1. "Database System Concepts" – Korth
2. "Fundamentals of Database Systems" – Navathe, Elmasri
3. "Database Mangt. Systems" – Ramakrishnan
4. "An Intro. to DB Systems" – C. J. Date.
5. SQL/PLSQL – The Programming Language of Oracle – Ivan Bayross

Database – a collection of related data (containing info. relevant to an enterprise, used by the application systems of some given enterprise)
• random collection of any data can't be correctly referred as database

Database management System (DBMS) – collection of related data and programs to create, maintain and utilize those data.
DBMS – a s/w system to manage database.
• primary goal of DBMS to provide a way to manage db both conviniently & efficiently.

utilize:
• retrieve specific data,
• update database to reflect changes
• generate reports
• share db to allow multiple users & programs to access db (simultaneously)

Applications

- Banking
- University
- Railway / Airlines (reservation system)
- Manufacturing
- Human Resources
- Telecommunication

- Data mining – applications that analyze large amounts of data searching for the occurrences of specific patters or relationships.

- Users of DB:-

1) End users / Naive users – who accen db interact via (online) application programs / interface provided as integral part of system. – menu-or form-driven interface.

2) Application programmers – implement the user/client requirements in programs, communicate with db. — so, they should be familiar with full range of capabilities (structure etc.) to do their tasks. The programs accen db by issuing appropriate requests (SQL statements) to the dbms. /commands

3) Database designers – responsible for identifying data to be stored in db and choosing appropriate structures to represent & store the data. So they have to communicate all db users to understand their requirements.

4) Database Administrators – As the db is used by many people, there is a need for a chief administrator to oversee & manage the database resources. He/she has the central control over both data & prog.s to access the data. (Practically, DBA functions are performed by a team of people, not just one) (for simplicity, we assume dba as a single individual). Functions of DBA include

- schema definition – creats original db schema
- define storage structure, access method
- defi granting authorization for data access
- routine maintainance
- periodically backing up database & maintaining logs of system activity to prevent data loss & facilitate recovery if system fails.
- coordinating & monitoring database use

Database & description of database are different.

- Description of database / overall design of db is called database schema, which is specified during db design and is not changed frequently (if req. change, then schema changes)

- Actual data in db may change frequently as new data is inserted & deleted. Data stored in db at a particular moment is called an instance / snapshot of db / a database state.

Data Models :- A data model is a collection of concepts used to describe the structure of a db (data types, data relationships, data semantics and constraints that should hold for data. (Most data models also include a set of basic operations of retrievals & updates)

Categories of data models :-

1) Relational Model :- represents database as a collection of relations. - main construct of this model is relation which can be thought as a set of records. We store relations / records in form of tables. Each row of table represents a record.
record → collection of related data value

- mathematically, relation is a subset of a cartesian product of a list of domains.
- table is subset of $D_1 \times D_2 \times D_3 \dots \times D_n$ (of n attributes)

* student is a subset of $D_1 \times D_2 \times D_3$

→ a record in student relation describes a student.

attributes → column headers → fields
set of permitted values for that attribute → domain
row of a table → tuple | relation is a set of tuples,
$R \subseteq A \times B$

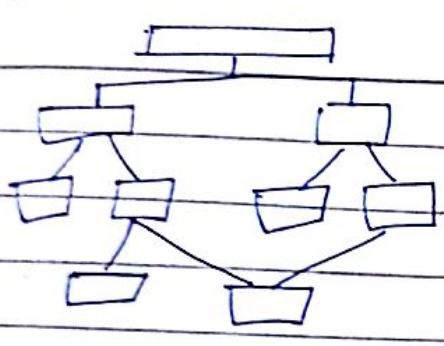
2) Entity Relationship Model - represents db as a collections of real world objects called entities and relationship among these entities.
Diagrammatic notation associated with ER Model ⇒ ERD

3) Object Based Data Model:- can be seen as extending ER model with notations of object oriented concepts — inheritance, encapsulation (info. hiding).

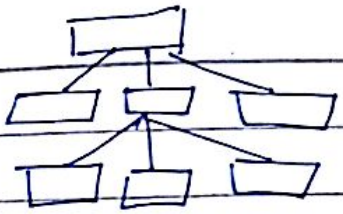
4) Semistructured data model:- individual data items of same type may have diff. sets of attributes. In previous models, each data item of a particular type has same set of attributes. XML (Extensible Mark up Language) is used to represent semistructured data.

5) Network & Hierarchical Model:- Network model is a flexible way of representing objects & their relationships. Here, schema viewed as graph where nodes → objects, arcs → relationships. While hierarchical model structures data as a tree of records with each record having one parent record and many children; the network model allows each

record to have multiple parent & child records, forming a generalised graph structure



Network model



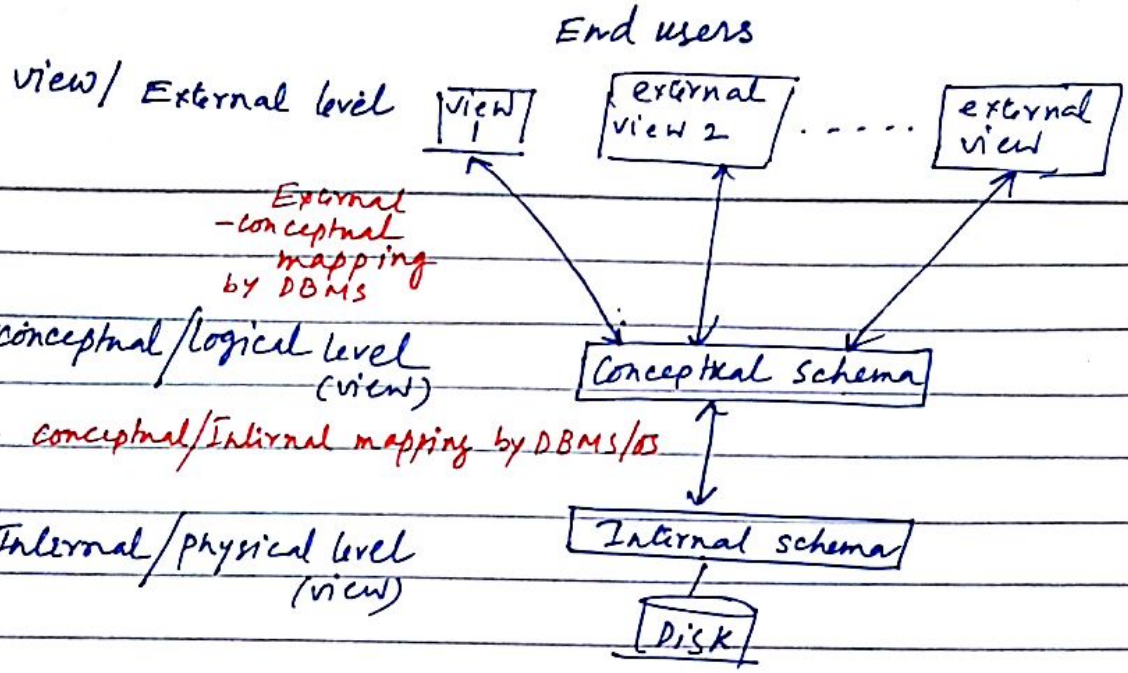
hierarchical Model

These 2 models preceded relational model etc.

Three-Schema Architecture:- goal of 3-level or 3-schema architecture is to provide levels of abstraction → to seperate user applications and physical db. For efficiency, complex ds is used to represent data in db, as many users not comptr trained and they need not to concern about how data stored, dievelopers hide these storage details through several levels of abstraction, to simplify users' interaction with system.

1) Internal level has an internal schema which describes details of physical storage structure of db in hard disk and access paths used by db. It's at the lowest level of abstraction

2) Conceptual/logical level has a logical schema that describes entities, records structure, relationships among data, constraints, data types.



conceptual view/level hides details of physical storage structure and gives somewhat abstraction in comparison with physical view.

- DBA concerns about logical & physical level.
  works at

3) External level is at the highest level of db abstraction. This level includes a no. of external views → external schemas. Each external schema describes that part of the db as represented /shown by that respective external view → users/application programs can only see that part of db.

- application programmers at external & logical level

* 3 schemas are only descriptions of data; the stored data actually exists in physical level. In a DBMS based on 3 schema archi, each user group refers to its own external schema/view. DBMS must transform a request on an external schema to a request

against conceptual schema, and then into a request on the internal schema for processing over stored db. For data retrieval request, data extracted from stored db must be formatted to match with user's external view. This process of transforming requests and results between levels are called mapping.

Data Independence:-

3 levels of abstraction provides 2 types of data independence — Logical & Physical. Data Inde. defined as the capacity to change the schema at one level of db system without changing schema at next higher level.

Logical data Independence:- indicates that logical schema can be changed without affecting existing external schemas. Logical schema can be changed by adding field/columns, changing constraints or deleting some field. In first & last case, external schemas that refer related to previous & remaining data should not be affected. Changes are absorbed by mappings that may need to be changed. So though conceptual schema undergoes a logical reorganisation, appli. prog. comes. & external schemas must work as before.

Physical data Independence:- physical schema can be changed without changing conceptual schema → hence, external schemas need not be changed at all. Physical files reorganisation or creating additional access structures to improve performance does not require changes in logical schema.

Advantage of DBMS:-

1) Controlling Redundancy & inconsistency:- In university db, course registration & accounting office — both keep data on students, redundancy in storing same multiple times leads to several problems — duplication of effort, storage space wasted, inconsistency arises.

2) Restricting Unauthorized Access → thus ensuring security.

3) easy access / efficient access

4) Multiple users - shared data - concurrent access anomalies doesn't occur.

5) Data Independence

6) Provide Backup & Recovery.

