17th July

UNIT-1

Introduction

Data :- Data may be considered as known packs which can be recorded and have an implicit meaning.

Database :- It is a collection of interrelated data, these data can be stored in terms of tables.

- A database can be of any size and varying complexity.
- A database may be generated and manipulated manually or it may be computerised.

Ex

s_id	s_name	s_contact no

A database is an organised collection of interrelated data so that it can be easily accessed, managed and updated.

Database Management Systems (DBMS) :- It is a set of programs used to define, administer and process databases and their associated applications.

Limitations of file Processing Systems :-

(i) **Data Redundancy :-** A single data has been used more than 1 time (primary key) Duplicacy.

(ii) Data Inconsistency :- It does not provide the updating feature from one table to another (foreign keys)

(iii) Unsharable Data :- Cannot share data and not able to access data.

(iv) Unstandardised Data :-

(v) Insecure Data :- Not providing backups.

Advantages of DBMS :-

(i) Reduce data duplicacy :- By using primary key concept we can reduce data redundancy in case of DBMS.

(ii) Control Data Inconsistence :- A centrally controlled data using foreign key.

(iii) Facilitate sharing of Data :- Same data can be shared among various applications and can be accessed by many users.

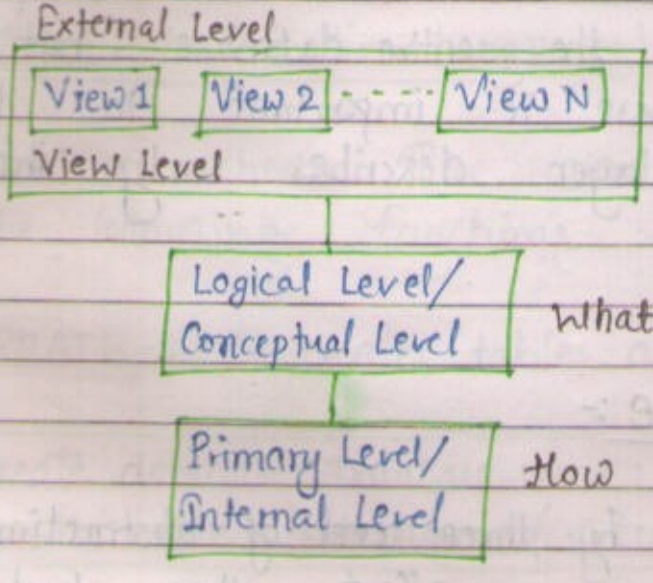
(iv) Encourse Standards (Validations of email & phone no.)

(v) Ensure data security : It prefers to protection of data against accidental or intensional damaged or access to unauthorised persons or modification

(vi) Maintains data integrity :- Database is consistent among many files.

18th July

Three Level Architecture / View Abstraction Level :-



The major purpose of database system is to provide users with an abstract view of data. Therefore, the database system hides details of how data is created, stored and maintained.

Primary / Physical / Internal Level :- The lowest level of abstraction describes how the data are actually stored. The physical level describes the complex low level data structure in detail.

Logical / Conceptual Level :- It describes what data are stored in the database and what relationship exists among these data. The logical level thus describe the entire database in terms of small number of relative simple structures.

This level also describes the constraints.

View / External Level :- It describes how users visualize the data. (GUI). It deals with the views of various users. It is also the closest layer to the user. In most cases users are not concerned with the entire database but with the part of database ie important to them. It means that this layer describes only the part of entire database.

Data Independence :-

Database is visualized by three level of abstraction. Any change at any level may affect other level of abstraction. Basically two types of data independencies are maintained in database :-

(i) Logical Data Independence :- It protects from changes in logical structure of data without changing external schema. Logical Data Independence is meant to provide insulation between external schema and conceptual schema.
Ex Addition of Row, Abs Subtraction or changes.

(ii) Physical Data Independence :- It can modify the internal schema without having to change logical schema. It means application program remains same even though the schema at physical level gets modified.

19th July

Database Languages :-

Database language is a language that permits a us to query and manipulate data on database system.

(i) DDL (Data Definition Languages) :- The DDL statements are used to define database schema. These DDL statements provide the following functions :-

(a) CREATE :- It creates tables and database.

Syntax :-
```
Create database Database-name;
Create table Table_name (values, ..... );
```

(b) Alter :- for alteration of table structure.

◉ To add a column to existing table

Syntax :-
```
alter table Table_name add (values);
```
Ex alter table Student add (s.add char);

◉ To rename any existing column

Ex
```
alter table Table_Name rename column sadd to location
```

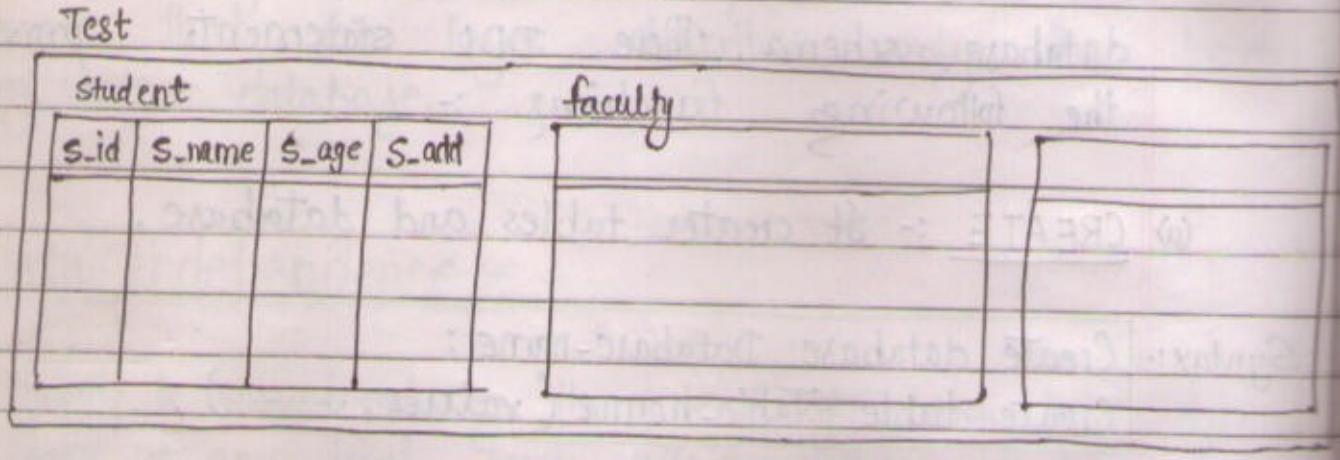
◉ To change datatype of any column or to modify its size.

Syntax :-
```
alter table Table-name modify (values datatype(size));
```

alter table Student modify (s_add varchar(50));

◉ Alter is also used to drop a column.

drop
alter table Student ~~delete~~ (s_add);



(c) Truncate :- It removes all records from a table but this command will not destroy the structure of the table. Truncate command actually reinitialize the table.

Syntax :- ~~alter table~~ truncate table Student ;

(d) DROP :- It will delete the table of student, It completely removes a table from database and will also destroy the structure of the table.

Syntax : drop table Student ;

(e) RENAME :- Rename the table name or modify the table name.

Syntax:- rename table Student to Student_record.

25/7/19

(iii) Data Manipulation Language (DML) :-

DML commands are used to query and manipulate existing objects like tables, procedures etc.
They are of two types :-

(i) Procedural DML :- User specifies what data is required and how to get those data.
Ex PL/SQL

(ii) Nonprocedural DML :- User specifies what data is required without specifying how to get these data.
DML commands perform following functions :-

Select :-

```
Select * from Student;
Select * from Student where s_phone = 789;
Select s_name from Student where s_add = 'Varanasi';
```

* → Select all the information

Insert :- Its syntax is insert into table_name <[column_name1], [column_name2] ... ... > values (list of values);

Ex

insert into Student <[s_id], [s_name], [s_age], [s_add]>
values (1, 'Nikita', 20, 'Varanasi');

OR

insert into Student values (1, 'Nikita', 20, 'Varanasi');

Inserting a null value to a Column.

Syntax: Insert into Student values (1, 'Nikita', null, 'Varanasi');

OR

Insert into Student values (sid, s_name, s_add) values (1, 'Nikita', 'Varanasi');

Update :- It modifies the data.

Syntax: update Student set s_add = 'Bareilly' where s_name = 'Nikita';

update Student set s_phn = '123', s_course = 'CDF' where s_name = 'A';

Delete :- It is used from deletion of record from database

Syntax: Delete from table_name where condition ;
ie Delete from Student where s_name = 'Nikita';

26/07/19

(iii) TCL (Transmission Control Language) :-

It is used to manage changes made by DML. It allows statements to be grouped together into logical transactions.

Commit :- It saves work done. It is used to make the change permanent. It also erases used savepoints in the transaction.

Syntax :- Commit work ;

Savepoint :- It identify a point in transaction to which you can later rollback. We must use distinct name of savepoints.
In case of similar savepoints the previous ones will be erased.

Syntax :- Savepoint savepoint-name ;

Rollback :- It is used to get the deleted data back which is saved by the savepoint.

Syntax :- Rollback to savepoint-name ;

(iv) **DCL (Data Control Languages)** :-

To control user access in a database we used DCL commands. It is related to security matters issues.

Grant :- It gives user's access priviledge to database. Basically seven types of priviledges can be grant by a normal user to another :-

alter, delete, index, insert, reference, select, update.

Syntax :- Grant <priviledge / ALL> on database-name to <*user> [with grant option];

Ex

Grant insert, select on Student to A with grant option;

With grant option :- This keyword conveys the privilege or role to a user with the right to grant same privilege to another user.

Revoke :- It allows to restrict the user from accessing data in database.

Syntax :- Revoke <privilege / ALL> ON database-name To <user> [cascade]

Grant select on Student to A with grant option;
Grant select on student to B;
Revoke select on Student to A cascade;

Cascade :- The owner of table (User A) can grant privilege to another user (User B). In the statements above, we revoke the select privilege from user 'A' but it doesn't revoked those privileges from user 'B'. after using the keyword 'cascade' at the time of revoking privileges from User 'A'.

30/7/19

Database Users and Administrators :-

A primary goal of database system is to retrieve information from and store new information into database.

Database User and Interface :- There are four types of database system user's differentiated by the way they expect to interact with the system.

(i) Naive Users :- They are unsophisticated users who interact with the system by invoking one of the application program that has been written previously

(ii) Application Programmers :- They are computer professionals who write application programs and interact with system to DML (Data manipulation Language) calls / commands.

(iii) Sophisticated Users :- They directly interact with the system without writing programs. Instead they form request in a database query language.

(iv) Specialised Users :- They are sophisticated users who write specialised database (like images, graphs, videos etc) application that do not fit ~~quite~~ into the traditional database framework.

Database Administrator (DBA) :- One of the main reason for using database management system is to have central control of both the data & the program that access these data. A person who has such central control over the system is called DBA.

The functions of DBA are :-

(a) Schema Definition :- DBA creates original database schema by executing a set of data definition statements in the DDL (Data Definition Language)

(b) Storage Structure and Access Method Definition :-

(c) Schema and Physical Organisation Modification :- DBA carries out changes to the schema and physical organisation to reflect the changing needs of Organisation or to alter the Physical Organisation to improve performance.

(d) Alter the Physical Organisation to improve performance.
Granting of Authorisation for data access.

(e) By granting different type of authorisation the

(f) DBA can regulate which part of database various users can access.

(f) Routine Maintenance :-

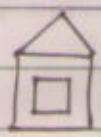
- Periodically backaned up of database.
- Ensuring that ~~ena~~ ~~in~~ enough free disk space is available for normal operations and upgrading disk space as required.
- Monitoring jobs running on the database and ensuring that performance is not degraded by expensive task submitted by some users.

31/07/19

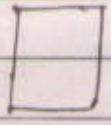
DATA MODEL :-

A collections of tools for describing - data, data relationships, data semantics, data constraints.

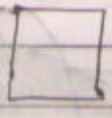
(i) Relational Model
(ii) ER Data Model
(iii) Object Based Data Model
(iv) Semi Structured Data Model (XML)
(v) Network Model
(vi) Hierarchical Model



Conceptual View — ER - Representation



Representation and Implementation — Tables / Relational Model



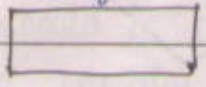
Logical Level or Physical Data Model — Using SQL

Entity Relationship Data Model :-

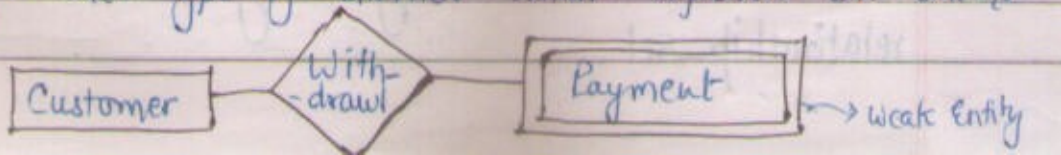
It models and enterprise as a collection of entities and relationships among them.

Entity (Noun) :- A thing or object in the enterprise that is distinguishable from other objects.
Described by a set of attributes.

- Strong Entity :- Such type of entities doesn't depend on others.



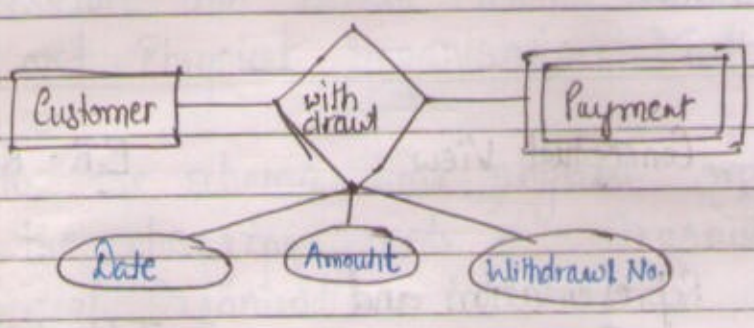
- Weak Entity :- Such type of entities which depends on others.

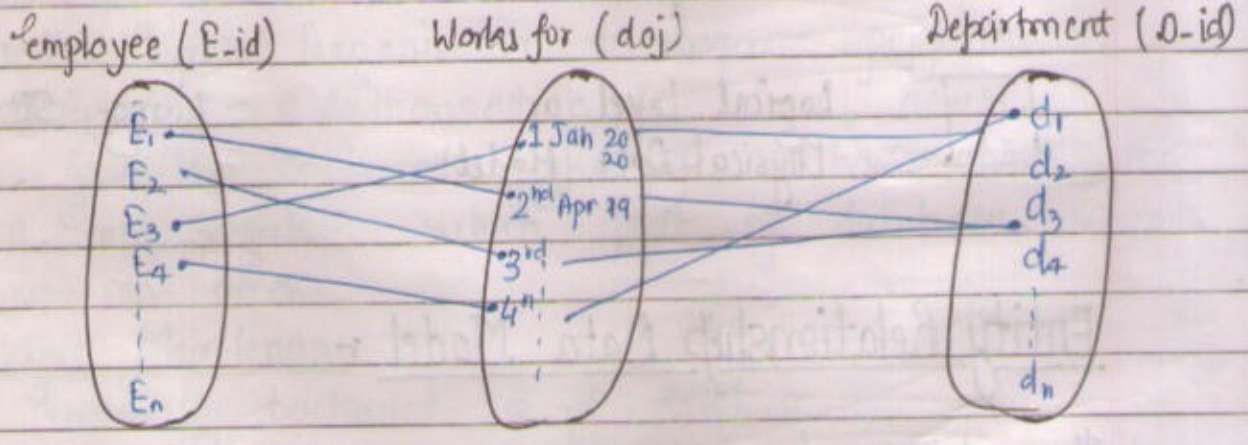


Entity Set :- An entity set is a set of entities of same type that shares same property or attributes.
Ex student, customer etc.

Relationship (Verb) :- An association among several entities.

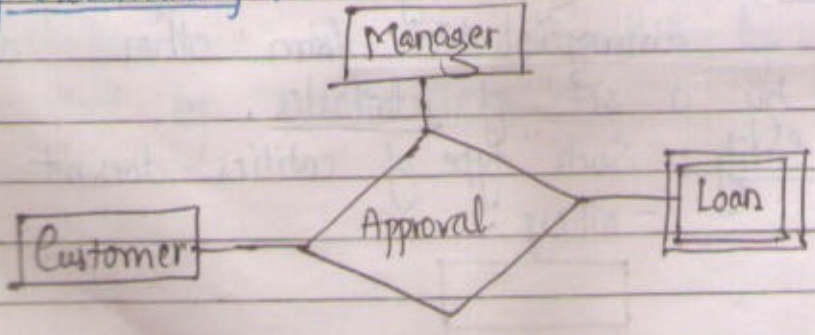
- **Relationship Attribute** :-





An attribute can also be a property of relationship set.

- **Degree of Relationship** :-

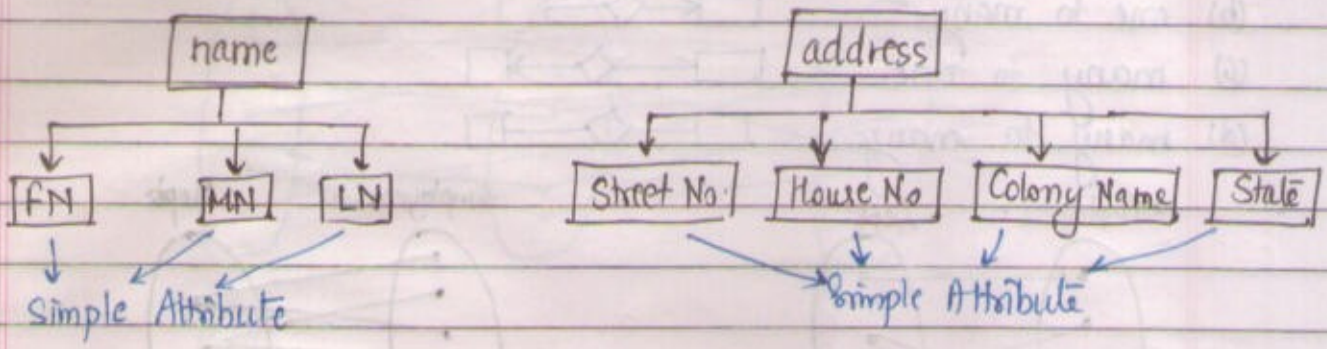


It refers to the number of entity sets that participate in a relationship set.

Attribute :- An entity is represented by a set of attributes that is descriptive properties possessed by all members of an entity set.

- Simple and Composite Attribute :- Collection of simple attributes is known as simple attributes.

Ex name, address etc.



- Single Valued and Multi Valued Attributes :-

↳ (multivalued is represented by double eclipse)

Ex roll no, age, etc (single valued)
phone_no, address, name, etc (multi valued)

- Derived attribute and Stored Attribute :- Derived : It can be computed from other attributes

Ex :- ~~dob~~, age given dob (derived attribute)
Ex :- dob (stored attribute)

↳ → Derive attribute (Dash eclipse)

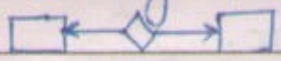
- Composite Complex Attribute :- Collection of composite & multi-valued attributes.

Ex address
↳ There should be more than two addresses (Multi Valued)
↳ Address contains more than one attributes. (Composite)

1/8/19

Cardinality Constraints :-

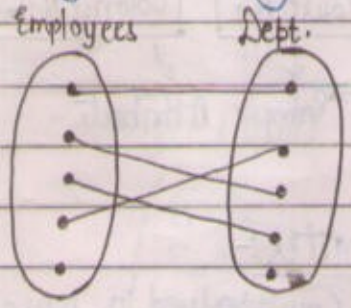
- Maximum number of relationships an entity can participate
- It express the number of entities to which another entity can be associated via a relationship set.
- Cardinality must be one of following type :-

(a) one to one 

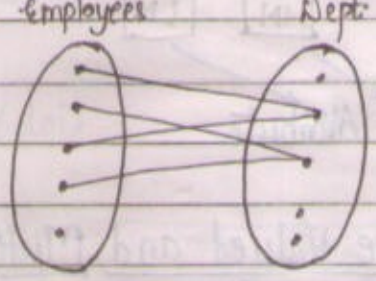
(b) one to many - 

(c) many to one 

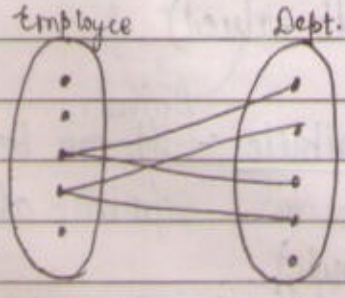
(d) many to many 



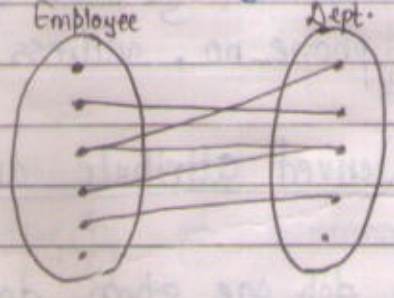
One to one



~~One to~~ many to One



~~Many to~~ one to many



Many to many

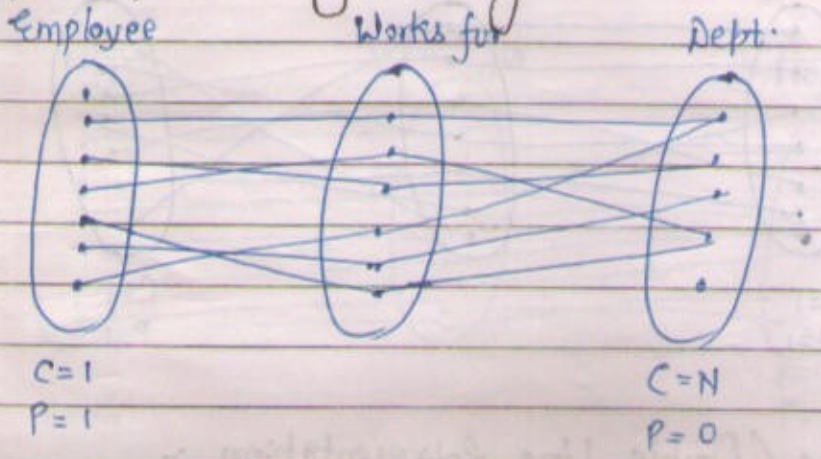
Structural Constraints / Participation / Existence :- Minimum number of relationships an entity can participate

There are following types of participation :-

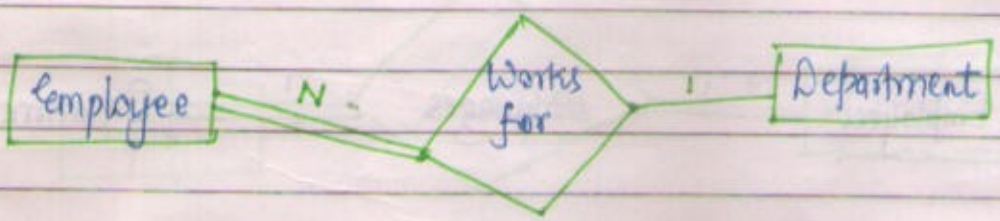
Total Participation :- If every entity in an entity set E participates in atleast one relationship in 'R'. (═══)

Partial Participation :- If only some entities in entity set 'E' participates in 'R'. (———)

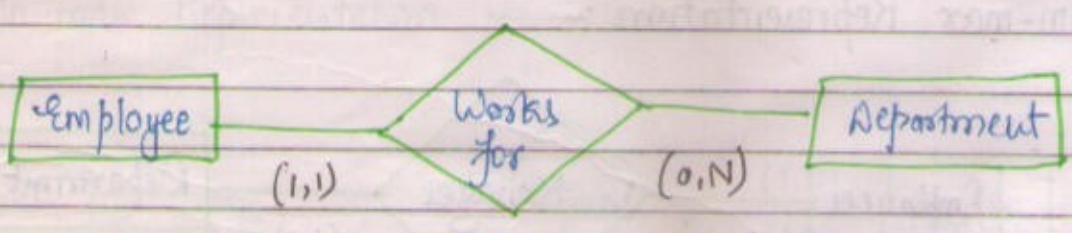
Que Assume a requirement anlalysis that every employee works for exactly one department and a department can have many employees. New department need not have any employee. Construct ER diagram for it & find the degree of relationship. Also calculate cardinality and participation of entity set.



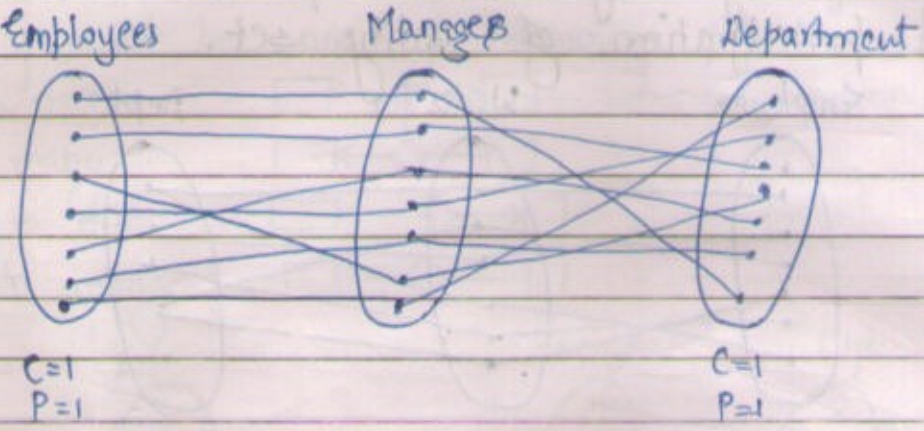
Single Line / Double Line Representation :-



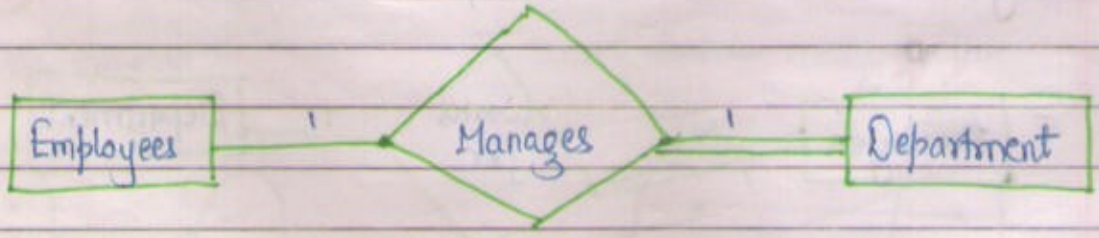
Min-Max Representation :-



Que Every dept. should have a manager and only one employee manages a dept. also an employee can manage one dept. find the degree of relationship and cardinality & participation of entity set.

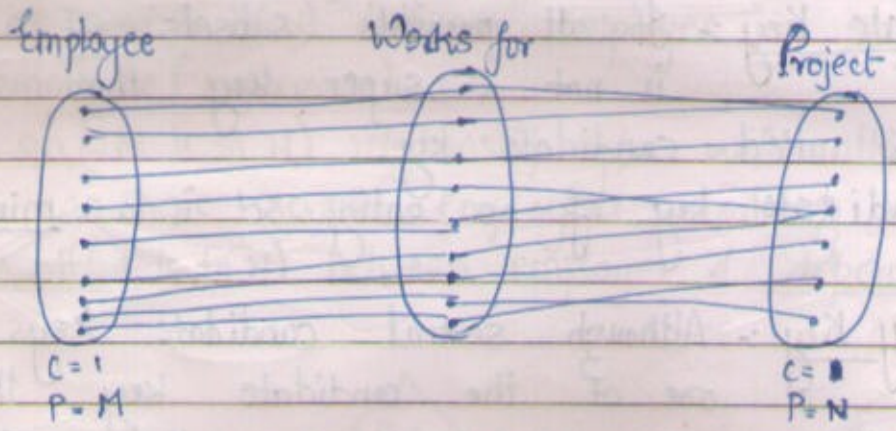


Single Line / Double Line Representation :-

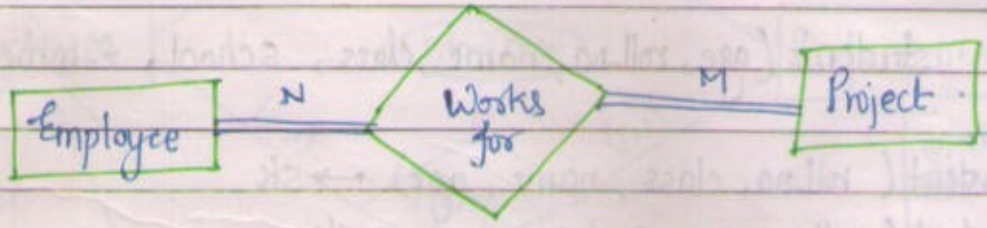


Min-max Representation :-

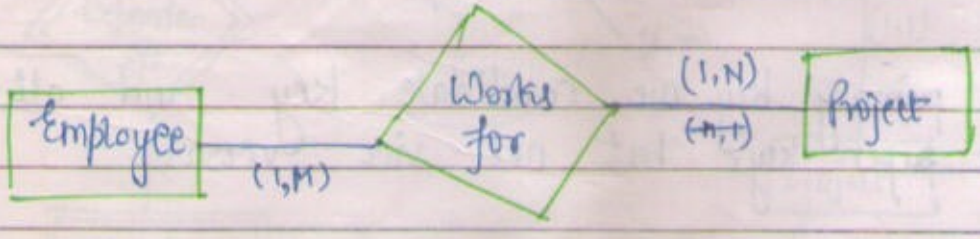
Que Every employee is suppose to work on atleast one project and every project is suppose to have many employees. A project is suppose to have atleast one employee. find the degree of relationship and cardinality, participation of entity set.



Single Line / Double Line Representation.



Min-max Representation ⇒



2/8/19

Keys :-

(i) **Super Key** :- A super key of an entity set is a set of one or more attributes whose values uniquely determined each entity.

(ii) **Candidate Key** :- If all possible subsets of a superkey is not a super key then the super key is called candidate key.
A candidate key of an entity set is a minimal superkey.

(iii) **Primary Key** :- Although several candidate keys may exist, one of the candidate key that be implement on our database is called primary key. and others are called alternate key.

