Paper Pattern

MODULE 1: Direct theory Q (5 marks)
Design Q (10 marks) {time consuming?
Query in SQL
Query in relational algebra
5m Serializibility Problem (15M) Q on transaction, ~~Q on~~ } mathematical problem
Q on normalisation
Q on query processing (theory)

- Data: Any raw fact collected for your application.
- Information: Data which is described

Eg: Data: 2021700001
Information: This is UID number assigned.

- Knowledge: How to use information
- Wisdom: How to use knowledge, decision making.

~~Eg: Knowledge: ???~~

- Database: Collection of data containing information relevant to an enterprise.

Eg: Employee database

- DBMS: It is a collection of interrelated data and set of programs to access the data. Goal: store and retrieve database info that is both convenient and efficient.
- Database management applications:

• Bank	• Education	• advertisements
• Airline	• Social media	• weather forecast
• Industry	• Medical purposes	• navigation systems

- Purpose of database:

• To store the data
• To provide structure for data (organization)
• To provide a mechanism for query, to create, read, modify and delete data. CRUD (create, read, update, delete) are basic operations. (management)

- Schema: Overall design of the database. Basic structure, does not changes as website should be user-friendly.
- Instances: Dif data added time after time. Collection of info stored in a database at a particular moment. Instances change.
- Advantages of file processing: (pen & paper file)
  - cheap
  - can be used by anyone, no experts required
  - Its easier when dealing with small amount of data.
- Disadvantages:
  - cannot be used by multiple users at the same time (Data isolation not possible)
  - Can easily be lost, human errors
  - Repetition of same data many number of times. (redundancy)
  - Time consuming, not environment friendly.
  - No back up, new file required for new task.
  - Inconsistency, manipulation of data is difficult.
  - ~~Data isolo~~ Hard to access (eg: to see roll no 54 we have to scan the data of 53 students first)
  - Data handling is difficult (multiple file formats)
- Disadvantages of database system:
  - cost is high, requires skilled labour
  - not required for small amt of data.

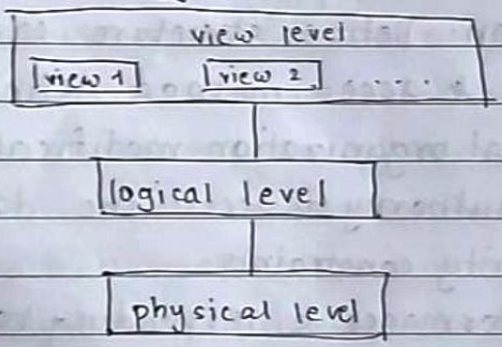
- Quiz:

(Q) One application of DBMS: Bank, Education...
(Q) Data duplication is termed as? Data redundancy.
(Q) Schema is? Structure
(Q) DBMS is? Database Management system
(Q) Collection of info stored in database at particular moment: Instance
(Q) Database is a collection of data containing info related to an enterprise.
(Q) DBMS is a collection of interrelated data & set of programs to access data.

(a) Data redundancy is also called as?
(b) Write any 2 purpose of database:

(c) Write anyone application of database:
(d) Information is ?
(e) Write 2 examples of database software : SQL, oracle, module
(f) CRUD stands for ? Create, read, up
(g) Write suitable application for file processing system:

- Data Abstraction: Hiding complexity from users to make the product user friendly. Three levels:



- Physical level: How the data is actually stored, where it is stored.
- Logical level: Just about the data and its relationship Eg: Name is of type string.
- View level: How data is organized according to user requirements.
- Physical level data independence: Modifying data at physical level without affecting logical & view level. Required to increase performance of the application.
- Logical level data independence: Modifying data at logical level without affecting view level. (Read its use from TB)
- Data independence: Ability to modify data at an abstraction level without affecting a higher level.

- Database users.
  - Naive users : Unsophisticated users who don't really know about the application but interact with it directly.
  - Application programmers: People who actually develop the software
  - Sophisticated users: Access the software by filing a query. ← manipulate the data, hv read/write permissions
  - Specialized users: ~~People~~ Skilled people who think about the knowledge of the ~~dsof~~ data and use it for various processes.
  - User behind the screen: Involved indirectly. Eg: people handling back up & recovering.
- Database Administrator (DBA): Owner. One who has central control over data and application. Duties:
  - Schema definition: Defines structure, can manipulate it
  - Storage structure & access method definition
  - Schema & physical organization modification
  - Granting user authority to access the database, permissions
  - Specifying integrity constraints
  - Monitoring performance & responding to changes in requirements i.e, physical level data independence.
  - Routine maintenance : Back up, free disk space
- ** Overall System structure (10 mark question) Draw & explain
  - Disk Storage • data, indices, data dictionary, statistical data
  - storage manager • Buffer manager, file manager, authorization & integrity manager, transaction manager.
  - Query processor • compiler & thinker, application program object code, query evaluation engine, DML queries, DML compiler and organizer, DDL interpreter
  - application interfaces, application programs, query tools, administration tools.

- 12 Codd's rules: (Read from TB, remember Rule No too)

(1) Information rule: represent explicitly at logical level

(2) Guaranteed Access: Each atomic value is guaranteed to be logically accessible by table name, primary key, column name.

(5) Comprehensive data sublanguage: There must be at least one language whose statements are expresible per some defined syntax.

(6) View updating rule: All views that are theoretically updatable are also practically updatable by the system.

(3) Systematic treatment of null values: All the missing info must be planed systematically.

(4) Dynamic online ~~catalog~~ model: Catalog based on relational model data.

(7) High level insert, update, delete: The capability of handling base relational or a derived as a

(8) Physical data independence: Application program & terminal activities remain physically unimpaired.

(9) Logical data independence: Application program & terminal activities remain logically unimpaired.

(10) Integrity Independence: (a) Entity integrity (b) Referential integrity

(11) Distribution Independence: It implies that users should not have to be aware of whether a database is distributed.

(12) Non subversion rule: If the relational system has a low level language, that language cannot be used to subvert or bypass the integrity rules in the higher level language.

Benifits of ER Model
- Easy to understand.
- Focuses only on essential parts
- Easy to identify alternative paths
- Easy to debug
- Easy to modularize

Limitation of ER:
- Cannot have relation with relation. Thats why we need ER (aggregation)
- Cannot focus on an entity of entity set. You hv employee but can't say what kind of employee. (Specialization, generalization not there)

classmate Date Page

★ INTRODUCTION TO ER MODEL:

- Entity Relationship model
- Phases of database modeling:
  Objects of interest, conceptual design (schema), logical design, physical design

(1) Conceptual design for a relational database:
- Input - Business requirements.
- Process - Mapping the requirements in terms of entities, relations, attributes, primary keys and cardinalities.
- Output - ER diagram

(2) Logical design for a relational database
- Input - ER diagram
- Process - Mapping the ER diagram ~~to~~ normalised ER model
- Output - Normalised relational model

(3) Physical design
- Input - Normalised relational model
- Process - Perform operations like CRUD on SQL.
- Output - Physical database.

- Components of ER model:
- Entity: Object ~~and~~ or thing which is distinguishable from other objects in the given world. It has properties:
  (1) Easily distinguishable
  (2) Shd hv its own properties
  (3) Shd play an important role. Shd hv atleast one key to access the entity ~~be accessed~~.
- Entity set: Collection of entities having similar properties.
- We work with entity sets, not entity, so that large amount of data is available.
- Two types:

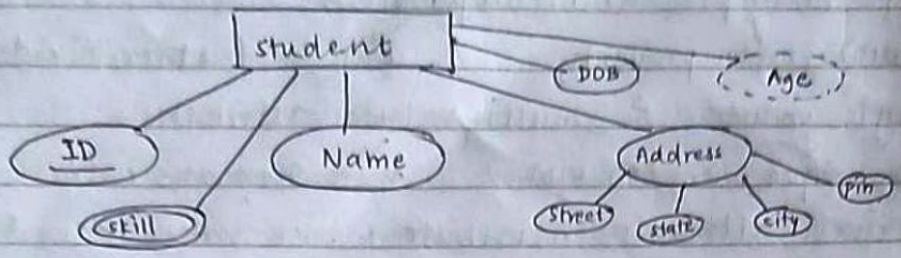
(1) Strong entity: Has primary key to access individual entity and differentiate it from an entity set. Unique & not NULL.
(2) Weak entity: Does not have a primary key. Dependent.
- Attributes: Properties which describes an entity. Its basically your columns in relational model

- Types (1) simple and composite.
  (Eg: VID) single property — can be further divided (Eg: Address).
  (2) Single valued & multi-valued attributes.
  One unique entry (eg: VID) — more than one entries allowed (Eg: Phone no).
  (3) Derived attribute: Attribute whose value can be calculated from another attribute. Eg: Age from DOB. ∴ Age is derived attribute but only when DOB attribute exists.
  (4) NULL attributes: Unimportant attributes which can be assigned null value, i.e, optional entry.
- Eg: Teacher (Name, address, Ph-No (can be NULL), Subject, faculty id, DOB, age)
  Name: composite; Ph-No: multi-valued; faculty id: single simple; age: derived
- Relationships: An association between 2 or more entities, of significance to the enterprise.
  Eg: employee —WORKS FOR→ company
- Symbols used in ER notation – Peter Chen

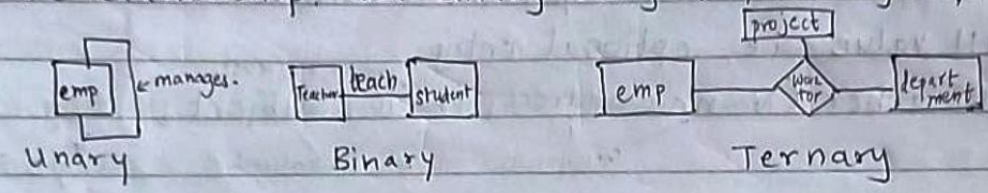
(1) E — Entity Set
(2) E — Weak entity set
(3) R — Identifying Relationship set for weak entity set
(4) A — Multivalued attribute
(5) R — E — Total Participation of Entity set in Relationship.
(6) A — ~~Dein~~ Discriminating attribute of weak entity set.
(7) R — Relationship set
(8) A — Primary key
(9) A — Attributes
(10) A — Derived attribute

- What shd an entity be?
  An object that will have many instances in the database, composed of multiple attributes, object we are trying to model, must have an identifier.
- What shd not be an entity?
  A user of database system, an output of database system
  Eg: report.
- Entity without attribute does not exist.

