. Thus numeric(3,1) allows 44.5 to be stored exactly, but neither 444.5 or 0.32 can be stored exactly.

→ real, double precision:- floating-point and double-precision floating-point numbers with machine dependent precision.

→ float(n): A f.p. numbers, with precision of at least n digits.

→ date: date containing 4 digits year, month and day of the month.
date can be specified like this
date '2002-04-30'.

E.g. of creating a table.

```
create table EMPLOYEE
        (EMP_NO integer not null,
         Name   char(25),
         Mobile char(10),
         pay_rate decimal(10,2));
```

null:- The null indicates value unknown or nonexistent

Alter :- The defn of an existing relation can be altered by using the alter statement. This statement allows a new column to be added to an existing relation. The existing tuples of the altered relation are logically considered to be assigned the null value for the added column. The physical alteration occurs to a tuple only during an update of the record. The syntax of the alter statement is

alter table existing table name
add column_name data type [---]

E.g. alter table EMP
add phone_number decimal(10);

Drop table :- To remove a relation from an SQL database we use the drop table command. The drop table command deletes all the information about the dropped relation from the database. The command is drop table r is a more drastic action than delete from r. The delete command retains relation r, but deletes all the tuples in r. The drop command deletes not only all tuples of r but also the schema for r. After r is dropped, no tuples can be inserted into r unless it is re-created with the create table command.

Null values :- SQL allows the use of null values to indicate absence of information about the value of an attribute.
→ we can use the special keyword null in a predicate to test for a null value.

Data manipulation : SQL :- To insert data into a relation, we either specify a tuple to be inserted or write a query whose result is a set of tuples to be inserted.

E.g.
```
create table Account
    ( Account-number char(5) not null,
      Branch char(10),
      Balance integer(10));

insert into account
    values ('A-001', 'BGH', 5000);
```

or
```
insert into account ( Account-number, branch, balance)
    values ('A-001', 'BGH', 5000)
```

or
```
insert into account values ('&Account-number',
    '&Branch', '&Balance');
```

```
SQL> Enter the value for Account-number : A-001
     "   "   "   "  Branch : CSE
     "   "   "   "  Balance : 1000

SQL> /
     Enter the value of Account-number : A-002
     "   "   "   "  Branch : ETC
     "   "   "   "  Balance : 2000

SQL> /
```

insert command :- insert command is used to insert rows in the table.

→ values can be inserted for all the columns or for the selected columns. To the insert into command, values can also be given through a query, however, the columns of the table being inserted must match the columns output by the subquery.

→ character data will be enclosed within quotes.

→ Null values are given as null, without any quotes.

SELECT COMMAND :- Request for information stored in a table can be done through the select command. This is generally referred to as querying the table. The select is the most frequently used command as access to information is needed all the time.

Syntax :- select column name from table name ;

To select all the columns of the table.

→ select * from tablename ;

→ select * from tab ; // It will show all the tables created. //

→ select emp-no, e-name, dept-no from emp ;

Then it will show the above mentioned column only.

→ select distinct e-name from emp ; then it will show only the unique names not the duplicate values.

EMP table :-

EMP-NO	E-name	JOB	Salary	Dept-no.
1	X Y	MGR	3000	10
2	Y X	CLERK	3000	20
3	X Z	clerk	3000	10
4	Z M	MGR	5000	20
5	A C	Salesman	2000	30
6	B D	Salesman	2000	30
7	C A	Accountant	7000	10
8	D C.	Accountant	7000	10

Select command with where clause :- The where clause is used along with the select statement to specify the condition, based on which the rows will be extracted from a table with select.

Operators used with where clause are :- =, >, <, >=, <=, !=, AND, OR, NOT etc.

→ Get the employees details working under dept no. 10.

select * from emp where dept-no = 10 ;

→ Get the employees name and no. whose salary is more than 1000.

select e-name, emp-no from emp where sal > 1000 ;

→ List the employee number and name of the manager.
* Select emp-no, e-name from emp where job = 'manager';

→ List the names of the clerks working in dept. 20.
* Select e-name from emp where job = 'clerk' AND
dept-no = 20;

→ List the names of the Accountants and salesman.
* Select e-name from emp where job = 'salesman' OR
job = 'Accountant';

→ List the employees name and salary, whose salary is between 1000 and 7000.
* Select e-name, salary from emp where salary BETWEEN
1000 AND 7000;

Matching a pattern with a column from a table :-

→ The LIKE operator is used only with char and varchar2 to match a pattern.
→ '%' represents a sequence of zero or more characters.
→ '_' (underscore) stands for any single character.
→ Both '%' and '_' are used with the like operator to specify a pattern.

→ List the employees whose names starts with "K".
* Select e-name from emp where e-name like 'K%';
Here, the like operator will match the ename column with an 'K', followed by any number of characters.

→ List the employees names ending with 'C'.
* Select e-name from emp where e-name like '%C';

→ List the names and JOB of employees whose names have exactly 2 characters.
* Select e-name, job from emp where e-name like '__';

→ List the name of the employee 'M' as the second character.
* Select e-name from emp where e-name like '_M%';

Using expressions with columns :- Arithmetic computations can be done on numeric columns. Alias names can be given to columns and or expressions on query outputs.

→ Alias names are displayed in place of column names.
→ Alias names are given to the right of a column name, enclosed within quotes.

→ List the name, salary and PF amount of all the employees (PF is calculated as 10% of salary)
* Select e-name, salary, salary * .1 from emp;

we can provide an alias to the column 'salary * .1' as "PF".

* select e_name, sal, salary * .1 "PF" from emp;

o/p e_name salary PF

Ordering the results of a query :-

→ SQL uses the order by clause to impose an order on the result of a query. Order by clause is used with select statement.

→ The order by clause orders the query output according to the values in one or more selected columns. Multiple columns are ordered one within another, and the user specifies whether to order them in ascending or descending order. Ascending is the default.

→ List the emp_no, e_name, sal in ascending order of salary.

* select emp_no, e_name, sal from emp order by salary;

* select emp_no, e_name, sal from emp order by salary desc;

* select emp_no, e_name, sal from emp order by 3;

Update command :- In certain situations, we may wish to change a value in a tuple without changing all values in the tuple. For this purpose, the update statement can be used.