• A query is a statement requesting retrieval of info.
→ Informally, all types of request (updation, creating tables) called queries.

• Database languages:-

DDL - to specify db schema ; o/p of DDL is placed in data dictionary, which contains meta data - data about data

DML - to express db queries & updates

/

DCL -

Entity-Relationship Model

E-R model → high level conceptual data model

Conceptual Schema - after requirements collected & analyzed, conceptual schemas for the db created → it's a concise description of data req.s of users & includes detailed description of entity types, relationships & constraints. → It doesn't include implementation details, easier to understand, used to communicate with end users - conceptual design phase.

- From conceptual schema to implementation data model → logical design
- physical design → internal storage structures, access paths, indexes, file organization for db specified.

Entity - thing in real world with an independent existence. It may be an object with physical existence (person, house) or with conceptual existence (company, project, course)

Attributes - particular properties that describe entity. An entity is represented by its attributes.

- A particular entity has a value for each of its attributes.
- collection or set of entities with same attribute → entity type
- Each simple attribute of an entity type is associated with a value set / domain of values → values that can be assigned

to that attribute for each individual entity. Value sets are specified using data types.

□ Composite - Simple (Atomic) attributes:-

Composite attributes can be divided into smaller subparts, which represent more basic attributes with independent individual meanings.

• Address — state, city, street-name, pin

Attributes not divisible → simple

• user may sometimes refer to composite attribute as a unit or other times refer specifically to its components.

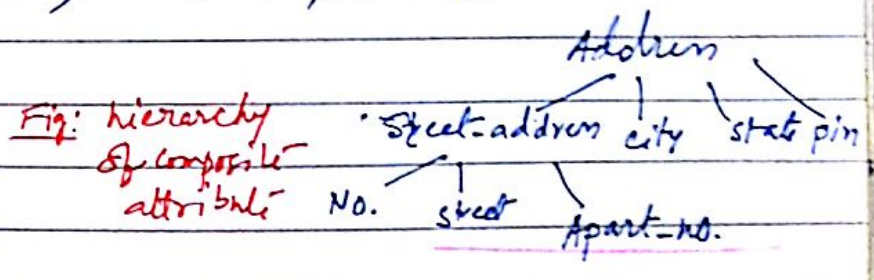


Fig: Hierarchy of composite attribute

Single-valued vs. Multivalued Attributes:-

attribute having a single value for a particular entity - single valued attribute having a set of values for same entity - multivalued

e.g- phone_no, colour of a newly launched car, degrees of a person

* An entity type describes schema for a set of entities that share same structure
* Entity type is represented in ER diagrams.
• Diagrammatic notation associated with ER model - ERD

To digrammatically represent the database schema modelled by E-R model ERD used.

Stored vs. Derived Attribute

Birth-date & Age - two attributes of an employee. If birth-date is known, age of that person can be determined. So, Age is a derived attribute → derivable from stored attribute dob

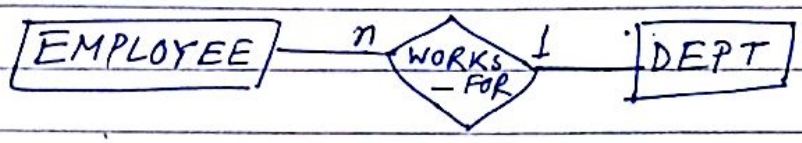
Null value :- In some cases, a particular entity may not have an applicable value for an attribute. E.g.:- Flat no is not applicable who have own residences / college degree may not applicable for an employee - For these cases, special value NULL is created. (If value of an attribute for a particular entity not known — NULL is used: home-phno may not be known, height of a person is missing

Key Attribute - There is must be some way to distinguish each entity among the entity set. An entity type usually has an attribute whose values are distinct for individual entity. That is key attribute - value of which used to identify each entity uniquely. (A key that uniquely identifies each record in a table - primary key of that table)

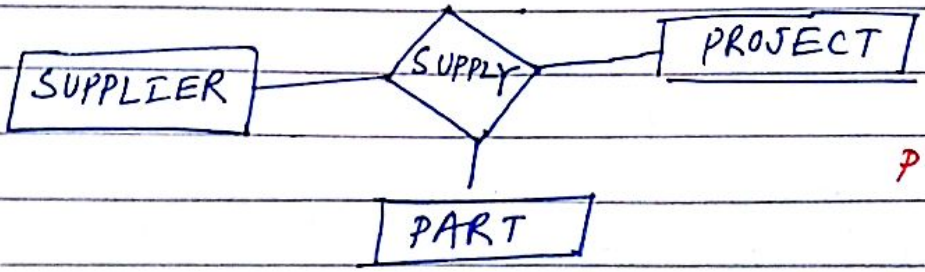
Relationship Set/type:- set of relationship instances r_i
mathematically, relationship type is a subset of cartesian product $E_1 \times E_2 \times \dots \times E_n$; Each of the entity types $E_1, E_2, \dots, E_n$ is said to participate in relationship type R.

- The no. of entity sets/types participate in a relationship type → degree of relationship
most relation is of degree 2 - binary
relationship of degree 3 → ternary

e.g.



⇒ WORKS_FOR is binary relationship



⇒ SUPPLY is a ternary Relationship where, S supplies part P to project j

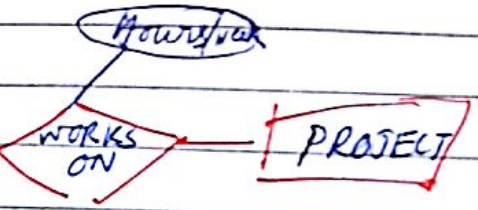
- Mapping Cardinality/
Cardinality Ratio for Binary Relationships — no. of entities to which another entity can be associated via a relationship set.

Mapping cardinality must be one of following.

One-to-many — in WORKS_FOR relationship betn DEPT & EMPLOYEE → DEPT: EMP is of cardinality ratio 1:N
vice-versa ~~1~~ N:1

, meaning each dept can be related to any no. of entities. but an emp. is related to only one dept. (zero/more)

- One-to-One – an employee MANAGES a dept. → one entity of employee entity set can be associated with at most one entity of dept entity set



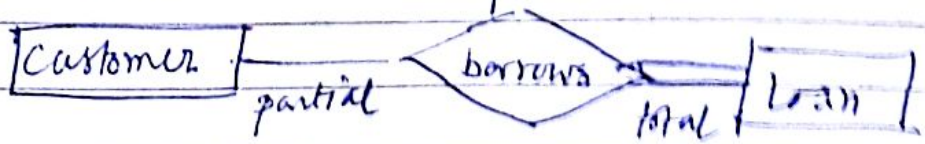
- Many-to-Many –

WORK_ON is of cardinality ratio M:N – an employee can work on several projects and a proj. can have several employees.

→ Participation Constraints: participation of an entity set E in a relationship set R is total if every entity in E participate in at least one relationship instance in R. e.g. participation of EMPLOYEE in WORKS_FOR is total.

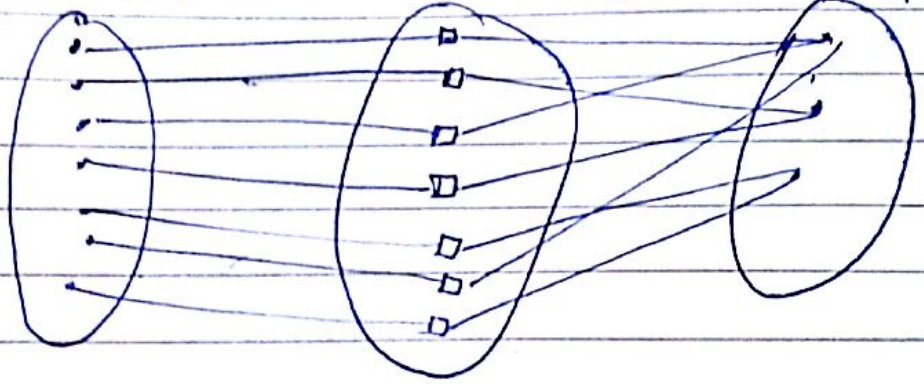
If only some entities in an Entity set E participate in relationships in R → participation of Entity is partial.

e.g. participation of EMPLOYEE in MANAGES is partial as every employee doesn't manage a dept., some employees are related to some departments via MANAGES relationship.



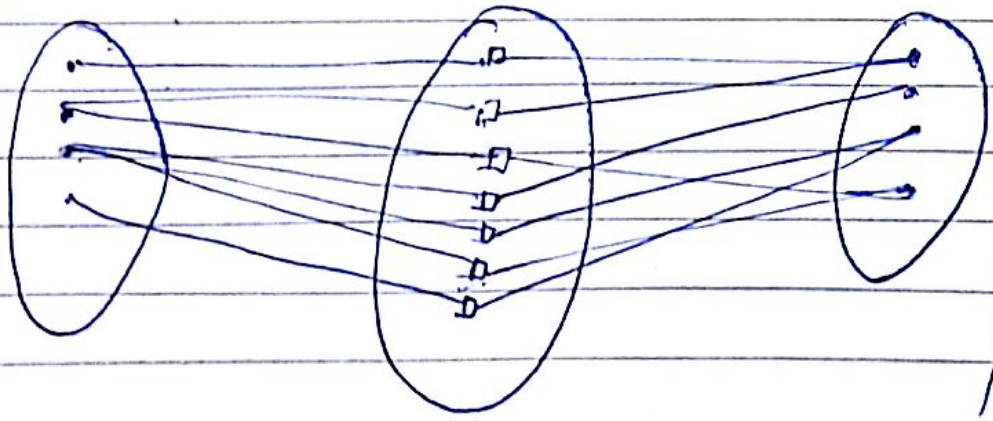
EMPLOYEE MANAGES DEPARTMENT 1:1

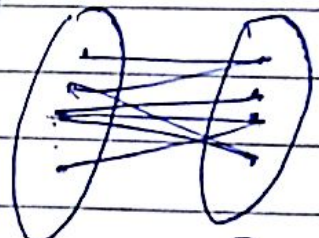
EMPLOYEE WORKS-FOR DEPT



1:N

EMP WORKS-ON PROJ.