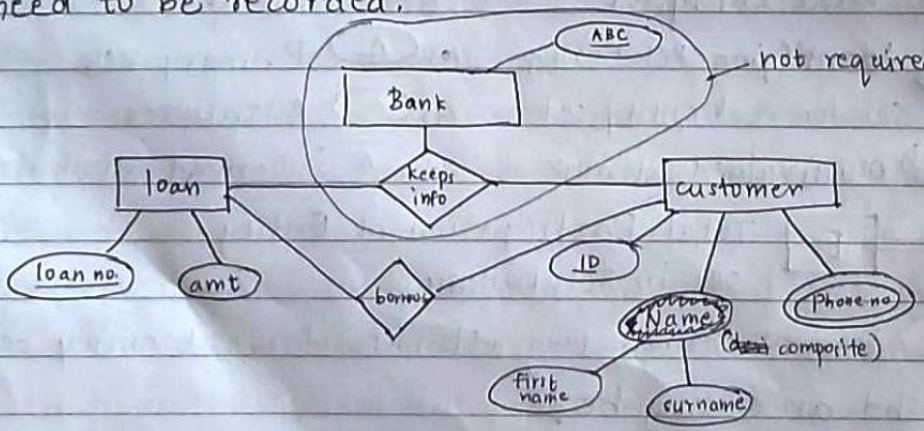
Eg:



- Degree of Relationship: No of entity types that are involved in the relationship. ~~tha~~ Unary (single), Binary (2), ~~Ter~~ Ternary (3)



- ER diagram eg: Draw ER model for ABC Bank. Bank need to keep the information about the customer such as ID, name, phone no. Bank also keeps a track of all the customers who have borrowed a loan. Loan information such as loan no, amt need to be recorded.

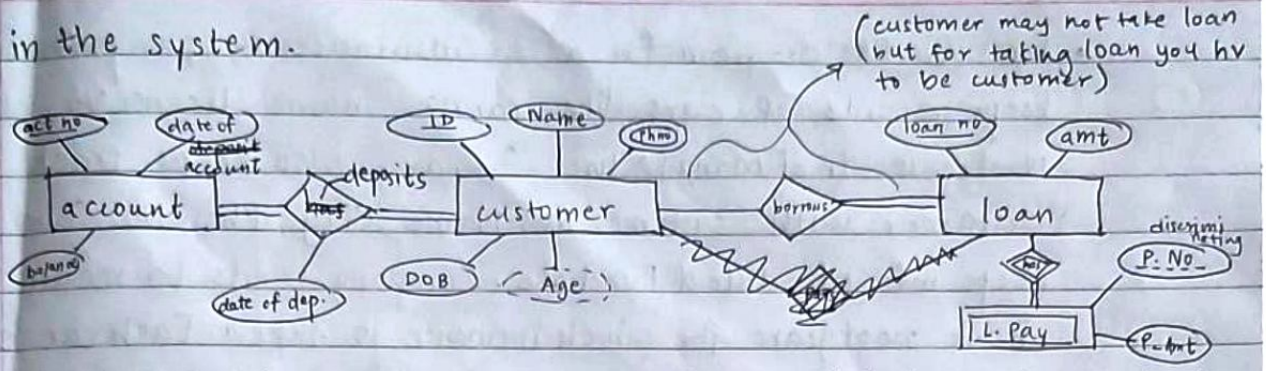


Correct answer: Bank shd not be an entity as it is an user for the software which keeps track of customers and loans. If you want to keep bank as entity, add registration no as primary key, define relationship only with customer, i.e, bank has customer.

- Extra part of same problem: Each customer has account identified by account no. The datae of deposit is recorded

partial key ⇒ discriminator

in the system.



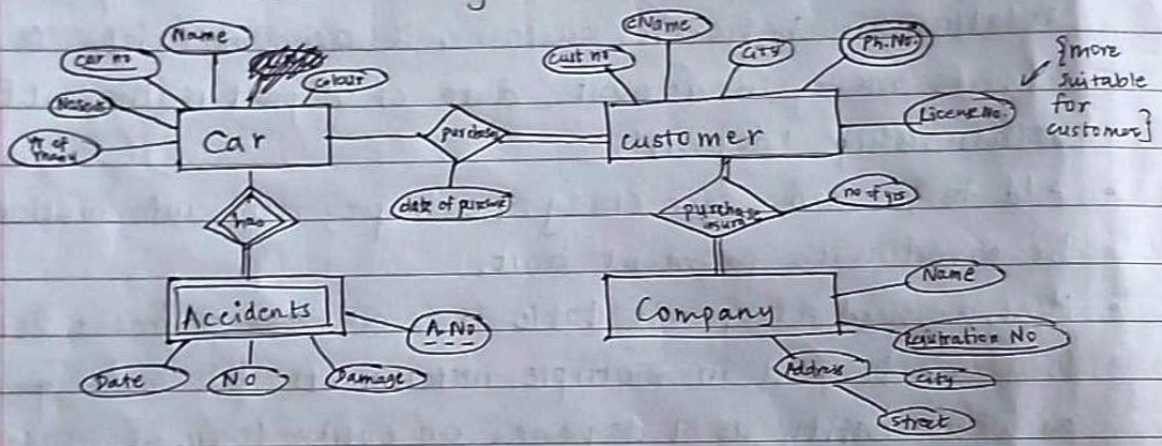
Note: Date of deposit is an operation which is performed when relationship between customer & account exists. As customer deposits money in account, date of deposit is the attribute of relationship 'deposit'.

- Add to the same case study: Loan payment information need to be saved with payment amt.
- Here, we need a seperate table to record ~~most~~ payment details as loan can be paid in multiple installments. We make ~~a~~ loan payment as weak entity as it depends on entity loan, i.e, there cannot be loan payment if there is loan.
- Foreign key. Primary key of one entity is referenced by another entity. Eg: You need loan no. in the table of payment details.
- Discriminating attribute. Only given to weak entity. In the table of payment details we have columns loan no & amt, to create rows, we need to define some serial nos which we are putting as P. No. Distinguishes between attributes of weak entity. Also called Partial key.
- Total Participation (double line between has & L-Pay): Every customer who has taken a loan ha~~s~~ to pay the loan. so for these customers loan payment will always be in relationship with loan, thus total participation. All customers will have an account.
- Partial Participation: You can be customer of bank even if you don't take a loan. Only some customers take a loan, thus partial participation.
- Read about Mapping Cardinalities.

– If Q comes: design system for some application
Ans: Write 3-4 lines case study & draw ER diagram.

- Draw an ER diagram for a car insurance company. Company keeps records of cars like car no., name, license No, colour, No of seats, Yr of Manufacture. Company also keeps track of customer with customer No, Name, City, PhoneNo. Company keeps info of date of Purchase. Company needs to record info about no of years for which insurance is taken. Each car is associated with 0 or any number of recorded accidents.

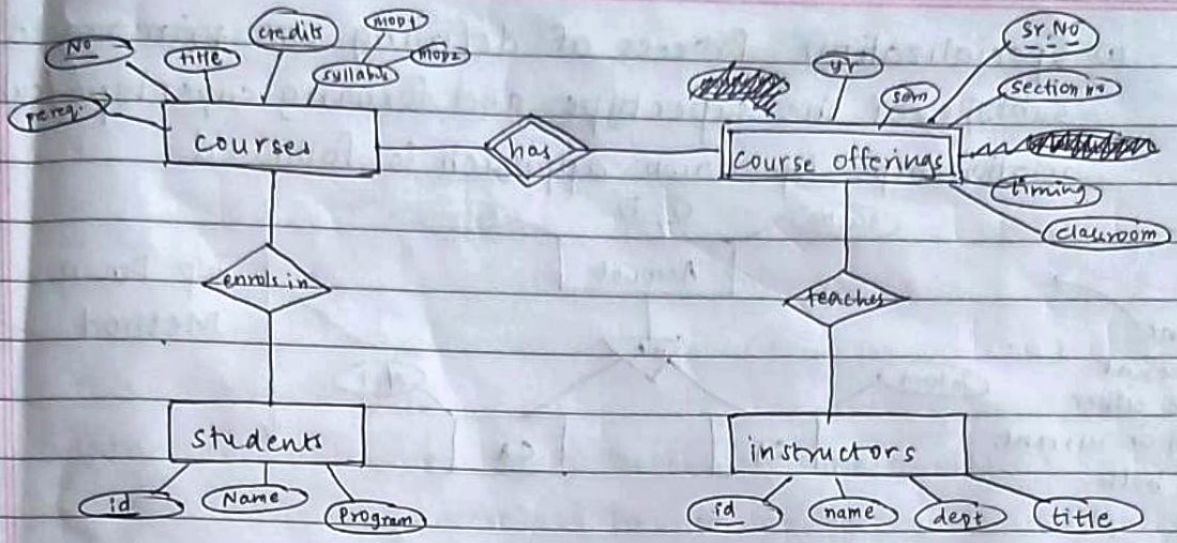
→



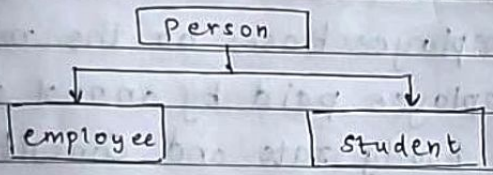
- A university registrar's office maintains data abt following entities:

(1) courses, having no, title, credits, syllabus, prerequisites.
(2) course offerings having course no, yr, sem, section no, instructor(s), timings, classroom.
(3) students having id, name, program.
(4) instructors having id no, name, department, title.

The enrollment of students in courses and grades awarded to students in each course they have enrolled for must be aptly modeled.
Construct ER diagram for registrar's office.



- Superclass, Subclass and Inheritance:

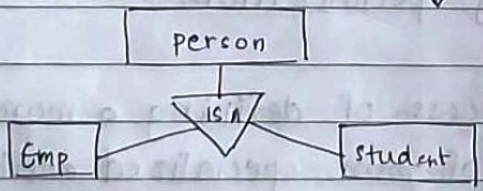


Here, Person entity is superclass which has its own attributes and entity employee and student are subclasses which have property of superclass but also have their own unique property.

NOTE: Subclass or entity without unique key or attribute cannot exist.

- Inheritance: The process of inheriting the property of superclass and having own attribute by subclass is called inheritance. It is represented by

The relationship 'IS A' exists between superclass & subclass.