→ As we could for insert and delete, we can choose the tuples to be updated by using a query.

→ update command is used to update the column in a table.

→ values of a single column or a group of columns can be updated. Updation can be carried out for all the rows in a table or selected rows. Update command sets specific values for such columns required to change.

→ increase everybody's salary by 10%.

* update emp set salary = salary + (salary * .1);

→ change the department of xy to 40.

* update emp set dept_no = 40 where e_name = 'xy';

Views :- A view is like a window through which data in a table can be viewed or changed. It is a virtual table that is, it does not have any data of its own, but designs the data from the table, it is associated with. It manipulates data in the underlying base table.

→ A view are database objects whose contents are derived from another table. A view contains no data of its own.

→ The changes in the tables are automatically reflected in the views.

→ A view is queried just like querying a table.

→ creating a view :- create view viewname as <query expression>

* create view emp11 as select emp_no, salary from emp;

Set operators :- Set operators are used to combine information of similar type from one or more than one table.
→ The types of set operators are: union, intersect, minus etc.

Union :- The union clause merges the outputs of two or more queries into a single of rows and columns.

Syntax : select <statement1>
union
select <statement2>

→ display the different designations in department 20 and 30.
* select job from emp where dept-no = 20
union
select job from emp where dept-no = 30;

job
clerk
MGR
salesman

Intersect The intersect operator returns the rows that are common between 2 sets of rows.

Syntax : select <statement1> intersect select <statement2>

→ List the jobs common in dept 20 and 30.
* select job from emp where dept-no = 20
intersect
select job from emp where dept-no = 30;

job
MGR
clerk

Minus :- Minus operator returns the rows unique to first query.

→ List the jobs unique to dept. 10.
* select job from emp where dept-no = 10
minus
select job from emp where dept-no = 20
minus
select job from emp where dept-no = 30;

job
Accountant

SQL functions functions can be used to perform complex calculations on data. Functions can modify individual data item
→ Functions can very easily manipulate o/p for group of rows.
→ Functions can alter date formats for display.
→ There are 2 types of functions are there. They are
* Scalar functions (single row functions)
* Group functions (Aggregate functions)
* select ln(2) from dual;
* select length('xyz') from dual;

Dual :- It is a table, which is automatically created by oracle along with the data dictionary. Dual table has one column defined to be as varchar2 data type and contain only one row with value 'x'. It is used for arithmetic calculations and date retrieval.
* select sysdate from dual;
* select user from dual; o/p :- Scott

* select abs(-15) from dual; o/p 15
* select sqrt(4) from dual; o/p 2
* select initcap('hello') from dual; Hello
* select lower('FUN') from dual; fun
* select upper('sun') from dual; SUN

Group functions :-

count :- It determines the number of rows present or non-null column values.

* select count(*) from emp;

→ List the no. of jobs available in the emp table.

* select count(distinct job) from emp; o/p - 4

sum :- The sum function returns the sum of values for the select list of columns.

→ List the total salary payable to employees.

* select sum(sal) from emp;

max :- max function returns the maximum value of the selected list of item.

→ List the maximum salary of the employee.

* select max(sal) from emp;

min :- min function returns the minimum value of the selected list of item.

* select min(salary) from emp;

AVG :- This function returns the average of column values.

→ List the average salary and no. of employees working in dept 20.

* select avg(sal), count(*) from emp where dept-no = 20;

CLE SCR — to clear screen

DESC table name.

TRANSACTION PROCESSING CONCEPTS

A transaction is a unit of program execution that accesses and possibly updates various data items. Usually, a transaction is initiated by a user program written in a high-level data-manipulation language or programming language for eg. C, C++, java etc, where it is delimited by statements or function calls of the form begin transaction and end transaction. The transaction consists of all operations executed between the begin transaction and end transaction.

→ To ensure integrity of the data, we require that the database system maintain some properties of transactions.

* Atomicity :- All operations of the transaction are reflected properly in the database.

* consistency :- Execution of a transaction in isolation preserves the consistency of the database.

* Isolation :- Even though multiple transactions may execute concurrently, the system guarantees that, for every pair of transactions T_i and T_j, it appears to T_i that either T_j finished execution before T_i started or T_j started execution after T_i finished. Thus each transaction is unaware of other transactions executing concurrently in the system.

* Durability :- After a transaction completes successfully, the changes it has made to the database persist, even if there are system failures. These properties are known as ACID properties, the name is derived from the first letter of each of the four properties.

Example :- Let a simplified banking system consist of several accounts and a set of transactions that access and update those accounts. Let's consider that the database permanently resides on disk, but some portion of it is temporarily residing in main memory. Then transaction access data using two operations :

→ read(X) :- It transfers the data item X from the database to a local buffer belonging to the transaction that executed the read operation. The buffer are the blocks of main memory.

→ write(X) :- It transfers the data item X from the local buffer of the transaction that executed the write back to the database.

* Let T_i be a transaction that transfers 5000 from account A to account B. This transaction can be defined as

```
Ti : read(A);
     A := A - 5000;
     write(A);
     read(B);
     B := B + 5000;
     write(B);
```

continues

Atomicity :— Sometimes a failure may occurs during the ~~transaction~~ execution of a transaction that failure prevents the transaction from completing its execution successful. Examples of such failures are the power failures, h/w failures and s/w errors etc. Because of the failure, the state of the system no longer reflects a real state of the world that the database is supposed to capture. Such a state is called an inconsistent state.

→ Such inconsistencies are not visible in a database system. But there will be some loss from that state. That is the reason for the atomicity requirement.

→ If the atomicity property is present, all actions of the transaction are reflected in the database.

→ The basic idea behind ensuring atomicity is this: The database system keeps track of the old values of any data on which a transaction performs a write, and if the transaction does not complete its execution, the database system restores the old values to make it appear as though the transaction never executed.

continues

Consistency :— The consistency requirement here is that the sum of A and B be unchanged by the execution of the transaction. Ensuring consistency for an individual transaction is the responsibility of the application programmer who codes the transaction.

continues

Durability :— Once the execution of the transaction completes successfully, and the user who initiated the transaction has been notified that the transfer of funds has taken place, it must be the case that no system failure will result in a loss of data corresponding to this transfer of funds.