Ex student (age, roll-no, name, class, school, f-name, add)

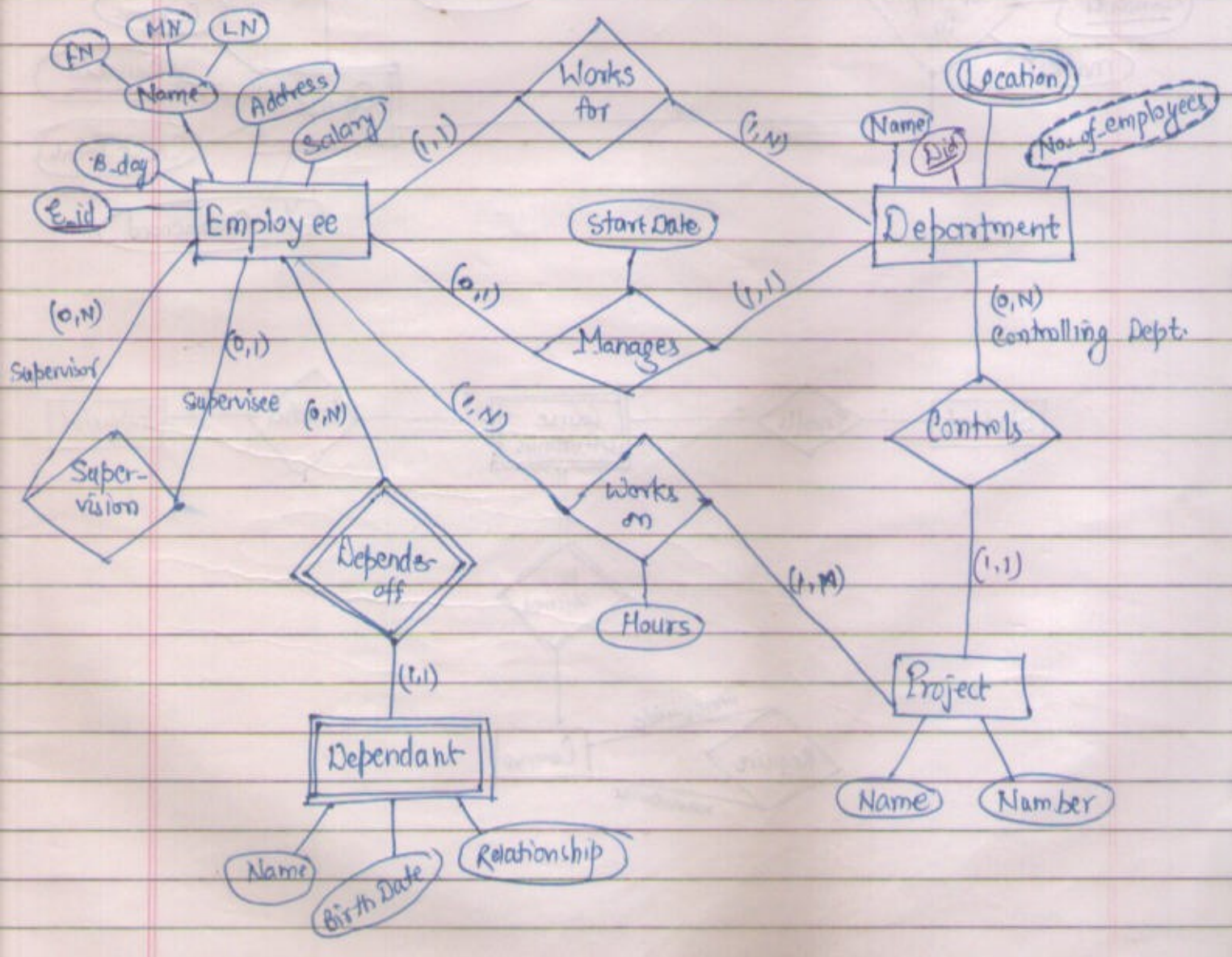
student (roll-no, class, name, age) → sk
student (roll-no, name class, age) → sk
student (roll-no, class) → sk, ck → pk
student (class, add) → ck → ak

NOTE :- All primary key are candidate key and all condidate keys are super key but not vice versa.

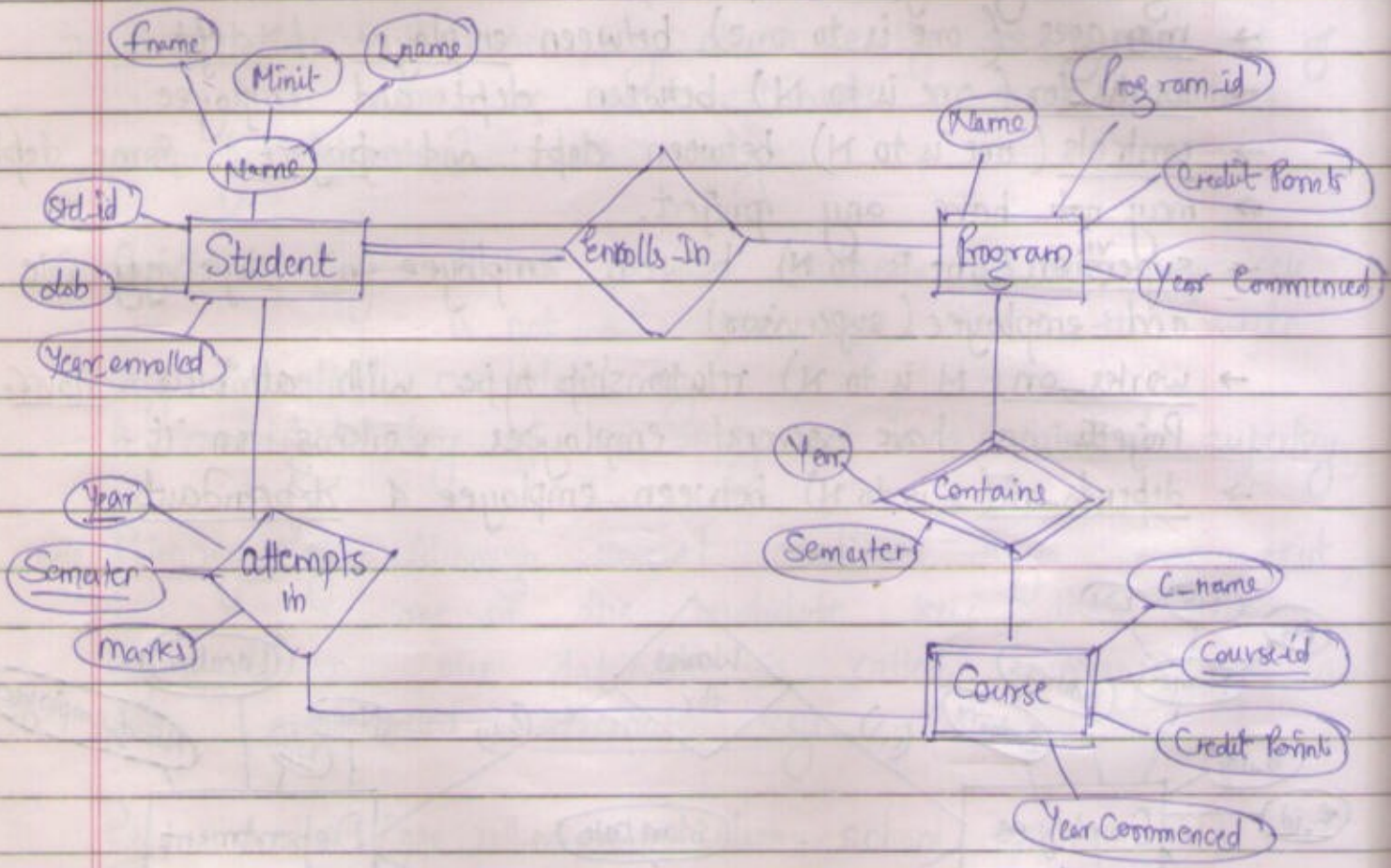
* A good ER Design does not contain redundant attributes.

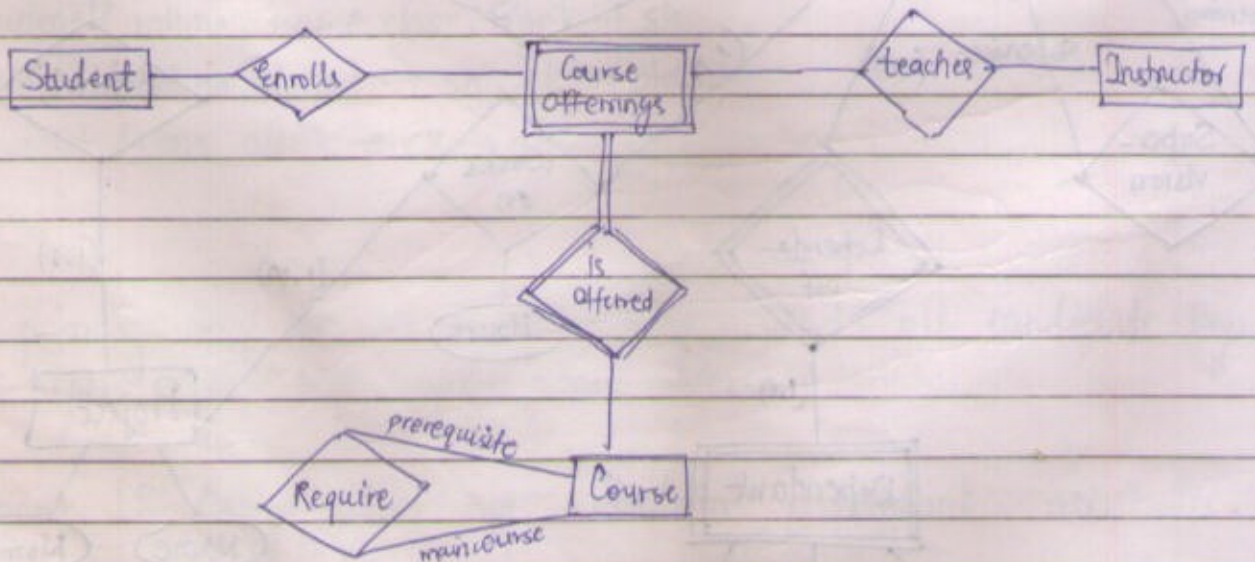
Que Design an ER diagram for company database with following entity types relationships :-

→ manages (one is to one) between employee & dept.
→ works_for (one is to N) between dept. and employee.
→ controls (one is to N) between dept. and project. Some depts. may not have any project.
→ supervision (one is to N) between employee in supervisor role and employee (supervisor)
→ works_on (M is to N) relationship type with attribute hours. Projects can have several employees working on it.
→ depends_off (1 is to N) between employee & dependant.



Q) Draw ER diagram for University Database Schema.

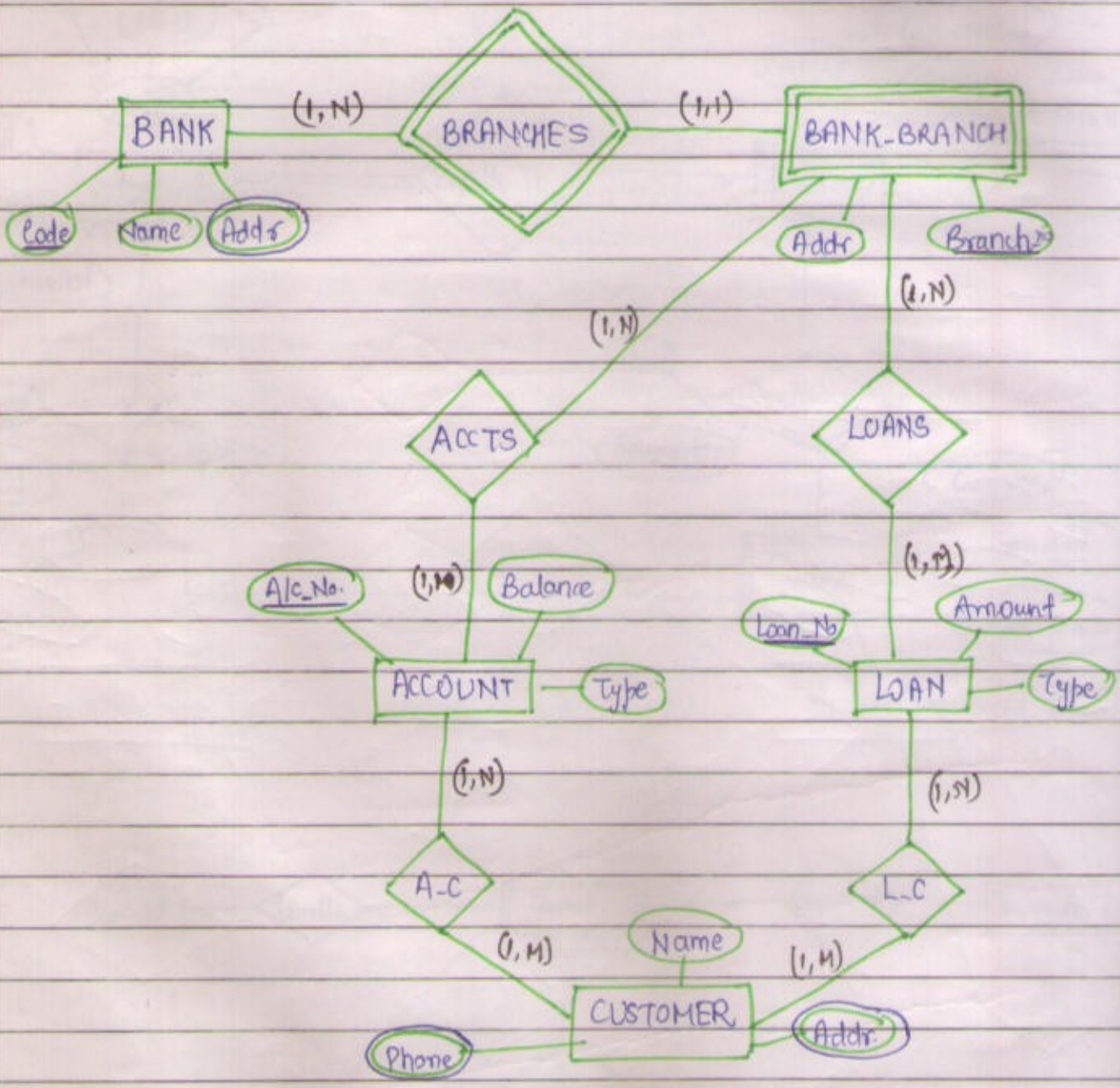




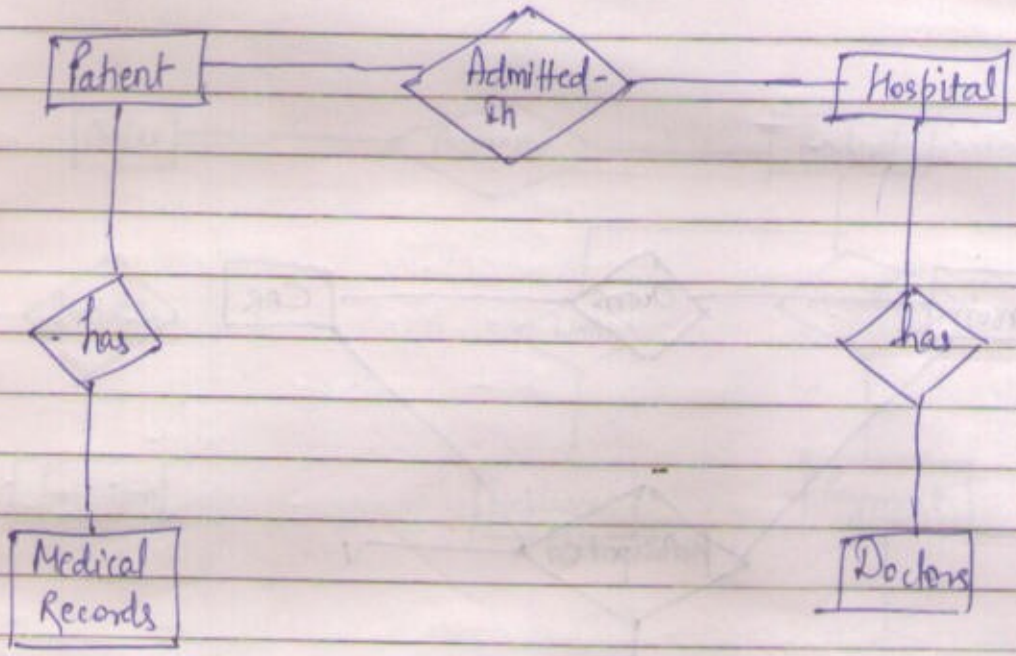
Date :
Page :

(2) Draw ER diagram for Airline Database Schema.

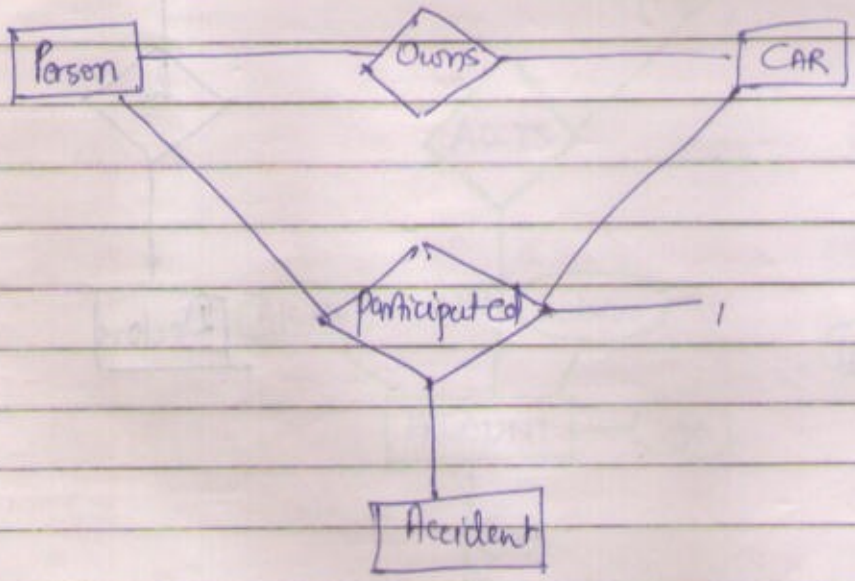
③ Draw ER diagram for Bank Database Schema.



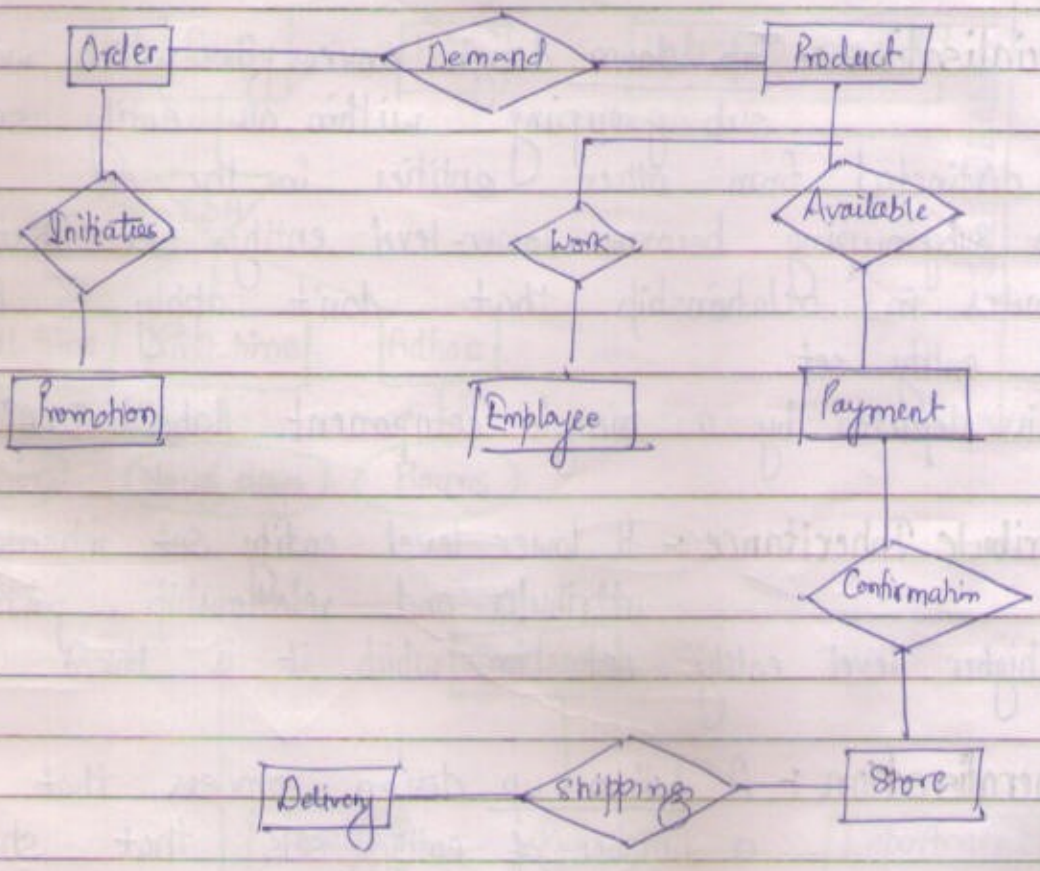
Q) Draw ER diagram for Hospital Management System.



5. Draw ER diagram for Car Insurance Company with a set of Customers - Each which owns a number of cars. Each car has a number recorded accidents associated with it.



6. Draw ER diagram for complete process for Selling the Product to the Customers which consists of salesman, customer, order, item, warehouse, etc.



7/08/19

Enhanced ER Model / Extended ER Model :-

The Extended ER Model includes the following 3 additional concepts:

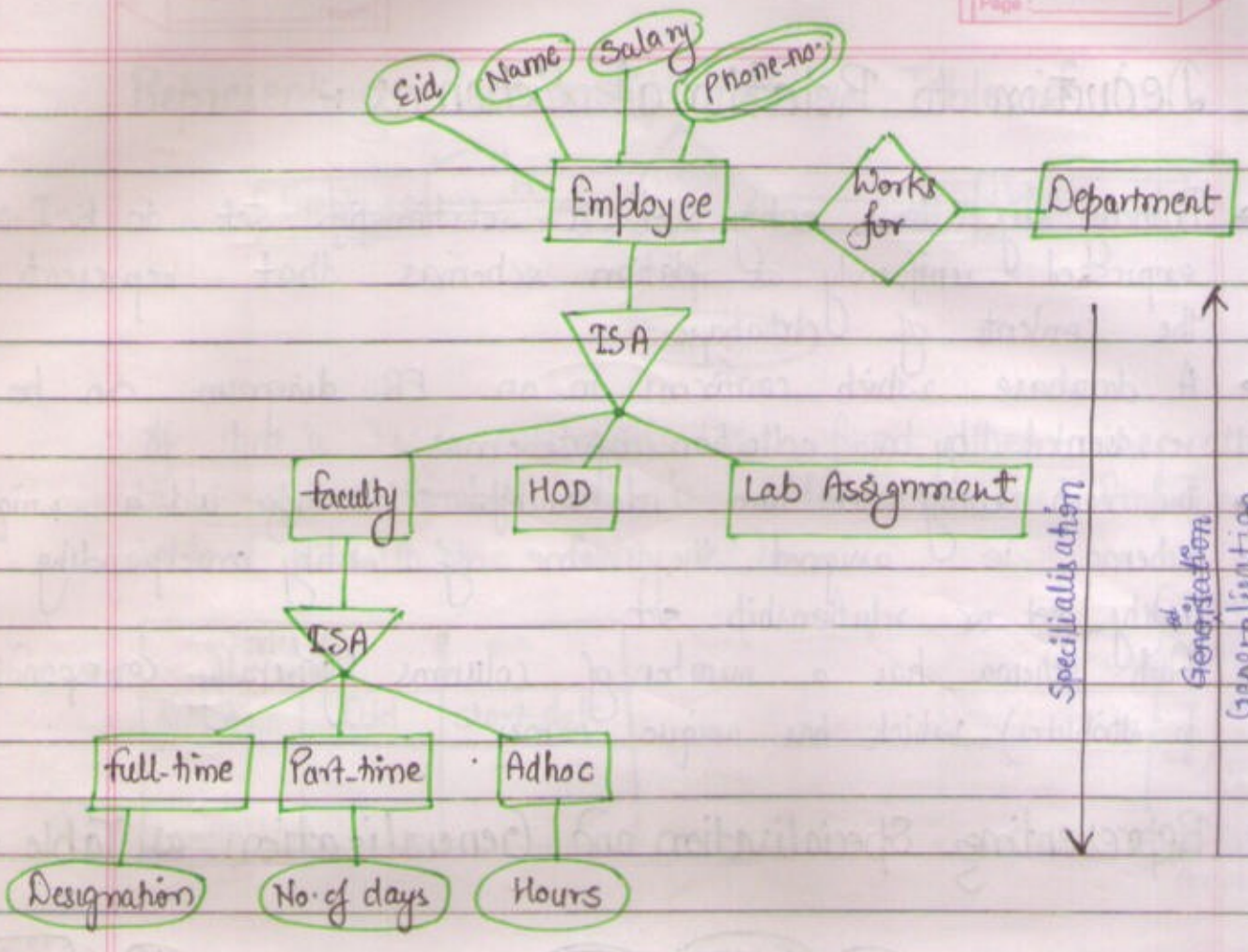
→ Specialisation
→ Generalisation
→ Aggregation

Specialisation :- Top down design process in which we designate sub grouping within an entity set that are distincted from other entities in the set.
These subgrouping become lower-level entity set that have attributes in relationship that don't apply to higher-level entity set.
It is depicted by a triangle component labelled 'ISA'.

- **Attribute Inheritance :-** A lower level entity set inherits all the attributes and relationship, participation of higher level entity set to which it is linked.

Generalisation :- A bottom up design process that combine a number of entity sets that shares the same features into a higher-level entity sets.
Specialisation and generalisation are simple inversions of each other.

Aggregation :- Aggregation is a form of Binary Association that specifies a whole part of relationship between an aggregate (whole-part) and the constituent part.
3 3



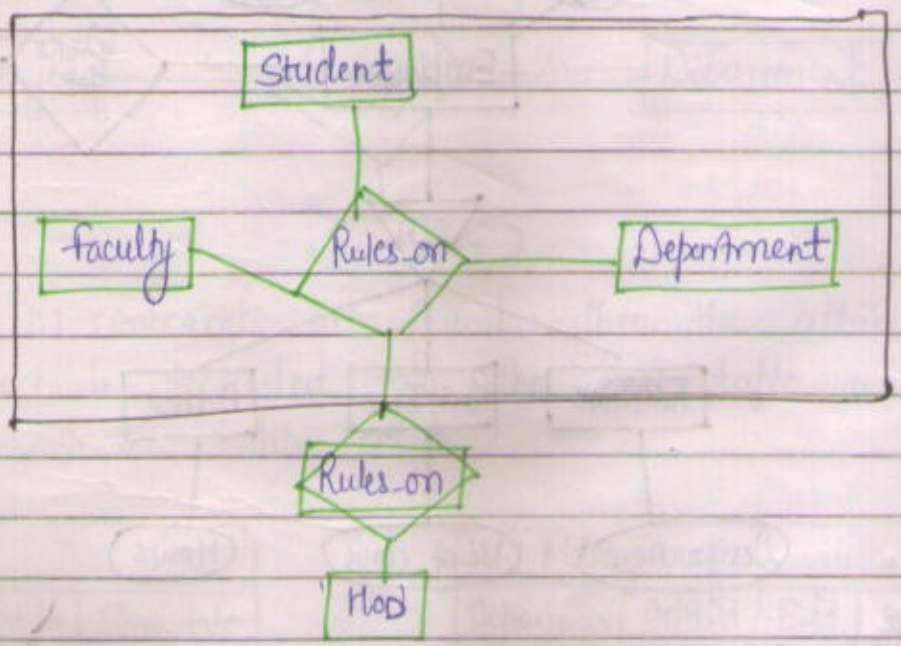


Fig 3 :- Aggregation

Deduction to Relational Schemas :-

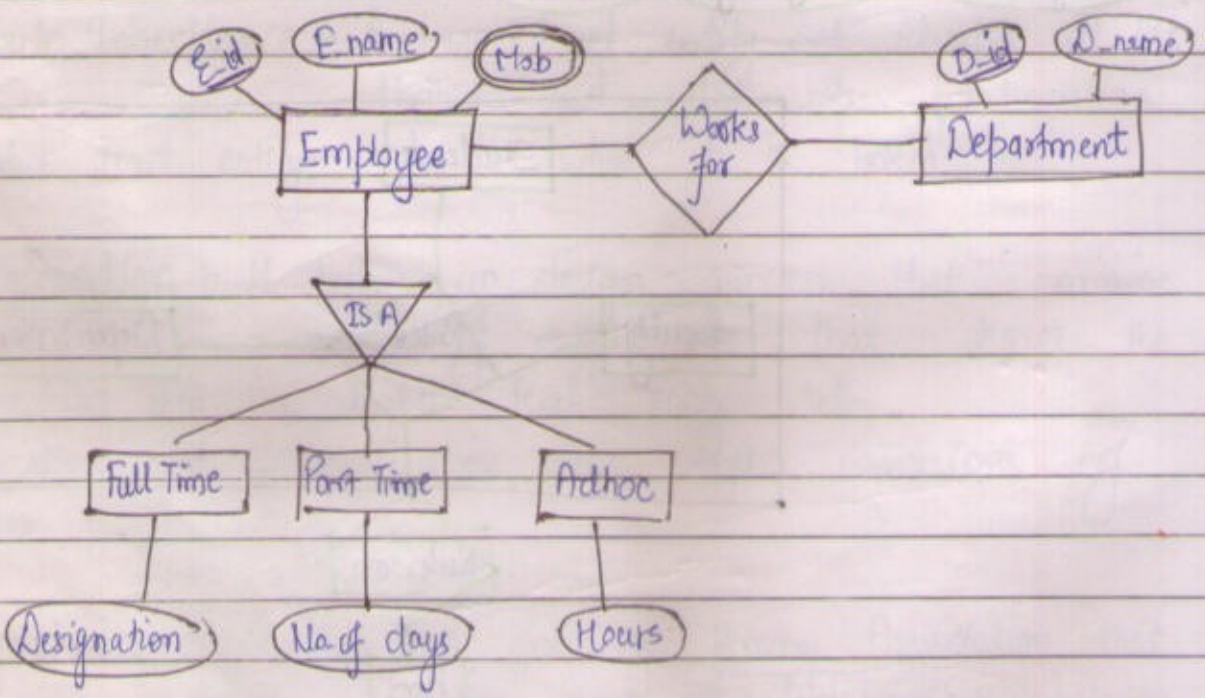
→ Primary key allows entity and a relationship set to be expressed uniformly as relation schemas that represents the contents of database.

→ A database which confirms to an ER diagram can be represented by a collection of schemas.

→ For each entity set and relationship set there is a unique schema ie assigned the name of entity corresponding entity set or relationship set.

→ Each schema has a number of columns (generally corresponding to attributes) which has unique names.