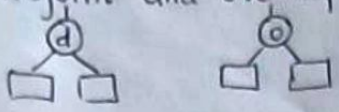
> The Enhanced Entity Relationship Model (EER)

→ Specialization

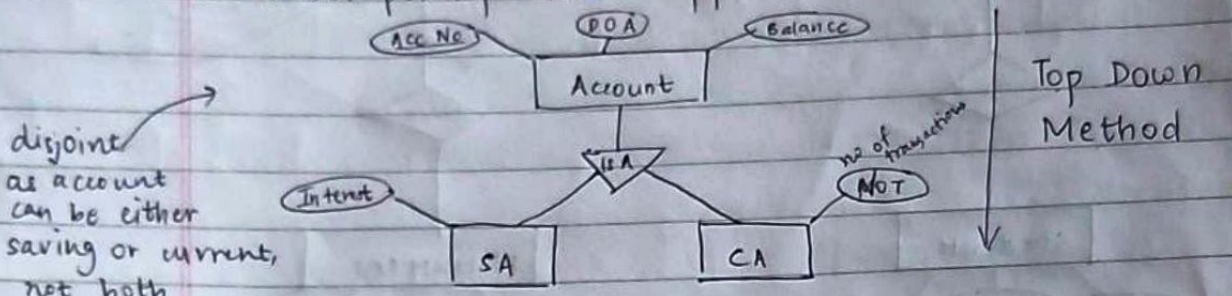
→ Generalization

→ Aggregation

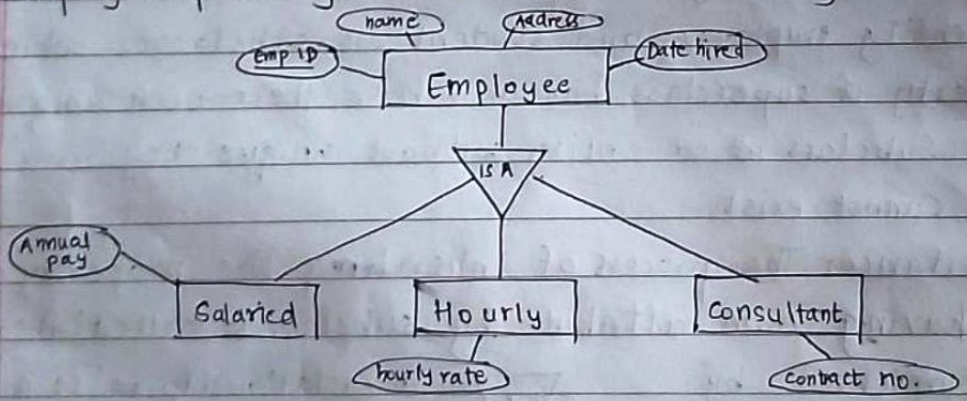
Read disjoint and overlap relation.



(1) Specialization: Process of defining one ~~primary~~ or more subtype of the supertype and forming supertype/subtype relationships. Top down approach is followed:



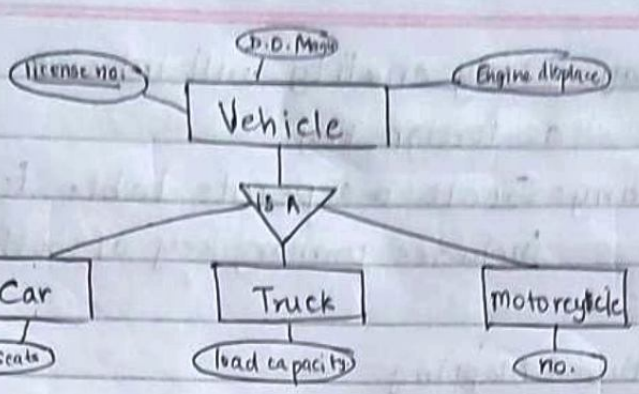
Eg: ABC company wish to keep the information of EMPLOYEE such ~~ds~~ as name, address and date ~~of~~ hired. Company keeps the details of employee bases on the method of pay. Such as salaried employee paid by annual salary, Hourly employee paid by hourly rate and Consultant employee paid by contract number billing rate.



NOTE: Avoid weak entity specialization.

(2) Generalization: The process of defining a more general entity type from a set of more specialized entity types. A bottom-up design process: combine a number of entity sets that share the same features into a higher level entity set.

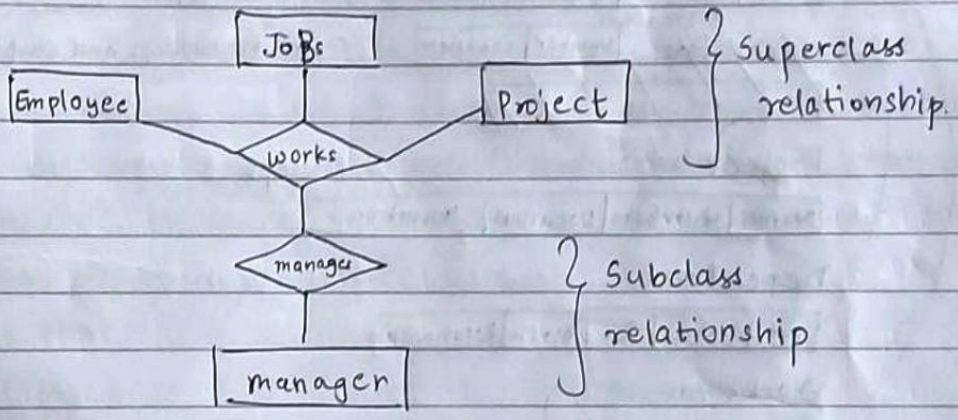
NOTE: Specialization and generalization are inversion of each other.



Bottom to top approach

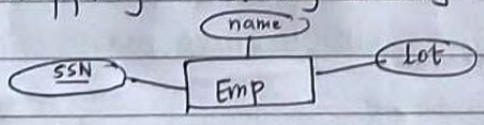
(3) Aggregation : the inheritance between relationships.
One relationship is superclass having regular relationship and subclass relationship which are specific.

Eg:



> ER to relational Mapping Algorithm:

- Mapping of strong entity:

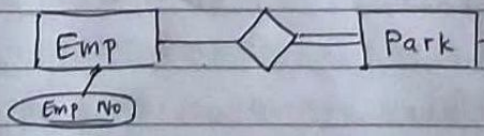


Emp

SSN	name	lot

- Mapping of weak entity: Include all attributes and also the primary key of strong entity as reference.

- Mapping of Relationship:



→ Parking includes the primary key as foreign key bcoz of total participation.

One to one relation.

Q: How to represent many to many in relational model?
Ans: Step 1: Create table for each relation.
2: Ref. primary key of other relation as foreign key.
3: Add attributes.

- One to many: Many entity will inherit the primary key as foreign key.
- Many to many: Create a seperate table for relation. It includes primary key of both entities.

Eg:1 ER to Relational Mapping.

Employee

SSN	Bdate	Name (F | M | L)	Address	Salary	Sex	Number (of dept.)

Department

Name	Number	~~Locations~~	~~No of emp~~	start date	ManagerSSN (you need to know who is managing.)

Works for (Locations: Number | Location)

can be put on dept and employee, anyone.

Project

Name	Number	Location	Number (of dept)

Dependent

SSN	Name	Sex	Birth	Relationship

Works on

SSN	Number	hours

NOTE: Derived attribute shd not be a column. Make new table for multivalued.

Eg:2 Doctors

DocNo	Quali	Ph_No	D_name	Dept Name.

Dept

Dept Name	Location

Patient

PatNO	Age	PhNo	Sex	DocNo	Address	Diagnosis	Status	Treatment

Regular doctor

DocNo	Salary

Doctor on call

Doc No	FS_PR_CL	PVMT

Patient_Admit

Pat No	ADV	Room	ADMTD	COND	MODE

Regular Patient

Pat No	Date	Pay	medicine

Patient_Discharged

Pat No	TR_ADVS	TR_GVN	Medicine

Operates On

Doc No	Pat No	operation	date	medicine

✱ Relational Algebra

- It is a procedure query language (specifies which data you want and by which method). It has set of operation which takes one or two relations as input and generates new relation as output.
- 6 basic operators:

(1) select: σ
(2) project: Π
(3) union: $\cup$
(4) set difference: $-$
(5) Cartesian product: $\times$
(6) rename: ρ
(1) Select: $\sigma_{p}(r)$

σ predicate (tablename) Eg: $\sigma_{salary > 5000}(Emp)$

It selects tuple that satisfies given predicate.

- Banking Eg:

branch (branch_name, branch_city, assets)
customer (customer_name, customer_street, customer_city)
account (account_number, branch_name, balance)
loan (loan_number, branch_name, amount)
depositor (customer_name, account_number)
borrower (customer_name, loan_number)

(Q) Find details of customer whose city is Mumbai.

→ σ customer_city = "Mumbai" (customer)

NOTE: use double quotes.

(Q) Find details of the loan which belongs to branch Andheri.

→ σ branch_name = "Andheri" (loan)

(Q) Find details of account with balance > 500 and account number = 101.

→ σ account_number = 101 $\wedge$ balance > 500 (account)

NOTE: Use $\wedge$, $\vee$, =, !=, $\geq$, $\leq$, $\neq$

(Q) Find details of customer who belongs to city Mumbai or name is Gopal.

→ σ customer_city = "Mumbai" $\vee$ customer_name = "Gopal" (customer)

(Q) Find details of Jones.

→ σ customer_name = "Jones" (depositor)

(2) Project: In order to find a specific column from a relation

$\Pi_{c_1, c_2, \ldots c_k}$ (tablename)

columns names.

Eg:

(Q) Find customer_name from customer relation

→ Π customer_name (customer)

(Q) Find loan_no and amount from banking database

→ Π loan_number, amount (loan)

(Q) Find customer name who lives in Mumbai.

→ Π customer_name (σ customer_city = "Mumbai" (Customer))

(Q) Find ~~details of~~ loan_no which belongs to branch Andheri & having amt > 5000.

→ Π loan_number (σ branch_name = "Andheri" $\wedge$ amount > 5000 (loan))

(3) Union: $\cup$

It's a binary operation. Take the union of 2 relation.

Conditions:

∧ ∨

∪ ∩

→ Both relations must have same no of attributes (arity).

→ The attribute domains must be compatible (Eg: 2nd column of r_1 is int then 2nd column of r_2 also should be int).

(Q) Find all customers (name) with either account or loan.

→ ~~Π customer (account) ∪ (loan)~~

Π customer_name (borrower) ∪ Π customer_name (depositor)

- For intersection use ∩. Same as union.

(4) Set difference : $r_1 - r_2$

Binary operation. Same conditions as union. It allows us to find tuple that are in one relation but are not in another relation.

$r - s = \{t \mid t \in R_1 \text{ and } t \notin R_2\}$

(Q) Find ~~details~~ cust_name who have an account but no t the loan.

→ Π custo_name (depositor) − Π cust_name (borrower)

(Q) Has loan but not account.

→ Π c_name (borrow) − Π c_name (deposit)

(Q) Find cc_name having both account & loan.

→ Π c_name (borrow) ∩ Π c_name (deposit)

(5) Cartesian / cross product : $r_1 \times r_2$

Binary. It results into a new relation having tuple in which every tuple of r_1 is paired with all tuple of r_2. It can be taken in any 2 relations which may not have similar (common) attributes.

The resultant schema contains attributes from r_1 as well as r_2.

Eg: Sailors (Sid, Sname, Rating, Age)

Reserve (sid, bid (boat id), Day)

(Q) Find ~~sid~~ sname having rating > 5.

→ Π sname (σ rating > 5 (sailors))

(Q) Find sid and sname whose Age is 20.

→ Π sid, sname (σ Age = 20 (sailors))

Join: Allows us to join data from one or more relations based on common attributes.

(Q) Find bid reserved on day Monday.

→ Π bid (σ day = "Monday" (reserve))

(Q) Find ~~sid~~ s-id the sailors who have reserved a boat.

→ ~~(Sailor) ∩ (reserve) Π sid (reserve)~~ Π sid (reserve) $\cap$ Π sid (sailor)

(Q) Find ~~sid~~ s-id the sailor who reserved a boat id 101.