→ The durability property guarantees that, once a transaction completes successfully, all the updates that it carried out on the database persist, even if there is a system failure after the transaction completes execution.

continues

Isolation :— Even if the consistency and atomicity properties are ensured for each transaction, if several transactions are executed concurrently, their operations may interleave in some desirable way, resulting in an inconsistent state.

→ A way to avoid the problem of concurrently executing transactions is to execute transactions serially—that is one after the other.

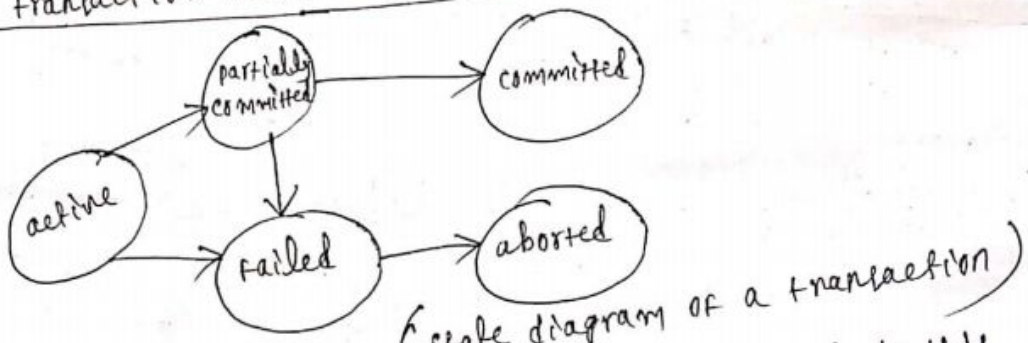
→ The isolation property of a transaction ensures that the concurrent execution of transactions results in a system state that is equivalent to a state that could have been obtained had these transactions executed one at a time in some order. Ensuring the isolation property of a transaction is the responsibility of a component of the database system called concurrency control component.

TRANSACTION STATE :- In the absence of failures, all the transactions complete successfully. However, a transaction may not always complete its execution successfully. Such a transaction is termed aborted. If we are to ensure the atomicity property, an aborted transaction must have no effect on the state of the database.

→ Thus any changes that the aborted transaction made to the database must be undone. Once the changes caused by an aborted transaction have been undone, then the transaction has been rolled back. It is the part of the responsibility of the recovery scheme to manage transaction aborts.

→ A transaction that completes its execution successfully is said to be committed. A committed transaction that has performed updates transforms the database into a new consistent state, which must persist even if there is a system failure. Once a transaction has committed, we can't undo its effects by aborting it. The one way to undo the effects of a committed transaction is to execute a compensating transaction.

A transaction must be in one of the following states :



(State diagram of a transaction)

Active :- the initial state, the transaction stays in this state while it is executing.

partially committed :- after the final statement has been executed.

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

Aborted :- After the transaction has been rolled back and the database has been restored to its state prior to the start of the transaction.

committed :- After successful completion.

→ The transaction has committed if it has entered the committed state. Similarly aborted only if it has entered the aborted state. A transaction is said to have terminated if has either committed or aborted.

→ A transaction starts in the active state. When it finishes its final statement, it enters the partially committed state. At this point, the transaction has completed its execution but it is still possible that it may have to be aborted,

since the actual o/p may still be temporarily residing in main memory, and thus a h/w failure may prevents the successful completion of the transaction. The database system then writes out enough information to disk that, even in the event of a failure, the updates performed by the transaction can be re-created when the system restart after the failure. when the last of this information is written out, the transaction enters the committed state.

→ A transaction enters the failed state after the system determines that the transaction can no longer proceed with its normal execution because of h/w or logical errors. Such a transaction must be rolled back. Then it enters the aborted state. At this point, the system has 2 options:

* It can restart the transaction, but only if the transaction was aborted as a result of some h/w or s/w error that was not created through the internal logic of the transaction. A restarted transaction is considered to be a new transaction.
* It can kill the transaction. It usually does so because of some internal logical error that can be corrected only by rewriting the application program, or because the input was bad, or because the desired data were not found in the database.

Concurrent Executions :- Transaction processing systems usually allow multiple transactions to run concurrently.

→ Allowing multiple transactions to update data concurrently causes several complications with consistency of data.

→ However, there are 2 good reasons for allowing concurrency

* Improved throughput and resource utilization :- A transaction consists of many steps. Some involve I/O activity others involve CPU activity.

→ The I/O operation can be done in parallel with processing at the CPU. The parallelism of the CPU and the I/O system can therefore be exploited to run multiple transactions in parallel.

→ Therefore while one read or write operation is carried out for one transaction another transaction can be running in the CPU, it increases the throughput means the no. of transactions executed in a given amount of time.

→ correspondingly, the processor and disk utilization also increase in other words, the processor and disk spend less time idle.

* Reduced waiting time :- There may be a mix of transactions running on a system, some short and some long. If transactions run serially, a short transaction may have to wait for a preceding long transaction to complete which can lead to unpredictable delays in running a transaction.

→ So the concurrent execution reduces the unpredictable delays in running transactions by operating the transactions on different parts of the database. The concurrent transactions shares the CPU cycles and disk accesses among them.

SCHEDULES :- When several transactions run concurrently, database consistency can be destroyed despite the correctness of each individual transaction.

→ Schedules represent the order in which instructions are executed in the system. A schedule for a set of transaction must consist of all instructions of those transactions and must preserve the order in which the instructions appear in each individual transaction.

T_1 : read(A);
A := A − 5000;
write(A);
read(B);
B := B + 50;
write(B).

Transaction T_2 transfers 10% of the balance from Account A to account B.

T_2 : read(A);
temp := A * 0.1;
A := A − temp;
write(A);
read(B);
B := B + temp;
write(B)

T_1	T_2
read(A)	
A := A − 5000	
write(A)	
read(B)	
B := B + 5000	
write(B)	
	read(A)
	temp := A * 0.1
	A := A − temp
	write(A)
	read(B)
	B := B + temp
	write(B)

→ The execution sequences of T_1 and T_2 transactions are called schedule. These schedule are serial.

Each serial schedule consists of a sequence of instructions from various transactions, where the instructions belonging to one single transaction appear together in that schedule.