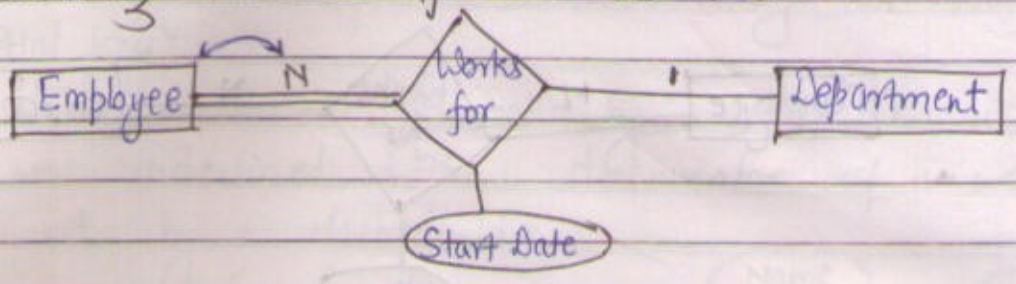
Representing Specialisation and Generalisation as Table :-



Full Time [Designation, E_id, Ename]
Part Time [No of days, E_id, Ename]
Adhoc [Hours, E_id, E_name]
Mob No. [E_id, number]
Department [D_id, D_name]

Representing Relationship Set as Table :-

Case. I :-



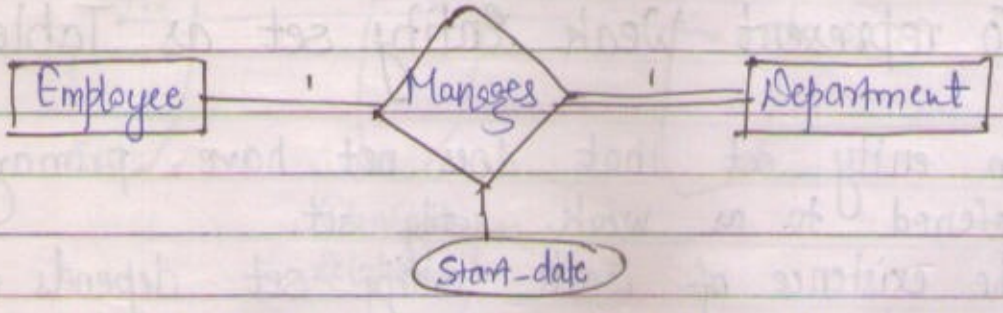
If there is N:1 constraints are found then the attribute table of relationship set is made into the Employee table (ie with the Entity set)

Works_for		
full_Time	Did	start_date

→ Full Time
→ Part Time
→ Adhoc
→ Department

⇒ 4 Table formed

Case. II



If there is 1:1 constraints are found then the attribute of relationship set is included into the Dept. table or Employee table (ie. with the Entity set)

Manages		
Employee_id	Dept_id	Start_date

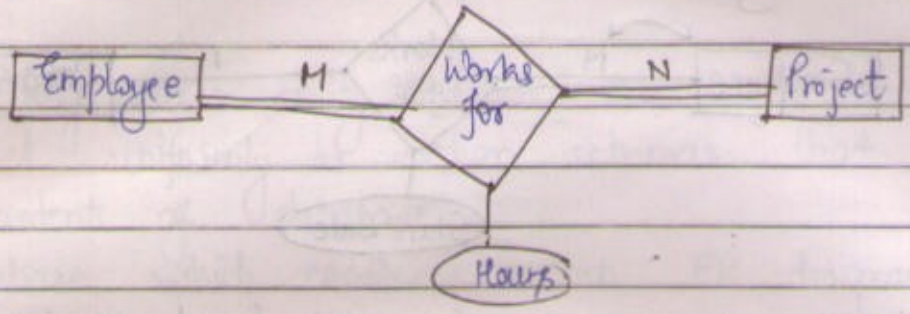
OR

Manages			
Department	Dept_id	E_id	Start_date

→ Full Time
→ Part Time
→ Adhoc
→ Department

→ Total 4 table formed.

Case: III



If there is M:N ratio is found then the table of the attribute of the Relationship is formed separately. (multivalued)

Manages		
E_id	P_id	Hours

ie
- → Full Time
- → Part Time
- → Adhoc
- → Department
- → Hours (Manager)

Total 5 tables are formed

10/08/19

To represent Weak Entity set as Table :-

→ An enity set that does not have primary key is refered to as weak entity set.

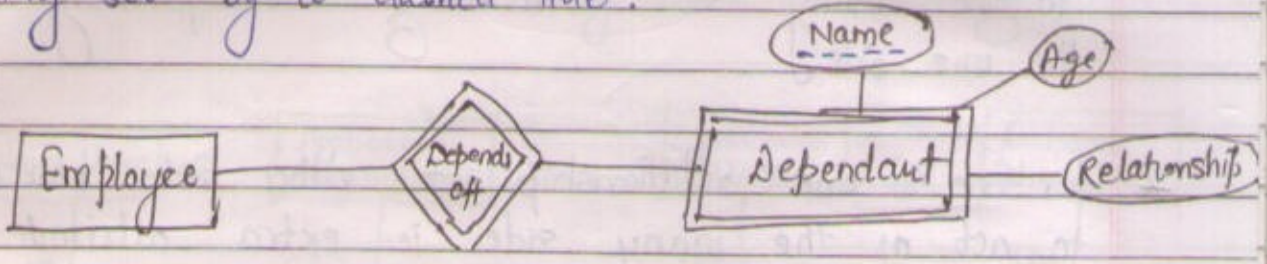
→ The existence of weak entity set depends on existence of identifying entity set.

- It must relate to the identifying entity set via a total, one to many relationship set from the identifying to the weak entity set
- Identifying relationship depicted using double diamond.
- The descriminator (or partial key) of a weak entity set is the set of attributes that distinguishes among all the entities of a weak entity set.
- The primary key of a weak entity set is formed by primary key of strong entity set on which the weak entity set's

existence dependant plus the weak entity set Descriminator (partial key).

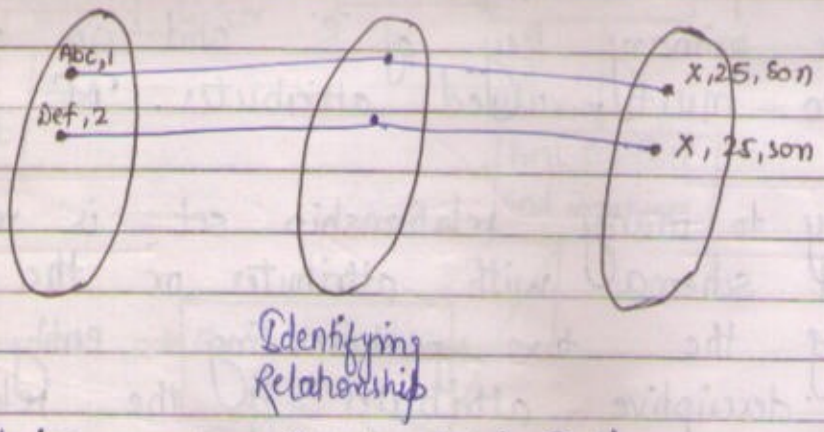
→ We depict weak entity set by a double rectangle and we underlined the descrimator of a weak entity set by a dashed line.

Ex



→ Weak entity set always have total participation but not every total participation relationship is weak entity set.

Ex



Dependent (Name, Age, Relationship, E-id)

Dependant			
Name	Age	Relationship	E-id

Many to One and One to Many

* Relationship sets that are total on the many side can be represented by adding an extra attribute to the many side containing the primary key of the one side.

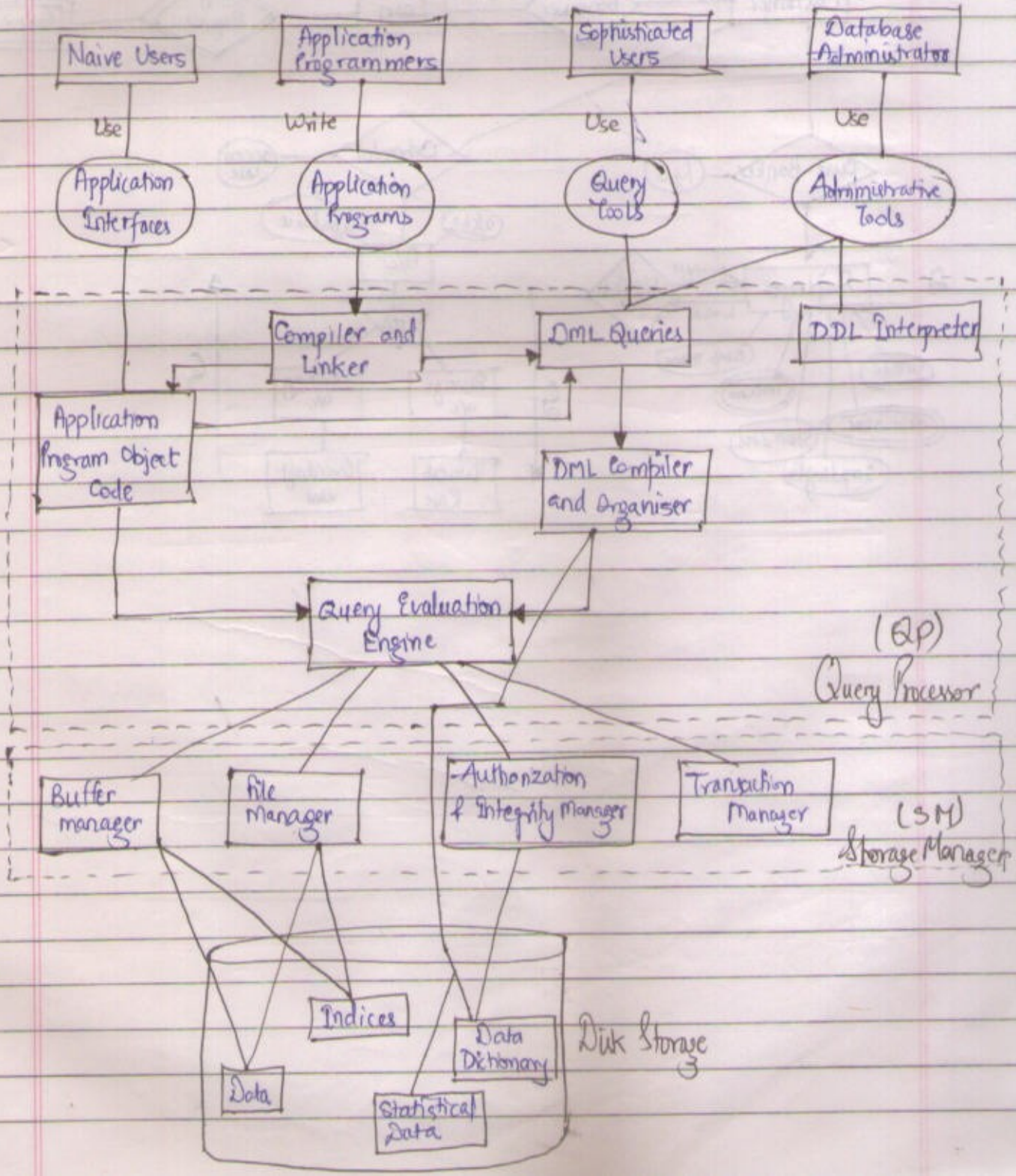
* for one to one relationship set either side can be choosen to act as the many side ie. extra attribute can be added to either of the table corresponding to the entity sets.

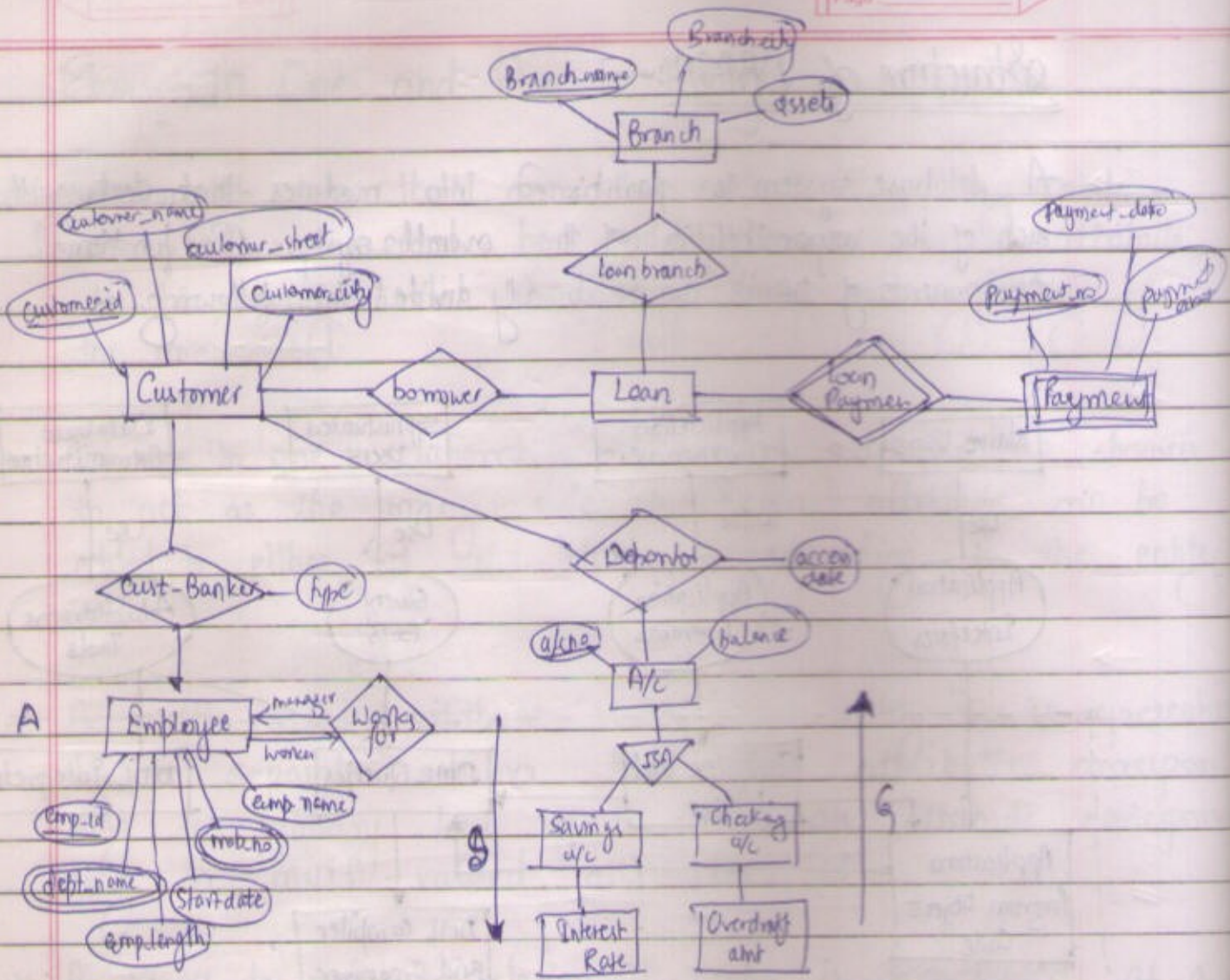
* A multi valued attribute 'M' of an entity 'E' is represented by a separate schema that has attributes corresponding to primary key of 'E' and an attribute corresponding to multi-valued attributes. 'M'.

* A many to many relationship set is represented as a different schema with attributes or the primary keys of the two participating entity sets and any desciptive attributes of the relationship set.

Structure of DBMS :-

A database system is partitioned into modules that deals with each of the responsibilities of the overall system. The functional components of DBMS can be broadly divided into following :-




Branch_city
Branch_name
assets
Branch
loan branch
Payment_date
Payment_no
payment_amt
customer_name
customer_street
Customer_id
Customer_city
Customer
borrower
Loan
Loan Payment
Payment
Depositor
account_no
Cust-Banker
type
balance
a/c no
A/C
A
Employee
manager
works for
worker
ISA
emp_name
Emp_id
mob_no
dept_name
Start_date
emp_length
Savings a/c
Checking a/c
Interest Rate
Overdraft amt

16/08/19

UNIT-02

Relational Model :-

The Relational Model represents the database as a collection of relations. Each relations resembles a table of values to some extent, a flat file of records. In the formal relational model terminology a row is called a 'tupple', a column header is called an 'attribute' and the table is called a 'relation'. The datatype describing the type of values that can appear in each column is represented by a domain of possible values.

Domain :- The domain 'd' is a set of atomic values by atomic we mean that each value in the domain is indivisible as far as relational model is concerned.
A datatype or format is also specified for each domain.
A relation schema 'R' is made up of relation name 'R' and a list of attributes $A_1, A_2 \ldots\ldots A_n$ and is denoted by :-

$$R(A_1, A_2, A_3 \ldots\ldots A_n)$$

'd' is called the domain of A_i (set of all attributes) and is denoted by :-

$$dom(A_i)$$

A relation schema is used to describe a relation : 'R' is called the name of relation and the degree (arity) of the relation is the number of attributes 'n' of this relation schema.

Relation State / Current Relation State / Instance Relation :-

A relation or relation state 'r' of relation schema $R(A_1, A_2 \dots A_n)$ also denoted by :-

$$r(R)$$

is a set of 'n' tuples $r = \{t_1, t_2 \dots t_n\}$. Each 'n' tuple is an ordered list of 'n' values $t = \{v_1, v_2, \dots v_n\}$, where each value v_i ; $1 \le i \le n$ is an element of domain $dom(A_i)$ or a special null value.

Student

S_id	S_name	S_phn	S_course	S_fn

- Relation - Table
- Tuple - Row
- Column Header - Attribute
- Domain - Column Values

The term Relation Intension (Instance) for schema 'R' and Relation Extension (Instance) for relation state 'r(R)' are also commonly used.

A relation or relation state r(R) is a mathematical relation of degree 'n' on the domain $dom(A_1)$, $dom(A_2)$ ... $dom(A_i)$ which is a subset of Cartesian Product of domains that defines R : $r(R) \le (dom(A_1) \times dom(A_2) \times \dots \times dom(A_n))$

The Cartesian Product specifies all possible combination of values from the underlined domains. Hence we denote the total no. of values or cardinality in a domain 'D' by :-

$|D|$

and the total number of tuples in the Cartesian Product is :-

$$|dom(A_1)| \times |dom(A_2)| \times \cdots \times |domain(A_n)|$$

Referential Integrity Constraints :-

Referential Integrity can works within a table also.

A constraint is a rule that is used for optimization purpose. Constraints enforce limits to the data or type of data that can be inserted/updated/deleted from a table.
The whole purpose of constraints is to maintain the data integrity during an update/delete/insert into a table.

Types of Constraints

- NOT NULL
- UNIQUE
- DEFAULT
- CHECK
- Key Constraints → Primary Key, Foreign Key
- Domain Constraints

20th Aug, 2019

Entity Integrity Constraints :-

Integrity means accuracy and consistency of particular data.
Enty integrity constraints states that no primary key value can be null. It is also known as Not Null Constraints.

Referential Integrity Conts Constraints :-

→ It is specified between two tables or relations and is used to maintain consistency among tuples in two relations.
→ To define differential integrity more formly we use the concept of Foreign Key.
✡ A set of attributes of foreign key in Relation Schema are R1 is a foreign key of R1 that references relation R2 if satisfies the following rules :-
→ The attribute in F_K have the same domain as the primary key attribute P_k of R2; the attributes F_K are said to referenced or refer to relation R2.
→ A value of F_K in tuple T1 of current state $r_1(R1)$ either occur as a value of P_K for some tuple T2 in current state $r_2(R_2)$ or is null.

So,

→ Within a table

$$t_1[F_K] = t_2[P_K]$$

and tuple T1 refers to tuple T2.

→ R1 is called Referencing Relation and R_2 is called Referenced Relation.

22/08/19

Actions upon Constraints Violation :-

(i) Reject
(ii) Cascade
(iii) Put either another value or null value

(i) **Reject** :-

Error shows in Domain Constraints :-

insert into St (rollno . int , name char)
values (Abc , xyz)

Rejects this value because we give the values ~~integer~~ character type at the place of integer type.

(ii) **Cascade** :-

Department
DID
1
2
3

Dependant

EID	Name	Age	Relation
1			Mother
1			father
1			wife

We cascade the EID 1 employee in Dependant table.

(iii) Put either another value or null value foreign key from another table in main table will be belongs to the another relation or it will be the null.

Relational Algebra :

The basic sets of operations for the formal relational model is relational algebra and these operations enable a user to specify basic retrieval request as Relational Algebra Expressions.
Relational Algebra is a procedural language.

Operations on Relational Algebra :-

- Select (σ) → Unary Operation
- Project [Π) → Unary operation
- Cross Product (x)
- Union (U)
- Intersection ($\cap$)
- (Minus or Set Difference (-)
- Rename (ρ) → Unary Operation

Implemented on (RA Interpreter)

Select (σ) :- Select operation is used to choose a subset of tuples from a relation that satisfies the selection condition.
It can be visualise as horizontal partition of relations into two sets of tuples.
Selection operation is denoted by :-

$$\sigma_{<cond^n>}(R)$$

Selection condition is made up of number of clauses of the form :-

- <attribute_name><comparison-operator><constant>
- <attribute-name><comparison_operator><attribute_name>

Employee

EID	E_name	DID	Salary

$\sigma_{DID=4}(Employee)$

$\sigma_{DID=4 \text{ AND } salary \geq 25000}(Employee)$

$\sigma_{salary \geq 25000}(\sigma_{DID=4}(Employee))$

- Select operator is unary ie. it is applied only on a single relation and is applied to each tuple individually.
- The degree of relation resulting from a select operation is its number of attributes and is same as degree of 'R'.
- Number of tuples in resulting relation is always less than or equal to number of tuples in 'R'.

Select operation is commutative.

ie. Ex

$$\sigma_{salary \geq 2500}(\sigma_{DID=4}(Employee)) \quad \because AB = BA$$

OR

$$\sigma_{DID=4}(\sigma_{salary \geq 2500}(Employee))$$

(Selected same no. of tuples)

- We can combine a cascade of select operation into a single select operation with a conjunction (AND), negation (NOT), destructs disjunction (OR).

ie. Ex

$$\sigma_{DID=4 \text{ AND } salary \geq 2500}(Employee)$$

- SQL Query for Select operation

Select * from Employee where DID=4 and salary ≥ 2500

23/08/2019

Project (Π) :- Project operation selects certain columns from a table and discards the other columns. It is also visualized as Vertical Partition.

It is denoted by :

$$\Pi_{<attribute\ list>}(R)$$

The degree of project operation has only the attributes specified in attribute list in the same order as they appear in the list.
It is also known as Duplicate Elimination.

Ex R

A	B	C
1	a	c
1	b	c
2	a	c
2	b	c

$\Pi_A(R)$ =

A
1
2

$\Pi_{AB}(R)$ =

A	B
1	a
1	b
2	a
2	b

$\Pi_B(R)$ =

B
a
b

$\Pi_C(R)$ =

C
c

- When attribute list is a super key then number of tuples after project operation will always be equal to the original relation.
- When attribute list is not a super key then number of ~~ty~~ tuples in ~~rel~~ relation after project operation will always be less than or equal to no. of tuples in original relation.

Ex

R

class	Name	F_N
M	a	ac
o	a	d
N	b	c
M	b	d

$\Rightarrow 4$

$\Pi_{Name, FN}(R) = 4$

$\Pi_{Name} = 2$

$\Pi_{FN} = 2$

$\Pi_{class} = 3$

- In SQL "distinct" keyword is used to retrieve the distinct values from relation column.
- Project operation is not commutative.

ie

$$\Pi_A(\Pi_{AB}(R)) \neq \Pi_{AB}(\Pi_A(R))$$

$\Pi_{<a_2>}(\Pi_{<a_1>}(R))$ is valid if and only if $a_2 \leq a_1$

QUERY:- Select distinct Name from R;

Rename (S):- We can write a single relational algebra expression known as in line expression as :-

$$\Pi_{FN, MN, LN}(\sigma_{DID=4 \text{ AND } Salary > 25000}(R))$$

But sometimes it is very hectic to understand this sequence so it is required to break down a complex sequence of operation by specifying intermediate relation result and for this we use rename operation.

If we want to rename a relation R to S(X, Y, Z) then its syntax is :-

$$\rho_{S(X,Y,Z)}(R)$$

If we want to rename only table not attributes then its syntax is :-

$$\rho_{S}(R)$$

If we wan to rename only attributes not table then its syntax is :-

$$\rho_{(X,Y,Z)}(R)$$

In SQL we use <u>alaising</u> for renaming and the keyword used is AS :-

QUERY :-
Select FN as F, MN as M, LN as L from R as Student where DID=4 and salary ≥ 25000.

Ex//

R

FN	MN	LN	Salary	DID

$$\rho_{Temp(F,M,L,S,D)}\left(\sigma_{DID=4 \text{ AND } salary \geq 2500}(R)\right)$$

$$\Pi_{F,M,L}(Temp)$$

Relational Algebra Operation from Set Theory :-

(a) Union
(b) Intersection
(c) Minus or Set Difference

24/09/19

(i) **Union** :- It is a binary operation. Two relations should be union compatible for union operations.

→ Number of attributes should be same (same degree).
→ Corresponding attribute domain should be same.
→ Names of attribute can be different.
→ Union operation removes duplicates.
→ If we don't rename output should be first operand.

Emp

Eid	E_name	D_id	sid
1	A	4	2
3	C	4	4
5	E	4	4

$R \cup S = R \{A_1, A_2, A_3 \ldots A_n\}$

Ex $\rho_{D\text{-}Emp}\left(\sigma_{DID=4}(Emp)\right)$

$R_1 \leftarrow \pi_{Eid}(D\_Emp)$

$R_2 \leftarrow \pi_{sid}(D\_Emp)$

$\rho(R_1 \cup R_2) \Rightarrow O/P = R$

Union is commutative and associate.

$R \cup S = S \cup R$

$R \cup (S \cup T) = (R \cup S) \cup T$

Student

FN	LN
A	K
B	Y
A	Z
C	G
D	W

Instructor

FN	LN
G	X
E	L
M	K
A	K
L	W

(ii) **Intersection** ($\cap$) :-

$\rho_{R_1(X,Y)} = \left(\Pi_{(FN,LN)}(Student)\right)$

$\rho_{R_2(M,N)} = (\Pi_{(FN,LN)}(\text{Instructor}))$

$\Rightarrow R_1 \cap R_2$

Intersection is also associative and commutative

Minus or Set Difference (−) :-

In SQL there are three operations - union, intersect and except (-) for those operation.

If we don't want to eliminate duplicates in SQL the we use union all, intersection all and except all in SQL.

$R \cup S - [(R-S) \cup (S-R)]$

27/08/19

Cartesian Product (×) / Cross Product :-

It is a binary operation that takes two relations and multiply them. Relations need not to be union compatible.

Emp

E_id	E_name
1	a
2	b

Dept.

D_id	M_id
s	t
u	v
1	2

Emp × Dept

E_id	E_name	D_id	M_id
1	a	s	t
1	a	u	v
2	b	s	t
2	b	u	v
1	a	1	2
2	b	1	2

- R×S will have (m+n) attributes where 'm' is the number of attributes in relation 'R' and 'n' is the attributes in 'S'.
- The total number of tuples in R×S will be :-

$$(p \times q)$$

where,

p is no. of tuples in R
q is no. of tuples in S

- Since to perform cross product in relations where no common attribute is of no use so we selection operation everytime with cross product.

$\sigma_{Eid = MID}(Emp \times Dept)$

E_id	E_name	Did	M_id
2	b	1	2

Join ($\bowtie$) :-

- It is a binary operation that is combination of cross product followed by selection.
- It combines relative tuples from two relations into a single larger tuple.
It is represented by :-

$$R \bowtie_{<condition>} S$$

We can have collection of conditions in join operations ie. represented as :-

$R \bowtie_{A_i \theta B_i \text{ and } C_i \theta D_i \text{ and } \ldots} S$

where, θ (Theta) is one of the comparison attribute, and this type of join operation with such a general join condition is called <u>theta join</u>.

Minimum number of tuples in join condition will be '0' when no tuple satisfies the condition whereas maximum number of tuple in join condition will be 'm x n' when all the tuples satisfies the condition.

Theta Join (θ)
Equi Join (=)
Natural Join (*)
Outer Join → Left Outer Join ($\bowtie$)
→ Right Outer Join ($\bowtie$)
→ Full Outer Join ($\bowtie$)

Natural Join (*)

Join Operation

Natural Join	Outer Join	Equijoin
A natural join is the set of tuples of all combinations in R & S that are equal on their common attribute names	The outer join operation is an extension of join operation. It is used to deal with missing information	It is also known as an inner join. It is the most common join. It is based on matched data as per the equality condition. It use the comparison operator

Inner and Outer Join :-

Join in which tuples are eliminated those who don't satisfies the condition is known as inner Join.

If we want all the tuples whether they satisfy all the conditions or not then the null comes into the table and it is known as Outer Join.

Emp

Eid	Ename	Edid
1	M	101
2	N	101
3	O	103
4	P	104

Department

Did	D_name	Mid
101	A	2
102	B	3
103	C	—
104	D	4

(i). Left Outer Join :-

$Emp \bowtie_{Eid=Mid} Dept.$

(ii) Right Outer Join

$Dept. \bowtie_{MID=DID} Emp$

Eid	E_name	Edid	Did	Dname	Mid
1	M	101	N	N	N
2	N	101	101	A	2
3	O	103	102	B	3
4	P	104	104	D	4

Eid	Ename	E_{DID}	Did	Dname	M_{ID}

full Outer Join :-

$$emp \bowtie_{Eid=mid} Dept$$

Eid	Ename	E_AD	Did	Dname	Min
1	M	101	N	N	N
2	N	101	101	A	2
3	D	103	102	B	3
4	P	104	104	D	4
N	N	N	103	C	N

Division (÷) :-

Retrieve the EID of Employees who work on all the projects that John Smith Works on.

- R ÷ S means whatever attributes we have in 'S' we substract them in R.
- Value of B (Remaining attribute of R) that is paired with every value of 'A' in 'S' (common attribute of R & S) will come in final result.