M:N

Recursive Relationship:- In some cases, the same entity type participates more than once in a relationship type in different roles. Here, role name read. for distinguishing meaning of each participation. Such relationship → recursive relationship

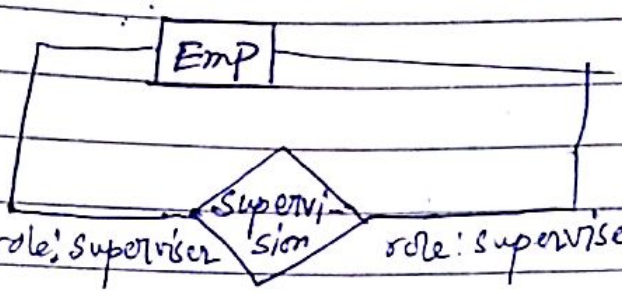
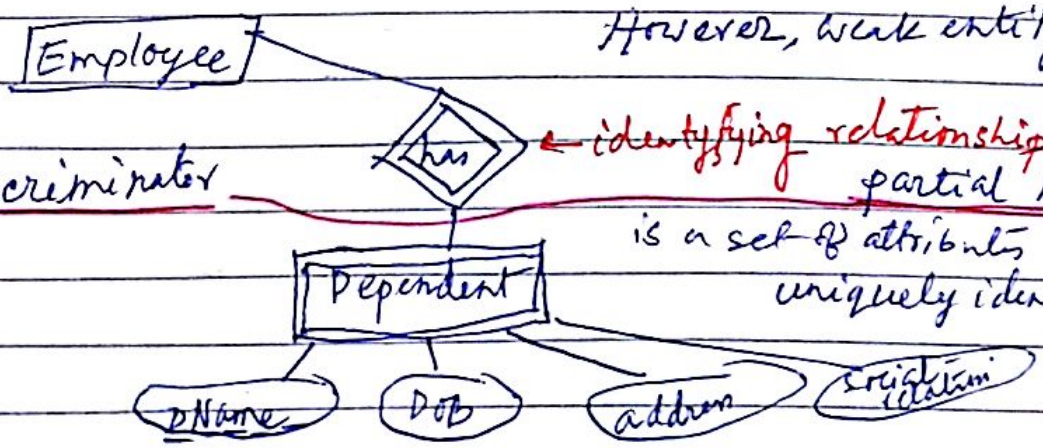


Fig: supervision relationship relates supervisor & supervisee — both are members of same Emp entity. So, Emp. entity participates Twice in SUPERVISION in two diff. roles.

Weak Entity type:

Entity type that don't have key attribute of its own.

Weak entity set must be associated with some another entity type — identifying/owner entity. Two dependents of 2 distinct entities may have same values for. name, dob, sex, relationship. So, they can be distinguished using key identifying entity.



However, weak entity type generally has a partial key — which is a set of attributes that can uniquely identify weak entities related to same owner entity

Attributes of Relation type:-

- Hours attribute of relation works_on. – can be determined by an employee-proj. comb. & not separately by either entity
- Start-date of manager can be migrated to one of participating entity types.
- attributes of 1:1 or 1:N relationship can be migrated to one of participating entity.

Min-Max:- it is written in L...h form, where L denotes minimum and h denotes maximum no. of relationship an entity participate in, but positioning of constraints is exactly the reverse of positioning of constraints in E-R diagram

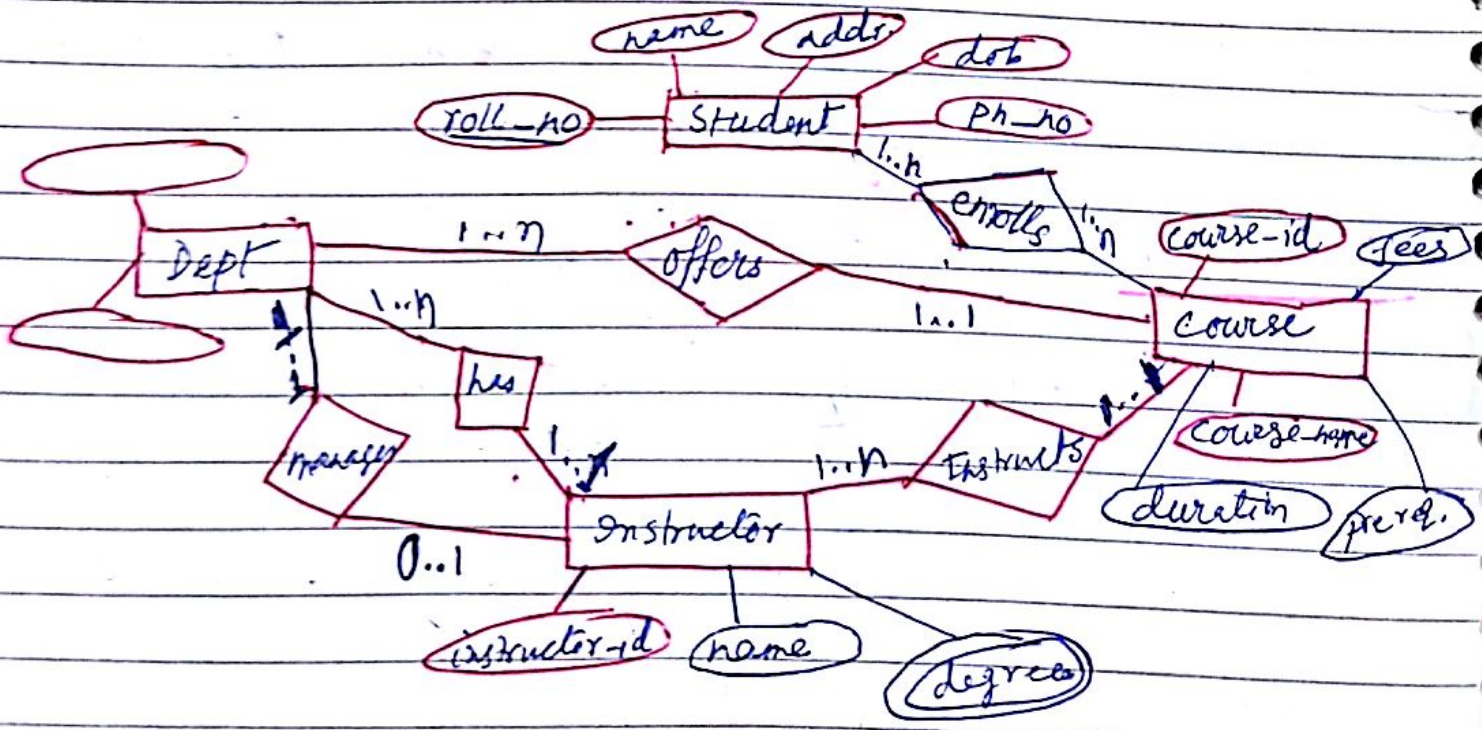
P-252-53 (KORTH)

- Foreign key:- table (R2)
  Primary key of a relation is set as foreign key of another attribute to relation (table) to relate them.
  R1 ⇒ referencing relation
  R2 ⇒ referenced "

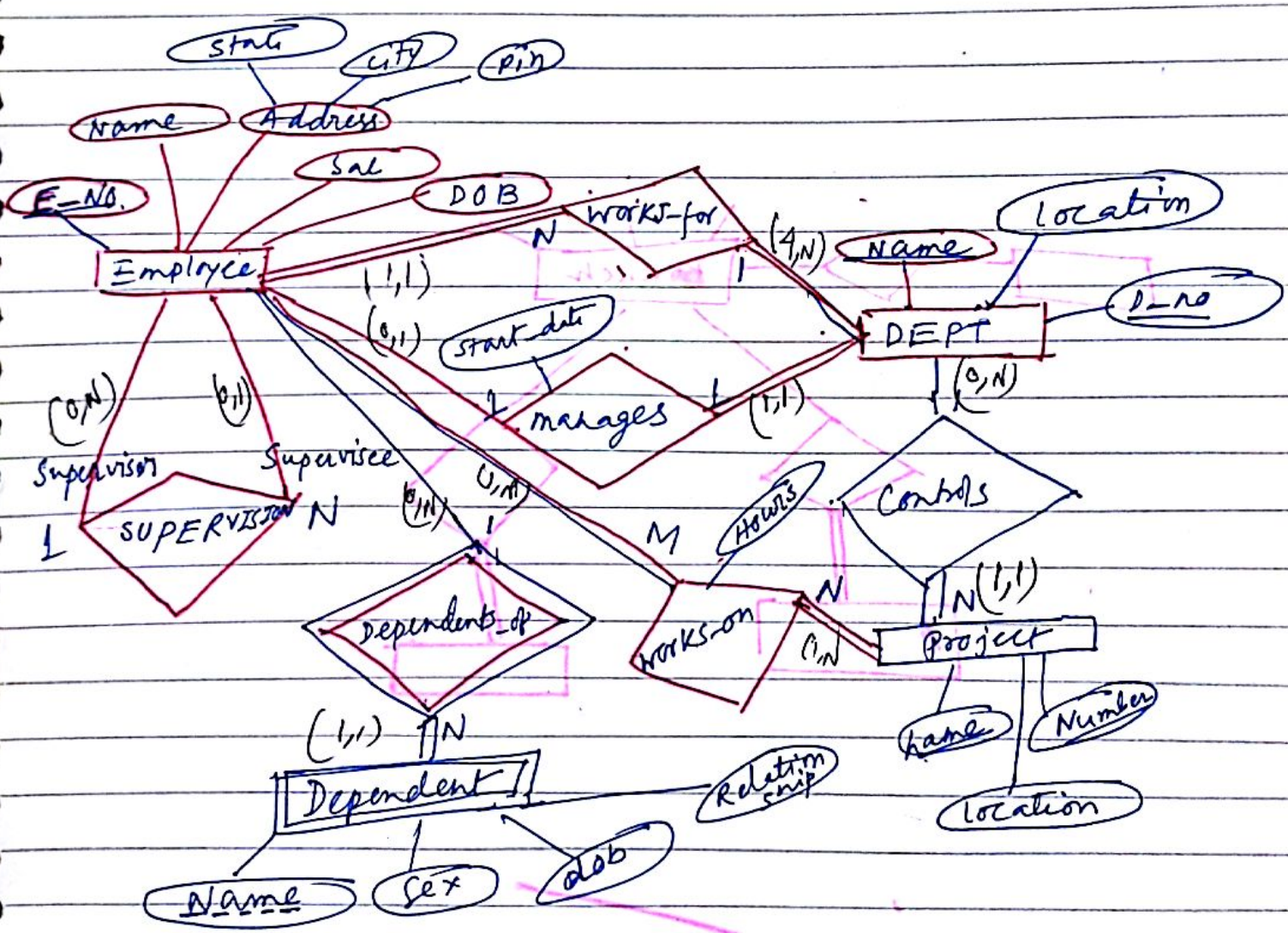
→ single values such as 1 or * may be written on edges. the single value 1 on an edge is treated as equivalent to 1...1, while * equivalent to 0...*