→ Concurrent Schedules :- When the database system executes several transactions concurrently, the corresponding schedule no longer needs to be serial.

→ If 2 transactions are running concurrently the OS may execute one transaction for a little while and switch to another and then again it may execute the first one.

→ The concurrent schedules then contain several execution sequences that are possible, since the various instructions from both the transactions may be interleaved.

T_1	T_2
read(A)	
A := A - 5000	
write(A)	
	read(A)
	temp := A * 0.1
	A := A - temp
	write(A)
read(B)	
B := B + 5000	
write(B)	
	read(B)
	B := B + temp
	write(B)

T_8	T_9
read(A)	
write(A)	
	read(A)
read(B)	

Recoverability :- Schedules are acceptable to maintain the consistency of the database, assuming implicitly that there are no transaction failures.

→ If a transaction T_i fails, we need to undo the effect of this transaction. In a system that allows concurrent execution, it is necessary also to ensure that any transaction T_j that is dependent on T_i is also aborted. To achieve this we need to place restrictions on the type of schedules.

Recoverable schedule :- A recoverable schedule is one where, for each pair of transactions T_i and T_j such that T_j reads a data item previously written by T_i, the commit operation of T_i appears before the commit operation of T_j.

Cascadeless schedules :- Even if a schedule is recoverable, to recover from failure of T_i we may need to roll back several transactions.

→ Such situations occur if transactions have read data written by T_i. Then all of those transactions also be need to roll back along with T_i. The phenomenon, in which a single transaction failure leads to a series of transaction rollbacks, is called cascading rollback.

→ It is desirable to restrict the schedules to those where cascading rollbacks can't occur. Such schedules are called cascadeless schedules. A cascadeless schedule is one where for each pair of transactions T_i and T_j such that T_j reads a data item previously written by T_i, the commit operation of T_i appears before read operation of T_j.

→ It is easy to verify that every cascadeless schedule is also recoverable.

CONCURRENCY CONTROL CONCEPTS

One of the fundamental property of a transaction is isolation. When several transactions execute concurrently in the database, however, the isolation property may no longer be preserved.

→ To ensure that it is, the system must control the interaction among the concurrent transactions, this control is achieved through the variety of mechanisms called concurrency control schemes.

→ The concurrency control schemes are all based on the serializability property. That is, all the schemes presented should ensure that the schedules are serializable.

Serializability :— The database system must control concurrent execution of transactions, to ensure that the database state remains consistent.

→ So we have to first determine that which schedules would ensure consistency and which schedule would not.

→ Since the transactions are programs, it is quite obviously difficult to determine exactly what operations a transaction performs and how operations of various transactions interact.

→ We therefore consider only 2 operations read and write.

→ The different forms of schedule equivalence leads to the notion of serializability. 2 types of serializibility are there conflict and view.

Conflict :— consider a schedule S in which there are 2 consecutive instructions I_i and I_j of transactions T_i & T_j respectively.

→ We say that I_i & I_j conflict if they are operations by different transactions on the same data item and at least one of these instruction is a write operation.

→ write (A) instruction of T_1 conflicts with the read (A) of T_2, however write (A) of T_2 does not conflicts with the read (B) of T_1, because 2 instructions access different data items.

T_1	T_2
read (A)	
write (A)	read (A)
read (B)	write (A)
write (B)	read (B)
	write (B)

* If a schedule S can be transformed into a schedule S' by a series of swaps of non conflicting instructions, we can say S & S' are conflict equivalent.

* The concept of conflict equivalence leads to the concept of conflict serializability. A schedule S is conflict serializable if it is conflict equivalent to a serial schedule.

View Serializability :- consider two schedules S and S', where the same set of transactions participates in both schedules.
→ The schedules S and S' are said to be view equivalent if three conditions are met!
* for each data item Q, if transaction Ti reads the initial value of Q in schedule S, then transaction Ti must in schedule S', also read the initial value of Q.
* for each data item Q, if transaction Ti executes read(Q) in schedule S, and if that value was produced by a write(Q) operation executed by transaction Tj, then the read(Q) operation of transaction Ti must in schedule S', also read the value of Q that was produced by the same write(Q) operation of transaction Tj.
* for each data item Q, the transaction that performs the final write(Q) operation in schedule S must perform the final write(Q) operation in schedule S'.
→ The concept of view equivalence leads to the concept of view serializability. We say that a schedule S is view serializable if it is view equivalent to a serial schedule.

Concurrency control scheme :- The schemes involves in concurrency control are locking, timestamp based order, optimistic scheduling and multiversion techniques.
→ The intent of locking is to ensure serializability by ensuring mutual exclusion in accessing data items.
→ In the timestamp-based ordering, the order of execution of the transaction is selected a priori by assigning each transaction an unique value.
→ In optimistic scheme it is recognized that the conflict between transactions, though possible, is in reality very rare, and it avoids old forms of locking.
→ In multiversion technique a data-item is never written over, each write operation creates a new version of a data item.

Locking Scheme :- From the point of view of locking, a database can be considered as being made up of a set of data items. A lock is a variable associated with each such data item.
→ Manipulating the value of a lock is called locking.
→ Locking the ~~transaction~~ items being used by a transaction can prevent other concurrently running transactions from using these locked items.
→ The locking is done by a subsystem of the DBMS usually called the lock manager.
→ Basically 2 types of locks are defined:
exclusive lock, shared lock.

E.Lock :- The exclusive lock is also called an update or a write lock. The intention of this mode of locking is to provide exclusive use of data item to one transaction.
→ If a transaction T locks the data item Q in an exclusive mode, no other transaction can access Q.

S.lock :- This lock is also called a read lock. The intention of this mode of locking is to ensure that the data item does not undergo any modifications while it is locked in this mode.
→ Any no. of transactions can lock a data item in the shared mode and can also access the data item, but non of these transactions can modify the data item.

current state of locking data item

Lock mode of request	unlocked	shared	Exclusive
unlock		yes	yes
shared	yes	yes	no
Exclusive	yes	no	no

T1
LockX(A);
Read(A);
A := A - 100;
write(A);
unlock(A);

→ If one data item is unlock we can have both S.lock and E.lock on that item.
→ If one data item is ~~in shared mode we can~~ unlock ~~and also provide shared lock on that but not~~ but if that item is in shared mode we can provide S.lock on that data item and in provide the shared lock on that item.
→ If one data item is unlocked we can provide exclusive lock on that item but we can't provide exclusive lock on the data item having a shared lock or exclusive lock.
→ LockS(A) means requesting for a shared lock on data item A.
→ LockX(A) → requesting for a exclusive lock on A.
→ unlock(A) → lock is released from A.