R

Eid	Pno.
101	1
101	2
102	1
103	2
104	3
101	4

S

John Smith

P_no.
1
2

R ÷ S

Eid
101

Que Consider the following relational schema:

branch (branch-name, branch city, assets)
customer (customer name, street, city)
account (a-no, branch-name, balance)
loan (loan-number, branch-name, amount)
depositor (customer_name, account_number)
borrower (customer-name, loan-number)

(i) find all loans of over $1200.

$$\sigma_{amount > 1200}\ (Loan)$$

(ii) find the loan-number of each loan of an amount greater than $1200

$$\Pi_{loan\_number}\left(\sigma_{amount > 1200}\ (Loan)\right)$$

(iii) find the names of all customers who have loan, account, all both from the bank.

$$\Pi_{customer\_name}(Borrower) \cup \Pi_{customer\_name}(Depositer)$$

(iv) find the names of all customers who have loan at Perry Branch.

$$\Pi_{Customer\_name}\left(\left(\sigma_{Branch\_name = "Perry"}(loan)\right) * Borrower\right)$$

(v) find the names of all customers who have loan at Perry Branch but don't have account at any other branch of the Bank.

$$\Pi_{Customer\_name}\left(\left(\sigma_{Branch\_name = "Perry"}(Loan) * Borrower\right) - \Pi_{Customer\_name}(Depositor)\right);$$

(vi) find the largest account balance.

$$\Pi_{Balance}(Account) - \Pi_{Balance}\left(\sigma_{Balance < D.balance}\left(Account \times \rho_{D}(Account)\right)\right)$$

ACCOUNT
1000
500
100
50
20

D
1000
500
100
50
20

X

A
500
100
100
50
50
50
20
20
20
20

31/08/2019

ID/ SSN (Social Security Number)

Aggregation :- It is used to find sum, average, minimum, maximum and count (total no. of values)

$G_1, G_2, \ldots G_n \; \mathcal{G} \; F_1(A_1), F_2(A_2) \ldots F_m(A_m) \; (\mathcal{E})$

where,

$\mathcal{E}$ is Relational Schema

$G_1, G_2 \ldots G_n$ are list of attributes on which two group

F_i is aggregate function and

A_i is attribute name

Que find sum of salaries of all instructor.

$\mathcal{G}_{sum(salary)} \; (Instructor)$

Que find total number of instructor who teach a course in Spring 2010 semester.

$\mathcal{G}_{count\_distinct(ID)} \; (\sigma_{semester = "Spring" \wedge year = 2010} \; (Instructor))$

Que find average salaries of employees of every department individually. Also find total number of employees of that department.

$D\_No. \; \mathcal{G}_{count\_distinct(Eid), Average(salary)} \; (Employee)$

Que Consider the following Schema :-

Employee (Fname, Minit, Lname, Ssn, Bdate, Address, Salary, Super_ssn, Dno)

Department (Dname, Dnumber, Mgr_ssn, Mgr_start_date)

Dept_location (D_number, D_location)

Works_on (Essn, Pno, Hours)

Project (P-name, Pnumber, Plocation, Dnum)
Dependant (Essn, Dependant_name, B-date, Relationship)

Que (i) Retrieve the name and address of all employees who worked for Research.

$R \leftarrow \sigma_{Dname='Research'}(Department)$
$RE \leftarrow R \bowtie_{Dnumber=Dno} (Employee)$
$\Pi_{F\_name, Minit, Lname, Address}(RE)$

$\Pi_{Fname, Minit, Lname, Address}\left(\left(\sigma_{Dname='Research'}(Department)\right) \bowtie_{Dnumber=Dno} Employee\right)$

(ii) for every project located at ~~Stephard~~ Stafford, list the project_number controlling department, dept manager last name, address and Birth date.

$R \leftarrow \sigma_{Pname='Stephard'}(Project)$
$RE \leftarrow R \bowtie_{Dnum=Dnumber} (Department)$
$SE \leftarrow (RE \bowtie_{mgr\_Ssn=Ssn} Employee)$
$\Pi_{Pno, Lname, address, Bdate}(SE)$

$\Pi_{Pnumber, Dname, Address, Bdate}\left(\sigma_{Plocation='Stafford'}(Project) \bowtie_{Dnum=Dnumber} (Department \bowtie_{mgr\_Ssn=Ssn} Employee)\right)$

(iii) find the name of employees who worked on ~~th~~ all the project controlled by Dept No 5.

$DP \leftarrow (\rho_{Pno}(\Pi_{Pnumber}(\sigma_{Dnum=5}(Project))))$
$EP \leftarrow \rho_{(Ssn, Pno)}(\Pi_{Essn, Pno}(Works\_on))$
$RE \leftarrow EP \div DP$
$\Pi_{Fname, Lname}(RE * Employee)$

(iv) Make a list of all Project numbers for projects that involve an employee whose last name is Smith either as a worker or manager of the dept. that controls the project.

(v) List the name of all employees with two or more dependants.

$A \leftarrow {}_{Essn}\mathcal{G}_{count\_distinct(Dependant\_name)}(Dependant)$

$\Pi_{Fname, minit, Lname}(\sigma_{count\_distinct > 2}(A \bowtie_{Essn=ssn} Employee))$

(vi) Name of employees with no dependant

$T_1 \leftarrow \Pi_{Essn}(Dependant)$

$T_2 \leftarrow \Pi_{ssn}(Employee)$

$T_3 \leftarrow T_2 - T_1$

$\Pi_{Fname, minit, Lname}(T_3 \bowtie_{Essn=Ssn} Employee)$

(vii) Name of managers who has atleast one dependant

$R_1 \leftarrow \Pi_{mgr\_ssn}(Department)$

$R_2 \leftarrow \Pi_{Essn}(Dependant)$

$R_3 \leftarrow R_1 \cap R_2$

$\Pi_{Fname, minit, Lname}(R_3 \bowtie_{Essn=Ssn} Employee)$

(viii)

(iv)

Worker

$Sm \leftarrow \Pi_{ssn}(\sigma_{Lname='Smith'}(Employee))$

$SW \leftarrow \Pi_{Pno.}(Sm \bowtie_{ssn=Essn} Works\_on)$

Manager

$M \leftarrow \Pi_{Lname, Dnumber}(Department \bowtie_{Mgr.ssn=Ssn} Employee)$

$MM \leftarrow \Pi_{Dnumber}(\sigma_{Lname='Smith'}(M))$

$SM \leftarrow \Pi_{Pnumber}(MM \bowtie_{Dnumber=Dnum} Project)$

$SW \cup SM$

7/09/19

Relational Calculus :-

RC

TRC (Tuple Relational Calculus)

DRC (Domain Relational Calculus)

Syntax :- $\{t \mid cond(t)\}$

$\{t \mid E_{salary} > 500\}$:- It retrieves all the tuples from universe whenever column is E_{salary} & greater than 500.

$\{t \mid Emp(t)$ AND $t.salary > 5000\}$:- It gives tuples from relation Emp and column name is E_{salary} where value is greater than 5000.

~~$\{t.eid, t.ename\}$~~

$\{t.eid, t.ename \mid Emp(t)$ and $t.salary > 5000\}$

(i) Retrieve the fname, minit, lname and address who works for Research Department.

$\{p.fname, p.minit, p.lname, p.address \mid Employee(p)$ AND

$\exists s (Department(s))$ AND $S.Dname = 'Research'$ AND

$p.Dno = S.Dnumber\}$

(iv) ~~{e.pnumber, e.name, e.address, BB.date | Department(d) AND~~

{p.pnum, p.Dnum, m.Lname, m.Bdate, m.add | Project(p) and Employee(m) and P.Plocation = 'stafford' and ((∃d)[Department(d) AND P.Dnum = d.Dnumber) AND dmgr.ssn = m.ssn}

(iii) find the first name & last name of each employee and immediate supervisor's first & last name.

Emp(P)

Fn	Ln	ssn	MSSn

& Emp(R)

Fn	Ln	

{ p.Fname, p.Lname, R.fname, R.Lname | Employee(P) AND Employee(R) AND S.superSSn = R.Ssn }

(iv) { e.fname, e.minit, e.Lname | Employee(e) AND ∃p(Project(p)) AND p.Dnum = 5 AND ∃w(Works(w)) AND P.Pnumber = w.Pno. AND w.Essn = e.SSn }

(v) Make a list of project no. ——— same as before

{ p.Pnumber | Project(p) And ((∃e)(∃w)(Employee(e)) And (Works_on(w)) And w.Pno = p.Pnumber And w.ESSn = e.ssn And e.Lname = 'Smith') OR ((∃m)(∃d)(Employee(m) and Department(d) AND p.Dnum = d.Dnumber And d.Mgr_Ssn = m.Ssn And m.Lname = 'Smith'))) }

(vi) { e.Fname, e.Minit, e.Lname | Employee(e) And (Not (∃d Departmen (d) And e.Ssn = d.Essn)) }

(vii) { e.fname, e.Lname, e.Minit | Employe (e) And ∃D Department(d) And D.mgr.ssn = e.ssn And (∃d Dependant(d) And e.Ssn = d.Essn }

find the name of Employees who worked on all the project controlled by Dept-No. 5.

{ e.Fname, e.Minit, e.Lname | Employee(e) And ((∀x) Project(And x.Dnum = 5) → (∃ w (Works on (w)) And w.Essn = e.Ssn And x.Pnum = w.Pno.) }

Relational Domain :-

1. List the Bdate & address of employee whose name is John-B. Smith. Employee (fname, Minit, Lname, Ssn, Bdate, Addr, sal, Ssn, Dno)

Employee (a, b, c, d, e, f, g, h, i)

$\{ e, f \mid (\exists a)(\exists b)(\exists c) (\text{Employee}(a,b,c,d,e,f,g,h,i)) \text{ And } a = \text{'John'} \text{ And } b = \text{'B'} \text{ And } c = \text{'Smith'} \}$;

[Relation Tuple → { t.Bdate, t.Addr | Employee (t) And t.Fname = 'John' And t.Minit = 'B' And t.Lname = 'Smith' }

2. Retrieve the name and address of employee who works for Research department. Dept (Dname, Dnumber, Mgr_ssn, Mgr_startdate)

Dept (a, j, k, l, m)

$\{ a, b, c, f \mid (\exists i)(\exists j)(\exists k) (\text{Employee}(a,b,c,d,e,f,g,h,i) \text{ And } \text{Department}(j,k,l,m) \text{ And } j = \text{'Research'} \text{ And } i = k) \}$;

3. for every project located in Stafford list the project no., dept no, dept manager last name, address, Bdate.

Project (Pname, Pnumber, Plocation, Dnum)

Project (p, q, r, s)

$\{ q, s, c, e, f \mid (\exists d)(\exists k)(\exists l)(\exists r)(\exists g) (\text{Employee}(a,b,c,d,e,f, g,h,i) \text{ And } \text{Department}(j,k,l,m) \text{ And } \text{Project}(p,q,r,s) \text{ And } r = \text{'Stafford'} \text{ And } d = j) \}$:

4. List the name of employee with no dependants.

Dependant (Essn, Dept name, Bdate, Relationship)

Dependant (j, k, l, m)

$\{ a, b, c \mid (\exists j)(\exists d) \text{ Employee}(a \ldots i) \text{ And Not } (\text{Dependant}(j,k,l,m) \text{ And } j = d) \}$

SQL

Datatypes :-

(i) Numeric :- int
float
decimal(i,j) / numeric(i,j)

(ii) Character String :- char[size]
varchar[size]

(iii) Bit String :- Bit(n) If we want value in 0 or 1 form then we use bit string where n is maximum value ie 1

(iv) Boolean :- This datatype has values True or false.

(v) Date :- YYYY:MM:DD
Ex Mgr.Ssn date.

(vi) Time Stamp :- This datatype include date and time field plus a minimum of six positions for decimal fractions of seconds and an optional with time zone qualifier.

Employee Table :-

```
Create table Employee
{
    Fname varachar(30) . Notnull ;
    Minit varchar(30),
    Lname varchar(30) Notnull ;
    Ssn int Not null,
```

```
    Dno int Notnull,
    Primary Key (Ssn),
    foreign Key (Dno) references Department (Dnumber),
    Bdate date,
    Address varchar(100),
    Salary decimal(10,2),
    Super_Ssn int, notnull,
    foreign Key (Super_ssn) references Employee (Ssn)
}
```

Department Table

```
Create table Department
{
    Dname varchar(50) notnull, unique,
    Dnumber int notnull,
    Primary key (Dnumber),
    Mgr_Ssn int notnull,
    foreign Key (Mgr_ssn) references Employee (ssn),
    Mgr_start_date date
}
```

Department Location

```
Create table dept_location
{
    Dnumber int notnull,
```

1. → Retrieve the name and address of Employee who works on Research Department.

```
select Fname, Lname, address from Employee, Department.
where Dname = 'Research';
```

2. → for every project located at Stafford list the project no, the controlling department the dept manager's last name, address, and Birth date.

```
Select pno, Dno, Fname, Minit, Lname, address, bdate from Employee,
Project, Department where Dname = Plocation = 'stafford And
Mgr_Ssn = Ssn And Dnumber = Dno.
```

3. → Make a list of all project no. or project that involve an employee whose last name is Smith either as a worker or as a manager of the dept that controls the project.

```
Select pno ((from Workson Project, Employee where Lname = 'Smith And
Essn = Ssn) OR ( from Project, Employee, Dept where Lname = 'Smith
And Mgr_Ssn = Ssn And Dnum = Dnumber))

Select pno from Workson
```

17/09/19

Attribute Constraints :-

Default :-

- Create table Employee
  {
  Dno int Default 1,
  foreign Key (Dno) references Department (Dnumber);
  }

- Check :-

Create table Department Dnumber int check (Dno >= 1 and Dno < 20)

- Unique :-

```
Create table Department
{
    Dname varchar(20) Notnull Unique;
}
```

Referential Triggered Action :-

```
Create table Employee
{
    Dno int Not Null Default 1,
    Constraint EmpPk Primary Key (SSn),
    Constraint EmpSuperFk foreign Key (Mgr_Ssn) References
      Employee (SSn), on delete set Null. on update Cascade,
    Constraint EmpDeptFk foreign Key (Dno) References
      Department (Dnumber) on Delete set Default. on update
      Cascade;
}
```

When we delete a tuple that references an domain of another table then in this case this deletion can violate referential integrity constraints. So in this case either we can reject the deletion operation or we can use :-

{ set Null, set Default, ~~set~~ Cascade }

command without rejecting deletion operation or on update operation.

Ex
```
Create Table Department
(
    Mgr_Ssn int Not Null Default 101,
    Constraint DeptPk Primary Key (Dnumber),
    Constraint DeptSk Unique (Dname),
    Constraint DeptMgrFk foreign Key (Mgr_ssn) References Employee(S
      on Delete set Default on Update cascade;
)
```

Aliasing (-As) :-

Query:- For each employee retrieve the Employee's first and last name and his immediate Supervisor's first and last name.

~~Create table Employee~~

Ex
```
Select EFN, ELN, MFN, MLN from Employee As E, Employee
    as M where E.Mgr_Ssn = M.Ssn
```

Employee

FN	LN

Employee

MFN	MLN

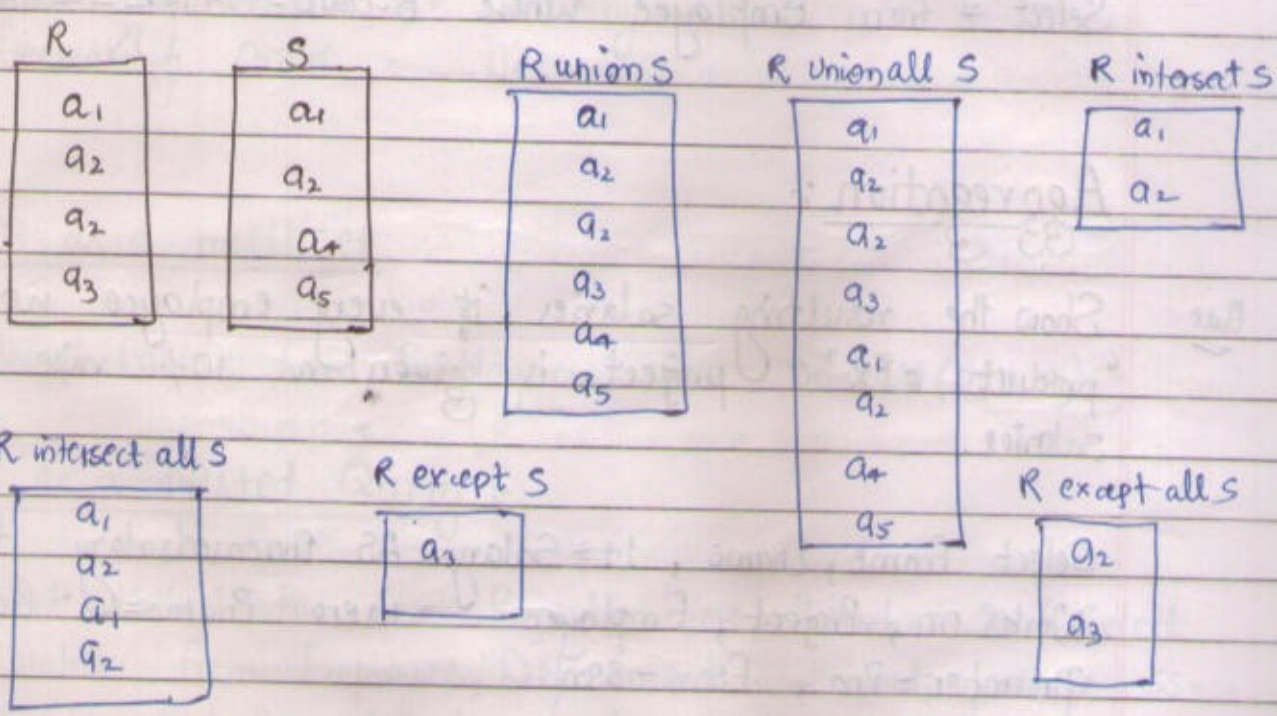
10/09/19

Union, Intersect, Except, Union All, Intersect All and Except All :-

Query Make a list of all project numbers that involve an Employee whose last name is a Smith either as a worker or manager of a department that controls the project.

(Select distinct Pnumber from Project, Department, Employee where Dnum=Dnumber and Mgr_ssn and Lname='Smith'.)
Union
(Select distinct Pnumber from Project, Works_on, Employee where Pnumber=Pno. and Essn=Ssn and Lname='Smith')

* Select operation is multiset in SQL.



Like :-

find all employees whose first name involve third alphabet 'v'.

```
Select * from Employee where Fname like '__v%';
```

- % → Number of zeroes or more characters are defined by % (percentage)
- _ → Single character are defined by _ (Undercore)

✡ If we have to retrieve a string that involve % or _ when we use escape(\) before % or _.

Que find all employees whose email is like ...AB-CD%.EF...

Select * from Employee where email like '%.AB_CD\%.EF%_'

Que Select all employees whose born during 1995.

Select * from Employee where B-date = '1995_____'

Aggregation :-

Que Show the resulting salaries if every employee working on 'product X' project is given a 10% raise in their salaries.

Select Fname, Lname, 1.1 * Salary AS Increasedsalary from Works_on, Project, Employee where Pname = 'Product X', Pnumber = Pno, Essn = Ssn

Between :-

Que find all employees in Dept. 5 whose salary in between 30,000 and 40000

Select Fname, Lname, Minit from Employee where Dno = "5" AND (salary between 30000 and 40000);

Order By :-

Que Select find all employees whose first name is in decreasing order and Lname is increasing order in Dept 5 whose salary in between 30,000 and 40,000

Select fname, Lname from Employee where Dno = 5 and (Salary Between 30000 and 40,000) ORDER BY Lname, Fname Desc;

Increasing Order :- Order by —
Decreasing Order :- — Desc

21/09/19

IN as a multiset :-

Select fname from Employee where Dno IN (2, 3, 4);

IN as a Nested Query :-

Select distinct Pno from Project where Pnumber IN (Select Pnumber from Employee, Department, Project where Ssn = Mgr_Ssn and Dno = Dnumber and Lname = 'Smith')

Que find fname and address of Employee who works for Department located at Stafford.

Select fname, Address from Employee, D-location where Dno = Dnumber and D_location = "Stafford";

OR

Select fname, address from Employee where Dno IN (select Dno from D_location where D_location = "Stafford");

→ We can use IN clause with relational operators {=, >, ≥, <, <=} and with { all, some, any } clause

IN → { =, >, ≥, <, ≤ } → { All, some, any }

Que find fname of Employees whose salary is greater than all employees who are working in Department no. 5.

Select fname from Employee where salary > All (Sttect salary from Employee where Dno = 5)

EXISTS / NOTEXISTS :-

Que Retriere fname of Employee who have Dependant and have same first name as their dependences name.

Select Fname from Employee AS E where Exists (Select * from Dependent AS D where E.ssn = D.ssn and E.Fname = D.Dependant_name);

Que find fname of Employees who have no dependant.

Select fname from Employee AS E where NotExists (Slect * from Dependent AS D where E.Ssn = D.Essn and E.fname =

D.Dependant_name);

Que Managers who have atleast one dependant.

Select fname from Employee AS E where Exists(Select * from Department where D.Mgr_Ssn = E.ssn) and ~~where~~ Exist (select * from dependant where Essn = MgrSsn)

Que Retrieve fname of each employee who works on all the projects controlled by Department number 5.

Select fname from Employee As E where Not Exist(Select Pnumber from Project Dno = 5) Except (Select Pno from Works_On AS W where E.ssn = W.Essn));

13/09/19

JOIN :-

find firstname and address of employees who works for Research Department using join.

Select Fname, Address from Employee Join Department on Dno = Dnumber where Dname = 'Research';

Using Natural Join :-

Select fname, Address from Employee Natural Join Department As D(Did, Dno, Dname, Daddress) where Dname = 'Research';

for full outer join, left outer join, or right outer join replace the Natural join keywords with all these keywords.

Aggregate Function :-

It have five functions namely :-

→ Avg
→ sum
→ Min
→ Max
→ Count

Minimum and maximum works on number, string and dates whereas sum works on only numbers

Que find sum of salaries and average of salaries of all the employees.

```
Select avg(salary), sum(salary) from Employee;
```

Que find the difference of minimum & maximum salary.

```
Select max(salary) - min(salary) from Employee;
```

Count :- Count is used to calculate number of entries in a row.

Que find all the number of employees.

```
Select count(*) As T_N_R from Employee;
```

Que find name of Employees whose salary is minimum.

```
Select fname from Employee where salary = min(Salary);
```

✡ min, max, sum, count always works with 'Select' clause

Select fname from Employee where salary = (Select min (Salary) from Employee);

Group By :-

Select * from Department group by ~~ES, EE~~ D-name (Attribute name)

24/09/19

3. NORMALISATION & FUNCTIONAL DEPENDENCY

Emp

Eid	Ename	Did

Dept.

Did	Dname

If we join these two tables in one table then information retrieval is easy.

Eid	Ename	Did	Dname
101	A	1	CS
	B	1	CS
102	C	1	CS
103			
104	D	2	IT
105	E	2	IT

ANAMOLIES

(i) Data Redundancy :- Data redundancy is possible when we join the two tables. Some more anamolies are also their :-

(ii) Insertion :- To insert a new employee tuple be include either the attribute value for the Department that he/she works for or null.

→ Suppose the employees working for DID = 2 and by mistake I have inserted Dname = EC then the DB is inconsistent.

→ If I have to insert a new Dept. details in which no employee is working then I can't insert.

(iii) Deletion :- If I delete the last employee detail who is working for Department = 3 then automatically will

Dept=3 will also be deleted and not exists.

(iv) Updation :- If I have to change name of Department whose DID=1 from CS to CSE then firstly I have to change same information in multiple tuples and secondly if somewhere it is left to be changed then our DB is inconsistent.

Functional Dependency :-

A functional dependancy is denoted by and it is a constraint

$$X \rightarrow Y$$

between two sets of attributes X and Y that are subset of R specifies a constraints on a possible tuple that can form a relation state of(R). The constraint is that, for any two tuples t1 and t2 if $t_1[X] = t_2[X]$ then $t_1[Y] = t_2[Y]$.

A	B	C
a	1	d
a	1	e
a	1	f
a	1	g
b	2	k

$A \rightarrow B$

If I have a functional dependancy

$A \rightarrow B$

then in this case if there are 100 'a' with then there will be 100 'b' and I have to change every value of 'a' if I need to change 'a' particular value of 'a'.

This redundant value can be solved if I decompose this table into two table like :

A	B
a	1
b	2

This is known as Normalisation.

Types of functional Dependancy :-

Functional Dependancy

Trivial	Nontrivial	Semi Trivial
$X \supseteq Y$	$X \cap Y = \phi$	$X \cap Y \neq \phi$
$AB \leftarrow A$	$A \leftarrow B$	$AB \leftarrow BC$
	$A \leftarrow BC$	

Attributes Participating in Keys :-

Attributes that are part of Candidate Key are known as Prime Attributes and remaining ones are known as Nonprime attributes.

Closure sets of Attributes :-

Functional Dependency :-
(FD-R(A B C D E))

Clues :-

$A \rightarrow B$

$B \rightarrow D$

$C \rightarrow DE$

$CD \rightarrow AB$

Closure

- One Attributes
  - $(A)^+ = ABD$ ✗
  - $(B)^+ = BD$ ✗
  - $(C)^+ = CDEAB$ ✓
  - $(D)^+ = D$ ✗
  - $(E)^+ = E$ ✗

- Two Attributes (Not a Candidate Key)
  - $(AB)^+ = ABD$ ✗
  - $(AD)^+ = ADB$ ✗
  - $(AE)^+ = AEBD$ ✗
  - $(BD)^+ = BD$ ✗
  - $(BE)^+ = BED$ ✗
  - $(DE)^+ = DE$ ✗

- Three Attributes
  - $(ABD)^+ = ABD$ ✗
  - $(ABE)^+ = ABED$ ✗
  - $(ADE)^+ = ADEB$ ✗
  - $(BDE)^+ = BDE$ ✗

Total number of candidate key possible are :- $2^n - 1$

- Four Attributes

  $(ABDE)^+ = ABDE$ (It is also not a candidate key)

  Only C is the candidate key in this functional dependency.

* Every candidate key is a super key.

Que R(EFGHITKLMN) By Shortest Method

EF → G

F → IJ

EH → KL

K → M

L → N

Whatever values we can retrieve we will tick that is from EF we get G.

E F G H I J K L M N

$(EFH)^+ = EFHGIJKLMN$

So EFH is a Candidate Key

$(EFH)^+ = E F H G I J K L M N$

So, EFH is a Candidate Key

Que R(A B C D E F G H)

CH → G

A → BC

B → CFH

E → A

F → EG

A B C D E F G H

$(D)^+ = D$ ✗

$(DA)^+ = DABCFHEG$ ✓

$(DB)^+ = DBCFHEGA$ ✓

$(DC)^+ = DCX$

$(DE)^+ = DEABCFHEG$ ✓

$(DF)^+ = DFEGABCH$ ✓

$(DG)^+ = DGX$

$(DH)^+ = DHX$

(DA, DB, DE, DF) are Candidate Key.

Now, we will not do pairing with D, A, B, E, F

$(DGC)^+ = DGCX$

$(DHC)^+ = DHCGX$

$(DHG)^+ = DHGX$

$(DGCH)^+ = DGCHX$

Therefore, Possible candidate keys are 4. They are DA, DB, DE, DF.

1. R(A B C D)

AB → CD

C → A

D → B

2. R(A B C D E)

AB → C

C → D

B → E

3. R(A B C D E F)

AB → C

C → D

D → E

E → F

F → A

4. R(A B C D E F)

A → B

C → D

E → F

5. R(A B C D E F)

AB → C

C → D

D → EB

E → F

F → A

6. R(A B C D E F)

A → BCDEF

BC → ADEF

DEF → ABC

7. R(A B C D E)

A → B

B → C

C → D

D → A

Equivalence of functional Dependancy :-

Que Check whether F is equivalent to G.

$F: \{A \to C, AC \to D, E \to AD, E \to H\}$

$G: \{A \to CD, E \to AH\}$

$F \supseteq G$ & $G \supseteq F$

$F = G$

$A^+ \to ACD$

$E^+ \to EACDH$

So $F \supseteq G$ — ①

$A^+ \to ACD$

$AC^+ \to ACD$

$E^+ \to EAHCD$

$G \supseteq F$ — ② —

from eqn ① & ②, we have

$F = G$

A set of functional dependency 'F' is set to cover another set of functional dependency 'E'. If every functional dependency in 'E' is also in F^+ closure ie: if every dependency in E can be inferred from F, alternatively we can say that 'E' is covered by F.

Two sets of functional dependencies 'E' & 'F' are equivalent if $E^+ = F^+$ ie 'ED can be infered in E can be inferd from F and functional dependency in F can be infered from E; ie if E is covered by F and F is covered by E then both are equivalent.

Que 2. $F: \{A \to B, B \to C, C \to D\}$

$G: \{A \to BC, C \to D\}$

Check whether $F \supseteq G$ or $G \supseteq F$

$A^+ = ABCD$

$C^+ = CD$

So, $F \supseteq G$ — (I)

$A^+ = ABC$

$B^+ = B$

$C^+ = .$

So $G \not\supseteq F$ — (II)

from eqn (I) & (II),

$F \neq G$

Que 3 $F: \{A \to B, AB \to C, D \to AC, D \to E\}$

$G: \{A \to BC, D \to AB\}$

Check whether $F \supseteq G$ or $G \supseteq F$

$A^+ = ABC$

$D^+ = DACBE$

So, $F \supseteq G$ — (1)

$A^+ = ABC$

$AB^+ = ABC$

$D^+ = DABC$ ✗

So, $G \not\supseteq F$ — (2)

from eqn (1) & (2), $F \neq G$

Que $F: \{A \to B, B \to C, C \to A\}$

$G: \{A \to BC, B \to A, C \to A\}$

Chak whether $F \supseteq G$ or $G \subseteq F$

Procedure to find minimal set of Functional Dependencies :-

1. Split the functional Dependencies such that RHS contain single attribute this is Canonical form of FD.

Ex $A \to BC$
$\hookrightarrow A \to B, A \to C$

2. find the redundant functional dependencies and delete them from the set

Ex $A \to B, B \to C, A \to C$
$\{A \to B, B \to C\}$

3. find the redundant attribute on LHS and delete them.

Ex If $AB \to C$ then A can be deleted only if B^+ contains A.

Ex $A \to C, AC \to D, E \to AD, E \to H$
$A \to C, AC \to D, E \to A, E \to D, E \to H$ (By step 1)
$A \to C, AC \to D, E \to A, E \to H$ (By step 2)
$A \to C, A \to D, E \to A, E \to H$ (By step 3)
$A \to CD, E \to AH$

Step 1 :- find Canonical form
Step 2 :- find the redundant values by finding closure values
Step 3 :- Redundant attribute on LHS & delete them.