Prob. Institute
College has many dept.s. Each dept. offers any no. of courses but a course is conducted only by a single dept. Instructors work for a particular dept. & dept. has several instructors. Each dept. has a head and an instructor can be head of only one dept. ~~For~~ Each instructor can take any no. of courses but a course can be taken by only one instructor. A student can take any no. of course & a course has many no. of students. Each dept. has unique dept. name & a location. ~~We keep track course~~

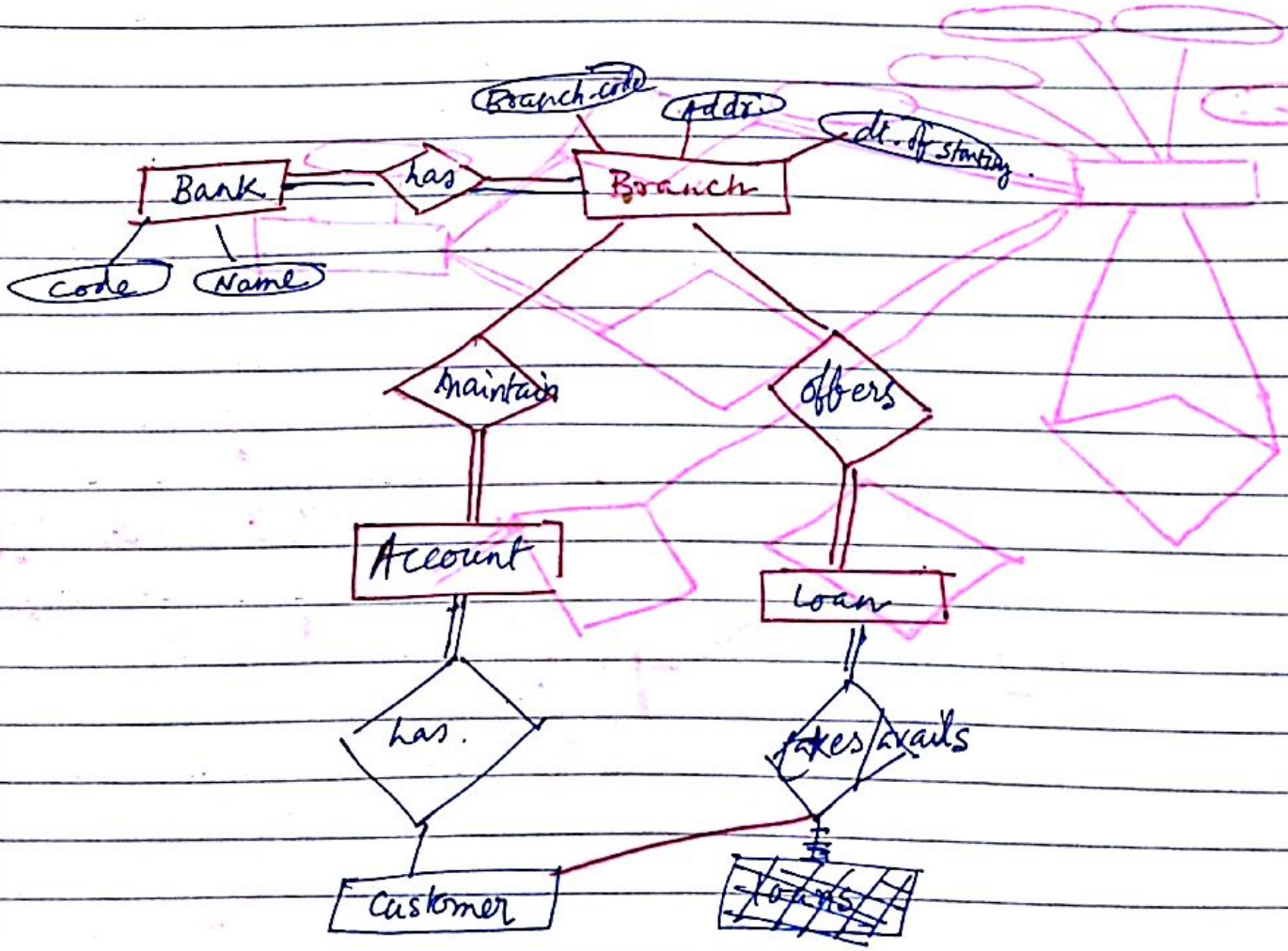
Course → course_id, duration, name, fees, pre-requisite
Instructor → instructor_id, name, ph_no
Student → roll_no, stu_id, name, dob, add, ph_no.



Prob: In a company, many employees work for ~~a~~ department. There are ~~are~~ no. of dept.s. Under a supervisor, there are a no. of supervisees. There is an employee who manages a dept. ~~for a dept.~~ manager from employee for a dept. Dept. controls many projects. An Employee works-on several projects, and a project has several employees. Employees has dependents - we have to keep track of them.

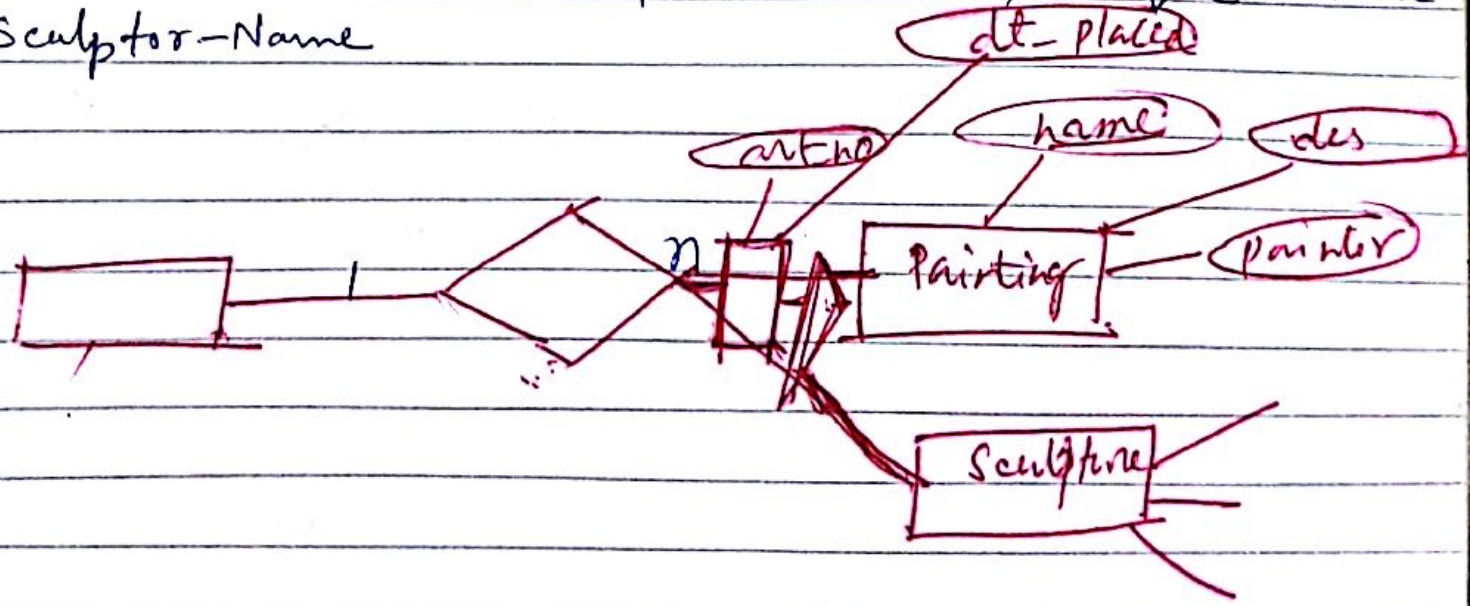


Prob: There are different banks. Each bank has several branches. Each branch maintains accounts of customers and customers have any no. of accounts. Each accounts has unique acc. no., ~~bin~~ balance, ~~check facility~~ etc. Each bank offer loans to customers. a customer can avail any loan account. Each loan account has unique loan_acc_no., amount, installment no., interest rate - Each Branch has unique branch_id., name, along with date of inauguration. Customer has custo. id, name, address, phone no.