Deadlock A system is in a deadlock state if there exists a set of transactions such that every transaction in the set is waiting for another transaction in the set.
→ If there exists a set of waiting transactions $\{T_0, T_1 \cdots T_n\}$ such that T_0 is waiting for a data item that T_1 holds and T_1 is waiting for a data item that T_2 holds and ... and T_{n-1} is waiting for a data item that T_n holds and T_n is waiting for a data item that T_0 holds then non of these transactions can make progress in such a situation.
The only method to recover from this is to take some drastic action such as rolling back some of the transaction involved in the deadlock. The rollback of the transaction may be partial

means a transaction may be rolled back to the point where it obtained a lock whose release resolves the deadlock.

→ There are two principal methods for dealing with the deadlock problem. We can use a deadlock prevention protocol to ensure that the system will never enter a deadlock state.

→ Alternatively, we can allow the system to enter a deadlock state and then try to recover by using a deadlock detection and deadlock recovery scheme.

→ Both methods may result in transaction rollback. Prevention is commonly used if the probability that the system would enter a deadlock state is relatively high; otherwise, detection and recovery are more efficient.

→ A detection and recovery scheme requires overhead that includes not only the run-time cost of maintaining the necessary information and of executing the detection algorithm, but also the potential losses inherent in recovery from a dead lock.

Deadlock prevention :— There are two approaches to deadlock prevention. One approach ensures that no cyclic waits can occur by ordering the requests for locks, or requiring all locks to be acquired together.

→ The other approach is closer to deadlock recovery, and performs transaction rollback instead of waiting for a lock whenever the wait could potentially result in a deadlock.

→ The simplest scheme under the first approach requires that each transaction locks all its data items before it begins execution.

→ There are 2 disadvantages associated with this scheme.

* It is often hard to predict, before the transaction begins, what data items need to be locked.
* Data item utilization may be very low, since many of the data items may be locked but unused for a long time.

→ Another approach for preventing deadlocks is to impose an ordering of all data items, and to require that a transaction lock data items only in a sequence consistent with the ordering.

→ Once a transaction has locked a particular item, it can't request locks on items that precedes that item in the ordering.

→ Two different deadlock prevention schemes are used. using the timestamps.

* The wait-die scheme is a nonpreemptive technique. When transaction Ti requests a data item currently held by Tj, Ti is allowed to wait only if it has a timestamp smaller than that of Tj otherwise Ti is rolled back (dies).

* The wound-wait scheme is preemptive technique. It is a counterpart to the wait-die scheme. When transaction T_i requests a data item currently held by T_j, T_i is allowed to wait only if it has a timestamp larger than that of T_j otherwise T_j is rolled back.

→ Whenever the system rolls back transactions, it is important to ensure that there is no starvation — that is, no transaction gets rolled back repeatedly and is never allowed to make progress.

Deadlock detection and Recovery :— To recover from the deadlock the system must:

→ maintain information about the current allocation of data items to transactions, as well as any outstanding data item requests.

→ provide an algorithm that uses this information to determine whether the system has entered a deadlock state.

→ Recover from the deadlock when the detection algorithm determines that a deadlock exists.

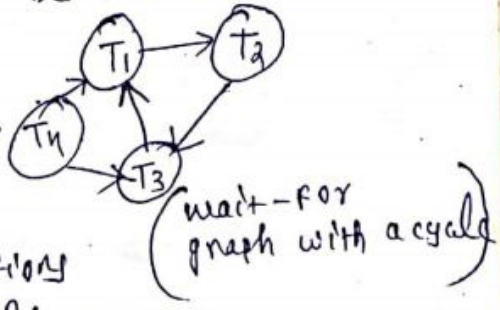
Deadlock detection :— Deadlocks can be described in terms of a directed graph called a wait-for graph. This graph consists of a pair $G = (V, E)$ where V is the set of vertices and E is the set of edges. The set of vertices consists of all the transactions in the system. Each element in the set E of edges is an ordered pair $T_i \rightarrow T_j$

⊛ $T_i \rightarrow T_j$ means transaction T_i is waiting for transaction T_j to release a data item that it needs.

→ A deadlock exists in the system if and only if the wait-for graph contains a cycle. Each transaction involved in the cycle is said to be deadlocked.

→ If deadlocks occur frequently, then the detection algorithm should be invoked more frequently.

→ Data items allocated to deadlocked transactions will be unavailable to other transactions until the deadlock is broken.



(wait-for graph with a cycle)

Recovery from deadlock :— When a detection algorithm determines that a deadlock exists, the system must recover from the deadlock. The most common solution is to roll back one or more transactions to break the deadlock.

Three ~~approaches~~ actions are there! —

* Selection of a victim :— Given a set of deadlocked transactions, we have to determine which transaction to roll back to break the deadlock.

→ we should rolled back those transactions that will result in minimum loss.

*Rollback :- Once we have decided that a particular transacti must be rolled back, we must determine how far this transaction should be rolled back.

→ The simplest solution is a total rollback. Then the transaction will be aborted and restarted.

→ It is more effective to roll back the transaction only as far as necessary to break the deadlock. Such partial rollback requires the system to maintain additional information about the state of running transaction. But the transactions must be capable of resuming execution after a partial rollback.

Starvation :- If one particular transaction is selected as victim repeatedly to be rollbacked then that transaction can never complete its task and thus there is starvation occurs. So we must ensure that transaction can be picked as a victim only a finite small no. of times.

Live lock :- It may so happen that B transaction requests for a lock and that is already hold by transaction A then the request may be denied. The transaction B has to wait until the lock has been released. But the system guarantees that B transaction need not to wait forever. Such a possibility is called live lock. The system guarantees to provide the lock to all transaction in some order.