Ex find minimal set of $\{A \to B, C \to B, D \to ABC, AC \to D\}$

Step 1 :- $A \to B, C \to B, D \to A, D \to B, D \to C, AC \to D$

Step 2 :- $A \to B, C \to B, D \to A, D \to C, AC \to D$

$A^+ = ACDB$	$D = DCB$	$AC = ACD$
$D = DBCD$	$D = DAB$	

Step 3 :- $A \to B, C \to B, D \to A, D \to C, AC \to D$
$A \to B, C \to B, D \to AC, AC \to D$

Que Check $\{AB \to C, D \to E, E \to C\}$ is the minimal cover of $\{AB \to C, D \to E, AB \to E, E \to C\}$

Step 1 :- $AB \to C, D \to E, AB \to E, E \to C$

Step 2 :- $D \to E, AB \to E, E \to C$

Step 3 :- $D \to E, AB \to E, E \to C$

Que $B \to A, D \to A, AB \to D$, find minimal cover of this.

Step 1 :- $B \to A, D \to A, AB \to D$

Step 2 :- $B \to A, D \to A, AB \to D$

Step 3 :- $B \to A, D \to A, B \to D$ (After applying Step 2)

Step 4 :- $D \to A, B \to D$

Que $G : \{A \to BCDE, CD \to E\}$ find minimal cover.

Step 1 :- $A \to B, A \to C, A \to D, A \to E, CD \to E$

Step 2 :-

Step 3 :-

$A \to BCD, CD \to E$

Decomposition :-

(i) Attribute Preservation :- Each attribute in R will appear in atleast one relational schema R_i in decomposition so that no attributes are lost.

(ii) Lossless (Lossless) Decomposition / (Non Additive) Joint Property :-

Ex 1

R(ABC)

A	B	C
a_1	b_1	c_1
a_2	b_1	c_1
a_1	b_2	c_2

$R_1(A)$ × $R_2(BC)$

A
a_1
a_2

B	C
b_1	c_1
b_2	c_2

=

A	B	C
a_1	b_1	c_1
a_1	b_2	c_2
a_2	b_1	c_1
a_2	b_2	c_2

Spurious Tuple

Ex 2 $R_1(AB)$

A	B
a_1	b_1
a_2	b_1
a_1	b_2

× $R_2(AC)$

A	C
a_1	c_1
a_2	c_1
a_1	c_2

=

A	B	C
a_1	b_1	c_1
a_1	b_1	c_2
a_2	b_1	c_1
a_1	b_2	c_1
a_1	b_2	c_2

Spurious Tuple

Ex :- Consider a relational schema R(ABC). Now after decomposition into two tables $R_1(A)$ & $R_2(BC)$ After performing natural join of two tables we have the above table.

Spurious Tuple :- Since this duple is not in main table so this is known as spurious tuple and our decomposition is lossy. We are calling such decomposition lossy although the data is not lost but some spurious tuples are also there.

Ex. 2. Now decompose the table $R_1(AB)$ and $R_2(AC)$ Now after performing natural join we have the drawn table.

Again we have some spurious tuples so this decomposition is also lossy.

Ex 3. Again we decompose the tables into $R_1(AB)$ $R_1(BC)$

$R_1(AB)$			$R_1(BC)$			A	B	C
A	B		B	C		a_1	b_1	c_1
a_1	b_1	×	b_1	c_1	⇒	a_2	b_1	c_1
a_2	b_1		b_2	c_2		a_1	b_2	c_2
a_1	b_2							

Now after performing Natural join we have above table and this table doesn't contain any supurious table tuple and is like the original table So the decomposition is lostless.

* If the common attribute is a key for any decomposed table then the decomposition is lossless.

(iii) Functional Dependency Preserving Property:- If each functional dependency $X \to Y$ specified in F either appeared directly in one of the relational schema R_i in the decomposition 'D' or could be infered from the dependencies that appear in some R_i then the decomposition dependency preserving. We want to preserve the dependencies because each dependency in F represents the constraints on the database. It is not necessary that exact dependency

$R(A_1, A_2, \ldots A_n)$

F^+

R_1 R_2

$F_1 \subseteq F^+$ $F_2 \subseteq F^+$

$F_1 \cup F_2 = F^+$

specified in F appear themselves in individual relation of decomposition 'D' but it is sufficient that the union of dependencies that hold on individual relation in D be equivalent to F.

Que: Check whether decomposition is valid or not.

R (ABC)

FD - {A → B, B → C, C → A}

D - R₁(AB), R₂(BC)

① Attributes are preserved because R₁(AB) and R₂(BC) contains the attributes of A, B & C.

② A B B C

$B^+ \rightarrow BCA$ $B^+ \rightarrow BCA$

It is lostless decomposition because B is candidate key.

③

AB	BC
✓ A → B	✓ B → C
✓ B → A	✓ C → B

$C^+ = CBA$

$F_1 \cup F_2 = F^+$

Que 2. Chek whether decomposition is valid or not.

R(A B C D)

FD = {A → B, B → C, C → D, D → A}

D = {AB, BC, CD}

① Attributes are preserved because we can obtain all attributes of A, B, C & D

②

{AB, BC, CD}

ABC CD

ABCD

$C^+ = CDAB$

$B^+ = BCDA$

It is lostless decomposition because C & B is the Candidate key

AB	BC	CD
A → B	B → C	C → D
B → A	C → B	D → C

Que Check the same as in above question.

R(ABCD)

FD = {AB → CD, D → A}

D = {AD, BCD}

10/10/19

Que R(ABCDEG)

F = { AB → C, AC → B, AD → E, B → D, BC → A, E → G }

D = { ABC, ABDE, EG }

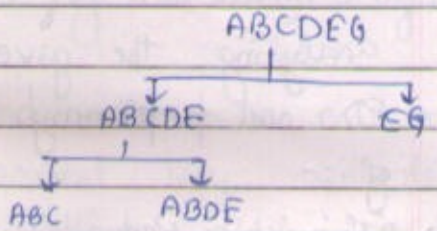
→ $R_1(ABC) \cup R_2(ABDE) \cup R_3(EG) \Rightarrow R(ABCDEG)$ ie Attribute Reserved.

→ Lossless Decomposition

$AB^+ = ABCDEG$ & $E^+ = EG$

(E is the Candidate Key)

(AB is the Candidate Key)



→ Functional Dependency Preserved

ABC	ABDE	EG
A → B	B → D	E → G ✓
B → C	AB → DE	G → E ✗
A → C	AD → E	
AB → C	for 3 attribute Not exits	
BC → A		
AC → B		

Que R(ABCDEG)

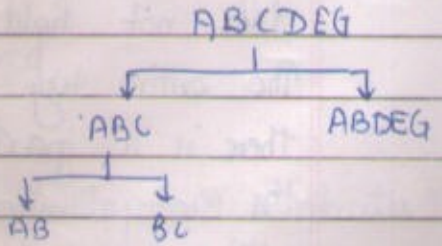
F = { AB → C, AC → B, AD → E, B → D, BC → A, E → G }

D = { AB, BC, ABDE, EG }

→ $R_1(AB) \cup R_2(BC) \cup R_3(ABDE) \cup R_4(EG) \Rightarrow R(ABCDEG)$

Attribute preserved

→ Lossless Preservation



10/10/19

NORMALISATION :-

First proposed by Codd in 1977 that takes a Relation Schema to a series of test to certified whether it satisfy a certain normal form.
Normalisation of data can be considered as process of analysing the given relation schema based on their FD and primary key to achieve the desired properties of :-

→ Minimising Redundancy
→ Minimising Insertion, Deletion and Updation Anamolies

✡ 1NF (First Normal form) :- A relational schema are is in 1NF if all the domains of all the attributes of relation R are atomic.
Domain is atomic if its elements are considered to be indivisible unit ie. multivalued and composite attributes are not allowed.

✡ 2NF (Second Normal Form) :- It is based on concept of full functional dependency.
A functional dependency $X \rightarrow Y$ is a Full FD if removal of any attribute A from X means that dependency does not hold any more.
The entire key should determine all attributes otherwise there is a partial dependency and partial dependency is not allowed in 2NF.
A FD $X \rightarrow Y$ is a partial dependency if some attribute $A \in X$ can be remove from X and the dependency still holds.

$B \rightarrow C$
$AB \rightarrow C$ ∴ AB is the candidate key

Part of Candidate Key

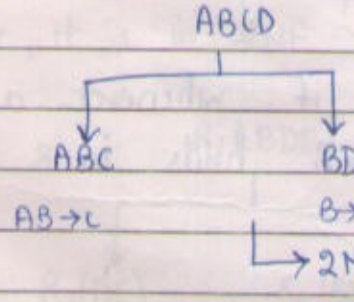
A relational schema are is in 2NF if every non-prime attribute A in R is fully FD on Candidate Key of R.

Test :- for relation where primary key contain multiple attributes then non key attribute (Non prime attribute) should not functionally dependent on part of Primary Key / Candidate Key.

Normalisation / Remedy :- Decompose and set up a new relation for each partial key with its dependent attribute. Make sure to keep a relation with original Candidate key and any attribute that are fully FD on it.

Ex R(ABCD)
FD = {AB → C, B → D}

Candidate Key AB
It is partially dependent because B → D

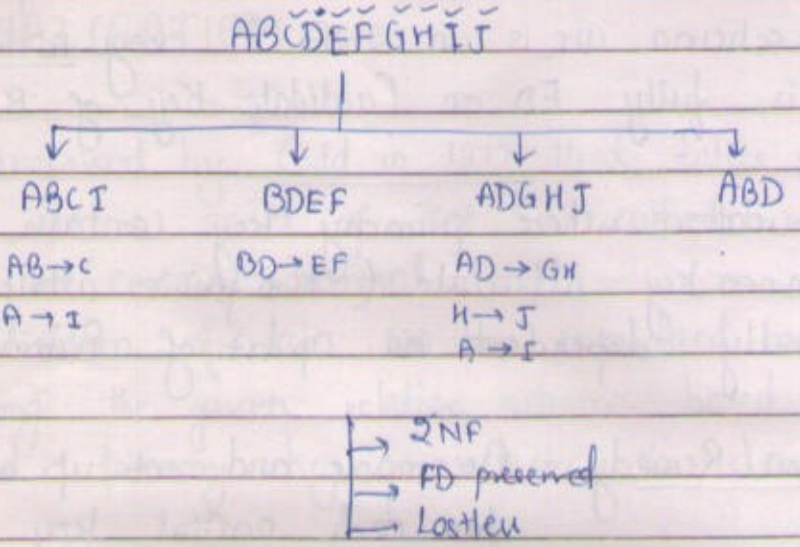


Lossless
Functional Dependency Preserved.

Ex R(ABCDEFGHIJ)

AB → C	PD	∵ Nonprime attributes retrieve depends by the part of the Candidate Key.
BD → EF	PD	
AD → GH	PD	
A → I	PD	
H → J	FD	

Candidate Key = ABD



✡ 3NF (Third Normal Form) :- It is based on concept of Transitive Dependency.

→ A FD $X \to Y$ in a relation schema R is a transitive dependency if there is a set of attribute 'z' ie. neither a candidate key nor a subset of any key of R ie and both $X \to Z$ and $Z \to Y$ holds.

→ A relational schema R are is in 3NF if it satisfies 2NF and no non prime attribute of R is transitively dependent on primary key.

→ A relational schema R is in 3NF if whenever a non-trivial functional dependency $X \to A$ holds in R either
- X is a superkey of R
- A is a prime attribute of R

Tests :- Relation should not have a non key attribute functionally determine by another non-key attribute. There should be no transitive dependency of non-key attribute on primary key.

Normalisation / Remedy :- Decompose and setup a relation that includes non key attribu

that functionally determine other non key attribute.

Que Normalise relational schema upto 3NF.

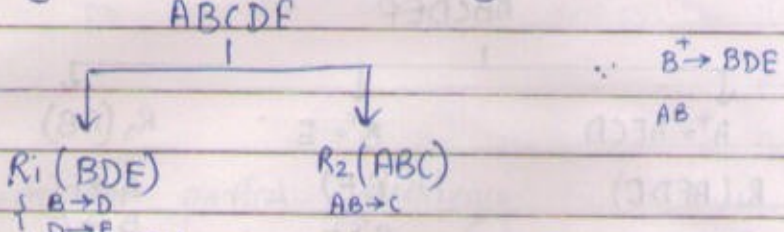
$R(ABCDE)$

$F = \{AB \to C, B \to D, D \to E\}$

$CK = AB$ (PD, TD)

→ It is in 1NF ie. multivalued and composite attributes are not allowed.

→ It is not in 2NF because a non-prime attributes (D) depends on the part of the Candidate Key (B) then decompose it.



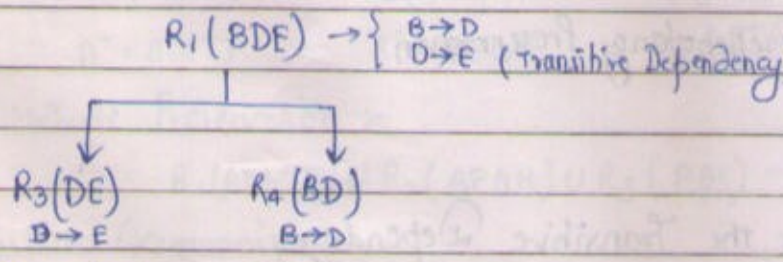
Now check it is valid or not.

→ Attribute Preservation (ie. B is common attribute ie $B^+ = BDE$)

→ Lostless (" ↙ ")

→ FD Preservation Property.

Hence, it is in 2NF form.



Now, check it is valid or not.

→ Attribute preservation ($R_3 \cup R_4 = R_1$)

→ Lostless (D is common attribute ie $D^+ = DE$)

→ FD Preserving Property

Ques R(ABCDEF)

$F = \{ A \to FC,\ C \to D,\ B \to E \}$

CK = AB (PD, TD, PD)

Normalise above Relational Schema upto 3NF.

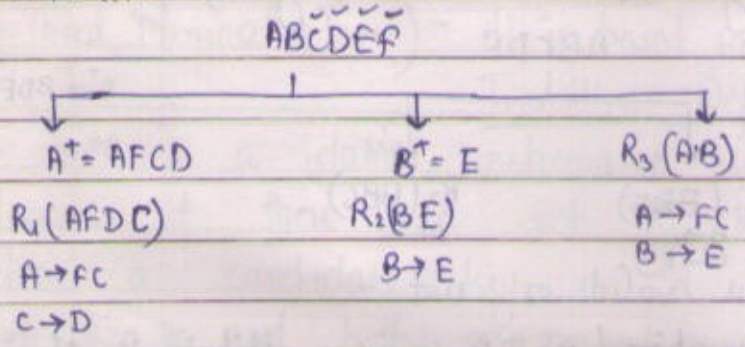
→ It is in 1NF (Multivalued and Composite Attribute are allowed)

→ for 2NF :-

Check for the Partial Dependency

Part of CK → N.P attribute

It is not in 2NF



- Attribute Preservation :-

$R_1 \cup R_2 \cup R_3 = R$

- Lossless Decomposition

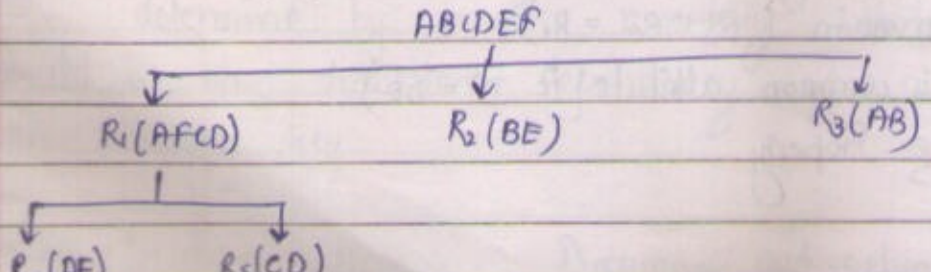
A is common attribute & B is also a common attribute

- Functional Dependency Preservation

→ for 3NF :-

Check for the Transitive Dependency

N.P attribute → N.P attribute



- Attribute Preservation
- Lossless Decomposition
- Functional Dependency Preservation

BCNF Normal Form :-
(Boyee Codd)

A relational schema ~~are~~ R in BCNF whenever a non-trivial functional dependency $X \to Y$ holds in R then X is a super key of R i.e. determinants (LHS) of all functional dependencies must be a super key.

Que $R(ABCDEFGHIJ)$

$F = \{ AB \to C, A \to DE, B \to F, F \to GH, D \to IJ \}$

(A → DE : PD; B → F : PD)

$CK = AB$

→ It is in 1 NF

→ For 2 NF check for partial dependency.

Part of ck. → n.p

It is not in 2 NF, so decompose it.

ABCDGEFHIJ

$R_1(ADEIJ)$ $A^+ = ADEIJ$

$R_2(BFGH)$ $B^+ = BFGH$

$R_3(ABC)$ $AB^+ = ABC$

① Attribute Preservation :-

$R_1(ADEIJ) \cup R_2(BFGH) \cup R_3(ABC) = R(ABCDEFGHIJ)$

② Lossless Decomposition

A is common attribute & B is also common attribute.

③ Functional Dependency Preservation

$R_1(ADEIJ)$ CK = A
$A \to DE$
$D \to IJ$
TD

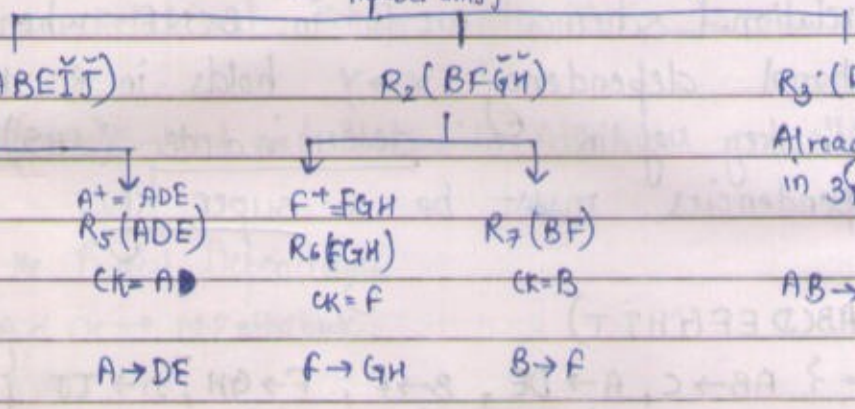
$R_2(BFGH)$ CK = B
$B \to F$
$F \to GH$
TD

$R_3(ABC)$
$AB \to C$

→ for 3NF :-

np. attribute → np. attribute (Transitive Dependency)

R_1 & R_2 are not in 3NF, so decompose it further.



① Attribute Preservation

② Lossless

③ functional Dependency Preservation

→ for BCNF :-

Check all dependies which is super key.

D, A, F, B, AB are all super key

ie. All decompositions are in BCNF form.

4.

Why Recovery is Needed?

Whenever a transaction is submitted to a DBMS for execution, the system is responsible for making sure that either all the operations in the transaction are completed successfully and their effect is recorded permanently in the database, or the transaction doesn't affect the DB or any other transactions. The DBMS must not permit some operations of a transaction T to a applied to the DB while other operations of T are not. This may happen if a transaction fails after executing some of its operations but before executing all of them.

Types of Failures :- failures are generally classified as transaction, system and media failures. There are several possible reasons for a transaction to fail in the middle of execution :-

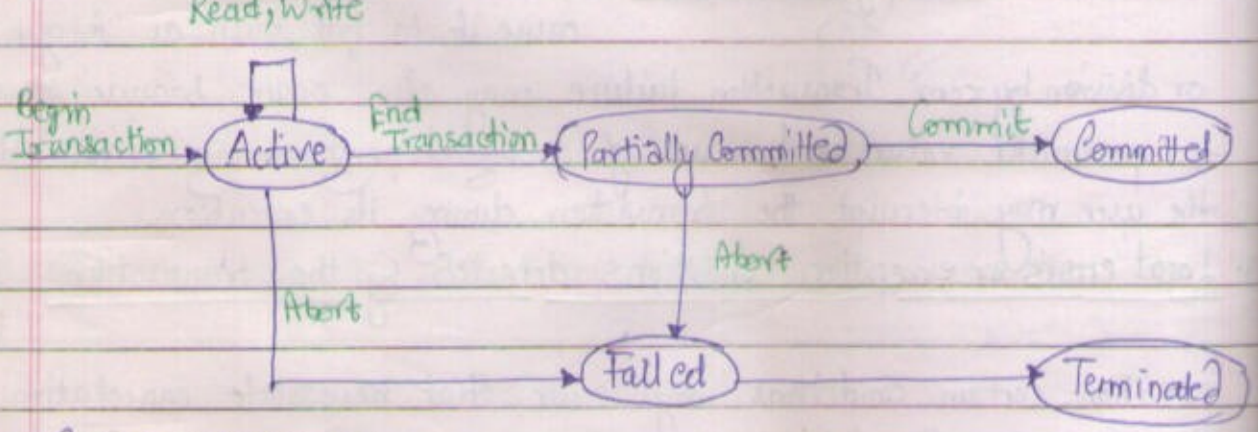
1. A Computer failure (System Crash) :- A hardware, software, or network error occurs in the computer system during transaction execution. Hardware crashes are usually media failures.
2. A transaction or System Error :- Some operation in the transaction may cause it to fail, such as integer overflow or division by zero. Transaction failure may also occur because of erroneous parameter values or because of a logical programming error. Additionally, the user may interrupt the transaction during its execution.
3. Local errors or exception conditions detected by the transaction :- During transaction execution, certain conditions may occur that necessiate cancelation of the transaction. Ex data for the transaction may not be found. Notice that an exception condition, such as insufficient account balance in a banking database, may cause a transaction, such as a fund withdrawal, to be canceled. This exception should be programmed in the transaction itself, and hence would be considered a failure.

4. Concurrency Control Enforcements :- The concurrency control method may be decide to abort the transaction, to be restarted later, because it violates serializability or because several transactions are in a state of deadlock.

5. Disk failure :- Some disk blocks may loose their data because of a read or write malfunction or because of a disk read/write head crash. This may happen during a read or a write operation of the transaction.

6. Physical problems and catastrophes :- This refers to an endless list of problems that includes power or air-conditioning failure, fire, theft, sabotage overwriting disks or tapes by mistake, and mounting of a wrong tape by the operator.

Transaction States and Additional Operations :-



- BEGIN-TRANSACTION
- READ OR WRITE
- END-TRANSACTION
- COMMIT-TRANSACTION
- ROLLBACK (or ABORT)

ACID Properties :-

Atomicity :- A transaction is an atomic unit of processing, it is either performed in its entirety or not performed at all.

Consistency Preservation :- A transaction is consistency preserving if its complete execution takes the database from one consistent state to another.

Isolation :- A transaction should appear as though it is being executed in isolation from other transactions. That is, the execution of a transaction should not be interfered with by any other transactions executing concurrently.

Durability or Permanency :- The changes applied to the database by a committed transaction must persist in the database. These changes must not be lost because of any failure.

Transaction :-

- A transaction is a logical unit of database processing that includes one or more database access operations, these include insertion, deletion, modification or retrieval operations.
- The database operations that form a transactions can be embedded within an application program.
- By specifying explicit begin transaction and end transaction we can specify the transaction boundaries.
- If the database operations in a transaction do not update the database but only retrieve data, the transaction is called a read-only transaction.

ACID Properties of Transaction :-

1. **Atomicity** :- It implies that either all of the operations of the transaction should execute or none of them should occur.
2. **Consistency** :- The state of database before the execution of transaction and

after the execution of transaction should be same.

(iii) Isolation :- A transaction must be aware of other transactions that are running parallel to it.

(iv) Durability :- Once a transaction is completed successfully. The changes made by transaction persist in database.

Ensuring the Atomicity :-

→ To ensure atomicity, database system keeps track of the old values of any data on which a transaction performs a write.

→ If the transaction does not complete its execution, the database system restores the old values.

→ Atomicity is handled by transaction management component.

Ensuring the Durability :-

→ Ensuring durability is the responsibility of a component called the recovery management component.

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

USEFULNESS of ACID Properties

Atomicity is useful to ensure that if for any reason an error occurs and the transaction is unable to complete all of its steps, then the system is returned to the state it was in before the transaction was started.

Consistency property is useful to ensure that a complete execute of transaction from beginning to end is done without interference of other transactions.

Isolation property is useful to ensure that a transaction should appear as though it is being executed in isolation from other transactions even though many transactions are executing concurrently.

Durability is useful to ensure that the changes applied to the database by a committed transaction must persist in the database.

15/10/19

The Problems due to Concurrent Transaction of Execution of Transactions :-

(1) LOST UPDATE PROBLEM :-

T_1	T_2
R(A)	
A = A - 50	
	R(A)
	x = A * 0.04
	A = A + x
W(A)	W(A)
R(B)	
B = B + 50	
W(B)	

(Write - Write Conflict)

Item A has an incorrect value because of its update by T_1 is lost (Overwritten).

Suppose that the operation of T_1 and T_2 are interleaved in such a way that T_2 needs the value of account 'A' before T_1 update, now when T_2 update the value of account A in the database, the value of account A updated by transaction T_1 is overwritten and hence lost.

(2) DIRTY READ PROBLEM :-

Reading an uncommitted data is called Dirty Read operation.

T_1	T_2
R(A)	
A = A-50	
w(A)	R(A)
	X = A * 0.04
	A = A + X
	w(A)
Read (B)	
B = B+50	
Write (B)	

Rollback

fail

(Write - Read conflict)

(3) UNREPEATABLE READ PROBLEM :-

T_1	T_2
R(A)	
	R(A)
	A = A - 15
	w(A)
R(A)	
A = A - 15	
w(A)	

(Read - Write Conflict)

When a transaction tries to read the value of data item twice and another transaction updates the same data time in between the two read operation of first transaction, as a result the first transaction reads varied values of same data item during its execution is known as unrepeatable read.

(4) PHANTOM TUPLE :-

T_1	T_2
Select * from Emp where salary > 30000	
	Insert into Emp('40', 35000)
Select * from Emp where salary > 30000	

Transaction T_1 may have read set of ~~these~~ rows from a table based ~~on~~ some condition, now suppose that a transaction T_2 inserts a ~~new~~ new row that also satisfies the condition of T_1, ~~when~~ then if T_1 is repeated then T_1 will see a rows that previously didn't exist and is called Phantom tuple.

(5.) INCORRECT SUMMARY PROBLEM.

One transaction is calculating an aggregate summary function on a number of records. While other transactions is updating some of these records then the aggregate function may calculate some values before they are updated and other values after they are updated & this results in incorrect summary.

T_1	T_2
	Sum=0
	R(A)
	Sum=sum+A
Read(x)	
X = X-N	
W(A)	
	R(X)
	Sum = Sum+X
	R(Y)
	Sum = Sum+Y
R(Y)	
Y=Y+N	
W(Y)	

SCHEDULE :- A schedule is a set of transaction with the order of execution of instruction in the transaction.

T_1	T_2
a_1	b_1
a_2	b_2

S_1	S_2	S_3	S_4	S_5	S_6
a_1	b_1	a_1	b_1	a_1	b_1
a_2	b_2	b_1	a_1	b_1	a_1
b_1	a_1	b_2	a_2	a_2	b_2
b_2	a_2	a_2	b_2	b_2	a_2

T_1	T_2	.....	T_n
n_1	n_2	.....	n_r

Total number of Schedules = $\frac{(n_1+n_2+n_3+\cdots+n_r)!}{n_1! * n_2! * \cdots * n_r!}$

Number of Serial Schedule = $n!$

16/10/19

The order in which instructions of transactions executed.

Number of Schedules :-

If there are two transactions T_1 and T_2 with a_1 & a_2 operations in Trasaction T_1 and b_1 & b_2 operations in Transaction T_2.

✡ Number of schedules = $\frac{(n_1+n_2+\cdots n_m)!}{n_1! * n_2! * \cdots * n_m!}$

If there are m transactions like :-

T_1	T_2	----------	T_m
n_1	n_2		n_m

then,

total no of schedules = $\frac{(n_1+n_2+\cdots+n_m)!}{n_1! \times n_2! \times \cdots \times n_m!}$

✡ Number of Serial Schedules will be m!

Some schedules can give some error. We want to allow only schedules which are guaranteed that DB result is going to be consistent & there will be no error.
Type of schedules that are always considered to be correct when concurrent transactions are executing is known as Serializable Schedules. If no interleaving of operations is permitted then there are only two possible outcomes :-
Execute all operations of transaction T_1 (in sequence) followed by all operations of transation T_2 (in sequence)
Execute all operations of transaction T_2 (in sequence) followed by all operations of transaction T_2 (in sequence

Characterizing Schedule :-

(i) Serial Schedule :-

T_1	T_2
R(A)	
A=A+10	
W(A)	
	R(A)
	A=A-50
	W(A)

T_1	T_2
	R(A)
	A=A-50
	W(A)
R(A)	
A=A-50	
W(A)	

Constant states of Output

T_1	T_2
R(A)	
A=A+10	
	R(A)
	A=A-50
W(A)	
	W(A)

Inconsistent states of O/P, transactions interleaving of operations

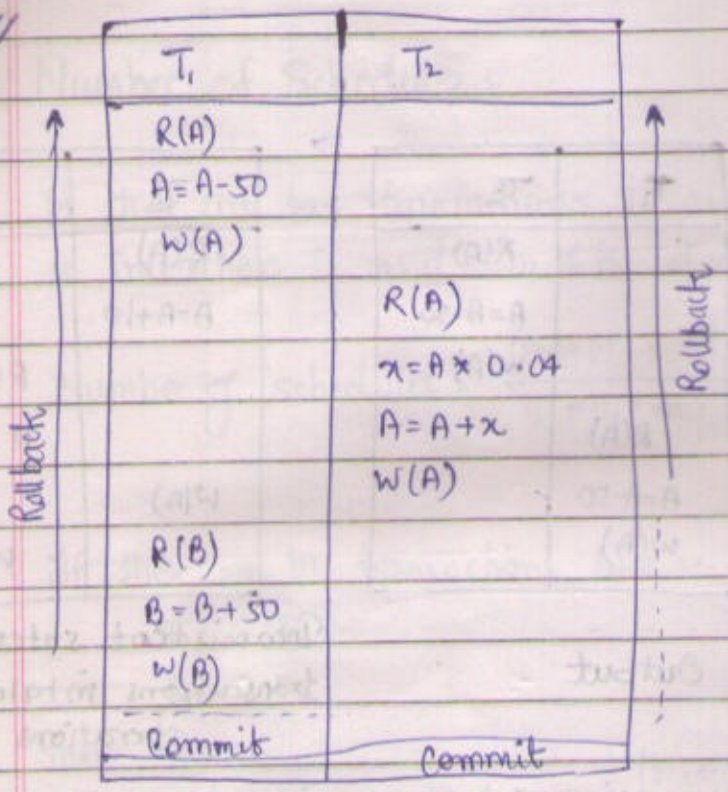
Serial schedules are those that does not interleave the actions of any operations of different transactions.