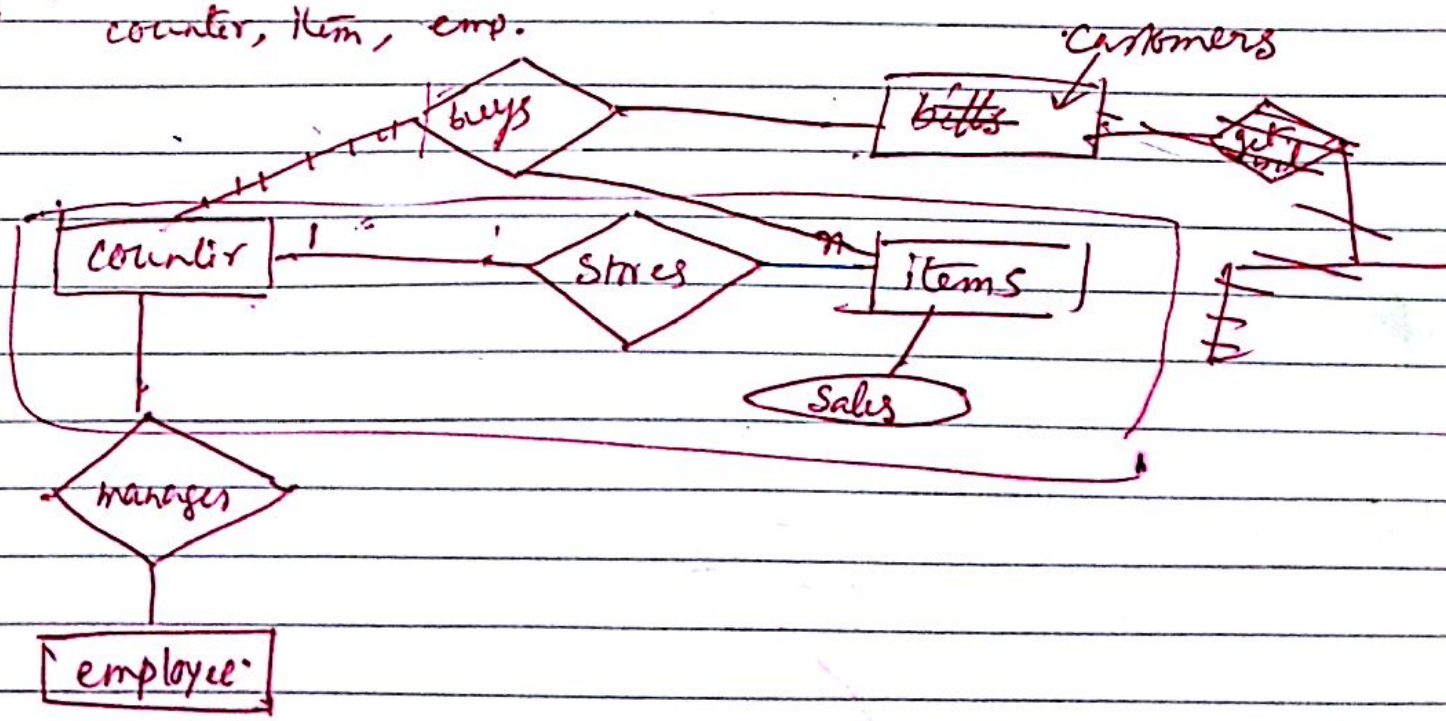
An Exhibiting organization keeps info about paintings and sculptures. Painting has Painting-Name, Painter-name, Painting-description. Sculpture has Sculptor-name, Sculpturename, and Sculpture-des. Painting and sculpture may appear in the same gallery. For the purpose of keeping track of location of items, each painting and sculptures ~~may appear in the same gallery~~ is given a unique identifier, Art-No. Each gallery has an identifier, Gallery-No, and a size. Each gallery can store any no. of art objects. Each art object appears in one art gallery only. The Dateplaced-In-Gallery is kept for both paintings and sculptures.

Note that the painting-name is unique within painter-name, and Sculpture-Name is unique within Sculptor-Name



Construct E-R diagram:-

A store has different counters managed by different employees. A counter has diff. items, but no two counters have common items. Customer buy from different counters but bills are prepared at bill counter only. Once in a month performance of persons managing counters is evaluated in terms of sales. Items are also reviewed and slow moving items are identified.



III

1. Reduce ER schema to ..

Describe entities, attributes, relationships & primary keys.

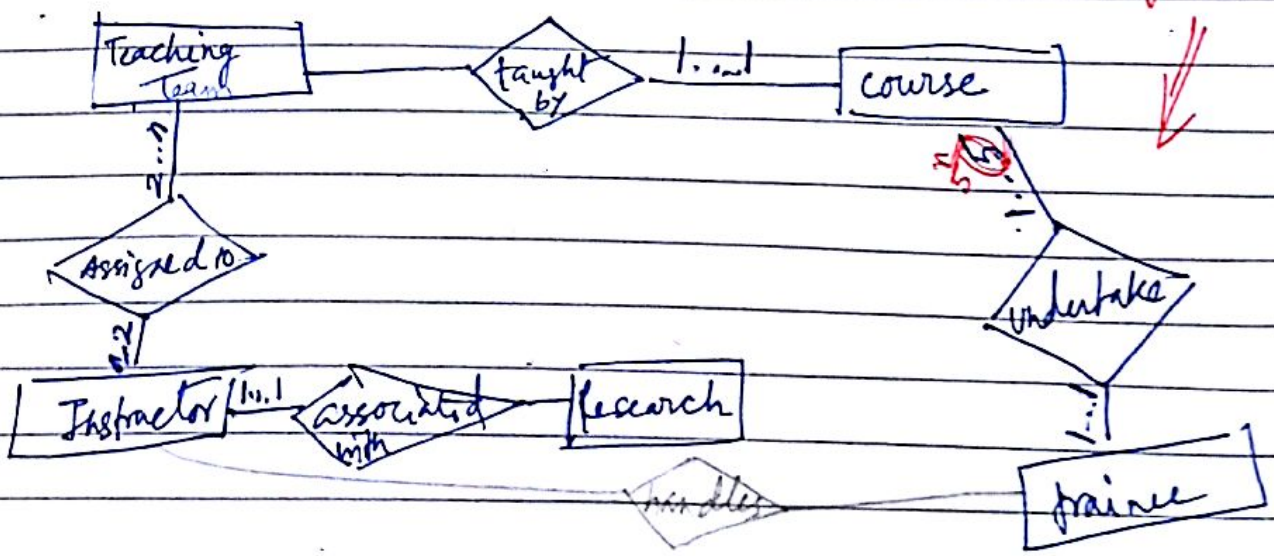
Draw an ERD for a travel agency consisting of following.
· customers, buses, drivers, conductors, guides, tickets, booking, agents, reservations, conducted tours & hotels.

11/ Draw an ERD for IT Training gr. (Kol) based on following info.

The IT Training gr. (Kol) has contacted you to create a conceptual model by using the ER data model for a db that will meet the info needs for its training prog. The company director has provided the following description of the training group's operating environment. The company has 12 instructors & can handle up to one hundred trainees per training session. The company offers five advnd tech. courses, each of which is taught by a teaching team of 2 or more ~~students~~ instructors. Each instructor is assigned to a max of 2 teaching teams or may be assigned to do research. Each trainee undertakes one advnd tech. course per training session.

⇒ Distinguish betwn. single valued/Multivalued and stored/derived attributes

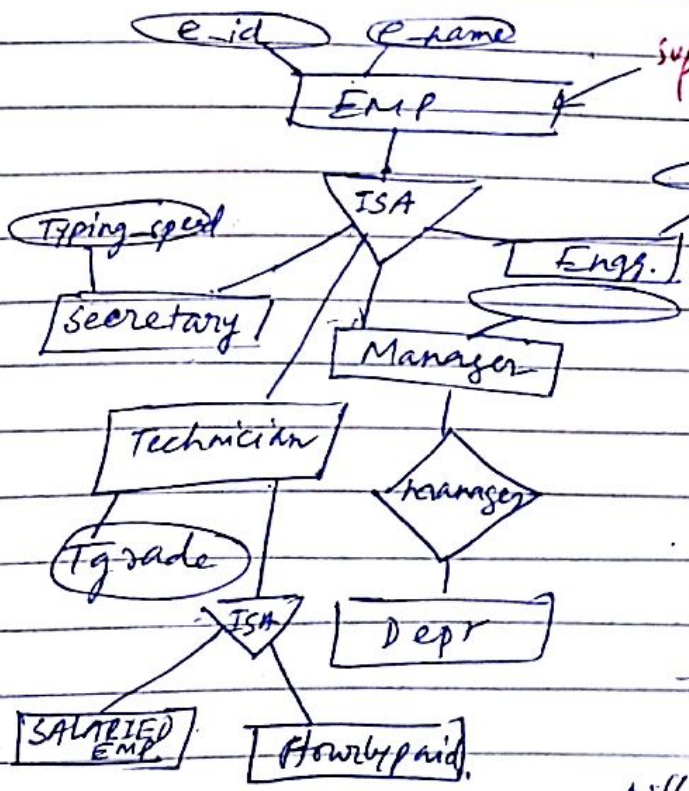
#1 trainee can enroll in 5 courses per training session.



ENHANCED ER FEATURES!

Specialization :- process of defining a set of subclasses of an entity type. set of subclasses defined on basis of some distinguishing characteristics → can't be shared by other entity subsets.