SECURITY AND INTEGRITY

→ Security in a database involves both policies and mechanisms to protect the data and ensure that it is not accessed, altered or deleted without proper authorization.
→ Integrity implies that any properly authorized access, alteration or deletion of the data in the database does not change the validity of the data.
→ Security and integrity concepts though distinct, are related.
→ Implementation of both integrity and security requires that certain controls in the form of constraints must be build into the system.

Security and integrity threats (Accidental)

→ A user can get access to a portion of the database not normally accessible to that user due to a system error or an error on the part of another user.
→ Failures of various forms during normal operation.
→ concurrent execution anomalies.
→ System error.
→ Improper authorization. The authorizer can accidentally give improper authorization to a user.
→ H/w failures.

Intentional :— A system programmer can intentionally bypass the normal security and integrity mechanisms, alter or destroy the data in the database or make unauthorized copies of sensitive data.
→ Authorized users could pass sensitive information for personal gain.
→ An unauthorized user can get access to the password of an authorized user and could destroy the database files.

3/ Authorization :— Authorization is expressed as a set of rules that can be used to determine which user has what type of access of which portion of the database. The person who is in charge of specifying the authorization is usually called the authorizer. The authorizer can be distinct from the DBA and usually is the person who owns the data.
→ The authorization is usually maintained in the form of a table called an access matrix. The access matrix contains rows called subjects and columns termed objects.
→ The entry in the matrix at the position corresponding to the intersection of a row and column indicate the type of access that the subject has with respect to the object.

Objects :- An object is something that needs protection and one of the first steps in the authorization process is to select the objects to be used for security enforcement.
→ A typical object in a database environment could be a unit of data that needs to be protected.
→ A data field, a record or a file could be considered as object.

Views as objects :- views or subscheme can be used to enforce security. A user is allowed access to only that portion of the database defined by the user's view. A number of user may share a view.
→ The advantage of this approach is that the no. of objects accessible to a class of users and the entry for it in the authorization matrix is reduced to one per view.
→ The disadvantage is that the entire class of users have the same access rights.

Subject :- A subject is an active element in the security mechanism; it operates on objects. A subject is a user who is given some rights to access a data object. We can also treat a class of users or an application program as a subject.
→ A user who belongs to or joins a class of users gets the access rights of that class of users.

Access types The access allowed to a user could be for data manipulation or control. The manipulation operations are read, insert, delete, update. The control operations are add, drop, alter, and propagate access control.
→ Read :- Allows reading only of the object.
→ Insert :- Allows inserting new occurances of the object type.
→ delete :- Allows deleting an existing occurrence of the object type.
→ update :- Allows the subject to change the value of the occurrence of the object.
→ Add :- Allows the subject to add new objects.
→ drop :- Allows the subject to drop the object types from the database.
→ Alter :- Allows the subject to add new data items or attributes to an existing record type or relation
→ propagate access control :- This is an additional right that determines if this subject is allowed to propagate the right over the object to other subjects.

1. To protect the database we must take security measures at several levels.

Physical :— The sites containing the computer system must be physically secured.

→ Security of the physical storage devices within the organization and when being transmitted from one location to another must be maintained.

→ User identifications and passwords have to be kept confidential.

Human factors :— The authorizer should grant proper database access authorization to the user community. Inadvertent assignment of authorization to a wrong class of users can result in possible security violations.

Administrative controls :— Administrative controls are the security and access control policies that determine what information will be accessible to what class of users, and the type of access that will be allowed to this class.

DBMS and OS security mechanisms :— The database depends on some of the protection features of the OS for security. The features are :—

→ The proper mechanisms for the identification and verification of the users. Each user is assigned an account number and a password. The OS ensures that access to the system is denied unless the number and password are valid.

→ The protection of data and programs both in primary and secondary memories.

Security constraints :— To prevent the database from unauthorized access an organization must apply some security policies or constraints to access the database.

→ Database security mechanisms are the functions used to enforce database security policies. These functions could be implemented by a combination of one or more of the following : administrative control procedures, h/w functions, s/w functions etc.

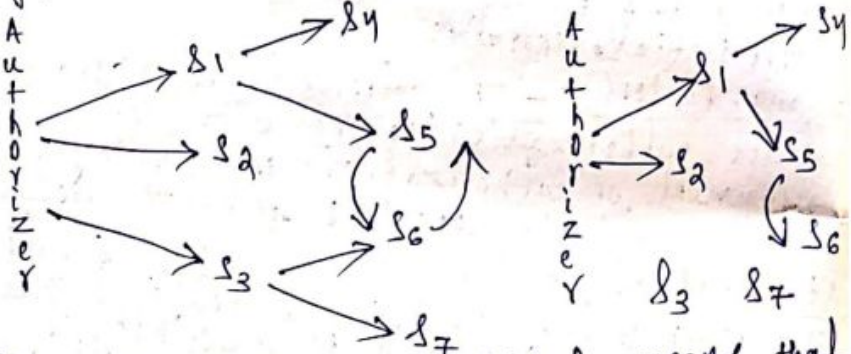
→ The administrative control procedures are the implementation of security policies to provide protection, external to the database, OS and computer h/w. An e.g. of such procedure is to have application programs written by one team and validated by a separate team.

→ Another rule is to change the password regularly.

→ Another mechanism is to choose the security features provided by the DBMS to adequately implement the security policies.

→ The other policy decision that has to be made is whether the focus of security administration is integrated with the DBA and whether security administration is centralized or decentralized.

Authorization grant tree :- when the authorizer grants a user some rights this may be granted with the propagate access control as well. This leads to an authorization grant tree. To properly revoke access rights, all paths in the access grant tree must start from the authorizer



If we revoke the access rights of subject S_3 means that subject S_7 also loses all rights. Subject S_6 retains those rights granted by S_5 and S_5 loses rights granted by S_6.

e.g. grant insert, update on table emp to personnel-manager';

e.g. grant insert, update, delete on table emp to 'clerk' with grant option;

e.g. revoke insert, update on table emp from clerk;

Identification and authentication :- Before making any request to access the database the user has to identify herself or himself to the system and authenticate the identification to confirm that the user is in fact the correct person.

→ The simplest and most common authentication scheme used is a password to authenticate the user. The user enters name or number and then authenticates herself or himself by the password.

→ Another method of authentication is that each user could be given an appropriately encoded card or key to be used for identification purposes.