→ Π sid (σ boat id = 101 (reserve))

(Q) Find id and name of all sailors from relation sailor who have reserved a boat with bid = 101.

→ ~~Π sid, sname (sailor)~~

Π s.sid, sname (σ bid = 101 $\wedge$ s.sid = r.sid (sailor x reserve))

> NATURAL - JOIN OPERATION ($\bowtie$)

- Binary operation that allows us to combine certain selection and cartesian product into one relation. It performs cartesian product of 2 arguments, performing a selection, forcing on equality on those attributes that appear in both the relation. It removes duplicates.

Eg: Emp

Name	City
Ram	Mumbai
Ramesh	Pune
Rohan	Mumbai

Emp - work

Name	Dept
Ram	IT
Rohan	CS
Gopal	EXTC

After natural-join operation:

Name	City	Dept
Ram	Mumbai	IT
Rohan	Mumbai	CS

← Emp $\bowtie$ Emp-work.

- ~~Extension~~ Outer - Join ~~$\bowtie$ left~~ : Extension of join operation. It deals with missing information. Types:

(1) Left $\bowtie$

(2) Right $\bowtie$

(3) Full $\bowtie$

(1) Left ⋈ : It takes all tuple from left relation, wherever the information is not present in right relation, it simply appends with NULL value.

Eg: Emp ⋈ Emp-work is same as Emp-work ⋈ Emp

Name	City	Dept
Ram	Mumbai	IT
Ramesh	Pune	NULL
Rohan	Mumbai	CS

(2) Right ⋈ : It takes all tuple from right relation, wherever the info is not present on left side, it simply appends with NULL value.

Eg: Emp ⋈ Emp-work is same as Emp_work ⋈ Emp.

(first data is fetched from Emp-work)

Name	City	Dept
Ram	Mumbai	IT
Rohan	Mumbai	CS
Gopal	NULL	EXTC

(3) Full ⋈ : It takes all details from both relations and append NULL value where info is missing.

Eg: Emp ⋈ Emp_work

Name	City	Dept
Ram	Mumbai	IT
Ramesh	Pune	NULL
Rohan	Mumbai	CS
Gopal	NULL	EXTC

> Examples:

(1) R:

A	B	C	D
α	1	α	a
β	2	γ	a
γ	4	β	b
α	1	γ	a
δ	2	β	b

S:

B	D	E
1	a	α
3	a	β
1	a	γ
2	b	δ
35	b	ε

(1) Cross Prod (x):

A	B	C	D	B	D	E
α	1	α	a	1	a	α
α	1	α	a	3	a	β
α	1	α	a	1	a	γ
α	1	α	a	2	b	δ
α	1	α	a	3	b	ϵ
β	2	γ	a	1	a	α
β	2	γ	a	3	a	β
β	2	γ	a	1	a	γ
β	2	γ	a	2	b	δ
β	2	γ	a	3	b	ϵ
γ	4	β	b	1	a	α
γ	4	β	b	3	a	β
γ	4	β	b	1	a	γ
γ	4	β	b	2	b	δ
γ	4	β	b	3	b	ϵ
α	1	γ	a	1	a	α
α	1	γ	a	3	a	β
α	1	γ	a	1	a	γ
α	1	γ	a	2	b	δ
α	1	γ	a	3	b	ϵ
δ	2	β	b	1	a	α
δ	2	β	b	3	a	β
δ	2	β	b	1	a	γ
δ	2	β	b	2	b	δ
δ	2	β	b	3	b	ϵ

(2) R ⋈ S

A	B	C	D	E
α	1	α	a	α.
α	1	α	a	y.
α	1	y	a	α.
α	1	y	a	y.
δ	2	β	b	δ

(3) R ⋈ S

A	B	C	D	E
α	1	α	a	α
α	1	α	a	y
β	2	y	a	-
y	4	β	b	-
α	1	y	a	α
α	1	y	a	y
δ	2	β	b	δ

(4) ~~(1) B ⋈ S~~ R ⋈ S

B	D	E	A	C
1	a	α	α	α
1	a	α	α	y
3	a	β	-	-
1	a	y	α	α
1	a	y	α	y
2	b	δ	δ	β
3	b	ε	-	-

(5) R ⋈ S

A	B	C	D	E
α	1	α	a	α
α	1	α	a	y
β	2	y	a	-
y	4	β	b	-
α	1	y	a	α
α	1	y	a	y
-	3	-	a	β
-	3	-	b	ε
δ	2	β	b	δ

SQL: STRUCTURE QUERY LANGUAGE

It supports the following languages:

DDL: Data definition DCL: Data Control

DML: Data manipulation TCL: Transaction

(1)- DDL: Operations on schema. Helps create/modify structure of a relational model.

→ Create

→ Alter ← Add / drop — modify

→ Drop

→ Truncate

(2)- DML: Operations on data

→ Insert

→ Select

→ Update

→ Delete

(1) DDL:

→ Create: Used to create new structure or schema required for particular domain. Syntax:

Create table tablename
(column name1 datatype (10), ← width
column name2 datatype ( ), ...)

* int has a default width. You have to assign width to char.
• HW: Explore all datatypes supported by SQL. Eg: varchar, string....
→ DESC for displaying structure. Describes schema, gives us the metadata. Metadata is stored in data dictionary.
→ Alter: Used to modify structure of existing relation.
(1) alter table add : It is used to add a new column in an existing structure. Syntax:
alter table tablename add (column name datatype (width))
(2) alter table tablename drop (~~column~~name) , to delete a column.
(3) alter table tablename modify (columnname datatype (width))
where you can change width, datatype, etc.
(4) Dif between drop, truncate and delete.
- Drop : The table is deleted entirely. You can create another table with same name.
- Truncate: Only data from table is deleted. The table (structure) still exists you cannot create another table with same name.
- Delete: For deleting one or more rows.
Truncate is like tearing the page whereas delete is cancelling every line one by one.

(2) DML: It is used to manipulate the data (Insert, select, modify, delete, conditional fetching of data) from the existing structure.

→ Insert: It is used to insert or add new record or tuple in the relation (table).

tuple: One row, i.e. it gives information about one entity of the entity set.

Syntax:

insert into ~~table~~ tablename values (value 1, v2, v3, ...)

→ Select:

Syntax:

Select * → for all

From tablename

for conditional empty → where columnname = value1 eg: where City = 'Mumbai' for fetching data of people living in Mumbai.

Conditions can be =, >, <, ≥, and, ≤, Or, not

→ insert into tablename (column1, column2) values (v_1, v_2)

Use this if you have incomplete data. eg: only name and roll no, we don't know city. In this case if you put the value you don't have as NULL or blank you won't be able to update it later.

→ Update: It is used to modify the data value in the existing table. Syntax:

update tablename set columnname = value.

where ← if you want any conditions.

→ Delete: Used to delete a tuple from a relation.

delete from tablename where ...

> COLUMN LEVEL CONSTRAINTS:

- Constraints are the rules that are used to control the invalid data entry in the column. The types of constraints are:

(1) Primary key
(2) Foreign key
(3) NULL
(4) Not NULL
(5) Unique
(6) Check
(7) Default

(1) Primary key: It is a simple key which has two special attributes. ⓐ It is unique throughout the column. ⓑ The value across the column cannot be left blank. A DBA is responsible for defining a primary key for a structure. There exists only one primary key for a particular structure.

create table tablename (columnname datatype primary key)

- If primary key isn't unique, error:
  unique constraint violation
- If you ~~enter key~~ put primary key as null, error:
  cannot insert NULL into

(2) Unique constraint: The unique key constraint ensures that info in column is unique or the value entered in specific column must not be repeated across the column. Unique key allows NULL values also.

You can have only one primary key in the table but multiple unique keys are possible.

Syntax:

create table tablename (column1 datatype primary key, column2 datatype (w) unique, column3 datatype (w))

(3) NULL : It ~~is used~~ allows values in particle column to be NULL whereas not NULL specifies that a column must have some non-zero value.

- columnname datatype () unique not NULL
  for making it both unique & not NULL so that it gets characteristics of primary key.

(4) Default: It specifies some default value if no value is entered by the user. Syntax:
create table tablename (column1 datatype () default 18)
(Age, int)

- We cannot use unique, not NULL, default together as default cannot be used with anything bcoz ~~it~~ it won't be an unique value.

(5) Check: It applies a condition to an input column value.

(6) Foreign key:
Syntax:
create table tablename2 (column1 datatype references tablename1)
This table has primary key as column1.

Eg:
create table student (RollNo. int primary key)
create table grades (RollNo. int references student) .

check syntax:
create table tablename (column datatype () check (column in ('value'))

> JOIN

Eg: Product (Pno, description, color)

P

Pno	description	color
P1	Mouse	Red
P2	Keyboard	Blue
P3	Disc	Green

Supplier S

SNo	Name	City
S1	A	Mumbai
S2	B	Pune
S3	C	Mumbai

Supplier product table

SP

SNo	PNo	Qty
S1	P1	NULL
S2	P1	100
S3	P1	200
S3	P2	300

(Q) Cartesian Product :

→ Select * from P, S

- Equijoin (Innerjoin) : Based on equality attribute. Selects all the rows from both tables as long as condition is satisfied.

(Q) Equijoin (Natural join)

→ Select * from P, SP where p.PNo = SP.PNo

: from table1, table2 where table1.matching column = table2.matching

- Left outer Join: Only unmatched rows from left side table are retained.

→ select * (write this) from table1 left outer join table2 on table1.matching = table2.matching

Eg: select * from P left outer join SP on P.PNo = SP.PNo

- Right outer Join: Only unmatched rows from right side table are retained.

select * from P right outer join SP on P.PNo = SP.PNo

- Full outer join : Unmatched rows from both tables are retained.

select * from P full outer join SP on P.PNo = SP.PNo

(Q) Find all PNo whose Qty < 300.

→ select PNo from P left outer join SP where Qty < 300.

$\Pi_{PNo}(\sigma_{qty<300}(SP))$

(Q) Find PNo, name, quantity of blue product.

→ select PNo, name, quantity from P full outer join SP and S full outer join SP on P.color = 'blue'.

select PNo, name, Qty from P, S, SP where color = 'blue'

(Q) Find name of all supplier who has supplied more than 200 parts.

→ select name from S, SP where S.SNo = SP.SNo ∧ Qty > 200

> Aggregate function:

- SQL provide aggregate function to summarize large amt of data. It takes collection of values & returns single value as result. Various aggregate functions are:

(1) Average : Avg() Syntax: select avg(column) as displayname from table
Function returns the avg value of a specified column.

(2) ~~Mean~~ Min()
Returns the smallest value from column.

(3) Max() : Returns largest value from column.

(4) Sum() : Returns the sum of given values.

(5) Count() : Returns total no of rows from specified relation.

(6) stddev() : Standard deviation.

- count(column) : All no of rows
count(distinct column) : Not considering duplicate.

- Group By: It is used in conjunction with the aggregate function to group the result set by one or more column.

Eg: select empid, sum(salary) from employee group by empid
Output: It will display salary of empid 1 as 20000 that is addition of duplicate

- Select having.