(ii) Complete Schedule :-

T_1	T_2
R(A)	
A=A-50	R(A)
W(A)	
commit	
	A=A+50
	W(A)
	Abort

A schedule is said to be complete if the last operation of each transaction is either abort or commit.

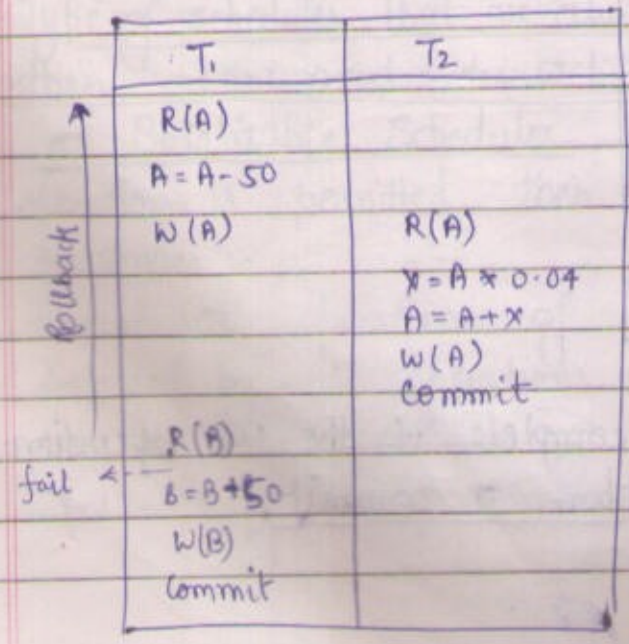
(iii) Recoverable Schedule :-



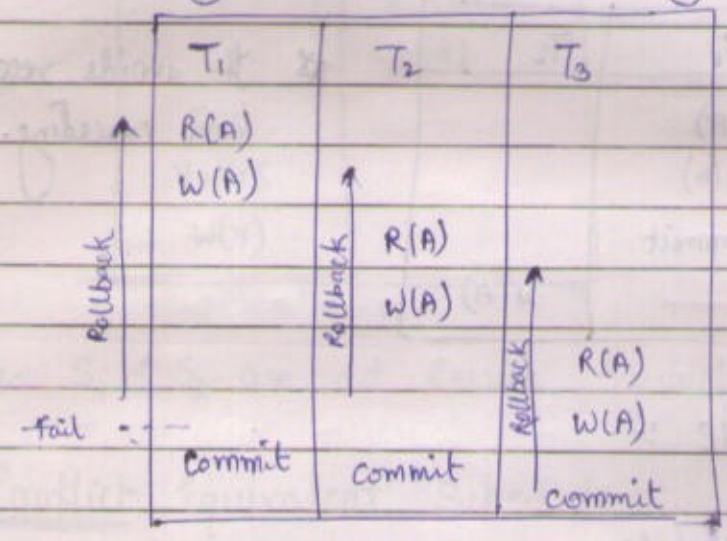
A recoverable schedule is one which for each pair of transaction T_i and T_j if T_j reads a data item previously written by T_i, the commit operation of T_i appears before the commit operation of T_j

A recoverable schedule is one where for each pair of transaction T_i & T_j such that T_j leaves the data item such that pass previously return by T_i then commit operation of T_i should appear before commit operation of T_j.

(iv) Non-recoverable Schedule :-



(v) Cascading Aborts :- If one transaction failure causes multiple transactions to rollback which is called Rate Cascading Rollback or Cascading aborts.



(vi) Cascadeless Schedule :- A cascadeless schedule is one where each pair of Transactions T_i & T_j such that T_j needs a data item ie. return by T_i then the commit operation of T_i should appear before read operation of T_j.

T_1	T_2	T_3
R(A)		
W(A)		
Commit	R(A)	
	W(A)	
	Commit	
		R(A)
		W(A)
		Commit

(vii) Strict Schedule :-

Schedule is restrict if a value return by a transaction T_i cannot be read or write until the transaction either aborts or commit.

T_1	T_2
R(A)	
W(A)	
Commit	
	W(A)

* It avoids recoverability and cascading.

16/10/19

(viii) Equivalent Schedule :-

(a) Result Equivalent Schedule :-

S_1	S_2
R(A)	R(A)
A = A + 10	A = A * 0.1
W(A)	W(A)

Two schedules S_1 & S_2 are said to be equivalent schedule if they produce the same final database state.

Que Is the following schedule are Result Equivalent for initial value of x & y (2, 5) respectively.

S_1

X	Y
2	5
7	10
21	

S_2

X	Y
2	5
6	10
11	

S_1

T_1	T_2
R(x)	
x = x+5	
W(x)	R(x
	x = x*3
	W(x)
R(y)	
y = y+5	
W(y)	

S_2

T_1	T_2
	R(x)
	x = x*3
	W(x)
R(x)	
x = x+5	
W(x)	
R(y)	
y = y+5	
W(y)	

ie. S_1 & S_2 are not Result Equivalent.

(b) Conflict Equivalent Schedule :-

Conflict Operations : Different transactions on same data items is having atleast one Write Operations.

Two schedules are said to be Conflict Equivalent if all Conflicting operations in both the schedules must be executed in same order.

Que Test whether the following schedules are Conflict Equivalent or not.

(i) S_1 : $R_1(A)$ $R_2(B)$ $W_1(A)$ $W_2(B)$

S_2 : $R_2(B)$ $R_1(B)$ $W_2(B)$ $W_1(A)$

S_1

T_1	T_2
R(A)	
	R(B)
W(A)	
	W(B)

S_2

T_1	T_2
	R(B)
R(B)	
	W(B)
W(A)	

R(B) → W(B) (Conflict)

These two are not Conflict Equivalent.

(ii) S_1 : $R_1(A)$ $W_1(A)$ $R_2(B)$ $W_2(B)$ $R_1(B)$
S_2 : $R_1(A)$ $W_1(A)$ $R_1(B)$ $R_2(B)$ $W_2(B)$

S_1

T_1	T_2
R(A)	
W(A)	
	R(B)
	W(B)
R(B)	

$W_2(B) \rightarrow R_1(B)$

S_2

T_1	T_2
R(A)	
W(A)	
R(B)	
	R(B)
	W(B)

$R_1(B) \rightarrow W_2(B)$

There both Conflicts are not Conflict Equivalent.

Conflict Serializability :-

A schedule is said to be conflict serializable if it is conflict equivalent to a serial schedule (SS)

Que find whether the following schedule is Conflict Serializable or not.

(i)

S_1

T_1	T_2
R(A)	
W(A)	
	R(A)
	W(A)
R(B)	
W(B)	
	R(B)
	W(B).

SS

T_1	T_2
R(A)	
W(A)	
R(B)	
W(B)	
	R(A)
	W(A)
	R(B)
	W(B)

S_1	S_2
✓ $R_1(A) \to W_2(A)$	$R_1(A) \to W_2(A)$
✓ $W_1(A) \to R_2(A)$	$W_1(A) \to W_2(A)$
✓ $W_1(A) \to W_2(A)$	$W_1(A) \to R_2(A)$
✓ $R_1(B) \to W_2(B)$	$R_1(B) \to W_2(B)$
✓ $W_1(B) \to R_1(B)$	$W_1(B) \to W_2(B)$
✓ $W_1(B) \to W_2(B)$	$W_1(B) \to R_2(B)$

∵ The conflicts of S_1 & S_2 are same. ∴ S_1 is conflict Serializable & DB will be in consistent state after schedule S_1.

(ii) S_1

T_1	T_2
R(A)	
W(A)	
R(B)	
	R(A)
	W(A)
	R(B)
W(B)	
	W(B)

S_1

$R_1(A) \to W_2(A)$
$W_1(A) \to R_1(A)$
$W_1(A) \to W_2(A)$
$R_2(B) \to W_1(B)$
$W_1(B) \to W_2(B)$

S_1 is not Conflict Serializable

S_2

T_1	T_2
R(A)	
W(A)	
R(B)	
W(B)	
	R(A)
	W(A)
	R(B)
	W(B)

S_2

$R_1(A) \to W_2(A)$
$W_1(A) \to R_2(A)$
$W_1(A) \to W_2(A)$
$R_1(B) \to W_2(B)$
$W_1(B) \to R_2(B)$
$W_1(B) \to W_2(B)$

∴ Bo

Tests for Conflict Serializability Using Precedance Graph:-

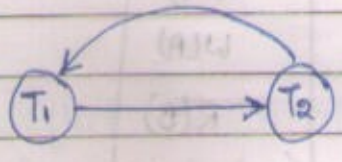
→ Construct a directed path where each vertices corresponds transaction and each directed edge represents the conflicted operation.

→ If the directed graph contain cycle then the concurrent schedule is not conflict serializable.

→ If the graph contains no cycle then the schedule is called Conflict Serializable.

S_1

T_1	T_2
R(A)	
W(A)	
R(B)	
	R(A)
	W(A)
	R(B)
W(B)	
	W(B)



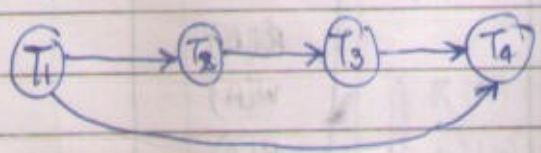
Cycle exists

ie S_1 is not Conflict Serializable

→ $R_2(x)\ W_3(x)\ W_1(y)\ R_2(y)\ W_2(z)\ R_4(x)\ R_4(y)$

S_1

T_1	T_2	T_3	T_4
	R(x)		
		W(x)	
W(y)			
	R(y)		
	W(z)		R(x)
			R(y)

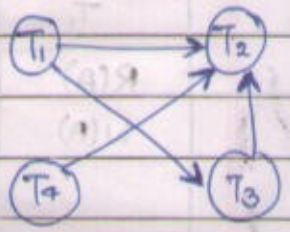


Serial Schedule : $T_1\ T_2\ T_3\ T_4$

→ $R_4(A)\ R_2(A)\ R_3(A)\ W_1(B)\ W_2(A)\ R_3(B)\ W_2(B)$

S

T_1	T_2	T_3	T_4
			R(A)
	R(A)		
		R(A)	
W(B)			
	W(A)		
		R(B)	
	W(B)		



$T_1\ T_4\ T_3\ T_2$
$T_1\ T_3\ T_4\ T_2$
$T_4\ T_1\ T_3\ T_2$

Number of Serial Schedules

View Equivalent Schedule :-

Two schedules S & S' are said to be view equivalent if following three conditions are met for each data item (say A) :-

(i) for each data item A if transaction T_i reads the initial value of A in schedule S then transaction T_i then must read initial value of A in schedule S'.

(ii) If transaction T_i executes read R(A) in schedule S and that was produced by transaction T_j then transaction T_i must in schedule S' also read the value of A that was produced by T_j.

(iii) for each data item the transaction that performs final right of A operation in S must also perform the operation in S'.

NOTE: All ~~read~~ Write-Read sequences operations to be maintained in same order in both S & S'.

Que

S_1

T_1	T_2
R(A)	
W(A)	
	R(A)
	W(A)
R(B)	
W(B)	
	R(B)
	W(B)

S_2

T_1	T_2
R(A)	
W(A)	
R(B)	
W(B)	
	R(A)
	W(A)
	R(B)
	W(B)

Initial Read
T_1 Read → write T_2
final value

	A		B	
	S_1	S_2	S_1	S_2
Initial ~~State~~ Read	T_1	T_1	T_1	T_1
final ~~State~~ Write	T_2	T_2	T_2	T_2

In S_1, A is produced by T_1 & consumed by T_2
In S_1, B is produced by T_1 & consumed by T_2.
In S_2, A is produced by T_1 & consumed T_2
In S_2, B is produced by T_1 & consumed by T_2.

Que Check whether following schedule is view equivalent or not

S_1

T_1	T_2
R(A)	
	W(A)
W(A)	

SS

T_1	T_2
R(A)	
W(A)	
	W(A)

	A		B	
Initial State →	S_1	SS	S_1	SS
	T_1	T_1		
final State →	T_1	T_2		

fail.

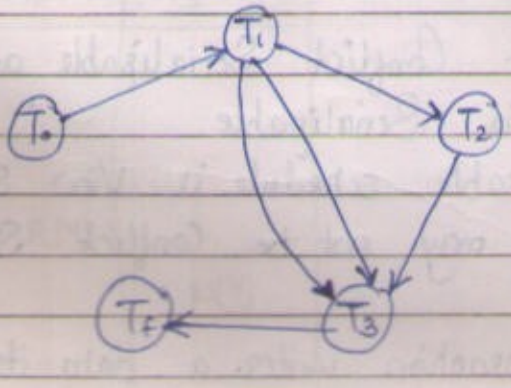
It is not view equivalent

Tests for View Serializable Schedule :-

(i) Using Poly Graph :- Check for WR operation for a data item then check for another W operation.

T_1	T_2	T_3
R(A)		
	W(A)	
W(A)		
		W(A)

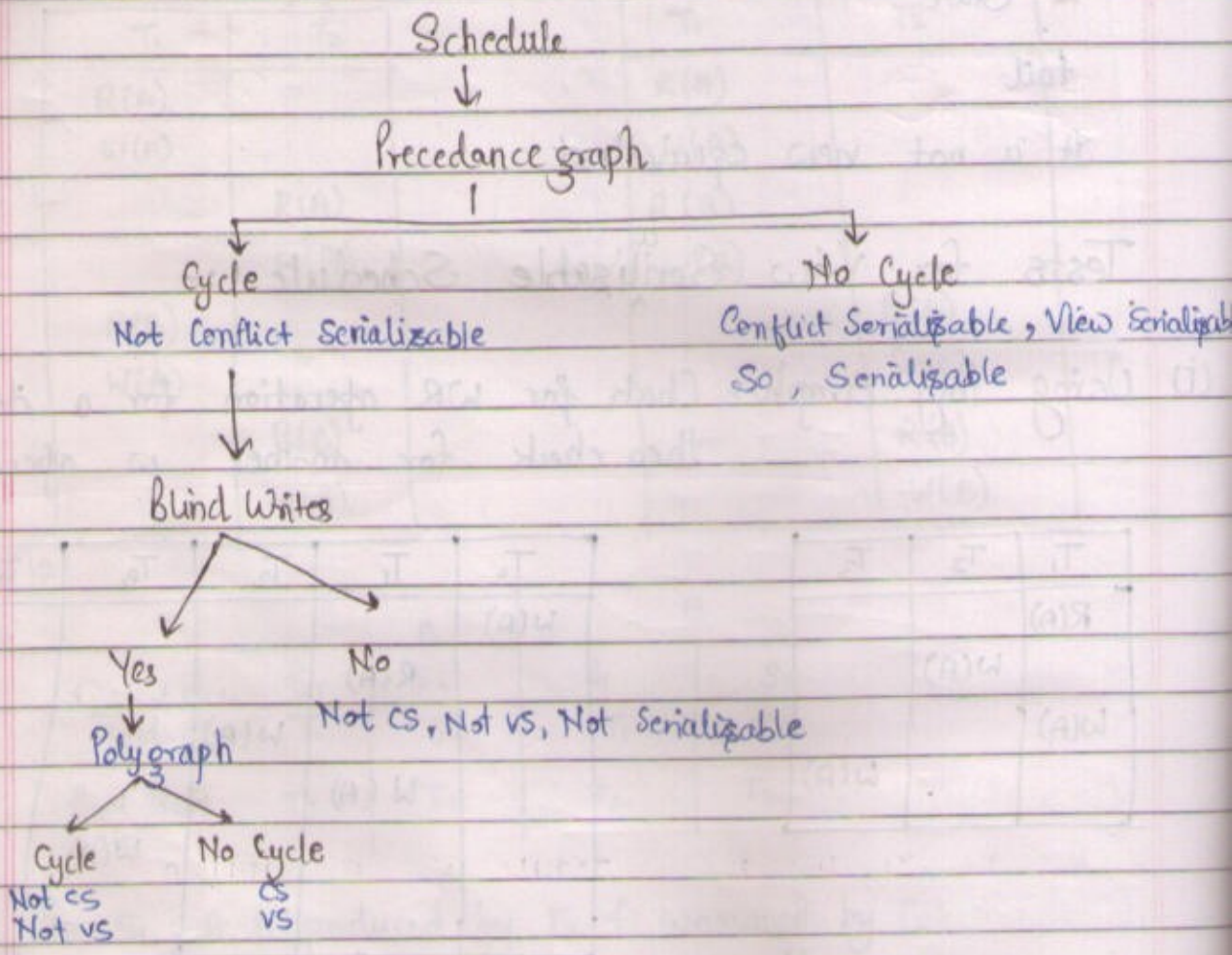
T_0	T_1	T_2	T_3	T_f
W(A)				
	R(A)			
		W(A)		
	W(A)			
			W(A)	
				R(A)



This schedule is view Serializable
T_0, T_1, T_2, T_3, T_f
T_1, T_2, T_3

Step1 :- Make two other transactions T_0 & T_f.
Step2 :- In T_0, first perform the write operations on each of every operands and in T_f, lastly perform the read operations on each of every operands.

Step-3 : Check for the RW or WR operations and make graph if there are cycle formed then it is View Serilizable and if it is not otherwise it is not view serilizable.



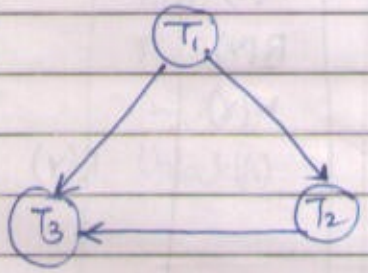
NOTE :
- If a schedule is either Conflict Serializable or View Serializable then it is Serializable.
- Every conflict Serialisable schedule is View Serialisable.
- Every View Serializable may not be Conflict Serializable

Blind Write : If the transaction writes a data item without reading it.

Que Check whether the given transaction is serialisable or not.

$S = r_1(A)\ w_2(A)\ w_1(A)\ w_3(A)$

T_1	T_2	T_3
R(A)		
	W(A)	
W(A)		
		W(A)



Precedance Graph (Not cs)

T_0 T_1 T_2 T_3 T_f

W(A)

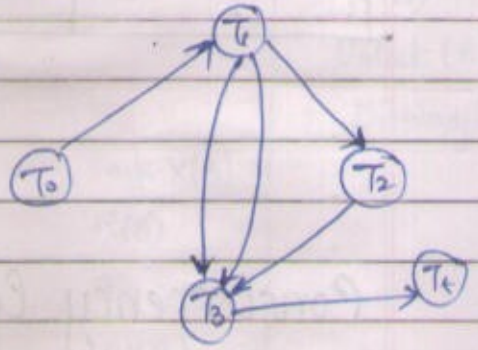
R(A)

T W(A)

W(A)

W(A)

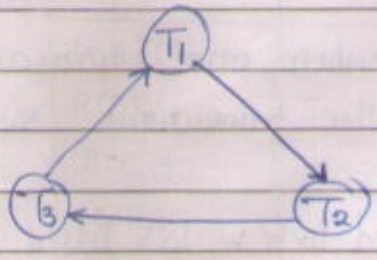
R(A)



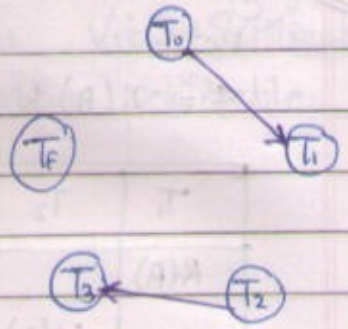
4/11/19

Que Check whether the given schedule is serializable or not

T_1	T_2	T_3
R(x)		
R(Y)		
W(X)		
	R(Y)	
		W(Y)
W(x)		
	R(Y)	



T_0	T_1	T_2	T_3	T_f
W(X)				
W(Y)				
	R(X)			
	R(Y)			
	W(X)			
		R(Y)		
			W(Y)	
	W(X)			
		R(Y)		
				R(X)
				R(Y)



Concurrency Control :-

For controlling the concurrent of transaction we use following protocol :-

(1) Lock Based Protocol :- It requires that all data items must be accessed in a mutually exclusive manner ie when one transaction is accessing data item no other transactions simultaneously update the same data item.

Shared lock :- lock based protocol (Lock S)

Exclusive Lock :- Both read and write operation (Lock X)

(1) Problem with Lock Based Protocol

5/10/19

T_1 : A $\xrightarrow{50}$ B

Lock X(A)
R(A)
A = A-50
W(A)
Unlock(A)
Lock X(B)
R(B)
B = B+50
W(B)
Unlock(B)

T_2 : Display (A+B)

Lock S(B)
R(B)
Unlock(B)
Lock S(A)
R(A)
Unlock(A)
Display (A+B)

User	S	X
S	True	False
X	false	false

Problem 1

T_1	T_2
Lock X(A)	
R(A)	
A = A-50	
W(A)	
Unlock(A)	
	Lock S(B)
	R(B)
	Unlock(B)
	Lock S(A)
	R(A)
	Unlock(A)
	Display (A+B)
Lock X(B)	
R(B)	
B = B+50	
W(B)	
Unlock(B)	

(2) Problem with ## Deadlock Condition

T_1	T_2
Lock X(A)	
R(A)	
A = A-50	
W(A)	
~~Unlock(A)~~	Lock S(B)
	R(B)
	Lock S(A)
	R(A)
	~~Display (B+A)~~
	~~Commit~~
	Unlock(B)
	Unlock(A)

T_1	T_2
Lock X(B)	
R(B)	
B = B+50	
W(B)	
Commit	
Unlock(B)	
Unlak(A)	

Two phase locking Protocol :-

It requires both locks and unlocks being done in two faces :-

(i) Growing Phase ("Obtain" Locks)
(ii) Shrinking Phase ("Release" locks)

Shrinking Phase :-

Lock Point :- It is the point at which the transaction has ~~opened~~ obtained final lock is called a lock point.

The lock point is used to determine the serializability order of a concurrent schedule.

Two-phase locking protocol ensures conflict serializability and serializability order is determined based on their lock points.

T_1	T_2	T_3
Lock X(A)		
R(A)		
W(A)		
Lock S(B)		
R(B)		
Unlock (A)		
:	Lock X(A)	
	R(A)	
	W(A)	
	Unlock (A)	
:		Lock S(A)
		R(A)
commit		

Two Phase Locking Protocol :-

- It is a procedure in which a transaction is said to follow the two-phase locking protocol if all locking operations precede the first unlock operation in the transaction.
- In 2PL, each transaction lock and unlock the data item in two phases :-

(a) Growing Phase :- In the growing phase, the transaction acquires locks on the desired data items.

(b) Shrinking Phase :- In the shrinking phase, the transaction releases the locks acquired by the data items.

According to 2PL the transaction cannot acquire a new lock, after it has unlocked any of its existing locked items.

T_1	T_2
Read-Lock(Y);	Read-lock(X);
Read-Item(Y);	Read-Item(X);
Unlock(Y)	Unlock(X);
Write-Lock(X);	Write-lock(Y);
Read Item(X);	Read-item(Y);
X = X+1;	Y = Y+1;
Write-item(X);	Write-Item(Y);
Unlock(X);	Unlock(Y);

→ Above two transactions T_1 and T_2 that do not follow the two phase locking protocol.

→ This is because the Write-lock(X) operation follows the unlock(Y) operation in T_1, and similarly the write-lock(Y) operation follows the unlock(X) operation in T_2.

→ If we enforce two-phase locking, the transaction can be rewritten as :

T_1	T_2
Redd-lock(Y)	
Read-item(Y);	Read-lock(X);
Write-lock(X);	Read-item(X);
Unlock(Y);	Write-lock(Y);
Read-item(X);	Unlock(X);
X = X+1;	Read-item(Y);
Write-item(X);	Y = Y+1;
Unlock(X);	Write-item(Y);
	Unlock(Y);

→ It can be proved that, if every transaction in a schedule follows the two-phase locking protocol, the schedule is guaranteed to be serializable, obviating the need to test for serializability of schedules any more

Cascading rollbacks may occur under two phase locking protocol and it can be avoided by a modification of two phase locking protocol

(i) Strict two phase locking Protocol (Strict 2PL) :- It requires that in addition to locking being two phase all exclusive mode locks taken by a transaction must be held until the transaction commit.

(ii) Rigorious two phase locking protocol (Rigorous 2PL) :- It requires that in addition to locking being two phase all locks must be held until the transaction commits.

In first and second it avoids cascading rollbacks but deadlock can occur.

(iii) Conservative two Phase Locking Protocol (Conservative 2PL) :- It requires the transaction to update all the locks before it starts & release all the locks after it commit.

* It avoids cascading rollback and deadlock.

3/11/19

Graph Based Protocol :-

The simplest model requires that we have prior knowledge about the order in which the database items will be accessed. To acquire such prior knowledge we impose partial ordering on set of all data items ie.

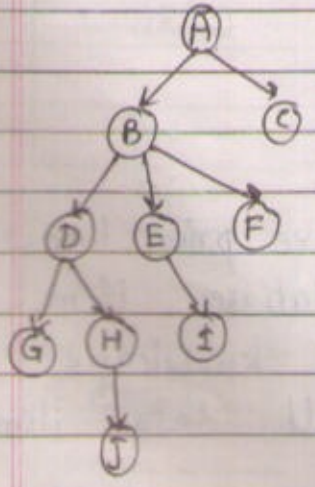
$d_i \rightarrow d_j$

is an ordering any transactions which required must require to access d_i before d_j.

The partial ordering is represented as an directed acyclic graph called a database graph. The simplest graph of a Graph Based Protocol is called Tree Based Protocol which requires to employ only exclusive mode locks

Tree Based Protocol :- In tree based protocol only lock instruction allowed is exclusive lock ie: lock X. On each transaction T_i can lock a data item atmost once and must observe the following rules :-

(i) The first lock on T_i may be on any data item.
(ii) The first Subsequently the data item can be locked if only if the parent of data item is currently locked by T_i.
(iii) Data Item may be locked at any time.
(iv) A data item that has been locked and unlocked by T_i cannot be relocked again by T_i.



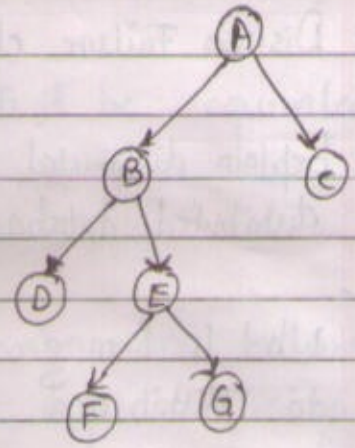
T_1	T_2	T_3
Lock X(B)		
	Lock X(B)	
	Lock X(H)	
	Unlock (B)	
Lock X(E)		
Lock X(D)		
Unlock (B)		
Unlock (E)		
		Lock X(B)
		Lock X(E)
	Unlock (H)	
Lock X(G)		
Unlock (D)		
		Unlock (E)
		Unlock (B)
Unlock (G)		

$T_1 \rightarrow T_2 \rightarrow T_3$

T_1	T_3
L(D)	
	L(E)

Test whether the following schedule can occur under Tree Protocol if possible then give serializable order.

T_1	T_2	T_3
Lock X(A)		
Lock X(B)		
Lock X(D)		
Unlock(B)		
	Lock X(B)	
Lock X(C)		
		Lock X(E)
Unlock(D)		
		Lock S(G)
Unlock(A)		
		Unlock(G)
		Unlock(E)
	Lock X(E)	
	Unlock(B)	
Unlock(C)	Unlock(E)	



ie. Transactions T_1, T_2 and T_3 follows the all rules of Tree Based Protocol.

$T_1 \rightarrow T_3 - T_2$

$T_3 - T_1 - T_2$

18/1/19

(1) What is Log? How it is maintained? Discuss salient features of deferred database modification and immediate database modification strategies in briefs.

(2) What are checkpoints?

(3) What is deadlock? How to prevent it and discuss deadlock detection and recovery scheme.

(4) Discuss Failure classification.

(5) Explain distributed database and outline the objectives of distributed database.

(6) What is homogenous and heterogenous database in reference to distributed database.

(7) How ~~can~~ Explain ~~cure~~ concurrency control in distributed DB.

(8) What is multiversion schemes of concurrency control? Descri with the help of an example. Discuss the various time stamping protocols for Concurrency Control

(9) What is validation based protocol.

(10) Working of multiple Granuality scheme in Concurrency Control.

(11) Trigger, Cursor, Views, Indexes

Timestamp based Protocols :-

Timestamp based protocol ensures serializability. It selects an ordering among transactions in advance using timestamps.

Timestamps :-

- With each transaction in the system, a unique fixed timestamp is associated. It is denoted by $TS(T_i)$.
- This timestamp is assigned by the database system before the transaction T_i starts execution.
- If a transaction T_i has been assigned timestamp $TS(T_i)$, and a new transaction T_j enters the system, then $TS(T_i) < TS(T_j)$.
- The timestamps of the transactions determine the serializability order. Thus, if $TS(T_j) > TS(T_i)$, then the system must ensure that in produced schedule, transaction T_i appears before transaction T_j.

To implement this scheme, two timestamps are associated with each data item Q.

(a) W-timestamp(Q) :- It denotes the largest timestamp of any transaction that executed write(Q) successfully.

(b) R-timestamp(Q) :- It denotes the largest timestamp of transaction that executed read(Q) successfully.

These timestamps are updated whenever a new read(Q) or write(Q) instruction is executed.

The timestamp ordering Protocol:-

The timestamp ordering protocol ensures that any conflicting read and write operations are executed in timestamp order. This protocol operates as follows :-

. Suppose that transaction T_i issues read(Q).

(a) If $TS(T_i) < W\text{-timestamp}(Q)$, then T_i needs a value of Q that was already overwritten. Hence, red read operation is rejected, and T_i is rollback.

(b) If $TS(T_i) \geq W\text{-timestamp}(Q)$, then the value of Q that T_i is producing was needed previously, and the sy then the read operation is executed, and R-timestam (Q) is set to the maximum of R-timestamp(Q) o $TS(T_i)$

Suppose that transaction T_i issues write(Q).

) If $TS(T_i) < R\text{-timestamp}(Q)$, then the value of Q that T_i is producing was needed previously, and the sys assumed that the value would never be produced. Hence, the system rejected write operation and rolls T_i back.

Time Stamp Based Protocol :-

It requires that ordering among the transaction is determined in advance based on their time stamps (ie. the time when it enters the system for execution).
Each transaction is given unique fixed time stamp denoted by $TS(T_i)$. If any transaction T_j enters after T_i then the relation between their time stamp will be like :

$$TS(T_i) < TS(T_j)$$

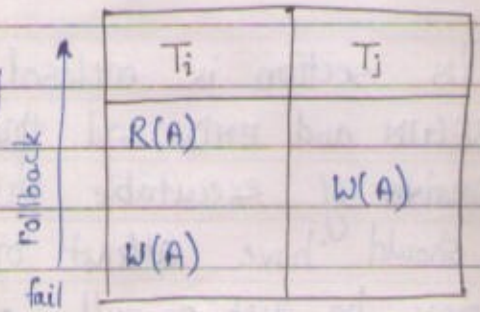
It means that producing schedule must be equivalent to serial schedule, T_i followed by T_j.

$$T_i \rightarrow T_j$$

Time Stamp Ordering Protocol (TSOP) :- In TSOP ensures that any conflicting Read and Write operation are executed in time stamp order. If not then such an operation is rejected and the transaction will be rolled back. The rolled back transaction will be restarted with a new time stamp.