→ Another method of authentication is the fingerprints or voiceprint etc.

Integrity constraints :- integrity constraints ensures that any properly authorized access, alteration, deletion, or insertion of the data in the database does not destroy the consistency and validity of the data.

→ Database integrity involves the correctness of data this correctness has to be preserved in the presence of concurrent operations and failures in h/w and s/w.

Domain or data item value integrity rules:
→ One of the most common integrity constraints that is specified and validated is to define the domain for each attribute or in case of network or hierarchical models define the value set for each data item.
→ For e.g. in the table emp we may have the domain of attribute salary between 12,000 to 30,000.

Implicit constraints :- The simplest e.g. of an implicit integrity constraint is that each record type must conform to the record declaration for that type.
→ The data model used by the DBMS implicitly builds in certain integrity constraints, such as the one-to-many relationship between a parent record type and the children record types in the hierarchical model.
→ The hierarchical model requires that the parent record type must exist for the child record type to be inserted in the database, the parent-to-child data structure is the implicit implementation of a one-to-many relationship.

Database constraints :- A database constraints specifies the legal value for a given database.

Encryption :- Storing and transmitting sensitive data in encrypted form is called data encryption. A data base system may make various authorization but may not provide sufficient protection for highly sensitive data, to avoid this difficulties data may be encrypted.
→ There are various techniques for the encryption of data are there. The simple techniques may not provide adequate security to the data. Since it may be easy for an unauthorized user to break the code.
→ One of the weak encrypted technique consist of the substitution of each character with the next character in the alphabet.
→ One of the better encryption scheme is the public key encryption. In this scheme both the encryption key and the encryption algorithm are public and readily available. This allow anyone to send a message in a coded form however the decryption key is secret and only the rightful recipient can decrypt the coded message.
→ The public key is a parameter of the encryption algorithm.

Functional Dependency : The functional dependency is a relationship that exists between two attributes.

→ It typically exists between the primary key and non-key attributes within a table.

→ The left side of FD is known as a determinant, the right side of the production is known as a dependent.

→ If column A of a table uniquely identifies the column B of the same table then it can represented as A → B (Attribute B is functionally dependent on Attribute A)

Types of functional Dependencies :

* Trivial Functional Dependency
* Non-trivial functional dependency
* Multivalued dependency
* Transitive dependency

Rules of Functional Dependencies

(Armstrong's Axioms)

* **Reflexive rule** : If X is a set of attributes and Y is a subset of X, then X holds the value of Y.
* **Augmentation rule** : When X → Y holds, and C is a attribute set, then ac → bc also holds, That is adding attributes which do not change the basic dependencies.
* **Transitivity rule** : This rule is very much similar to the transitive rule in algebra. If X → Y holds and Y → Z holds, then X → Z also holds. X → Y is called as functionally that determines Y.

Secondary Rules:

Union Rules:

If A holds B and A holds C, then A holds BC.
If $\{A \to B\}$ and $\{A \to C\}$ then $\{A \to BC\}$

Decomposition Rules:

If A holds BC and A holds B, then A holds C.
If $\{A \to BC\}$ and $\{A \to B\}$, then $\{A \to C\}$

Pseudo Transitivity:

If A holds B and BC holds D then AC holds D.
If $\{A \to B\}$ and $\{BC \to D\}$, then $\{AC \to D\}$ holds.

Trivial Functional dependency:

If A holds B $\{A \to B\}$, where A is a subset of B then it is called a trivial Functional dependency.
→ Trivial always holds Functional dependency.

Non - trivial Functional dependency:

If A holds B $\{A \to B\}$, where B is not a subset of A, then it is called as a Non - trivial Functional dependency.

Example of Functional dependency:

Project cost

ProjectID	project cost
001	1000
001	5000

EMPproject

EMPID	projectID	Days
E001	001	120
E002	002	180

The above relations states that:
Days are the number of days spent on the project.
So here EMPID, projectID, projectcost → Days
However, it is not fully functionally dependent. Whereas the subset {EMPID, projectID} can easily determine the {Days} spent on the project by the employees.
→ This summarizes and gives our fully functional dependency.
{EMPID, projectID} → Days.

closure of functional dependency:

The closure is essentially the full set of values that can be determined from a set of known values for a given relationship using its functional dependencies.
→ Given R and F a set of FDs that holds in R: The closure of F in R (denoted F+) is the set of all FDs that are logically implied by F.

closure of a set of Attributes:

closure of a set of Attributes X with respect to F is the set X+ of all attributes that are functionally determined by X using F+.

Example: consider the following list of FDs. We are going to calculate a closure for A From this Relationship.

1. A → B
2. B → C
3. AB → D

The closure would be as follows:
a) A → A, B → B, C → C (by reflexivity rule)
b) A → AB
c) A → B, B → C then A → C (transitivity rule)
d) AB → D then A → D and B → D holds (Decomposition rule)
e) A → BC (union rule)

Example : consider a relation R (A, B, C, D, E, F, G) with the functional dependencies :

$A \rightarrow BC$

$BC \rightarrow DE$

$D \rightarrow F$

$CF \rightarrow G$

Find the closure of some Attributes and attribute sets :

Ans : closure of Attribute A :

$A^+ = \{A\}$

$= \{A, B, C\}$ (using $A \rightarrow BC$)

$= \{A, B, C, D, E\}$ (using $BC \rightarrow DE$)

$= \{A, B, C, D, E, F\}$ (using $D \rightarrow F$)

$= \{A, B, C, D, E, F, G\}$ (using $CF \rightarrow G$)

Thus, $A^+ = \{A, B, C, D, E, F, G\}$

closure of Attribute D :

$D^+ = \{D\}$

$= \{D, F\}$ (using $D \rightarrow F$)

$D^+ = \{D, F\}$

closure of Attributes set $\{B, C\}$:

$\{B, C\}^+ = \{B, C\}$

$= \{B, C, D, E\}$ (using $BC \rightarrow DE$)

$= \{B, C, D, E, F\}$ (using $D \rightarrow F$)

$= \{B, C, D, E, F, G\}$ (using $CF \rightarrow G$)

Thus, $\{B, C\}^+ = \{B, C, D, E, F, G\}$

closure of Attribute set $\{C, F\}$

$\{C, F\}^+ = \{C, F, G\}$

using $(CF \rightarrow G)$

Thus $\{C, F\}^+ = \{C, F, G\}$

Normalization

Normalization is a process of organizing the data in database to avoid data redundancy, insertion anomaly, update anomaly and deletion anomaly.