Eg: select empid, sum(salary) from employee group by empid having sum(salary) > 30000.

(Q) Create table: [(Q) Find name of emp who has max(salary) after increasing salary] Assignment
Emp (Eno, Name, Salary, City)

(1) Increase salary of all employees by 1000. (Just see, dont have to change in database, for that we use update).
→ select (salary + 1000) from emp.

(2) Decrease by 1000.
→ select (salary - 1000) from emp

(3) Display salary of all employee with 10% bonus.
→ select (salary * 1.1) from Emp.

(4) Find the highest salary employee.
→ select max(salary) from Emp.

(5) Salary divide by 10.
→ select (salary/10) from Emp.

> DUAL TABLE: It is a small oracle work table which consists of only one row and one column, and contain value 'x' in that column. It supports data retrieval and its formatting, arithmetic operations, etc. We can use dual table with select statement, where clause, and for retrieving the function results.

(Q) Find system date (using Dual).
→ select sysdate from dual

- Various data functions are:

(1) ADD_Months : To add specific no of months to a ~~data~~ date.

(2) LAST_DAY: Returns last day in the month of the specified date.

(3) MONTHS_BETWEEN: It calculates the no of months between 2 days.

(4) SYSDATE : Returns current date & time in oracle server.

~~(5) Add~~

(1) select ADD_MONTHS ('12-JAN-2022', 3) from DUAL

→ Output : 12-APRIL-2022

(2) select ADD_MONTHS ('12-JAN-2022', -12) from DUAL

→ Output : 12-JAN-2021

* (3) select ADD_MONTHS ('28-FEB-1989', 1) from DUAL

→ Output : 31-MAR-2021 (it will add 1 month i.e. 31 days)

(4) select LAST_DAY ('12-JAN-2022') from DUAL

→ Output : 31-JAN-2022

(5) Select LAST_DAY ('31-JAN-2022') from DUAL

→ Output : 31-JAN-2022

(6) select LAST_DAY (SYSDATE) from DUAL

→ Output : 31-OCT-2022

(7) select MONTHS_BETWEEN ('31-JAN-2022', '31-MAR-2022') from DUAL.

→ Ans: 2

(8) select MONTHS_BETWEEN ('31-MAR-1995', '28-FEB-1994') from DUAL.

→ Ans: 13

(9) select MONTHS_BETWEEN ('31-JAN-2022', '1-MAR-2022') from DUAL. NOTE: Performs calculation with fraction

→ Ans: 1.····· (some decimal ans)

(5) NEXT_DAY: Returns the date of first day after the specified

date, which falls on specified day in week. (If today is Monday, it will find next Monday).

(9) select NEXT_DAY ('20-OCT-2022', 'Thursday') from DUAL

→ Output : 27-OCT-2022

> SET OPERATIONS:

Set operations supported by SQL are:

(1) UNION

(2) UNION ALL

(3) INTERSECTION

(4) SET DIFFERENCE

(8) Create table BOOK with ~~bo~~ bid as integer.

BOOK bid	AUTHOR bid
101	101
102	102
103	103
104	
105	

(1) Union: It will display rows from both tables.

→ select * from BOOK UNION ~~AUTH~~ select * from AUTHOR

~~(2) Select~~ Ans: 101, 102, 103, 104, 105.

(2) select * from BOOK UNION ALL select * from AUTHOR

→ Ans: 101, 102, 103, 104, 105, 101, 102, 103.

(3) select * from BOOK INTERSECT select * from AUTHOR

→ 101, 102, 103

(4) select * from Book minus select * from AUTHOR

→ 104, 105

(5) select * from AUTHOR minus select * from BOOK

→ NULL

> ~~BETWEEN~~

> VARIOUS OPERATORS IN SQL:

(1) In : ~~Gives all records~~ Selects one of the set value.

→ IN (value1, value2, ...)

(Q) select * from Emp where salary IN (1000, 2000)

→ It will display all records from table with salary 1000, 2000.

(2) Not In : All records from table, not having the specified value in the list.

(Q) select * from Emp where salary IN (1000, 2000) · Emp with sal 1001 will also be displayed.

→ All emps not having salary 1000, 2000.

(3) Check when value is between specified value : BETWEEN.

BETWEEN (val1, val2)

(Q) select * from Emp where salary BETWEEN (1000, 2000)

→ All emps from 1001 to 1999 will be displayed.

(Q) select * from Emp where salary NOT BETWEEN (1000, 2000)

→ 1001 to 1999 will not be displayed. 1000, 2000 will be displayed.

> VIEW : View is a logical table of data extracted from existing i.e base table. It can be called as virtual table. The table upon which view is based is called the base table. A query can be made just as a base table on virtual table but does not require disc space.

- Use of view:

(1) To hide sensitive data.

(2) To provide additional level of security to the database.

(3) To isolate the application from changes in definition of base table.

- Syntax:

create view viewname as (query expression)

- first create table Emp with Eno, name, city, DOB, qualification, salary.

(Q1) Create view on emp table to display Eno, Name, City, Salary.

→ create view empview1 as (select Eno, Name, City, Salary from Emp)

(Q2) Create a view on emp to display Name, Eno of all emp belonging to city Mumbai & having salary > 5000.

→ create view empview2 as (select Eno, Name from Emp where city='Mumbai' and salary>5000)

- Lab Question for view: Create any base table from project. Create a view based on query of logical operator. Insert, update, delete in view table and observe changes in base table. Insert, update, delete in base table & Observe changes in view table. Try inserting duplicate primary key in view and write the error. If you delete one row from view, will that row get deleted from base.

(Q3) Insert a record in emp view1.

→ insert into empview1 values (101, Ram, Mumbai, 1000)

- DESC empview1 to display schema.
- To drop a view : Drop view viewname
- Join doesn't work while creating view. Try join, aggregat, set on view.

- Subquery: In, not in, between, not between.

Emp (ENO, Fname, Lname, Email, Ph No, Joiningdate, leavingdate, jobid, salary, comm, managerid, deptid)

(Q1) Write query to display Name (fname, lname), salary, deptid, jobid for emps who work in same designation as emp works for empno=197

(Q2) Write query to display Fname, Lname for emp who gets more salary than emp whose no is 163.

(Q3) Display Fname, lname, salary, deptid for emps who earn such amt of salary which is lowest salary of any dept.

(Q4) Display Eno, Fname for all emp who earn more than avg salary

(Q5) Display Fname, Eno, Salary of all emp reporting to "Rajan".

(Q.6) Display all info of emp whose salary & reporting person id is 5000 & 129 respectively

(Q7) Display all info of emp whose id is any of the no from 135, 169, 188.

Answers:

(7) select * from Emp where ENO IN (135, 169, 188)

(6) select ~~* from~~ Fname, Lname, salary, dept id, job id ~~from Emp where~~ INTERSECT select jobid from Emp where EmpNO = 187.

(1) select Fname, Lname ..... from Emp where job id = (select Jobid from emp where ENO = 187)

(2) select Fname, Lname .... from Emp where salary > (select salary from Emp where ENO = 163)

> TRIGGER: Statement executed automatically by the system as a side effect of modification to the database. Triggering events are insert / update / delete. Various types of triggers are:

(1) Before: Fires trigger before executing trigger statement.

(2) After: Fires trigger after executing

(3) For each row: It specifies that trigger fires once per row.

- Syntax (to replace already existing trigger with new one.)

```
create or replace trigger triggername
before/after
insert/update/delete
on tablename
for each row
begin {
}
end
```

- Variables :new.columnname
  :old

Anushka Acharya
UID: 2021700001

10

- Schema:

Branch (bid, bname, bcity)
Customer (cname, cstreet, ccity)
Depositer (cname, Acct_no)
Loan (lno, bname, amount)
Borrower (cname, lno)
Account (Acct_no, bname, balance)

(1) Find branch name of branch 101.

→ $\Pi_{bname}(\sigma_{bid=101}(Branch))$ ✓ (1)

select bname from Branch where bid=101.

(2) Find details of customers belong to city "GOA".

→ $\sigma_{ccity = "GOA"}(Customer)$ ✓ (1)

select * from customer where ccity = 'GOA'

(3) Find names of all customers who have a loan at the bank along with loan number and amount.

→ $\Pi_{cname, lno, amount}(Borrower \bowtie loan)$ ✓ (1)

(4) Find name of all branches with customers who have an account in the bank and who belongs to city "Mumbai".

→ $\Pi_{bname}(\sigma_{ccity = "Mumbai"}(Customer \times (Branch \bowtie Account)))$ ✗ (0)

(Customer ⋈ Account ⋈ Depositor)

(5) Find all customers who have both a loan and an account at the bank.

→ $\Pi_{cname}(depositor) \cap \Pi_{cname}(borrower)$ ✗ (0)

$\Pi_{cname}(borrower \bowtie Depositor)$

$\Pi sid(\sigma_{rating \geq 8}(S)) \cup \Pi sid(\sigma_{bid...}$

- Schema:
  S (sid, sname, rating, age)
  B (bid, bname, color)
  R (sid, bid, date)

(1) Find colour and name of boat 209.
→ $\Pi_{color, bname}(\sigma_{bid=209}(B))$ ✓ (1)
select colour, bname from B where bid=209.

(2) Find age of sailor 109.
→ $\Pi age(\sigma_{sid=109}(S))$ ✓ (1)
select age from S where sid=109.

(3) Find all sid having rating more than 4.
→ $\Pi sid(\sigma_{rating>4}(S))$ ✓ (1)
select sid from S where rating > 4

(4) Find the colors of boats reserved by Rohan.
→ $\Pi color(\sigma_{sname="Rohan"}(S \bowtie B \bowtie R))$ ✓ (1)
$\Pi color((\sigma_{name="Rohan"}(S)) \bowtie R \bowtie B)$

(5) Find all sailors id's of sailors who have a rating of atleast 8 or reserved a boat 103.
→ $\Pi sid(\sigma_{rating \geq 8}) \cup \Pi sid(R)$ ✗ (0)
$\Pi_{sid}(\sigma_{rating \geq 8}) \cup \Pi_{sid}(\sigma_{bid=103}(B))$.

$\Pi sname(\Pi sid(S) -$

(6) Find the name of sailors who have not reserved a red boat.
→ ~~$(S \times B \times R)$ -~~
$\Pi sname(S \times B \times R) - \sigma_{color="red"}(S \times B \times R)$ ✗ (0)
$\Pi_{sname}((\Pi_{sid}(S) - \Pi_{sid}(\sigma_{color="red"}(b) \bowtie r)) \bowtie S)$.

$\Pi_{sid}(\sigma_{age>20}(S)) - \Pi_{sid}((\sigma_{color='red'}(B)) \bowtie R)$

(7) Find the sailor id of sailor with age over 20 who have not reserved a red boat. ✗ ⓪

→ $\Pi_{sid}(\sigma_{age>20}(S \bowtie B \bowtie R)) - \sigma_{color="red"}(S \bowtie B \bowtie R)$

$\Pi_{sid}(\sigma_{age>20}(S)) - \Pi_{sid}(\sigma_{color="red"}(S) \bowtie r)$

- Schema:

Customer (cid, cname, city)

Product (pid, pname, color)

Catalog (cid, pid, cost)

$\Pi_{cname}(\Pi_{cid}(\Pi_{pid}(\sigma_{color="black"}(P)) \bowtie Catalog) \bowtie Cust)$ ✗ ⓪

(1) Find name of customer who supply some black product.