Advantages :-
- Conflict Serializability
- No deadlock

Disadvantages :-
- Not recoverable
- Starvation



$$Ts(T_i) < TS(T_j)$$

$$T_i \rightarrow T_j$$

If the transaction T_i is rolled by the concurrency control scheme, the system assigns it a new timestamp and restarts it.

(b). If $TS(T_i) < W\text{-timestamp}(Q)$, then T_i is attempting to write on obsolete value of Q. Hence the system rejects the write operation and rollsback T_i.

(c). Otherwise, the system executes the write operation and sets W-timestamp(Q) to $TS(T_i)$

Thomas Write Rule :- It requires ignore all obsolete write operations.

T_1	T_2
	R(A)
W(A)	
	W(A)

$T_2 \rightarrow T_1$

TOP

Not Allowed	Allowed
$R_1(A)\ W_2(A)$	$R_2(A)\ W_1(A)$
$W_1(A)\ W_2(A)$	$R_1(A)\ R_2(A)$
$W_1(A)\ R_2(A)$	$W_2(A)\ R_1(A)$
	$W_2(A)\ W_1(A)$
	$R_2(A)\ R_1(A)$

TWR

$R_1(A)\ W_2(A)$	$R_1(A) \cdot R_2(A)$
$W_1(A)\ R_2(A)$	$W_1(A)\ W_2[A)$

Suppose that $TS(T_2) < TS(T_1)$

PLSQL Bo

Basic Syntax :-

(i) Declaration :- This section starts with keyboard keyword DECLARE. It is an optional section and defines all variables, cursors, subprograms and other elements to be used in the program.

(ii) Executable Command :- This section is enclosed between keyword BEGIN and END and this section is ~~mad~~ mandatory. It consists of executable PLSQL statements of the program. It should have atleast one executable line of code which may be just a null command. To indicate that nothing to be executed.

(iii) Exception Handling :- This section starts with the keyword EXCEPTION. This optional section contains exception that handles error in the program.

The basic structure of PL/SQL is :-

```
DECLARE
    <declare section>
BEGIN
    <executable command(s)>
EXCEPTION
    <exception handling>
END;
```

Ex
```
DECLARE
    message varchar(20)='Hello World!';
BEGIN
    dbms_output.put_line(message);
END;
```

PL/SQL Delimeters :-

PL/SQL Program Units :- A PL/SQL unit is one of the following

(i) PL/SQL Block
(ii) function
(iii) Package
(iv) Package Body
(v) Procedure
(vi) Trigger
(vii) Type
(viii) Type Body

PL/SQL Triggers :- Triggers are stored programs which are automatically executed when some events occur. Triggers are written to be executed in any of the following events :-

(ii) A database manipulation statement (insert, delete update)
(iii) A database definition statement (create, alter, drop)
(iv) A database operation (Server error, logon, logoff, startup or shutdown)

Ex of Trigger -

PL SQL Cursor :-

Oracle creates a memory area known as Context Area for processing an SQL statement which contains all the information needed for processing the statement.
Ex The number of row processed etc.

A cursor is a pointer to this Context Area. PL-SQL controls the Context area through a cursor. A cursor holds the rows one or more returned by a SQL statement. The set of rows the cursor holds is refered to as active set.
There are two types of cursors :-
(i) Implicit Cursors
(ii) Explicit Cursors

Implicit Cursors :- Implict cursors are automatically created by Oracle whenever an SQL statement is executed, when there is no explicit for the statement. Programmers cannot control the implicit cursors and informa in it.

Syntax of creating Trigger :-

```
CREATE [OR REPLACE] TRIGGER Trigger_name
{ BEFORE | AFTER | Instead of }
{ INSERT [OR] | UPDATE [OR] | DELETE }
[OF COL_NAME]
ON Table_Name
[Referencing OLD AS o NEW AS n]
[FOR EACH ROW]
WHEN (condition)
DECLARE
    declaration statement
BEGIN
    executable statement
EXCEPTION
    Exception Handling
END;
```

Ex (TRIGGER)

```
CREATE [OR REPLACE] TRIGGER display_salary_changes
BEFORE Delete or Insert or Update
ON CUSTOMERS
for each row
    when (NEW.ID>0)
    DECLARE
        Sal_diff number;
```

```
BEGIN
    sal_diff := new salary - :OLD.salary;
    dbms_output.put_line ('Old_salary is : ' || :OLD.salary);
    dbms_output.put_line ('New_salary : ' || New_salary);
    dbms_output.put_line('Salary_diff : ' || sal_diff);
END;
```

Whenever a DML statement (Insert, Update & Delete) is issued, an implicit cursor is associated with this statement.

→ for Insert Operation, the cursor holds the data that needs to be inserted.

→ for Update and Delete Operation, the cursor identifies the rows that would be affected.

Implicit cursors as a SQL cursor, which always has attribute such as :-

%FOUND, %ISOPEN, %NOTFOUND, %ROWCOUNT

%FOUND :- It returns true if an insert, update or delete statement affected one or more rows.

'SELECT INTO' STATEMENT RETURNED one or more rows otherwise it returns false.

%NOTFOUND :- The logical opposite of %FOUND. It returns True if an insert update statement affects no rows or a 'SELECT INTO' statement returned no rows otherwise it returns false.

%ISOPEN :- It always returns false for implicit cursors because Oracle closes the SQL cursor after executing its associated SQL statement.

%ROWCOUNT :- It returns the number of rows affected by an insert updat or delete returned by a 'SELECT INTO' statement.

Any SQL cursor attribute will be accessed as :-

SQL %attribute_name

Customers

ID	Name	Age	Address	Salary
1	A	30	K	20
2	B	31	L	30
3	C	32	M	40
4	D	33	N	50
5	E	34	O	60
6	F	35	P	70

Select * from Customer.

```
DECLARE
    total_rows number(2);
BEGIN
    update Customers
    set salary = salary + 500;
    IF sql%NOTFOUND then
        dbms_output.put_line('No customer related');
    ELSEIF sql%FOUND then
        total_rows := sql%rowcount;
        dbms.output.put_line(total_rows || 'Customer Selected');
    ENDIF;
END;
```

Explicit Cursors :- Explicit cursors are programmer's defined cursor for gaining more control over Content area. An explicit cursor should be defined in the declaration section of PL/SQL block.

The syntax for creating an explicit cursor is :-

```
CURSOR Cursor_name IS select statement;
```

Working with an explicit cursor includes the following steps :-

Step 1 :- Declaring the cursor for initializing the memory.

Ex

CURSOR C_Customer IS
Select id, name, address from Customer;

Step2:- Opening the cursor for allocating the memory.

Ex
```
OPEN C_Customers;
```

Step3 fetching the cursor for retreiving the data.

Ex
```
FETCH C_Customer INTO C_id, C_name, C_address;
```

Step 4 Closing the cursor for to release the allocated memory;

Ex
```
CLOSE C_Customer
```

```
DECLARE
    c_id Customers.id %type;
    C_name Customer.name %type;
    C_address Customer.address %type
    CURSOR C_Customers IS
        SELECT id, name, address. from Customer;
BEGIN
    OPEN C_Customer;
    LOOP
    FETCH C_Customer INTO C_id, C_name, C_address;
    EXIT when C_Customer %NOTFOUND
    dbms.output.put_line(c_id || ' ' || C_name || ' ' || C_address);
    ENDLOOP;
    CLOSE C_Customer;
END;
```

Validation Protocol in Concurrency Control :-

Validation protocol in concurrency control consists of following three phase :-

(1) Read Phase :-

- During this phase, the system executes transaction T_i
- It reads the values of the various data items and stores them in variables local to T_i.
- It performs all write operations on temporary local variables, without updates of the actual database.

(2) Validation Phase :-

Transaction T_i performs a validation test to determine whether it can copy to the database, the temporary local variables that hold the results of write operations without causing a violation of serializability.

(3) Write Phase :-

If transaction T_i succeeds in validation phase, then the system applies the actual updates to the database, otherwise, the system roll back T_i.

All three phases of concurrently executing transactions can be interleaved.

To perform the validation test, we should know when the various phases of transactions T_i took place. We shall, therefore, associate three different timestamps with transaction T_i :

- Start (T_i), the time when T_i started its execution.
- Validation (T_i) the time when T_i finished its read phase and started its validation phase.
- finish (T_i), the time when T_i finished its write phase.

Multiple Granularity Protocol of Concurrency Control :-

Multiple granularity protocol is a protocol in which we to lock the data items in top-down order and unlock them in bottom-up order.

In multiple granularity locking protocol, each transaction T_i can lock a node Q in any locking mode by following certain rules, which ensures serializability. These rules are as follows :-

(a) T_i must follow the compatibility matrix as shown in fig. to lock a node Q.

(b) It first locks the root of the tree and then locks the other nodes.

(c) It can lock a node Q in S or IS mode only if it currently has the parent of Q locked in either IX or SIX mode.

(d) It can lock node Q in X, SIX or IX mode only if it currently has the parent of Q locked in either IX or SIX mode.

(e) It can lock a node if it has not previously unlocked any node.

(f) It can unlock a node Q only if it currently has none of the children of Q locked.

Requested Mode	X	SIX	IX	S	IS
X	NO	NO	NO	NO	NO
SIX	NO	NO	NO	NO	YES
IX	NO	NO	YES	NO	YES
S	NO	NO	NO	YES	YES
IS	NO	YES	YES	YES	YES

X = Exclusive; SIX = Shared Intention Exclusive; IX = Intension Exclusive; S = shared; IS = Intention Shared

Advantages :-
This protocol enhances concurrency
It reduces lock overhead.

Disadvantages:
Deadlock is possible in this protocol.

Multiple Granularity :- It can be defined as heirarchically breaking up the database into blocks which can be locked.

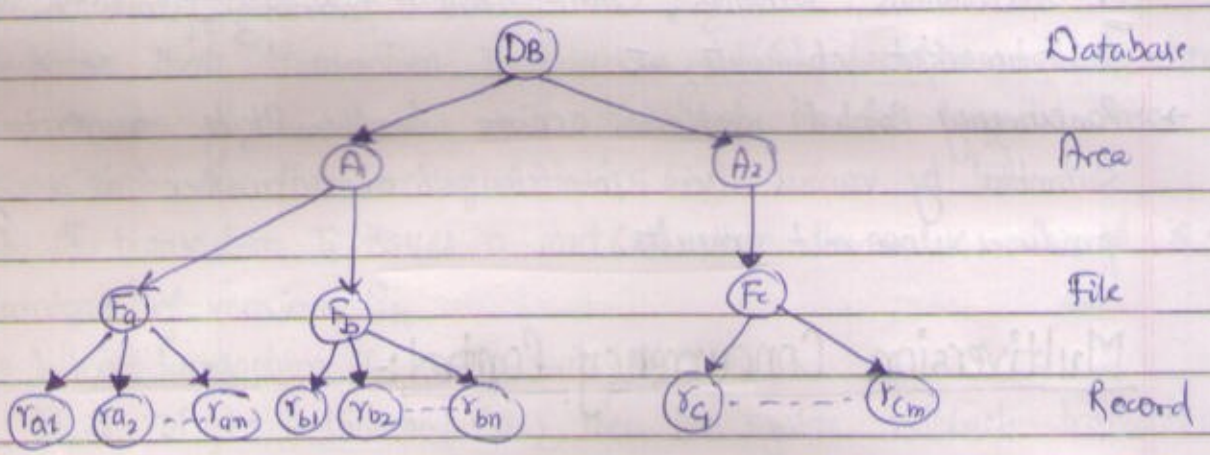
- This multiple granularity protocol enhances concurrency and reduces lock overhead.
- It maintains the track of what to lock and how to lock.
- It makes easy to decide either to lock a data item or to unlock a data item.

Implementation :-

(i) Multiple granularity is implemented in transaction system by defining multiple levels of granularity by allowing data items to be of various sizes and defining a heirarchy of data granularity where the small granularities are nested within larger ones.
(ii) In the tree, a nonleaf node represents the data associated with its descendents.
(iii) Each node is an independent data item.
(iv) The highest level represents the entire database.
(v) Each node in the tree can be locked individually using shared or exclusive mode locks.
(vi) If a node is locked in an intention mode, explicit locking is being done at lower level of the tree (ie at a finer granularity).

(vii) Intention locks are put on all the ancestors of a node before that node is locked explicitly.

(viii) While traversing the tree, the transaction locks the various nodes in an intention mode. This hierarchy can be represented graphically as a tree.



(ix) When a transaction locks a node, it also has implicitly locked all the descendents of that node in the same mode

Circumstances for multiple granularity locking :- The multiple granularity locking protocol is used when processing a mix of transaction that includes :-

→ Short transaction that access only a few items (records or fields)
→ Long transactions that access entire files

Granularity locking :- Granularity locking is a concept of locking the data item on the basis of size of data item.
It is based on the hierarchy of data where small granularities are nested within larger one. The lock may be granted at any level from bottom to top.

TRANSACTION PROCESSING SYSTEM :-

→ Transaction provides a mechanism for describing logical units of database processing.

→ These are the systems with large databases and hundreds of concurrent users that are executing database transactions
Ex Reservations, banking, credit card processing, stock markets, supermarket checkouts, etc.

→ Concurrency control problem occurs when multiple transactions submitted by various users interfere with one another in a way that produces incorrect results.

Multiversion Concurrency Control :-

→ Multiversion concurrency control is a schemes in which each write(Q) operation creates a new version of Q.

→ When a transaction issues a read(Q) operation, the concurrency-control manager selects one of the version of Q to be read.

→ The concurrency control scheme must ensure that the version to be read is selected in manner that ensures serializability.

Multiversion timestamping protocol :-

- The most-common transaction-ordering technique used by multiversion schemes is timestamping.
- With each transaction T_i in the system, we associate a unique static timestamp, denoted by $TS(T_i)$
- This timestamp is assigned before the transaction starts execution.
- Concurrency can be increased if we allow multiple versions to be stored, so that the transaction can access the version that is consistent for them.
- With this protocol, each data item Q is associated with a sequence of versions $< Q_1, Q_2, Q_3, \ldots, Q_m >$

- Each version Q_k contains three data fields
  (a) Content is the value of version Q_k
  (b) W-timestamp(Q_k) is the timestamp of the transaction that created version Q_k
  (c) R-timestamp(Q_k) is the largest timestamp of any transaction that successfully read version Q_k
- The scheme operates as follows:
  Suppose that transaction T_i issues a read(Q) operation. Let Q_k denote the version of Q whose write timestamp is the largest write timestamp less than or equal to $TS(T_i)$.
  (a). If transaction T_i issues a read(Q), then the value returned is the content of version Q_k.
  (b) When transaction T_i issues write(Q):
    (i) If $TS(T_i) < R\text{-timestamp}(Q_k)$, then the system rollback transaction T_i.
    (ii) If $TS(T_i) = W\text{-timestamp}(Q_k)$, the system overwrites the contents of Q_k; otherwise it creates a new version of Q.
    (iii) This rules forces a transaction to abort if it is "too late" in doing a write

Advantages of multiversion timestamping protocol:
→ Read request never fails.
→ Read request never made to wait.

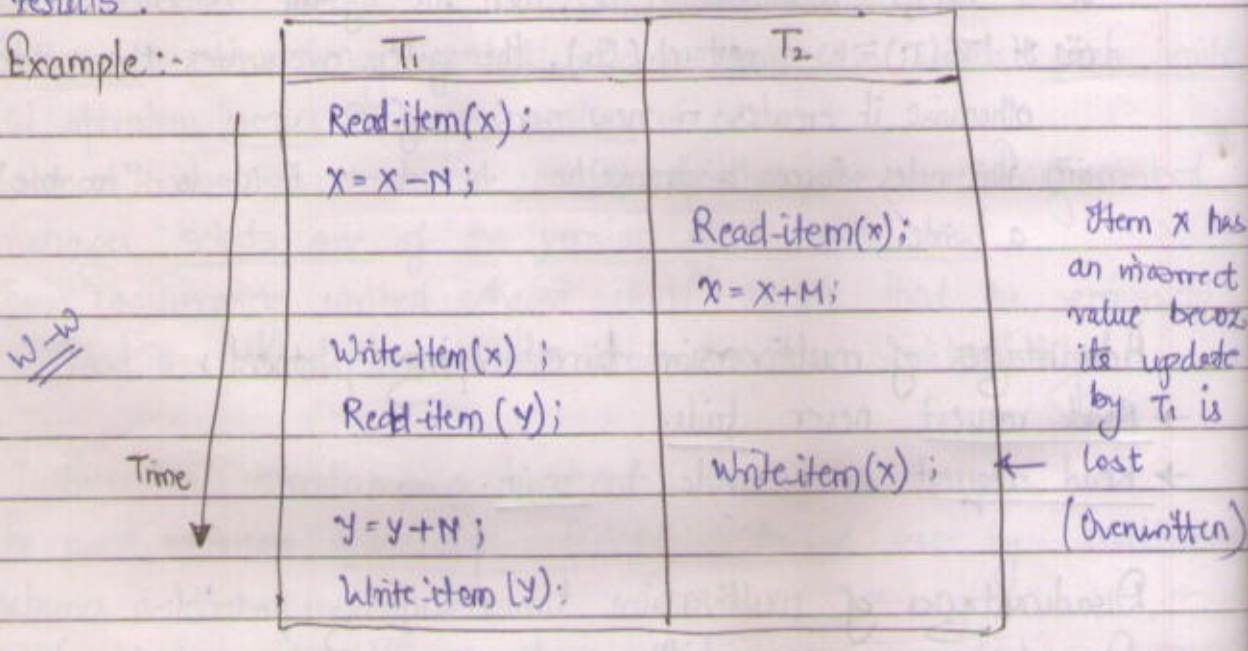
Disadvantages of multiversion timestamping protocol:-
- Do not ensure recoverability and cascadelessness
- It is expensive.
- Access more than one disk.

Concurrent Transaction :-

Concurrent transactions means multiple transactions are active at the same time.
Following problems can arise if many transactions try to access a common database simultaneously.

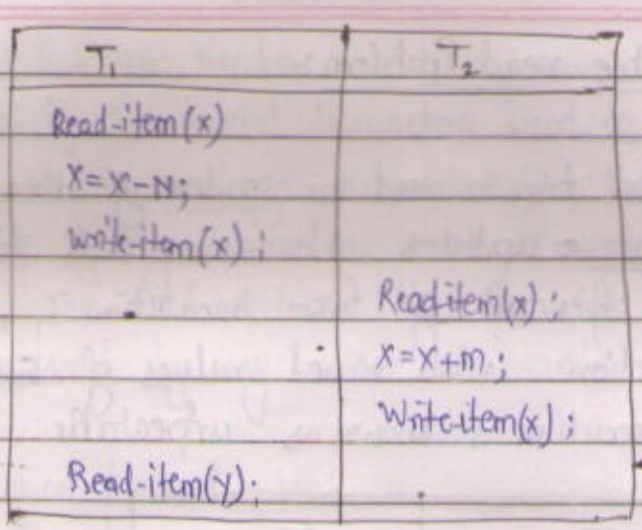
1. The lost update problem :-
A second transaction writes a second value of a data item on top of a first value written by a first concurrent transaction, and the first value is lost to other transactions running concurrently which need, by their precedence, to read the first value.
The transactions that have read the wrong value end with incorrect results.

Example :-



2. The dirty read problem (temporary update) :-
- Transactions read a value written by a transaction that has been later aborted.
- This value disappears from the database upon abort, and should not have been read by any transaction ("dirty read")
- The reading transaction end with incorrect results.

Example:-



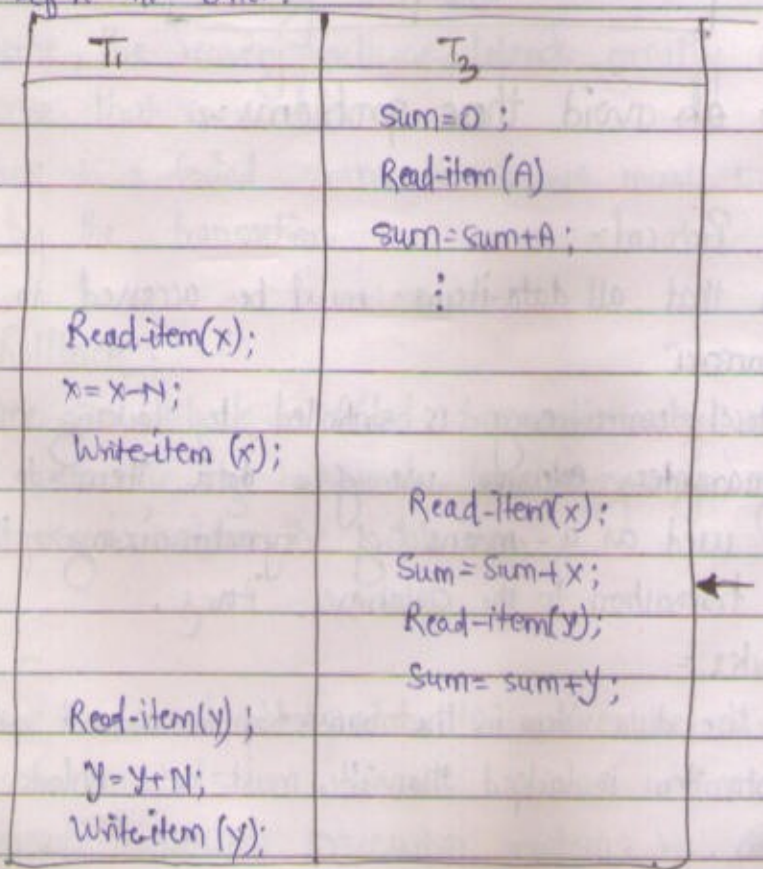
Transaction T_1 fails and must change the values of X back to its old value, meanwhile T_2 has read the temporary incorrect value of X.

3. The incorrect Summary Problem:-

While one transaction takes a summary over the values of all the instance of a repeated data item, a second transaction updates some instances of that data item.

The resulting summary does not reflect a correct result for any (usually needed for correctness) precedence order between the two transactions (if one is executed before the other.

Example:-



T_3 reads X after N is subtracted and reads Y before N is added; a wrong summary is the result (off by N)

4. Then unrepeatable read Problem :-

When a transaction tries to read the value of data item twice and other transaction updates the same data item in between the two read operation of first transaction T_1, as a result the first transaction reads varied values of same data item during its execution is known as unrepeatable read problem.

Example :

T_1	T_2
Read-item(x);	
	Read item(x);
	x = x-10;
	write item(x);
Read-item(x);	
x = x-10 ;	
Write item(x);	

Methods to ~~eb~~ avoid these problems :-

(i) Lock-based Protocol :-

- It requires that all data-items must be accessed in a mutually exclusive manner
- In this protocol, concurrency is controlled by locking the data items.
- A lock guarantees exclusive use of a data item to current transaction
- Locks are used as a means of synchronizing the access by concurrent transaction to the database items.

Role of locks :-

→ It locks the data item in the transaction in correct order.

→ If any data item is locked then it must be unlock at the end of the operation.

Two types of lock are used :-

(i) Shared Lock :- If the several transaction want to access the same data item x for reading purpose only, then we use shared lock.

(ii) Exclusive lock :- If any transaction want to update the value of x then we used exclusive lock. The data item with exclusive lock cannot be accessed by any other transaction until we release it.

2. Timestamp based protocal.

3. Multiversion scheme.

Recovery with Concurrent Transactions :-

Recovery from concurrent transaction can be done in the following four ways :-

(i) Interaction with concurrency control :

→ In this scheme, the recovery scheme depends greatly on the concurrency control scheme that is used.

→ So to rollback to a failed transaction, we must undo the updates performed by the transaction.

2. Transaction Rollback :

→ In this scheme we rollback a failed transaction by using the log.

→ The system scans the log backward, for every log record found in the log the system restores the data item.

3. Checkpoints :-

→ In this scheme we used checkpoints to reduce the number of log records that the system must scan when it recovers from a crash.

→ In a concurrent transaction processing system, we require that the checkpoint log record be of the form < checkpoint L>, where 'L'

is a list of transactions active at the end time of the checkpoint.
→ A fuzzy checkpoint is a checkpoint where transactions are allowed to perform updates even while buffer blocks are being written out.

4. Restart Recovery :-
→ When the system recovers from a crash, it constructs two lists.
→ The undo-list consists of transactions to be undone, and the redo-list consists of transactions to be redone.
→ The system constructs the two lists as follows :- Initially, they are both empty. The system scans the log backward, examining each record, until it finds the first <checkpoint> record.

Oracle :-

- The Oracle database (commonly referred to as Oracle RDBMS or simply Oracle) consists of a relational database management system (RDBMS)
- Oracle is a multi-user database management system. It is a software package specializing in managing a single, shared set of information among many concurrent users.
- Oracle is one of many database servers that can be plugged into a client / server equation.
- Oracle works to efficiently manage its resource, a database of information, among the multiple clients requesting and sending data in the network.

Storage :-
→ The oracle RDBMS stores data logically in the form of table spaces and physically in the form of data files.
→ Table spaces can contain various types of memory segments, such as data segments, index segments etc

Terms of Oracle :-

(i) Tablespace :- Tablespace is a logical portion of an Oracle database used to allocate storage for table and index data.

- Each tablespace corresponds to one or more physical database files.
- Every Oracle database has a tablespace called SYSTEM and may have additional tablespaces.
- A tablespace is used to group related logical structures together.

(ii) Package :- Packages are a method of encapsulating and storing related procedures, functions, and other package constructs together as a unit in the database. While packages provide the database administrator or application developer organizational benefits, they also offer increased functionality and database performance.

- Calling a public procedure or function ie part of package is no different than calling a standalone procedure or function, except that we must include the ~~programmer's~~ package name as a prefix to the program name.

  Ex: PackageName.function/ProcedureName

(iii) Schema :- A schema is a collection of table definitions or related objects owned by one person or user.

- SCOTT is schema in the Oracle database.
- Schema objects are the logical structures that directly refer to the database's data.
- Schema objects include such structures as tables, views, sequences, stored procedures, synonyms, indexes, clusters and database links.

SQL Plus :-

- SQL plus is the front end tools for Oracle.
- The SQL-Plus window looks much like as a Dos Window with a white background similar Notepad.
- This tool allows us to type in our statements, etc, and see the results

SQL * Net :-

- This is Oracle's own middleware product which runs on both the client and server to hide the complexity of the network.
- SQL * Net's multiprotocol interchange allows client / server connections to span multiple communication protocols without the need for bridges and routers, etc. SQL * Net will work with any configuration design.

SQL * LOADER :-

- A utility used to load data from external files into Oracle tables.
- It can load load data from as ASCII fixed format or delimited file into on oracle table.

Serializability :-

Serializability is a property of a transaction schedule which is used to keep the data in the data item in consistent state. It is the classical concurrency scheme.

Serializability is Required :-

→ To control concurrent execution of transaction
→ To ensure that the database state remains consistent.

Serializability of Schedule (history) :-

- In DBMS, the basic assumption is that each transaction preserves database consistency.
- Thus, the serial execution of a set of transaction preserves database consistency.
- A concurrent schedule is serializable if it is equivalent to a serial schedule.

Schedule S

T_1	T_2
R(A)	
W(A)	
	R(A)
	W(A)
R(B)	
W(B)	
	R(A)
	W(B)
	Commit

Conflict Serializability :

→ Let us consider a schedule S, in which there are two consecutive instructions I_i and I_j of transaction T_i and T_j respectively ($i \neq j$)

→ If I_i and I_j refer to different data items, then we can swap I_i and I_j without affecting the results of any instruction in the schedule.

→ However, if I_i and I_j refer to the same data item Q, then the order of the two steps may matter.

Following are four possible cases :-

I_i	I_j	Swapping Possible.
Read (Q)	Read(Q)	Yes
Read (Q)	Write(Q)	No
Write(Q)	(Read(Q)	No
Write(Q)	Write(Q)	No

→ I_i and I_j conflict if there are operations by different transactions on the same data item, and atleast one of these instructions is a write operation.

for Example

Schedule S

T_1	T_2
R(A)	
W(A)	
	R(A)
	W(A)
R(B)	
W(B)	
	R(B)
	W(B)

The write(A) instruction of T_1 confits with read(A) instruction of T_2. However, the write(A) instruction of T_2 does not conflict with the read(B) instruction of T_1 as they access different data items.

Schedule S'

T_1	T_2
R(A)	
W(A)	
	R(A)
R(B)	
	W(A)
W(B)	
	R(B)
	W(B)

Since the W(A) instruction of T_2 in schedule S' does not conflict with the R(B) instruction of T_1, we can swap these instructions to generate an equivalent schedule.

→ If a schedule S can be transformed leads to the concept of swaps of non-conflicting instructions, we say that S & S' are not conflict equivalent.

→ The concept of conflict equivalence leads to the concept of conflict serializability and the schedule S is conflict serializable.

View Serializability :-

The schedule S and S' are said to be view equivalent if following three conditions are met :-

(a) for each data item Q, if transaction T_i reads the initial value of Q in schedule S, then T_i in schedule S', must also read the initial value of Q.

(b) for each data item Q if transaction T_i executes read(Q) in schedule S and if that value was produced by a write(Q) operation executed by transaction T_j, then the read(Q) operation of transaction T_i, in schedule S', must also read the value of Q that was produced by the same write(Q) operation of transaction T_j.

(c) for each data item Q, the transaction (if any) that performs the final write(Q) operation in S must perform the final write(Q) operation in schedule S'.

Conditions (a) and (b) ensure that each transaction reads the same values in both schedules and therefore, performs the same computation. Condition (c), coupled with condition (a) and condition (b) ensure that both schedules result in the same final system state.

- The concept of view equivalence leads to the concept of view serializability.
- We say that schedule S is view serializable, if it is view equivalent to serial schedule.
- Every conflict serializable schedule is also view serializable but there are view serializable schedules that are not conflict serializable

Schedules S_1 and S_2 are view equivalent as :-

Schedule S_1

T_1	T_2
R(A)	
W(A)	
R(B)	
W(B)	
	R(A)
	W(A)
	R(B)
	W(B)

Schedule S_2

T_1	T_2
R(A)	
W(A)	
	R(A)
	W(A)
R(B)	
W(B)	
	R(B)
	W(B)

→ T_1 reads initial value of data item A in S_1 & S_2.
→ T_2 reads value of data item A written by T_1 in S_1 and S_2.
→ T_2 writes final value of data item in A in S_1 and S_2.