Specialization is depicted by triangle labelled ISA → Manager 'is a' employee

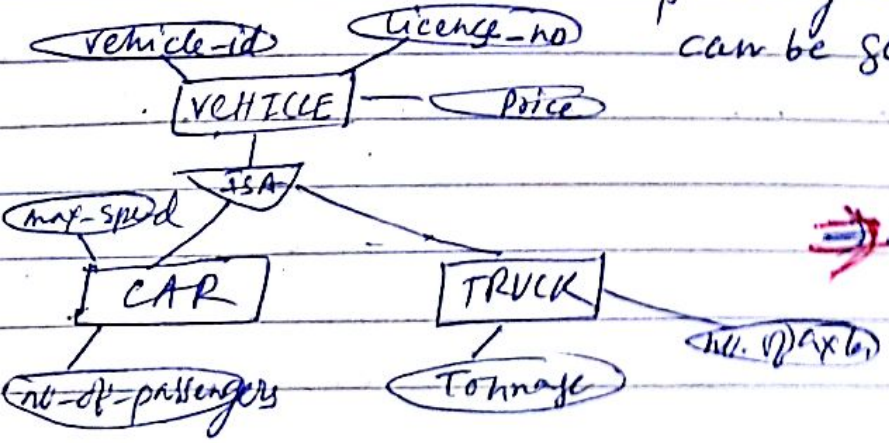


Here specialization is done based on Job_type

Generalization :-

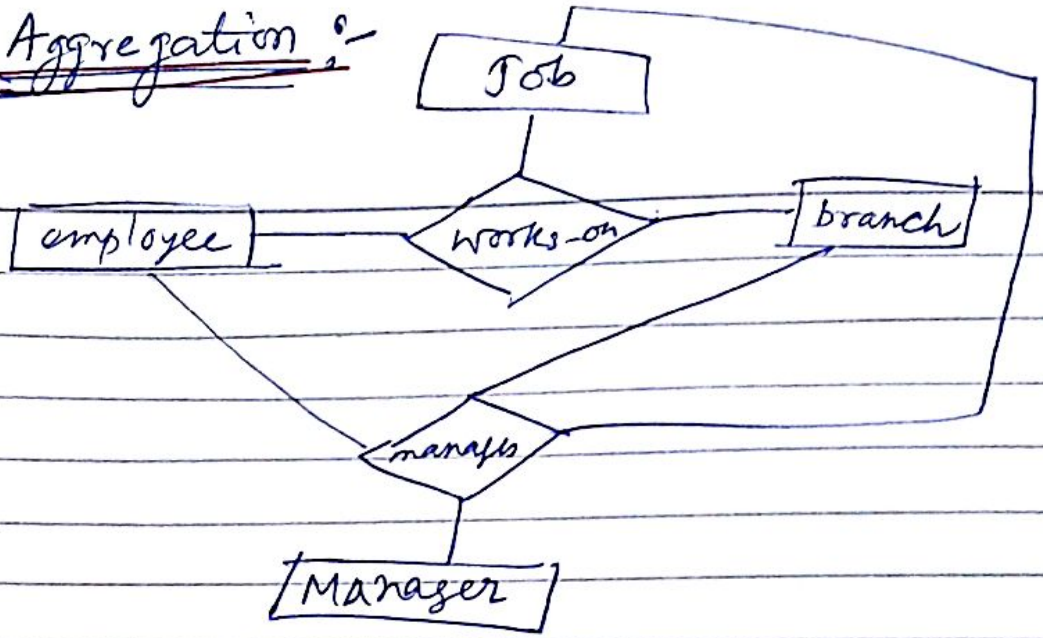
reverse process of abstraction where we identify common features & generalize the subclasses into a superclass.

Diff. entity sets - secretary, manager, engg can be generalized to employee.

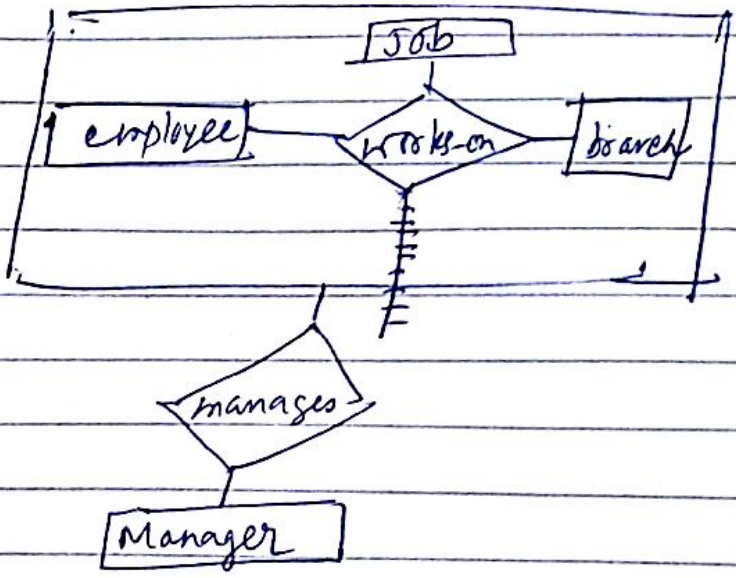


⇒ show 2 entities with seperate attributes & then merge.

Aggregation :-



- manager manages employee, job, branch → bc employee works a particular job at a branch



Normalization

Normalization process - proposed by Codd.
Normalization is reqd. for good db design. It's a process of analyzing given relational schemas based on their FDs & primary keys to achieve desirable properties of 1) minimizing redundancy 2) minimizing insertion, deletion, update anomalies.
After analyzing, unsatisfactory relation schemas that don't meet certain conditions are decomposed into smaller relation schemas to meet the conditions & desirable properties.

keys :-

A superkey of relation schema $R = \{A_1, A_2, A_3, \ldots A_n\}$ is a set of attributes $S \subseteq R$ with the property that for any 2 distinct tuples t_1 & t_2 in any legal relation state r of R, (we have the constraint)

$t_1[S] \neq t_2[S]$.

No 2 distinct tuples should have same set of values for (these subset of attributes) SK. → SK specifies a uniqueness constraint.
/this attribute set

- superkey can have redundant attributes.

{Eno, Ename, Bdate}, {Eno, Ename}, {Eno} ⇐ superkeys.

- degree of normalization — highest normal form cond^n. schema meets.
- Every relation has at least one default superkey — set of all its attributes

- A candidate key / key K is a superkey with property that removal of any attribute from K will cause ~~K~~ not to be a superkey any more. (if we remove any attri. from K then K won't be a superkey anymore)
- Candidate key is a minimal superkey.
- There may be many Candidate key → we choose one of them as primary key, others are secondary key.
- 2 individual tuples shouldn't have same value on key attributes at same time.

prime attribute :- if an attribute is a member of some candidate key ....

if the attribute is of a prime attribute.

if

non-prime attribute -

License_NO	Engine_serial_no	Maker	Model	Yr

car Relation

⇒ License_NO & Engine_serial_no both are candidate keys → one can be chosen as PK

- 1st Normal Form :-

A relational schema is in 1st normal form if the domains of all attributes are ~~are~~ atomic (atomic → simple, indivisible) / include atomic values)

1st normal form disallows multivalued & composite attributes.

DEPT

D-NO	D-Name	D-PhNo.	Dlocations
1			
2			
3			

not in 1st NF

⇒ Decomposing above table in 2 1st NF relation

D-NO	Dlocation

primary key → {D-NO, Dlocation}

D-NO	D-Name	D-PhNo.

P-key

Sol^n 1) place d-location in seperate table along with p-key D-NO. as d-location violates 1NF

2) make seperate tuple for each location in the original table ⇒ introduces redundancy

3) Replace D-location attribute (multivalued) with 3 atomic attributes — D-location1, D-location2, D-loc 3. ⇒ introduces NULL for most depts. if they have fewer than 3 location.

* complex objects & XML (semi structured) used for storing multivalued / composite attributes ⇒ RDBMS doesn't allow.

2nd Normal Form

It's based on full functional dependency.

Functional dependency:- denoted by $X \rightarrow Y$.

FD between two sets of attributes X & Y, that are subsets of $R = \{A_1, A_2, \ldots, A_n\}$, specifies a constraint (on the possible tuples that can form a relation state r of R. Constraint is,) for any two tuples t_1 & t_2 in r that have $t_1[X] = t_2[X]$, they must also have $t_1[Y] = t_2[Y]$.

This means values of Y component of a tuple in r. depend on or are determined by, the values of X component. Alternatively values of X component of a tuple uniquely/functionally determines the values of Y component. ⇒ functional dependency from X to Y or Y functionally dependent on X.
(we can say that for this value of X component, value of Y comp. must be this)

- If X is a candidate key, then $X \rightarrow Y$ for any subset of attributes Y in R. ⇒ Candidate key functionally determines all non-candidate attributes.
- $X \rightarrow Y$ doesn't imply $Y \rightarrow X$.

~~E NO~~ E_no → E_name, Pno → {Pname, Plocation}
{Eno, Pno} → hrs.

⇒ value of an employee's id uniquely determine the emp's name / so on.
⇒ for a given value of Eno, we can know value of E_name, so on.
⇒ FD specifies constraints on attributes of relational schema R that holds all time

- A FD $X \rightarrow Y$ Full FD if removal of any attribute A from X means that dependency doesn't hold anymore; i.e., for any attribute $A \in X$, $(X - \{A\})$ doesn't functionally determine Y. After removal of A if dependency still holds ⇒ partial dependency

For eg., $\{Eno, Pno\} \rightarrow hrs$ ⇒ full dependency
neither $Eno \rightarrow hrs$. nor $Pno \rightarrow hrs$ holds
but $\{Eno, Pno\} \rightarrow Ename$ partial as $Eno \rightarrow Ename$

⇒ A relation schema R is in 2NF if every nonprime attribute A in R is fully functionally dependent on PK of R & R must be in 1 NF.

EMP-PROJ

PK ⇒ {eno, Pno}
⇐ not in 2NF.