→ $\Pi_{cname}(\sigma_{colour="black"}(Customer \bowtie (Product \bowtie Catalog)))$

$\Pi_{cname}(\Pi_{cid}((\Pi_{pid}(\sigma_{color="black"}(P)) \bowtie Catg) \bowtie Cust)$

$\Pi_{cname}(\Pi_{cid}((\Pi_{pid}(\sigma_{colou}$

(2) Find name of customer who supply some black or pink product. ✗ ⓪

→ $\Pi_{cname}(\sigma_{colour="black" \wedge colour="pink"}(customer \times product)$

$\Pi_{cname}(\Pi_{cid}(\Pi_{pid}(\Pi_{color="black" \wedge color="pink"}(prod)) \bowtie catalog) \bowtie cust)$

(3) Find city of customer Gopal.

→ $\Pi_{city}(\sigma_{cname="Gopal"}(customer))$ ①

select city from Customer where name='Gopal'

(4) Find product name with color black and red.

→ $\Pi_{pname}(\sigma_{colour="black" \wedge colour="red"}(Product))$

select pname from Product ①

where color='black' and color='red' ✓

(5) Find cost of product id 205.

→ $\Pi_{cost}(\sigma_{pid=205}(Catalog))$ (1)

Select cost from Catalog where pid=205.

UID :- 2021700036.

✱ Transaction

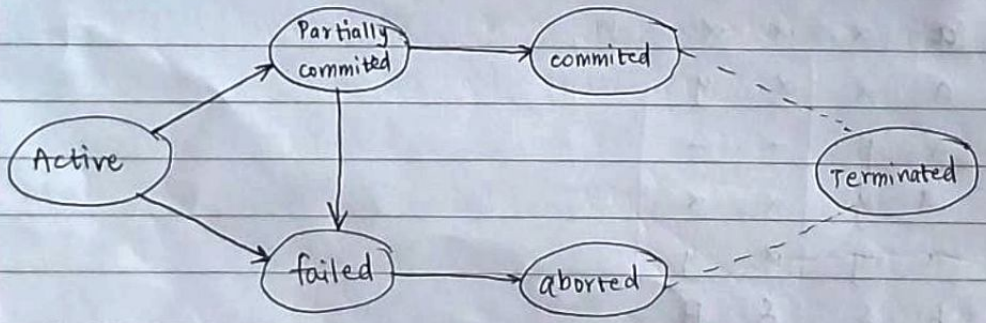
- Definitions
- Transaction: Collection of operations that form a single logical unit of work.
- Role of transaction manager: Checks whether an instruction is executed completely or not at all.
- Buffer manager:

Eg: Transaction of ₹50 from Acct A to B

```
read (A)
A = A - 50          } operation 1
write (A)
read (B)            } operation 2
B = B + 50
write (B)
```

Set of operations = Transaction

- Transaction States:



Active: When you start the transaction.
Partially commited: When changes have been made but it isn't finalised, or not approved by DBA.
Commited: Finalised changes, approved by DBA.
Failed: When any step cannot be executed, the transaction fails.
Abort: Undo all operations, go back to start.

- Properties of Transaction (ACID):
- Atomicity: Transaction is executed completely or not at

A = 0.9A

all 0% or 100% success.

- Consistency: If data is consistent initially then it shd remain consistent after transaction.
- Isolation: At a time, only one user can perform one transaction.
- Durability: Makes sure changes after successful transaction are retained in the database.

(Q) Transfer 10% Balance from Account A to B.

read (A)	read (A)
A = A - 0.1A	Temp = A * 0.1
~~write (A)~~	A = A - Temp
read (B)	write (A)
B = B + 0.1A	read (B)
write (B)	B = B + Temp
~~write (A)~~	write (B)
write (A)	

Wrong

Whenever any operation is performed on data before transfer, store it in some temporary variable.

(Q) Transaction where one user wants to transfer ₹50 from A to B and other user wants to transfer 10% balance from A to B simultaneously.

→ Can't be done simultaneously. One after another or concurrently.

- Format:

Begin Transaction
  Process
  Commit.

- Schedule: Sequence in which transactions are executed.

- Serial schedule eg: $T_2 \rightarrow T_1$ T_2 followed by T_1
- Concurrent schedule eg: $T_1 \rightrightarrows T_2$
- Equivalent schedule: Output after 2 schedules is same.

- Dirty read: In concurrent transactions, if T_2 is reading the value committed by T_1 & T_1 fails then it is called dirty read.

Assumption: Data is in consistent state

- Serializability: A schedule which is equivalent to serial schedule. It allows multiple (concurrency) transactions to be executed at a time unlike serial schedule. Two types:
  - Conflict ~~serializability~~ equivalent: If schedule s can be transformed into s' by series of swaps of non-conflicting instructions

T_1	T_2	
read(A)	read(A)	Non conflicting
read(A)	write(A)	Conflicting.
write(A)	read(A)	Conflicting.
write(A)	write(A)	Conflicting.

Conflict serializability: schedule is conflict equivalent to serial schedule.

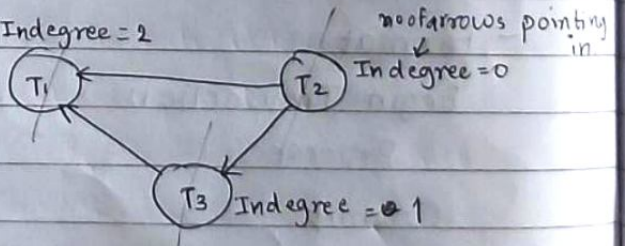
- Ways to derive serial schedule from a concurrent schedule:

(1) Precedence graph (V, E)

Check conflict pairs (R→W, W→R, W→W) in other transaction and draw edge. Nodes are transactions.

Eg:

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



No loop.

Loop ≡ T_A ⇄ T_B

No loop ⇒ Conflict serializability
⇒ Serial schedule exists
⇒ consistent.

6 possible ~~conflict~~ combinations ($T_1 \to T_2 \to T_3, \dots$)

~~Conflict~~ Serial combination

= $T_2 \to T_3 \to T_1$

indegree = 0 ; indegree 0 after removing T_2

NOTE: (T1 T1) Not allowed. Don't check conflicts in same transaction.

(Q) Find whether the given schedule is conflict serializable or not. If it is, then write the given schedule in serial form.

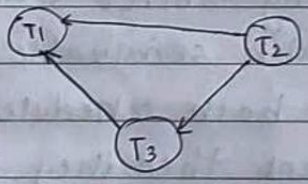
→ Serial schedule:

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

NOTE:
5 marks ⇒ Conflict serial or not
7 marks ⇒ Also write serial
10 marks ⇒ At beginning write about dirty read & dirty write using some numerical example.

(Q)

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



No loop.
⇒ Conflict serializability.
⇒ Serial schedule exists
⇒ Consistent

Serial schedule : $T_2 \rightarrow T_3 \rightarrow T_1$

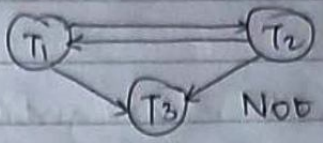
(Q) T_1 T_2 T_3

R(A)

W(A)

W(A)

W(A)



Not conflict serial

Loop ⇒ Can't determine serializability

~~Not serializable~~

(Q) T_1 T_2 T_3 T_4

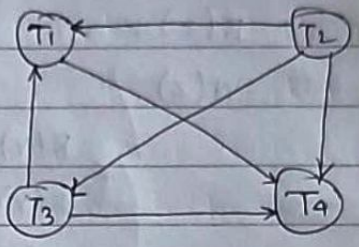
R(x)

W(x)

W(x)

W(Y)

R(Z)



R(x)

R(Y)

No loop ~~i.e. Seri~~

⇒ Conflict serializability.

⇒ Serializable : $T_2 \rightarrow T_3 \rightarrow T_1 \rightarrow T_4$

- View serializability : If schedule is non-conflict serializable, then using conflict serializability method, you cannot say whether there exists serializable schedule or not. View serializability says that if both schedules (S & S') produce same result, i.e, are ~~view~~ equivalent then they are view serializable.

To convert given schedule into view equivalent.

(1) Check initial read (from database) ⇒ Will always be first

(2) Check final write operation ⇒ Will always be last

(3) Check intermediate write & read operation sequences.

(4) Apply blind W & blind R operation on both schedule and prove that converted schedule is view equivalent.

Eg: T_1 T_2 T_3 ← S

R(A)

W(A)

W(A)

W(A)

T_1 T_2 T_3 ← S'

R(A)

W(A)

W(A)

W(A)

(Q)

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

$T_1 \rightarrow T_2$

⇒ Conflict serializable.

⇒ Serializable.

> CONCURRENCY PROTOCOL:

- Lock-Based Protocols : Two types - shared mode lock or exclusive (R+W) mode lock.

lock S(A)

unlock (A)

R \ S	S	X
S	✓	×
X	×	×

(only R)

← Compatibility matrix.

Shared & shared is allowed.

Exclusive is not allowed with anything. It has to be unlocked

- Growing phase : Any no of locks can be introduced.
- Shrinking phase : Starts when locks are unlocked. No locks allowed.

- Advantages of 2-Phase Lock:
  - Always ensures serializibility
- Disadvantages
  - Not free from irr-recoverability.
  - Not free from deadlock
  - Not free from starvation
  - Not free from Cascading Rollback.
- Irr-recoverability

Eg:

T_1	T_2
X(A)	
R(A)	
W(A)	
unlock → U(A)	
	S(A)
	R(A)
	commit
Fail	

~~One trans~~ T_2 read value from T_1 but T_1 failed. Thus T_2 has dirty read value. This is called irr-recoverable schedule.

NOTE: IF T_1 is S(A) & T_2 is S(A) then it does not matter if T_1 fails, T_2 will commit as in shared-mode only read is allowed, data won't change.

- Cascading rollback: T_2 reads value from T_1, T_3 from T_2 & T_4 from T_3. T_1 fails but T_2, T_3, T_4 commit. T_2, T_3, T_4 have dirty reads so we need to undo them.
- 2 phase is not free from irr-recoverability & cascading roll back as it allows T_2 to read after T_1 unlocks data item, but unlock is partially committed state not fully committed.
- Strict 2-phase locking protocol: It should satisfy the basic 2 phase locking protocol & all exclusive locks shd hold until committed or aborted. Free from irr-recoverability & cascading.