Difference between Conflict and View Serializability:

Conflict Serializability	View Serizability
→ Easy to achieve.	Difficult to achieve.
→ Cheaper to test.	Expensive to test.
→ Every conflict serializable is view serializable.	Every view serializable may not be conflict serializable.
→ Used in most concurrency control scheme.	Not used in concurrency control scheme.

Log Based Recovery :-

- The log/system is a sequence of log records, recording all the update activities in the database.
- Various types of log records are denoted as :-
  - $\langle T_i \text{ start} \rangle$: Transaction T_i has started.
  - $\langle T_i, X_j, V_1, V_2 \rangle$: Transaction T_i has performed a write on data item X_j. X_j had value V_1 before the write, and will have the value V_2 after the write.
  - $\langle T_i \text{ commit} \rangle$: Transaction T_i has commited.
  - $\langle T_i \text{ abort} \rangle$: Transaction T_i has aborted.
- Whenever a transaction performs a write, it is essential that the log record for that write be created before the database is modified.

Log based Recovery :- Log based recovery is a method to ensure atomicity using log when failure occurs. In log based recovery, following two techniques are used to ensure atomicity and to maintain log :-

1. Deferred database modification :-

→ The defered database modification technique ensures transaction atomicity by recording all database modifications in the log, but deferring the execution of all write operations of a transaction until the transaction partially commits.

→ When a transaction partially commits, the information on the log associated with the transaction is used in executing the deferred write.

features :-

- All logs written onto the database is updated when a transaction commits.
- It does not require old value of data item on the log.
- It do not need extra I/O operation before commit time.
- It can manage with large memory space
- Locks are held till the commit point.

2. Immediate database modification :-

→ The immediate database modification technique allows database modifications to be output to the database while the transaction is still in the active state.

→ Data modification written by active transactions.

Features :-

- All logs written onto the database is updated immediately after every operation.
- It requires both old and new value of data item on the log.
- It needs extra I/O operation to flush out block buffer.
- It can manage with less memory space.
- Locks are released after modification.

Checkpointing :-

→ It is a process of saving a snapshot of the application's state, so that it can restart from that point in case of failure.

→ Checkpoint is a point of time at which a record is written onto the database from the buffers.

→ Checkpointing shortens the recovery process.

Types of checkpointing techniques:-

(i) Consistent checkpointing :-

- It creates a consistent image of the database at checkpoint.
- During recovery, only those transactions which take place after last checkpoint are undone or redone.
- The transactions that takes place before the last consistent checkpoint are already committed and need not be processed again.
- The actions taken for checkpointing are :-
  - All changes in main-memory buffers are written onto the disk.

- A "checkpoint" record is written in the transaction log.
- The transaction log is written to the disk.

2 Fuzzy Checkpointing :-

- In fuzzy checkpointing, at the time of checkpoint, all the active transactions are written in the logs.
- In case of failure, the recovery manager processes only those transaction that were active during checkpoint & later.
- The transactions that have been committed before checkpoint are written to the disk and hence need not be redone.

Significance of Checkpointing :-

- Output onto stable storage, all log records currently residing in main memory.
- Output to the disk all modified buffer blocks.
- Output onto stable storage a log record of the form <checkpoint L>, where L is a list of transactions active at the time of the checkpoint.
- The presence of a <checkpoint L> record in the log allows the system to streamline its recovery procedure.

Deadlock :-

→ A deadlock is a situation in which two or more transactions are waiting for locks held by the other transaction to release the lock.

→ Every transactions is waiting for another transaction to finish its operations.

Methods to handle a Deadlock :-

(i) Deadlock Prevention Protocol :- This protocol ensures that the system will not go into deadlock state. There are different methods that can be used for deadlock prevention :-

(a) Pre-declaration method :- This method requires that each transaction locks all its data item before it starts its execution.

(b) Partial ordering method :- In this method, system imposes partial ordering of all data items and requires that a transaction can lock a data item only in the order specified by partial order.

(c) Timestamp method :- In this method, the data item are locked using timestamp of transaction.

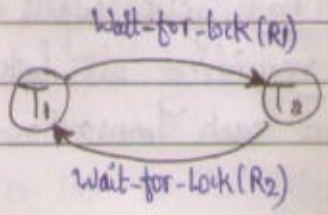
(ii) Deadlock Detection :-

→ When a transaction waits indefinitely to obtain a lock, system should detect whether the transaction is involved in a deadlock or not.

→ Wait-for-graph is one of the methods for detecting the deadlock situation.

→ In this method a graph is drawn based on the transaction and their lock on the resource.

→ If a graph is created has a closed loop or a cycle, then there is a deadlock.



(iii) Recovery from deadlock :-

(a) Selection of victim :- In this we determine which transaction (or transactions) to rollback to break the deadlock. We should rollback those transactions that will incur the minimum cost.

(b) Rollback :- The simplest solution is a "total rollback". Abort the transaction and then restart it.

(c) Starvation :- In a system where selection of transactions, for rollback, is based on the cost factor, it may happen that the same transactions are always picked up.

(iv) Deadlock Avoidance :-

(a) Serial access :- If only one transaction can access the database at a time, then we can avoid deadlock.

(b) Autocommit transaction :- It includes that each transaction can only lock one resource immediately as it uses it, then finishes its transaction and releases its lock before requesting any other resource.

(c) Ordered Updates :- If transactions always request resources in the same order (Ex numerically ascending by the index value of the row being locked) then system do not enters in deadlock state.

(d) By rolling back conflicting transactions.

(e) By allocating the locks where needed.

Deadlock Prevention Protocols :-

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. This approach requires that each transaction locks all the data items

before it begins execution. It is required that, either all data items should be locked in one step, or none should be locked.

- Second approach for preventing deadlocks is to use preemption and transaction rollbacks. To control preemption we assign a unique timestamp to each transaction. The system uses these timestamps only to decide whether a transaction should wait or rollback.

Two different deadlock prevention schemes using timestamps are:-

(a) Wait-die scheme:- In this scheme, if a transaction request to lock a resource (data item), which is already held with conflicting lock by some other transaction, one of the two possibilities may occur:-

→ If $TS(T_i) < TS(T_j)$, ie. T_i, which is requesting a conflicting lock, is older than T_j, T_i is allowed to wait until the data item is available.

→ If $TS(T_i) > TS(T_j)$, ie. T_i is younger than T_j, so T_i dies. T_i is restarted later with random delay but with same timestamp.

This scheme allows the older transaction to wait

(b) Wound-wait scheme:-

→ In this scheme, if a transaction request to lock a resource (data item), which is already held with conflicting lock by some other transaction, one of the two possibilities may occur.

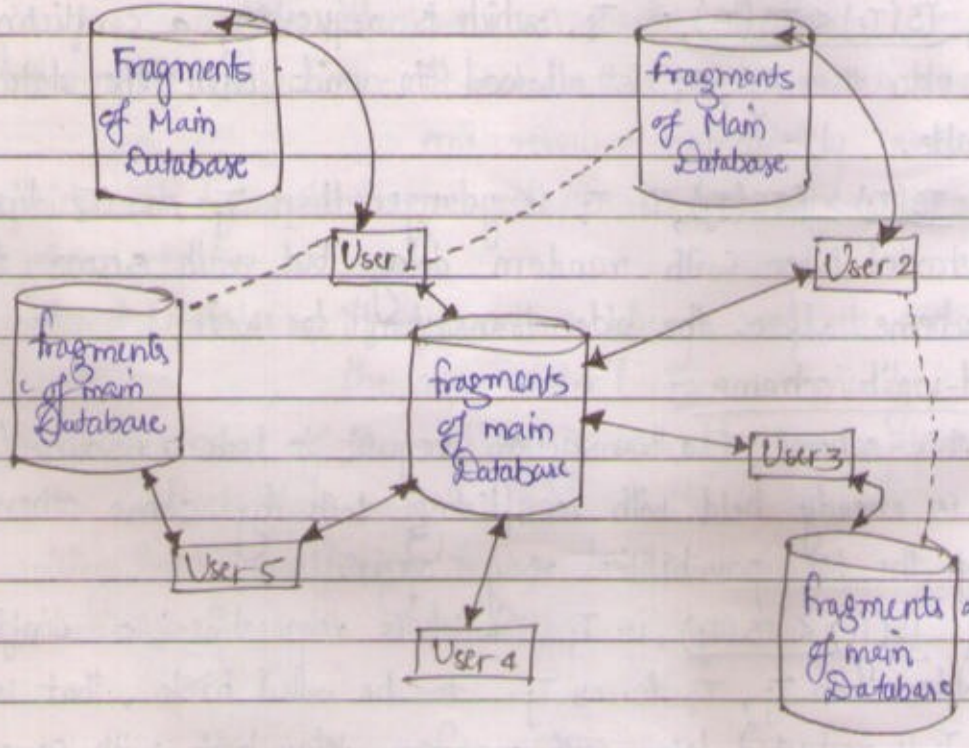
→ If $TS(T_i) < TS(T_j)$, ie. T_i, which is requesting a conflicting lock, is older than T_j, T_i forces T_j to be rolled back, that is T_i wounds T_j. T_j is restarted later with random delay but with same timestamp.

↳ If $TS(T_i) > TS(T_j)$ ie., T_i is younger than T_j, T_i is forced to wait until the resource (ie. data item) is available.

This scheme, allows the younger transaction to wait but when an older transaction request an item held by younger one, the older transaction forces the younger one to abort and release the item. In both cases, transactions which enters late in the system, is aborted.

Distributed DBMS :-

- A distributed database system consists of collection of sites, connected together through a communication network.
- Each site is a database system site in its own right and the sites have agreed to work together, so that a user at any site can access anywhere in the network as if the data were all stored at the user's local site.
- Each site has its own local database.
- A distributed database is fragmented into smaller data sets.
- DDBMS can handle both local and global transactions.



Advantages :-

- DDBMS allows each site store and maintain its own database, causing immediate and efficient access to data.
- It allows access to the data stored at remote sites. At the same time users can retain the control to its own site to access the local data.

- If one site is not working due to any reason (i.e. communication link goes down) the system will not be down because other sites of the network can possibly continue functioning.
- New sites can be added to the system anytime with no or little efforts.
- If a user needs to access the data from multiple sites then the desired query can be subdivided into sub-queries and the parts equivocal in parallel.

Functional Dependency :-

Functional dependency is a set of constraints between two attributes in a relation. Functional dependency says that if two tuples have same values for attributes $A_1, A_2 \ldots A_n$ then those two tuples must have to have same values for attributes $B_1, B_2 \ldots B_n$.

Functional dependency is represented by :-

$$X \rightarrow Y$$

where X functionally determines Y.

Armstrong Axioms :- If F is a set of functional dependencies then the closure of F, denoted as F^+, is a set of all functional dependencies logically implied by F.

Armstrong axiom's are set of rules, that when applied repeatedly generates a closure of functional dependencies :-

→ **Reflexive Rule :-** If α is a set of attributes and β is subset of α then α holds β.

→ **Augmentation Rule :-** If $a \rightarrow b$ holds and y is attribute set, then $ay \rightarrow by$ also holds. That is adding attributes in dependencies does not change the basic dependencies.

→ **Transitivity Rule :-** Same as transitive rule in algebra, if $a \rightarrow b$ holds and $b \rightarrow c$ holds then $a \rightarrow c$ also holds.

Trivial Functional Dependency

Trivial	Non-trivial	Completely Non-Trivial
$X \rightarrow Y, Y \subset X$	$X \rightarrow Y, Y \not\subset X$	$X \rightarrow Y, X \cap Y = \phi$

Normalization :-

↔ Normalization is the process of minimizing redundancy from a relation or set of relations. Redundancy in relation may cause insertion, deletion and updation anomalies.

↔ Normalization split a large table into smaller tables and define relationships between them to increase the clarity in organizing data.

↔ If a database design is not perfect, it may contain anamolies which are like a bad dream for any database administrator. Managing a database with anamolies is next to impossible.

Update anomalies :- If data items are scattered and are not linked to each other properly, then it could lead to strange situations.

Ex When we try to update one data item having its copies scattered over several places, a few instances get updated properly while a few others are left with old values. Such instances leave the DB in an inconsistent state.

Deletion anomalies :- We tried to delete a record, but parts of it was left undeleted because of unawareness, the data is also saved somewhere else.

Insert anomalies :- We tried to insert data in a record that does not exist at all.

(i) First Normal form :-

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

Example :-

Employee

Emp_id	Emp_name	Emp_Address	Emp_mob
101	A	New Delhi	9415868171
102	B	Kanpur	9458707573
103	C	Madras	9458707573
			7599908337
104	D	Banglore	7269095504

Employee

Emp_id	Emp_name	Emp_address	Emp_mob
101	A	New Delhi	9415868171
102	B	Kanpur	9458707573
103	C	Madras	9458707573
103	C	Madras	7599908337
104	D	Banglore	7269095504

This table is not in 1NF as the rules says "Each attribute of a table must have atomic value" the Emp_mob values for c Employee violates the rule

Second Normal form (2NF):-

A table is said to be in 2NF if both the following conditions hold:-
Table is in 1NF (first Normal form)

* No non-prime attribute is dependent on the proper key subset of any
* candidate key of table.

An attribute that is not part of any candidate key is known as non-prime attribute.

Ex

Teacher_id	Subject	Teacher_age
101	Maths	38
101	Physics	38
222	Biology	38
333	Physics	40
333	Chemistry	40

teacher_subject

id	subject
101	Maths
101	Physics
222	Biology
333	Physics
333	Chemistry

teacher_details

id	age
101	38
222	38
333	40

Candidate Key:- {teacher_id, subject}
Non-prime attribute :- {teacher_age}

Third Normal form (3NF) :-

A table design is said to be in 3NF if both the following conditions hold :-

* Table must be in 2NF.
* Transitive functional dependency of nonprime attribute on any super key should be removed.

Any attribute that is not part of any candidate key is known as non-prime attribute.

Ex

Emp_id	Empname	Emp_zip	Empstate	Empcity	Emp_district
1001	John	282005	UP	Agra	DayalBagh
1002	Ajeet	222008	TN	Chennai	M-city
1006	Lora	282007	TN	Chennai	Urrapakkam
1101	Lilly	292008	UK	Pauri	Bhagwan
1201	Steve	222999	mp	Gwalior	Ratan

Super Key :- {empid}, {empid, emp_name}, {empid, emp_name, emp_zip}----

Candidate Key :- {emp id}

Employee

empid	empname	emp_zip
1001	John	282005
1002	Ajeet	222008
1006	Lora	282007
1101	Lilly	292028
1201	Steve	222999

Employee_zip table :-

emp_zip	emp_state	emp_city	emp_district
28005	UP	Agra	DayalBagh
222008	TN	Chennai	M-city
282007	ChennTN	"	Urrapakkam
292028	UK	Pauri	Bhagwan
222999	MP	Gwalior	Ratan

Boyce Codd Normal form (BCNF):-

It is an advance version of 3NF thats why it is also refered as 3.5 NF. BCNF is stricter than 3NF. A table complies with BCNF if it is in 3NF and for every functional dependency $X \rightarrow Y$, X should be the superkey of the table.

Ex

emp_id	emp_nationality	emp_dpt.	emp_type	dept_type	dept_no_of_emp
1001	Austrian	Production	D001		200
1001	"	Planning	D001		250
1002	American	Design	D134		100
1002	"	Purchasing	D134		600

functional dependencies :-

emp_id → emp_nationality

emp_dept → {dept_type, dept_no_of_emp}

Candidate Key :- {emp_id, emp_dept}

Emp_nationality Table

emp_id	emp_nationality
1001	Austrian
1002	American

Emp_dept_table

emp_dept	dept_type	dept_no_of_emp
Production	D001	200
Planning	D001	250
Design	D134	100
Purchasing	D134	600

Emp_dept_mapping table

emp_id	emp_dept
1001	Production
1001	Planning
1002	Design
1002	Purchasing

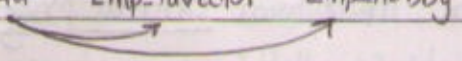
Forth Normal form (4NF) :-

Fourth Normal form comes into picture when Multi-valued dependency occur in any relation.

Rules for 4NF form :-

* It should be in Boyce-Codd Normal form.
* And, the table should not have any Multi-valued-Dependency.

Ex

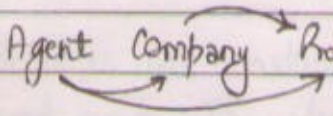


Fifth Normal form / Projected Normal form :-

A relation R is in 5NF if and only if satisfies the following conditions :-

* R should be already in 4NF.
* It should cannot be further non-loss decomposed (Join Dependency)

Ex



PL/SQL Programming Language :-

PL/SQL programming language was developed by Oracle Corporation in the late 1980's as procedural extension language for SQL and the Oracle relational database.

→ PL/SQL is a completely portable, high performance transaction processing.

→ PL/SQL provides a built-in interpreted and OS independent programming environment.

→ PL/SQL ~~can~~ call can also be made from external programming language calls to database.

→ PL/SQL's general syntax is based on that of ADA and Pascal programming language.

→ Apart from Oracle, PL/SQL is available in Times Ten in-memory database and IBM DB2

Features :-

- PL/SQL is tightly integrated with SQL.
- It offers extensive error checking.
- It offers numerous datatypes.
- It offers a variety of programming structures.
- It supports structured programming through functions & procedures.
- It supports OOP.
- It supports the development of web applications & server pages

Syntax of PL/SQL :-

PL/SQL is a block structured language, ie: PL/SQL programs are divided and written in logical blocks of code. Each block consists of three sub parts :-

(i) Declarations
(ii) Executable Commands
(iii) Exception Handling

Syntax:
```
DECLARE
    < declaration section>
BEGIN
    < executable command(s)>
EXCEPTION
    <exception handling>
END;
```

Ex
```
DECLARE
    message varchar(20):= 'Hello, World!';
BEGIN
    dbms_output.put_line (message)
END;
```

PL/SQL Identifiers :-

PL/SQL identifiers are constants, variables, exceptions, procedures, cursors, and reserved words. The identifier consists of a letter optionally followed by more letters, numerals, dollar signs, underscores and number signs and should not exceed 30 characters.

→ Identifiers are not case sensitive.

The PL/SQL Delimeters :-

A delimeter is a symbol with special meaning.

+, -, *, /	Addition / Subtraction / Multiplication / Division
%.	Attribute Indicator
'	Character String Delimeter
.	Component Selector
:	Host variable Delimeter
,	Item Separator

=	Relational Operator
@	Remote Access Indicator
;	Statement terminator
:=	Assignment Operator
||	Concatenation operator
/*, */	Multi line comment
--	Single line comment

PL/SQL Program Units :-

→ PL/SQL Block
→ function
→ Package
→ Package body
→ Procedure
→ Trigger
→ Type
→ Type Body

Datatypes in PL/SQL :-

- Scalar (Number, Date, Bodean)
- Large Object (LOB) Ex text, graphic images, audio clips, video clips
- Composite (collections or records)
- Reference (Pointers)

Failure Classification :-

To find that where the problem has occurred, we generalize a failure into the following categories :-

→ Transaction failure
→ System Crash
→ Disk failure

Transaction failure :- The transaction failures occurs when it fails to execute or when it reaches a point from where it can't go any further. If a few transaction or process is hurt, then this is called transaction failure.
Reasons for a transaction failure could be :-

→ Logical Error
→ Syntax Error

System Crash :- It can occur due to power failure or other h/w or s/w failure.
Ex Operating System error.

Disk failure :- It occurs when hard disk drives or storage drives used to fail frequently. It was a common problem in early days of technology evaluation.
Disk failures occurs due to the formation of bad sectors, disk head crash, and unreachability to the disk or any other failure, which destroy all part of disk storage

Log Based Recovery — Recovery using Log Records

- Deferred Database Modification
- Immediate Database Modification

Checkpoints

16/08/19

DBMS

(1) Lodging is a many to many relationship. Rent, payment to be made by persons occupying different hotel rooms should be added as an attribute to

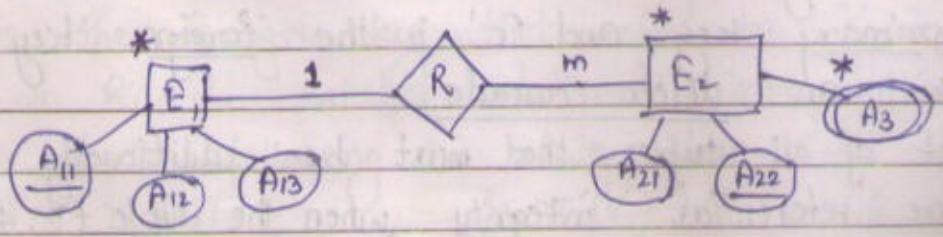
(a) Hotel
(b) ~~Log~~ Lodging
(c) Person
(d) NONE

(2) Given the basic ER and relational model. Which of the following is incorrect.

(a) an attribute of an entity can have more than one value.
(b) an attribute of an entity can be composite
(c) in a row of a relational table an attribute can have more then one value.
(d) In a row of a relational table an attribute can have exactly one value or a null value.

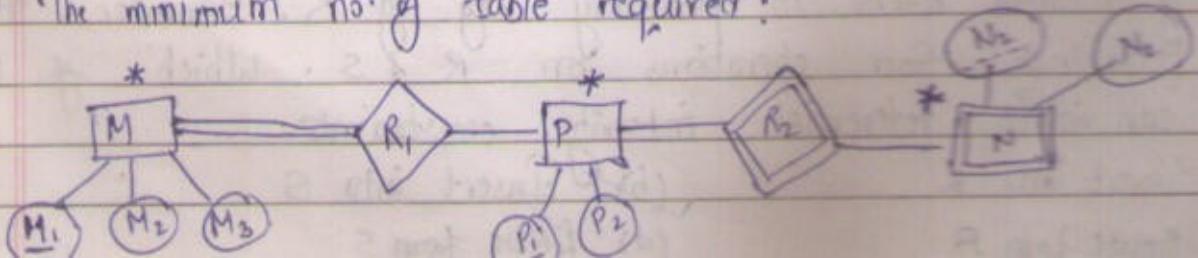
(3) The minimum no. of tables for following ER diagram is :-

(a) 1 (b) 2 (c) 3 (d) 4



(4) E_1 & E_2 are two entities having R_1 & R_2 two relationships b/w E_1 & E_2. R_1 is one to many & R_2 is many to many. E_1 & R_2 doesn't have their own attribute. What is minimum no. of tables required?

(5) The minimum no. of table required?



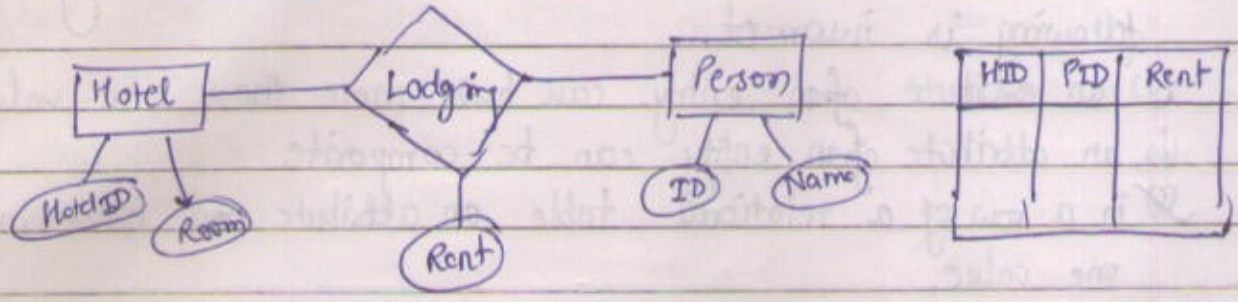
(a) 2 (b) 3 (c) 4 (d) 5

Set for one of the tables for the correct answer to the above question is

(a) $M_1 M_2 M_3 P_1$ (b) $M_1 P_1 N_1 N_2$

(c) $M_1 P N$ (d) $M_1 P_1$

Ans (1)



(2)

23/09/19

1. The following table has two attributes 'A' & 'C' where A is the primary key and 'c' is the foreign key referencing 'A' with on delete cascade.

The set of all tuples that must be additionally deleted to preserve referential integrity when the tuple (2,4) is deleted as

(a) (3,4) & (6,4)

(b) (5,2) & (7,2)

(c) (5,2) & (7,2) & (9,5)

(d) (3,4) & (4,3) & (6,4)

(PK) A	C (FK)
2	4
3	4
4	3
5	2
7	2
9	5
6	4

2. Let R(a,b,c) and S(d,e,f) be two relation, d is foreign key of 'S' that refers to primary key of 'R'. Consider the following four operations on R & S. Which of these can violate referential integrity constraints.

(a) Insert into R (b) Insert into S

(c) Delete from R (d) Delete from S

→ systemctl stop firewalld
→ setenforce 0
→ systemctl start httpd
→ cd /var/www/html

3. Which of the following query transformation is incorrect. R_1 & R_2 are relations, C_1 & C_2 are conditions & A_1 & A_2 are attributes of R.

(a) $\sigma_{C_1}(\sigma_{C_2}(R_1)) \rightarrow \sigma_{C_2}(\sigma_{C_2}(R_1))$

(b) $\sigma_{C_1}(\Pi_{A_1}(R_1)) \rightarrow \Pi_{A_1}(\sigma_{C_1}(R_1))$

(c) $\sigma_{C_1}(R_1 \cup R_2) \rightarrow \sigma_{C_1}(R_1) \cup \sigma_{C_2}(R_2)$

(d) $\Pi_{A_2}(\sigma_{C_1}(R_1)) \rightarrow \sigma_{C_1}(\Pi_{A_2}(R_1))$

4. Suppose $R_1(A,B)$ and $R_2(C,d)$ are two relational schema with r_1 and r_2 be the corresponding relation instances. 'B' is the foreign key that refers to 'c' in R_2. If data in R_1 & R_2 satisfied relational integrity constraints which of the following is always true.

(a) $\Pi_B(r_1) - \Pi_C(r_2) = \phi$
(b) $\Pi_C(r_2) - \Pi_B(r_1) = \phi$
(c) $\Pi_B(r_1) = \Pi_C(r_2)$
(d) $\Pi_B(r_1) - \Pi_C(r_2) \neq \phi$

A theta join allows for arbitrary comparison relationships (such as ≥)
An equi join is a theta join using the equality operator.
A natural join is an equijoin on attributes that have the same name in each relationship.

OPERATIONS OF RELATIONAL ALGEBRA

Operation	Purpose	Notation
SELECT	Select all tuples that satisfies the selection condition from a relation R	$\sigma_{\langle \text{selection condition} \rangle}(R)$
PROJECT	Produces a new relation with only some of the attributes of R_1 and removes duplicate tuples	$\pi_{\langle \text{attribute list} \rangle}(R)$
THETA JOIN	Produces all combinations of tuples from R_1 & R_2 that satisfies the join condition	$R_1 \bowtie_{\langle \text{join condition} \rangle} R_2$
EQUIJOIN	Produces all the combinations of tuples from R_1 and R_2 that satisfies a join condition with only equality comparisons.	$R_1 \bowtie_{\langle \text{join condition} \rangle} R_2$ OR $R_1 \bowtie_{(\langle \text{join condition 1} \rangle), (\langle \text{join attribute 2} \rangle)} R_2$
NATURAL JOIN	Same as EQUIJOIN except that the join attributes of R_2 are not included in the resulting relation; if the join attributes have the same names, they do not have to be specified at all.	$R_1^* \bowtie_{\langle \text{join condition} \rangle} R_2$ OR $R_1^* * R_2$, or $R_1 *_{(\langle \text{join attribute 1} \rangle), (\langle \text{join attribute 2} \rangle)} R_2$
UNION	Produces a relation that includes all the tuples in R_1 or R_2 or both R_1 and R_2; R_1 and R_2 must be union compatible	$R_1 \cup R_2$
INTERSECTION	Produces a relation that includes all the tuples in both R_1 and R_2; R_1 and R_2 must be union compatible	$R_1 \cap R_2$
SET DIFFERENCE	Produces a relation that includes all the tuples in R_1 that are not in R_2; R_1 and R_2 must be union compatible	$R_1 - R_2$
CARTESIAN PRODUCT	Produces a relation that has the attributes of R_1 and R_2 and includes as tuples all possible combinations of tuples from R_1 & R_2.	$R_1 \times R_2$

DIVISION Produces a relation $R(x)$ that includes all tuples $t[x]$ in $R_1(z)$ that appear in R_1 in combination with every $R_1(z) \div R_2(z)$
tuple from $R_2(y)$, where $z = x \cup y$.

Differentiate between homogenous and heterogenous distributed database management system :-

Homogenous DDBMS	Heterogenous DDBMS
• Every site has identical DBMS s/w. (softwares)	Different sites may use different DBMS s/w.
• Every site aware of one another.	No sites aware of one another.
• All sites have common schema.	All sites have different schema
• Easy to manage	Difficult to manage.
• Easy to design	Difficult to design.

Cursors :-

→ A cursor is a temporary work area created in the system memory when a SQL statement is executed.

→ A cursor contains information on a select statement & the rows of data accessed by it.

→ A cursor can hold more than one row, but can process only one row at a time

→ The set of rows the cursor holds is called the active set.

Implicit Cursors:- These are created by default when DML statements like INSERT, UPDATE & DELETE executed.

They are also created when a SELECT statement that returns just one row is executed.

Explicit Cursors :- They must be created when we are executing a SELECT statement that returns more than one row.

When we fetch a row the current row position moves to next row

Triggers :-

→ A trigger is a procedure (code segment) that is executed automatically when some specific events occur in a table/view of a database.
→ Triggers are mainly used for maintaining integrity in a database.
→ Triggers are also used for enforcing business rules, auditing changes in the database and replicating data.
→ Most common triggers are DML triggers. These triggers are specially used for auditing.

DML Triggers (AFTER, INSTEAD OF)

Triggers is a statement that a system executes automatically when there is any modification to the database. In a trigger, we first specify when the trigger is to be executed and then the action to be performed when the trigger executes. Triggers are used to specify certain integrity constraints that cannot be specified using the constraint mechanism of SQL.

TYPES OF TRIGGERS :-

AFTER INSERT
AFTER UPDATE
AFTER DELETE
BEFORE INSERT
BEFORE UPDATE
BEFORE DELETE

Benefits of Triggers

→ Generating some derived column values automatically.
→ Enforcing referential integrity
→ Event logging and storing information on table access.
→ Auditing
→ Synchronous replication of tables
→ Imposing security authorizations
→ Preventing invalid transactions