Here all nonprime attributes not fully functionally dependent on PK. $\{Eno, Pno\} \rightarrow Ename$ partial as $Eno \rightarrow Ename$ holds
Pname & Ploc are also partially dependent on PK.

⇒ So, given relation can be 2nd normalized by decomposing it a no. of 2NF relations in which non prime attributes are associated only with that part of PK on which they

are fully dependent.
So, FD1, FD2, FD3 lead to decomposition of EMP-PROJ in 3 schemas, each of which in 2NF.

Eno	Pno	Hrs

Eno	Ename

Pno	Pname	Plocation

⇒ Test for 2NF involves testing whether Left hand side of FD is part of PK. If PK contains a single attribute, test is not applied at all.

- For another example, see next page.

Redundant Info. in tuples & Update Anomalies

If to minimize storage space we join two base relation EMP & DEPT instead of keeping them seperately, then attribute values for particular departments are repeated for every employee who works for that dept. If we keep seperately 2 relations, then dept's info appears once in dept relation, foreign key maintained in emp etc.

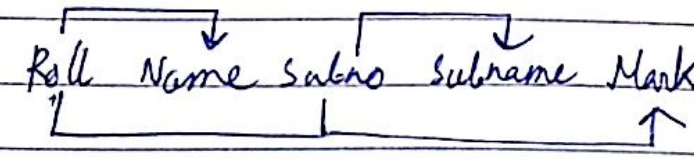
Another serious problem is Update Anomalies (can be classified into insertion, deletion & modification anomalies).

Insertion Anomalies:

- We must enter attribute values for a particular dept (eg. 5) correctly so that they are consistent with values for dept 5 in other tuples in EMP-DEPT.
- It's difficult to insert a new dept that has no employee as yet in EMP-DEPT. Then place NULL for attributes of employee. but emp-no is pk → pk values are null → not good.

Deletion Anomalies :- If we delete from EMP_DEPT an employee tuple that is the last one working for that dept, info regarding that dept will be lost from db.

Modification Anomalies :- If we change the value of an attribute of a particular dept, then we must update the tuples of all employees who work for that dept. otherwise db will be inconsistent — two diff. employee shows two diff. value for same attribute of a particular dept.



{Roll, Subno} → Marks
PK

Roll → Name
Subno → Subname

decomposition accord. to fds.

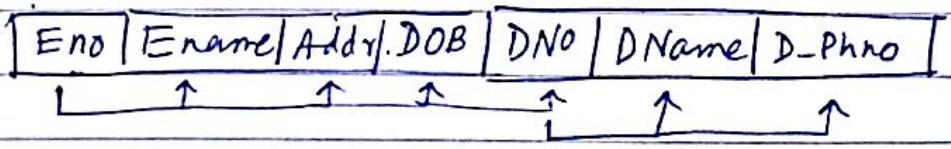
superkey

- Candidate key alway functionally determines any subset of attributes

3rd Normal Form : A relation schema should strive to achieve 3NF atleast

• Trivial dependency :- A FD $X \to Y$ is trivial if $X \supseteq Y$.
$A \to A$ / $AB \to A$ are trivial. They are satisfied by all relations.

• Transitive dependency : if $X \to Y$ & $Y \to Z$ then $X \to Z$
Y is not a candidate key or subset of any key.



EMP-DEPT

Eno → DNO
Dno → Dname
Eno → Dname
⇓ transitive

EMP_DEPT is in 2NF so no partial dependency exist.

3NF : A relation schema is in 3NF if it satisfies 2NF and no nonprime attribute of R transitively dependent on PK.

EMP-DEPT is not in 3NF as transitive dependency of dname (also d-phno) on E-no via D-no. dname, d-phno are nonprime & they are transitively dependent on PK Eno. We can normalize EMP-DEPT by decomposing it as follows - 2 (3NF) Schema: ED1 & ED2. Intuitively ED1, ED2 are two independent entities. A natural join operation recovers EMP-DEPT without generating spurious tuples.

• check that left hand side is part of PK / non key attribute ⇒ problematic.

Alternative def^n for 3NF: R is in 3NF if, whenever a nontrivial fd $X \to A$ holds in R, either (a) X is a superkey of R, or (b) A is a prime attribute of R.

Now R violates this def^n of 3NF if a fd holds in R that violates both cond^n's (a) and (b) of 3NF. Violating (b) means A is a non-prime attribute. violating (a) means that X is not a superset of any key of R ⇒ i.e., X could be nonprime or it could be a proper subset of a Key of R. Now if X is nonprime (also A is part nonprime), there is a transitive dependency that violates 3NF whereas if X is a part of key ⇒ partial dependency that violates 3NF (also 2NF)

In the previous example, Dno → Dname does not satisfy the cond^n (a) & (b).

Thus two def^n of 3NF matches

BCNF (Boyce-Codd Normal Form):-

This NF eliminates all redundancy that can be discovered based on fds. BCNF is stricter than 3NF ⇒ every relation in BCNF is also in 3NF, but a relation in 3NF not necessarily in BCNF. Generally, BCNF applied when there are large volume of data.

Def^n:- R is in BCNF if whenever a nontrivial fd $X \to A$ holds, then X is a superkey of R.

• Diff. betw BCNF & 3NF :- cond^n (b) of 3NF which allows A to be prime, is absent from BCNF. If $X \to A$ holds in R with not being a superkey & A being a prime attribute, then R in 3NF not in BCNF.

Advantage of 3NF that it can obtained without sacrificing losslessness or dependency preservation. Nevertheless, there are disadvt to 3NF! NULL values may be used to represent some of possible meaningful relationships among data items and there is prob. of repetition of info. (refer to Korth for eg. Navathe)

TEACH:

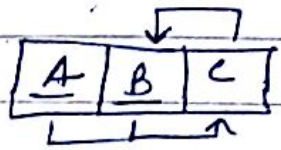
Student	Course	Instructor
X	DBMS	Mark
Y	DBMS	Navathe
Y	OS	Ammar
Y	N/W	Schulman
Z	DBMS	Mark
Z	OS	Ahamed
W	DBMS	Dale
M	DBMS	Navathe
X	OS	Ammar

PK/SK

FD1: {student, course} → Instructor

FD2: Instructor → course

prime attribute



So, TEACH is 3NF not in BCNF.

* all decompositions lose fd1

1) {stu, ins} & {stu, cour}
2) {cour, ins} & {cour, stu}
3) {ins, course} & {ins, stu}

Decomposition to get BCNF → (Instructor, course) & (Instructor, Student)

Though it loses FD1, among others (refer Navathe) it desirable as it won't generate spurious tuples.

* All-key relations are by default in BCNF as they have only trivial dependencies. ⇒ one cond^n for being BCNF

CLOSURE of set of FUNCTIONAL DEPENDENCY

Given a set of fds F, we can prove that certain other fds hold. — such fds are 'logically implied' by F. Every relation instance satisfying F satisfies f.

Suppose: $R = (A, B, C, G, H, I)$ and the set of fds

$A \to B$,
$A \to C$, $CG \to H$, $CG \to I$, $B \to H$.

fd $A \to H$ logically implied.

Suppose t_1, t_2 tuples, $t_1[A] = t_2[A]$ holds, then $t_1[B] = t_2[B]$ must hold as $A \to B$.
As $B \to H$, so $t_1[H] = t_2[H] \Rightarrow A \to H$.

Closure of F, denoted by F+, is the set of all fds logically implied by F.

There are 6 rules of inference:-

- Reflexivity rule: if $\beta \subseteq \alpha$, then $\alpha \to \beta$ // Trivial
- Augmentation rule: if $\alpha \to \beta$ holds and γ is set of attributes then $\gamma\alpha \to \gamma\beta$
- Transitivity rule: if $\alpha \to \beta$, $\beta \to \gamma$ then $\alpha \to \gamma$

These first 3 rules are called Armstrong's axioms ⇒ be proved.
Applying these rules repeatedly, we can get F+

- use $\alpha\beta$ to denote $\alpha \cup \beta$

Armstrong's axioms are sound, as they don't generate any incorrect fds. They are complete, as for a set of fds F, they allow us to generate all logically implied fds.

Although these axioms are complete, it is tiresome to use them directly to compute F+. There are 3 additional rules.

- union rule :- if $\alpha \to \beta$, $\alpha \to \gamma$, then $\alpha \to \beta\gamma$
- decomposition rule :- if $\alpha \to \beta\gamma$, then $\alpha \to \beta$ and $\alpha \to \gamma$.
- Pseudotransitivity rule :- if $\alpha \to \beta$, $\gamma\beta \to \delta$, then $\alpha\gamma \to \delta$

→ We take this for example.

$A \to H$. as $A \to B$ & $B \to H$ (transitive rule)

$CG \to HI$, as $CG \to H$ & $CG \to I$ (union rule)

$AG \to I$, as $A \to C$ & $CG \to I$ (pseudotransitivity rule)

or as $A \to C$, so $AG \to CG$, & $CG \to I$.

$AG \to I$ (transitivity).

CLOSURE of ATTRIBUTE SET

or whether $\alpha \to \beta$ holds if β is in α^+

To test whether a set α is a superkey, we must devise an algo. to compute the set of attributes functionally determined by α. One way → from F^+, take fds whose left hand side is α & take union of right hand sides of all such dependencies, but doing so is expensive as F^+ can be large.

set of all attributes functionally determined by α under set FD fds → closure of α under F (denoted by α^+).

```
result := α;
while (changes to result) do
    for each fd β → γ in F do
        [begin]
            if β ⊆ result then result := result ∪ γ
        end
```

To compute (AG^+) with the previous (given in last page) dependencies using this algo/pseudocode.