Eg:

T_1	T_2
X(A)	
R(A)	
W(A)	
Commit	
U(A)	
	S(A)

> Timestamp-based Protocols

- Unique value to every transaction
- TS(T_i). Timestamp issued when T_i enters system.
- If an old Ti has TS(Ti), a new Tj is assigned TS(Tj) such that TS(Ti) < TS(Tj)

9:00	9:15	10:00
T_1	T_2	T_3
10	20	30
Old	Younger	Youngest

- Read Timestamp RTS(A): TS of last read performed
- Write Timestamp WTS(A): TS of last write performed

T_1(10)	T_2(20)	T_3(30)
R(A)		
	R(A)	
		R(A)

RTS(A) = 30

T_1(10)	T_2(20)	T_3(30)
W(A)		
	W(A)	
		W(A)

WTS(A) = 20

- Cases:

(1) Ti issues R(A)

(a) If WTS(A) > TS(Ti) Rollback Ti

(b) Otherwise execute R(A)

Set RTS(A) = Max {RTS(A), TS(Ti)}

(2) Ti issues W(A)

(a) If RTS(A) > TS(Ti) Rollback

(b) If WTS(A) > TS(Ti) Rollback

(c) Else execute W(A)

set WTS(A) = TS(Ti)

> Validity-based Protocols: Execution of Ti is done in 3 phases:

(1) Read and execute phase: Write to temp variable

(2) Validation phase:

Validity test: Whether Ti can complete without violating serializability.

(3) Write phase: Write can be issued. No rollback.

- Each transaction Ti has 3 timestamps:

(1) Start(Ti)

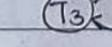
(2) Validation(Ti)

(3) Finish(Ti)

> DEADLOCK:

T3	T4
lock X(B)	
read(B)	
B = B-50	
write(B)	
	lock S(A)
	read(A)
	lock S(B)
lock X(A)	

- Neither T3 nor T4 can make progress
- Executing lock S(B) causes T3 to wait for T4 to release its lock on A.



Loop present. Deadlock.

- To handle deadlock, one of the transactions must be rolled back and its locks released.

> Two principle methods:

(1) Deadlock prevention protocols: ensures system never gets deadlock.

(2) Deadlock detection & recovery: Allows deadlock but trys to recover the system.

- Prevention: Requires that each T locks all its data items before it begins execution (predeclaration). But, it is not practical:
  - Often hard to predict before T begins, what data item needs to be locked.
  - Data item utilization is low.
- Strategies: Preemptive: T holding a lock is aborted to make data item available. Two prevention strategies:

(1) Wait-die scheme (non-preemptive): When Ti requests data item held by Tj, Ti is allowed to wait only if it has T.S. smaller than Tj, else it is rolled back (dies). Younger T doesn't wait for old ones, they are rolled back instead.

(2) Would-wait (preemptive): When Ti requests data held by Tj, Ti waits only if TS is larger than Tj, else Tj is rolled back. Older T forces rollback of younger T instead of waiting. Young T may wait for old T. Counterpart of 'wait-die' scheme.

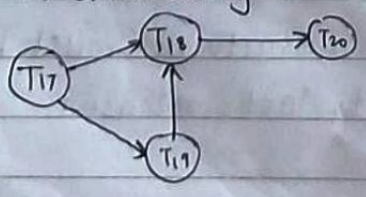
- Deadlock detection: Deadlocks described as a wait-for graph $G = [V, E]$ (V ≡ Vertices, E ≡ Edges)

(1) If $T_i \rightarrow T_j$ is in E, there is directed edge from Ti to Tj ⇒ Ti is waiting for Tj to release data item.

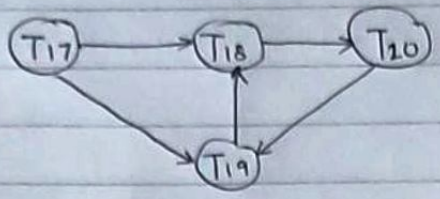
(2) Edge is removed only when Tj is not holding data that Ti wants.

(3) Deadlock if and only if the wait for graph has a cycle.

Eg:



No deadlock as no cycle.



T_{18} waiting for T_{20} waiting for T_{19}
Wait for graph with a cycle.
∴ Deadlock.

Either 18, 19 or 20 is rolled back.

- Deadlock recovery:

(1) Selection of victim: Select T that incurs min cost.
- how long transaction has computed
- how many data items used.
- how many more data items the transaction needs for it to complete.
- how many transactions are involved in rollback

(2) Rollback:
- Total rollback: abort transaction and restart it.
- Partial rollback: roll back the transaction only as far as necessary to break deadlock.

(3) Starvation:
If same transaction is always chosen as a victim.
- Include no of rollbacks in the cost factor to avoid starvation.

(Q) Consider schedules S_1 and S_2. S_1 has T_1 and T_2.

S_1

T_1	T_2
lock X(A)	
R(A)	
W(A)	
U(A)	
	lock S(B)
	R(B)
lock S(B)	
R(B)	
U(B)	
	U(B)

S_2

T_1	T_2
lock X(A)	
R(A)	
W(A)	
	lock X(B)
	R(B)
	lock S(A)
lock X(B)	
R(B)	
W(B)	
U(B)	
U(A)	
	U(B)
	U(A)

Answer: S_2 suffers deadlock
Roll back T_2.

✱ Normalisation.

- Process that improves database design by generating relations that are of higher normal forms. To create relations where every dependency is on the key, the whole key and nothing but the key.
  (whole = primary key)
- Functional dependency: (FD)
  table R with attributes A & B. B is functionally dependent on A if you can find B when A is given. $A \rightarrow B$
  (A = determinant)
  A determines B.
  Trivial: $X \rightarrow Y$ is true when Y is subset of X.
  Eg: Sid → Sid , Sid Sname → Sid.
  ∴ $LHS \cap RHS \neq \phi$ will imply $X \rightarrow Y$.
  Non trivial: when you are not sure about $X \rightarrow Y$.
  $LHS \cap RHS = \phi$ will imply non trivial case.
  Attribute B is func. dependent on A if and only if for each value of A no more than one value of B is associated. Value of A uniquely determines value of B.
- Inference rules for FDs.
- Given a set of FDs F, we can infer additional FDs that hold whenever FDs in F hold.
- Armstrong's inference rule.

A1. Reflexive: If Y is subset of X, $X \rightarrow Y$.

A2. Augmentation: If $X \rightarrow Y$ then $XZ \rightarrow YZ$
($XZ \rightarrow XUZ \rightarrow YUZ$)

A3. Transitive: If $X \rightarrow Y$ and $Y \rightarrow Z$ then $X \rightarrow Z$

- Decomposition:
  If $X \rightarrow YZ$ then $X \rightarrow Y$ and $X \rightarrow Z$
  NOTE: $XY \rightarrow Z$ does not mean $X \rightarrow Z$ and $Y \rightarrow Z$
- Union: $X \rightarrow Y$ and $X \rightarrow Z$ means $X \rightarrow YZ$
- Pseudotransitivity: If $X \rightarrow Y$ and $WY \rightarrow Z$ then $WX \rightarrow Z$
- Closure of a set F of FDs is set F+ of all FDs than can be inferred from F.

(Q) Shipment (BoxNo, DeliveryAdd, Date, Payment). With functional dependency ① BoxNo → DeliveryAdd

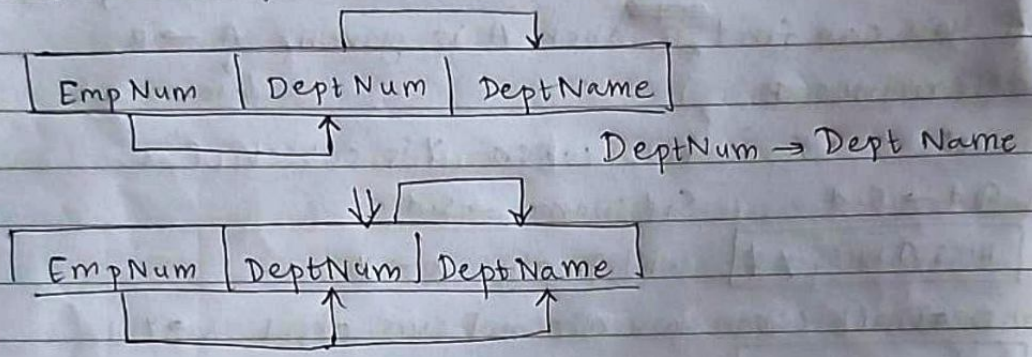
② {Date, DeliveryAdd} → Payment

Identify any other FD derived from this.

→ Pseudotransitivity : {Date, BoxNo} → Payment.

- Transitive Dependency:

EmpNum → DeptNum



(Q) Given: R(A, B, C, D)

FD = (A → B, B → C, C → D)

Find Candidate key (key on which other attributes depend)

$A^+ \Rightarrow \{A\ B\ C\ D\}$ (closure of A) $\quad C^+ \Rightarrow \{C\ D\}$

$D^+ \Rightarrow \{D\}$

$B^+ \Rightarrow \{B\ C\ D\}$

CK ⇒ {A}

PA ⇒ {A} Prime attributes ⇒ (used to form) Candidate keys

NPA ⇒ {B, C, D} Non Prime attributes.

(Q) R(A, B, C, D)

FD = (A → B, B → C, C → D, D → A)

$A^+ \Rightarrow \{A, B, C, D\}$ $\quad C^+ \Rightarrow \{C, D, A, B\}$

$B^+ \Rightarrow \{B, C, D, A\}$ $\quad D^+ \Rightarrow \{D, A, B, C\}$

CK = {A, B, C, D}

PA = {A, B, C, D}

NPA = { }

(Q) R (A, B, C, D, E)

FD = { A → B, BC → D, E → C, D → A }

↳ AC → D

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

$B^+ = \{B\}$

$C^+ = \{C\}$

$D^+ = \{D, A, B\}$

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

$DE^+ = \{DECAB\}$

$AE^+ = \{AEBCD\}$

$BE^+ = \{BECDA\}$

$CE^+ = \{CE\}$

While checking, note that E is getting determined only by E so E has to be there on LHS.

CK = {AE, DE, BE}

PA = {A, E, D, B}

NPA = {C}

(Q) R (A, B, C, D) (Prev eg)

A → B, B → C, C → D

CK = {A} * But {AB} can also be CK .... These are super keys don't consider.

- Candidate key: Minimal key which helps us to determine all attributes of given relation. Any attributes which you add with candidate key is called super key.

(Q) Consider the following relational schema:

R (CNo, DataUsed, DataRemain, LogId, Date, Hours)

① CNo → DataUsed

② LogId → DataRemain

③ CNo → LogId

④ DataUsed → Date

Which of the FDs must be added to make CNo as candidate key?

→ $(CNo)^+ = \{CNo, DataUsed, LogId, DataRemain, Date\}$

Add: CNo → Hours OR LogId → Hours OR any such combination where RHS is Hours.