Anomalies in DBMS:

There are three types of anomalies that occur when the database is not normalized. These are insertion, update and deletion anomaly.

→ An insert anomaly occurs when certain attributes cannot be inserted into the database without the presence of other attributes.

→ A deletion anomaly occurs when we delete a record that may contain attributes that should n't be deleted.

→ An updation anomaly is a data inconsistency that results from data redundancy and partial update.

Normal Forms: There are 4 most commonly used normal forms are there.

They are:
* First Normal Form (1NF)
* Second Normal Form (2NF)
* Third Normal Form (3NF)
* Boyce-codd Normal Form (BCNF)

First Normal Form (1NF):

As per the rule of first normal form, an attribute (column) of a table cannot hold multiple values. It should hold only atomic values.

Example Emp table

emp-id	emp-name	address	emp-mobile
101	XY	BGH	8994512334
102	AB	CTC	1234567890 3434567891
103	CD	SBP	2334512234
104	MN	BBSR	1234562290 3456781234

This table is not in INF as the rule says "each attribute of a table must have atomic values, but here the emp-mobile values for employees MN and AB violates the rule.

→ To make the table complies with INF we should have the data like this:

Table in INF:

emp-id	emp-name	address	emp-mobile
101	XY	BGH	8994512334
102	AB	CTC	1234567890
102	AB	CTC	3434567891
103	CD	SBP	2334512234
104	MN	BBSR	1234562990
104	MN	BBSR	3456781234

Second normal form: (2NF)

→ Second normal form is a normal form used in database normalization... A relation is in the second normal form if it fulfills the following two requirements: It is in first normal form.

→ It does not have any non-prime attribute that is functionally dependent on any proper subset of any candidate key of the relation.

→ To be in second normal form, a relation must be in first normal form and relation must not contain any partial dependency. A relation is in 2NF it has NO partial dependency.

Example

Consider the following table:

Stud NO	course-NO	course-Fee
1	C_1	1000
2	C_2	1500
1	C_4	2000
4	C_3	1000
4	C_1	1000
2	C_5	2000

Here, there are many courses having the same course fee.

Here, course-fee can't alone decide the value of course-NO or Stud-NO, course-fee together with Stud-NO can't decide the value of course-NO, course-fee together with course-NO can't decide the value of Stud-NO,

→ course-fee would be a non-prime attribute, as it does not belong to the one only candidate key { Stud-no, course-no };

→ But course-no → course-fee i.e. course-fee is dependent on course-no, which is a proper subset of the candidate key.

→ Non-prime attribute course-fee is dependent on a proper subset of the candidate key, which is a partial dependency and so this relation is not in 2NF.

→ To convert the above relation to 2NF. we need to split the above table into two tables such as:

Table 1 : Stud-no, course-no

Table 2 : course-no, course-fee

Table 1

Stud-no	course-no
1	C_1
2	C_2
1	C_4
4	C_3
4	C_1
2	C_5

Table 2

course-no	course-fee
C_1	1000
C_2	1500
C_3	1000
C_4	2000
C_5	2000

Third Normal Form (3NF)

→ Although 2NF relations have less redundancy than those in 1NF, they may still suffer from update anomalies.

→ If we update only one tuple and not the other, the database would be in an inconsistent state.

→ A relation is in third normal form, if there is no transitive dependency for non-prime attributes as well as it is in second normal form.

→ A relation is in 3NF if atleast one of the following condition holds in every non-trivial function dependency $X \rightarrow Y$:

1. X is a super key.
2. Y is a prime attribute.

→ In other words, A relation that is in first and second normal form and in which no non-primary-key attribute is transitively dependent on the primary key, then it is in 3NF.

→ If $A \rightarrow B$ and $B \rightarrow C$ are two FDs, then $A \rightarrow C$ is called transitive dependency.

Example

Stud-no	Stud-name	Stud-State	Stud-country	Age
1	XY	Odisha	INDIA	20
2	XY	punjab	India	19
3	YZ	Manipur	India	21

Functional dependency set:

Stud-no → Stud-name, Stud-no → Stud-state,
Stud-state → Stud-country,
Stud-no → Stud-age

Here candidate key : Stud-no

For this relation Stud-no → Stud-state and Stud-state → Stud-country are true. So Stud-country is transitively dependent on Stud-no.

→ It violates the third normal form. To convert it in third normal form, we will decompose the relation Student (Stud-no, Stud-name, Stud-phone, Stud-state, Stud-country, Stud-age)

Student (Stud-no, Stud-name, Stud-phone, Stud-state, Stud-age)

state-country (State, country)

Boyce-Codd Normal Form (BCNF)

Boyce-codd normal form (BCNF) is based on functional dependencies that take into account all candidate keys in a relation, however, BCNF also has additional constraints compared with general definition of 3NF.

→ A relation is in BCNF, iff, every determinant is a form (BCNF) candidate key.

Example (Emplouee table)

Emp-id	emp-country	emp-dept	Dept-type	dept-no
001	India	designing	D001	203
001	India	testing	D001	204
002	UK	Stores	D002	232
002	UK	developing	D002	233

In the above table Functional dependencies are as follows:

emp-id → emp-country

emp-dept → {dept-type, dept-no}

candidate key : { emp-id , emp-dept }

The table is not in BCNF because neither emp-dept nor emp-id alone are keys. To convert the given table into BCNF, we decompose it into three tables.

Emp_country table

Emp-id	Emp-country
001	India
001	India

Emp-dept table

Emp-dept	dept-type	Dept-no
Designing	D001	203
testing	D001	204
Storeg	D002	232
developing	D002	233

Emp-dept-Mapping table

EMP-id	dept-no
001	203
001	204
002	232
002	233

Functional dependencies

emp-id → emp-country

emp-dept → { dept-type ; dept-no }

candidate keys :

For the first table : emp-id

For the second table : emp-dept

For the third table : { emp-id , emp-dept }

Now, this is in BCNF because left side part of both the functional dependencies is a key.