- Lossless / Lossy Join Decomposition:

Rules:

- $R_1 \cup R_2 = R$
- $R_1 \cap R_2 = \phi$

Eg: R

A	B	C
1	2	1
2	2	2
3	3	2

$R = R_1 R_2$ Let $R_1 = (AB)$ $R_2 = (BC)$

R_1

A	B
1	2
2	2
3	3

R_2:

B	C
2	1
2	2
3	2

$R_1 \times R_2$

A	B	B	C
1	2	2	1
1	2	2	2
1	2	3	2
2	2	2	1
2	2	2	2
2	2	3	2
3	2	2	1
3	2	2	2
3	2	3	3

R_1 Natural Join R_2

A	B	B	C
1	2	2	1
1	2	2	2
2	2	2	1
2	2	2	2
3	3	3	2

A	B	C
1	2	1
1	2	2
2	2	1
2	2	2
3	3	2

} Lossy as not in R.
∴ Not correct decomposition

Eg 2: BC, AC

R_2

B	C
2	1
2	2
3	2

R_1

A	C
1	1
2	2
3	2

$R_1 \times R_2$

A	C	B	C	
1	1	2	1	✓
1	1	2	2	✗
1	1	3	2	✗
2	2	2	1	✗
2	2	2	2	✓
2	2	3	2	✓
3	2	2	1	✗
3	2	2	2	✓
3	2	3	2	✓

A	B	C	
1	2	1	
2	2	2	
2	3	2	
3	2	2	Lossy.
3	3	2	

$\underset{R_1}{AB}, \underset{R_2}{AC}$ is lossless as it is correct decomposition.

NOTE: Take common attribute as one which doesn't hv duplicates. (Here, A)

- Whenever we decompose a table, there must be atleast one common attributes between 2 decompositions & this common attribute should be candidate key or super key.

(Q) Company Database R=(Company, Material, Type, cost)

① Company → Material

② material → Type

③ Type → Cost

(i) R_1 = (Company, Type, Cost)

(ii) R_2 = (material, Type)

→ $Type^+ \Rightarrow \{Cost, Type\}$

Common attribute is not candidate key. Thus, lossy.

(Q) Work (Incharge, Dept, ~~Emp~~ Exp)
(Incharge: duplicate; Dept: unique; ~~unique~~ Exp: duplicate)

① Incharge → Dept ⑤ ~~Exp~~ → Incharge

② Incharge → Exp ④ Dept → Exp

Incharge	Dept	Exp
Roy	Prod	5
Sinha	XYZ	2
Roy	HR	5
Sinha	Testing	4

> Anamolies:

- Updated Anamoly: To update one value, many rows need to be updated.
- Insertion Anamoly: Only some column entries cannot be inserted independently, primary key has to be inserted. Eg: For inserting Faculty into, you need student id.
- Deletion Anamoly: ~~Few values Do~~ To delete one value, entire row is deleted.

Eg:

Sid	Sname	Cid	Cname	Fid	Fname
1	A	C_1	D	F_1	F
2	B	C_2	E	F_2	G
3	C	C_1	D	F_1	F

∴ Break the table into components & to remove anamolies.
Normalize

> First Normal Form (1NF): All columns contain only scalar values i.e, no multiple values.

- Decomposition:

(1) Place all items that appear in repeating group in new table.
(2) Designate primary key for each new table ~~of~~ produced.
(3) Duplicate in new table, primary key of the table from which repeating group was extracted or vice versa.

Eg.

ENo	Ephone	EDegree
333	123	BA, PhD
679	456	BSc, MSc

Employee

ENo	Ephone
333	123
679	456

EmpDegree

ENO	EDegrees
333	BA
333	PhD
679	BSc
679	MSc

> Second Normal (2NF): Conditions:

(1) Database should be in 1NF

(2) All nonkey (non prime attributes) attributes should be fully functionally dependent on primary key. There should not be any partial dependency.

Eg:

Cust ID	store ID	Location
1	1	Delhi
1	3	Mumbai
2	1	Delhi
3	2	Bangalore
4	3	Mumbai.

Here, location is a non prime attribute. It only depends on store ID not Cust ID ∴ Partial dependency ⇒ not in 2NF.
In order to convert the given relation in 2NF, decompose it as

Cust ID	store ID
1	1
1	3
2	1
3	2
4	3

store ID	Location
1	Delhi
2	Bangalore
3	Mumbai

∴ Both tables are in 2NF now.

- Rule for 2NF: LHS of FD shd be proper subset of CK AND RHS shd be non-prime attribute

(Q) R (A B C D E F) Identify if it is in 2NF:

FDs: $C \to F$ $C^+ = \{F, C\}$ $B^+ = \{B\}$ $F^+ = \{F\}$

$E \to A$ $E^+ = \{A, B, E\}$ $EC^+ = \{E, C, D, A, B, F\}$

$EC \to D$ $A^+ = \{A, B\}$ ↑ Candidate key.

$A \to B$ $D^+ = \{D\}$ Non prime = {A B D F}

Partial dependency ∴ Not 2NF

(Q) R (A B C D) Check if it is in 2NF.

$AB \to CD$ $\xrightarrow{AB \to C}$ $\xrightarrow{AB \to D}$

$D \to A$

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

$AB^+ = \{A B C D\}$, $BD^+ = \{A B C D\}$

Candidate key = {AB, DB} Prime = {A, B, D}. Non prime = {C}

∴ No transitive dependence, it is in 3NF. (by the rule given below)

> Third Normal (3NF): Conditions:

(1) Should be in 2NF

(2) No attribute is transitively dependent on primary key.

(Q) Find if it is in 3NF R (A B C D)

$AB \to C$

$C \to D$

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

$AB^+ = \{A B C D\}$ Non-prime = {C, D}

AB is the candidate key but there exists transitive relation between D and AB. Thus, not in 3NF.

~~(Q) Scheme~~

- Decomposition of 3NF:
  - Move all items involved in transitive relation to new table
  - Find primary key for new entity
  - Reference it as foreign key on original table

∴ Decomposition of above Q: ABC and scond table is ACD OR BCD depending on which primary key you select as foreign key.

- Rule: To check relation in 3NF:

For each FD, LHS must be a candidate/super key OR RHS is a prime attribute.

(Q) (SNo SName S_State SCountry SAge) R

SNo → SName

SNo → S_State

S_State → SCountry

SNo → SAge

$B \rightarrow C$ ✗

BE = {BEF}

$SNo^+ = \{SNo, SName, Sstate, SAge, Scountry\}$

CK = {SNo} prime = {SNo} non-prime {SName, Sstate, SAge, Scountry}

Not in 3NF as transitive relation exists.

Decomposition: ① SNo, SName, S_State, SAge

② SNo, S_State, S_Country

> BCNF (Boyce Codd): Does not allow dependencies between attributes that belong to the ~~prim~~ candidate key. It is like 3NF, drops the restriction of non-key attribute from 3NF. 3NF & BCNF are not same if:

① Table has 2 or more CKs

② Atleast 2 CKs are composed of more than one attribute

③ Keys are not disjoint. (~~CKs~~ Composite CKs share same attributes)

– Decomposition:

- Place 2 CKs in seperate entities.
- Place each of remaining data item in one of the resulting entities according to its dependency on the primary key.

– Rule: LHS of each FD shd be CK or super key.

(Q) Rollno → Name

Rollno → Voterid

Voterid → Age

Voterid → Roll no

$RollNo^+ = \{RollNo, Voterid, Age, Name\}$

$Voterid^+ = \{Voterid, Age, RollNo, Name\}$

CK = {Rollno, Voterid}

All FDs satisfy rule ⇒ Given FDs are ~~relation is~~ in BCNF.

Relation may or may not be in BCNF.

(Q) R = (A, B, C, D, E, F) Check highest Normal form

✗ $AB \rightarrow C$ $F \rightarrow A$ $A^+ = \{A\}$ $B^+ = \{B\}$ $C^+ = \{CDEFA\}$

$C \rightarrow DE$ $AB^+ = \{ABCDEF\}$ $BC^+ = \{BCDEFA\}$ $BF^+ = \{BFACDE\}$

$E \rightarrow F$ CK = {AB, BC, BF, BE} Prime = {A, B, C, F, E} Non-prime = {D}

	AB→C	C→DE	E→F	F→A
BCNF	✓	×	×	✓
3NF	✓	×	×	✓
2NF	✓	×	×	✓
1NF				

	AB→C	C→DE	E→F	F→A	Status
BCNF	✓	×	×	×	No
3NF	✓	×	✓	✓	No
2NF	×	×	×	×	No
1NF					

Answer: With the help of normalisation rule, we observe that given relation is not in BCNF, 3NF, 2NF. For 1NF, we require all scalar values for each attributes. In given relation, with assumptions of having all values as scalar for each attribute, you can conclude that given relation is in 1NF.

(Q) R [A B C D E)

$A \rightarrow CB$

$E \rightarrow CBD$

$DCB \rightarrow A$

$A^+ = \{A\ CB\}$ $B^+ = \{B\}$ $C^+ = \{C\}$ $D^+ = \{D\}$ $E^+ = \{ECBD\}$

CK = {E} Prime = {E} Non-prime = {A B C D}

	A→CB	E→CBD	DCB→A	Status
BCNF	×	✓	×	No
3NF	×	✓	×	No
2NF	×	✓	×	No
1NF				

> 4NF: Conditions:

① ~~Rule:~~ Must be BCNF

② A given relation may not contain more than one multi-value attribute.

(Q) R = (A B C D E F G)

$A \rightarrow E$

$BC \rightarrow GD$

$D \rightarrow F$

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

$AC^+ = \{A\ C\ E\}$

$ABD^+ = \{A\ B\ D\ E\ F\}$

① $(A,C) \rightarrow F$

② $(B,D,E) \rightarrow B$

③ $(A,B,D) \rightarrow C$

④ $(A,B,C) \rightarrow (E,F)$

(Q)

X	Y	Z
1	1	a
1	1	a
2	1	b
3	2	b

Check whether following FDs are true:

(1) $X \rightarrow Y$

$Z \rightarrow X$

(2) $Y \rightarrow Z$

$Z \rightarrow X$

(3) $X \rightarrow Y$

$X \rightarrow Z$

(4) $Y \rightarrow X$

$X \rightarrow Z$

(Q) Relation student has RegNo, Name, Address, PhNo, Cid.

FD: RegNo → Name, Address

Address, PhNo → Cid

Name → PhNo

Cid → RegNo

$RegNo^+ = \{RegNo\ Name\ Address\ PhNo\ Cid\}$ ✓

$Cid^+ = \{Cid\ RegNo\ Name\ Address\ PhNo\}$ ✓

ck = {RegNo, Cid, Address PhNo} Prime: {RegNo, Cid, Address, PhNo} Non prime = {Name}

(Q) R = (Phone, Name, Address, Room, Floor, stay)

Phone, Name → Address

Phone → Room

Name → Floor, stay

Highest NF?

Phone$^+$ = {Phone, room}

Name$^+$ = {Name, floor, stay}

(Phone, Name)$^+$ = {Phone, Name Address Room Floor stay}

CK = {Phone, Name}

~~Prime~~ Non-prime = {Address Room floor stay}

				Status
BCNF	✓	x	x	No
3NF	✓	x	x	No
2NF	✓	✓	✓	Yes
1NF				