We start with result = AG. First time when we execute while loop to test each dependency, we find that-

- $A \to B$ causes to include B in result / $A \subseteq$ result (AG) result := result ∪ B
- $A \to C$, result = ABCG, as $A \subseteq$ result
- $CG \to H$, result = ABCGH, as $CG \subseteq$ result
- $CG \to I$, result = ABCGHI

~~second time~~ For $B \to H$, no change in result, so algo terminates

$R = \{A, B, C, D\}$, $FD = \{AB \to C, C \to D, D \to A\}$. List candidate keys.

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

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

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

So, candidate keys are : AB, BC, BD

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

Now, check only $ACD^+ = \{A, C, D\}$ No Superkey

as, ACB, ADB $\supseteq$ AB, ADB $\supseteq$ BC, Now AB, BC are candidate keys, so need not to check those.

$R = \{A, B, C, D\}$, $FD = \{AB \to C, BC \to D, CD \to A, AD \to B\}$

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

$AB^+ = \{AB\} \Rightarrow AB^+ = \{ABC\}$, $AB^+ = \{ABCD\}$ ✓

$BC^+ = \{BCDA\}$ BC, $CD^+ = \{CDAB\}$ C, $DA^+ = \{ADBC\}$ C

$ABC^+ = \{ABCD\}$ S, $BCD^+ = \{BCDA\}$ S, $CDA^+ = \{CDAB\}$ S

$BDA^+ = \{BDAC\}$ S

$BD^+ = \{BD\}$

$AC^+ = \{AC\}$

(Navathe)

Multivalued dependency & 4th NF.

Multivalued depen. (MVD) are consequence of 1NF, which disallows an attribute in a tuple to have a set of values. If we have 2 or more multivalued independent attributes in the same relation schema, we get into a prob. of having to repeat every value of one of the attributes with every value of the other attributes to keep the relation state consistent and to maintain the independence among the attributes involved. — This constraint is specified by MVD.

EMP

EName	PName	DName
Smith	X	John
Smith	Y	Anna
Smith	X	Anna
Smith	Y	John

PName & DName independent of each other. To keep consistency, separate tuple to represent every combination of an employee's dependent and an employee's proj. ⇒ MVD.

{EName, PName, Dname} ⇒ PK

Defn:- A mvd $X \twoheadrightarrow Y$ specified on relation schema R, where X & Y are subsets of R, specifies the following constraint on any relation state r of R: If 2 tuples t_1 and t_2 exist in r such that $t_1[X] = t_2[X]$, then 2 tuples t_3 & t_4 should also exist in r with the following property, where Z is $(R - (X \cup Y))$ ⇒

- $t_3[X] = t_4[X] = t_1[X] = t_2[X]$
- $t_3[Y] = t_1[Y]$ & $t_4[Y] = t_2[Y]$
- $t_3[Z] = t_2[Z]$ & $t_4[Z] = t_1[Z]$.

$X \twoheadrightarrow Y$ means X multidetermines Y ∥

- Values of Y determined only by ~~values of~~ X alone, doesn't depend on remaining attributes Z of R.

If for same value of X, 2 tuples have distinct Y values, then these values of Y must be repeated in seperate tuples with every distinct Z values that occurs with that same X values.

In emp relation, the values 'x' & 'y' are repeated for each of dname (and similarly values 'Anna', 'John' repeated for each pname) This is redundancy undesirable.

EMP is in BCNF (All-key relations are in BCNF)
4NF is stronger than BCNF.

ENam, Pname, Dname → Ename
all are trivial → Pname
→ Dname
so in BCNF

A MVD is trivial if a) Y is a subset of X, or b) XUY = R.
$X \twoheadrightarrow Y$. MVD that doesn't satisfy any cond^{n} ⇒ Non trivial

(*) mvd may be trivial also for 4NF

4NF :- R_1 is in 4NF if for every non trivial multivalued dependency $X \twoheadrightarrow Y$, X is a superkey for R.

EMP is in BCNF but not in 4NF. as Ename →→ Pname
Ename →→ DName are non trivial mvd. and Ename not superkey, but

So, decompose EMP in EMP-PROJ. & EMP-DEP; those are in 4NF as mvds are trivial.

EName	PName
Smith	X
Smith	Y

EName	DName
Smith	John
Smith	Anna

Relational Algebra

The basic set of operations for a relational model that take one or two relations as i/p and produce a new relation as their o/p.

Relational calculus provides a higher level declarative notation for specifying relational queries.

Unary Relational Operations:-

SELECT: this operation is used to select a subset of tuples from a relation that satisfies a selection cond^n. ⇒ SELECT is called as a horizontal partition. (Discard other tuples)

$\sigma_{\langle selection\ cond^n \rangle}(R)$ // lower case sigma

$\sigma_{DNO=4}(EMP)$

$\sigma_{SAL>3000}(EMP)$

$\sigma_{(DNO=4\ AND\ SAL>3000)\ OR\ (DNO=5\ AND\ SAL>4000)}(EMP)$

SELECT operation is commutative.

$\sigma_{cond1}(\sigma_{cond2}(R)) = \sigma_{cond2}(\sigma_{cond1}(R))$

$\sigma_{cond1}(\sigma_{cond2}(\cdots(\sigma_{condn}(R))\cdots) = \sigma_{cond1\ AND\ cond2\ AND\ cond3}(R)$

PROJECT: this operation selects certain columns from tables & discards other columns.

- We use project operation
  ⇒ vertical partition

$\Pi_{\langle attribute\ list\rangle}(R)$ // $\Pi_{NAME, SAL}(EMP)$

(upper case Pi)

Combination of these two operations.

(PPO)

Composition of Relational operations.

$\Pi_{name,sal}(\sigma_{DNO=4}(EMP))$

// $\sigma_{DNO=4}(EMP) \rightarrow X$

$\Pi_{name,sal}(X)$

Rename: $\rho_E(EMPLOYEE)$

UNION, INTERSECTION AND SET DIFFENCE

Several set operations used to merge elements of 2 sets in various ways, UNION.

- binary operations — applied on 2 sets.

2 relations are

$R(A_1, A_2, \ldots, A_n)$ and $S(B_1, B_2, \ldots)$ are union compatible if they have same degree n and if $dom(A_i) = dom(B_i)$ for $1 \le i \le n$

$X \leftarrow \sigma_{DNO=5}(EMP)$

Result1 $\leftarrow \pi_{ENO}(X)$

Result2

Union: $\pi_{st\_name}(CSE1) \cup \pi_{st\_name}(CSE2)$

Set-diff: $\pi_{cust\_name}(depositor) - \pi_{cust\_name}(borrower)$

CARTESIAN PRODUCT (CROSS PRODUCT) :-

ln-no	branch_name	amount
L-170	Downtown	3000
L-230	Redwood	4000
L-260	Perry	5000

LOAN

Cust-name	Lon-no
Jones	L-170
Smith	L-230

BORROWER.

Inner join: join 2 relations matching values of common attribute

Loan_no.	branch_name	amount	cust_name	ln_no
L-170	- - -	- - -	-	L-170
L-230				2-230

Outer join: deals with missing info.

Left outer join (⟕) takes all tuples in left relation that doesn't match with tuples in Right relation & fills/pads the attribute values for right relation with null, then add the tuples with result of inner join.

L170	✓	✓	Jones	L-170
L-230	✓	-	Smith	L-230
L-160	Penridge	-	null	NULL

Right outer join (⟖) opposite ⇒ it takes tuples from right relation that doesn't match with tuples of left one and pads with null values.

Ln No	branch_name	amount	Cust_name	Ln-no
L-170	⌣	⌣	Jones	L-170
L-230	⌣	⌣	Smith	L-230
NULL	null	NULL	Hayes	L-155

In case of natural join we result to give one cond^{n}. & 2 common columns are merged in result.

Full outer join:

Theta join ⇒ θ is the condition.

Equi join ⇒ When in θ-join only equality cond^{n}, then equi join

Aggregate funct.s & grouping.

Aggregate funct.s take a collection of values and return a single value as a result. Suppose, total no. of employees

count (emp_no). ⇒ $\mathcal{G}_{count(emp\_no)}(EMP)$

count(distinct emp_name) ⇒ $\mathcal{G}_{count\_distinct(emp\_name)}(EMP)$

Now, we want to count no. of emp. in each dept seq, we need to partition relation EMP in groups based on dept_no. & apply aggregate funct.n on each group.

select count(emp_name) from emp where <cond.>.

select dno, count(emp_no) from emp group by dno.

select dno, count(emp_no) from emp group by dno having count(emp_no) > 3

dno $\mathcal{G}_{count(emp\_no)}(EMP)$

select branch_name, avg(bal) from account gr by branch_name having avg(bal) > 1200.

⇒ to apply cond.n on groups, we use having clause. having used with group by.

⇒ select dname count(emp_name) from emp join dept on <cond.>. group by d_name.

⇒ select dname, count(emp_name) from emp, dept where <cond.> group by d_name

- depositor → cust_name, acc_no
account → acc_no, branch name, bal.
customer → cust_name, street, city.
branch → br_name, br_city, assct

~~select branch_name, count(distinct customer_name) from depositor~~
join account on depositor.acc_no = account.acc_no.
group by branch_name.

- select cust_name, avg(bal) from depositor, account.
depositor.
depositor on depositor.acc_no = account.acc_no join
customer on depositor.cust_name = customer.cust_name,
where cust_city = 'Bombay'
group by depositor.cust_name having
count(depositor.acc_no) >= 3.

→ List avg. bal for each customer who lives in Bombay and has atleast three accounts.

Nested subquery:

Select cust_name from Borrower where customer_name not in (select cust_name from depositor)

- Find customers having both an account and a loan at Perryridge branch.

Select cust_name from borrower, loan where borrower.loan_no = loan.loan_no and branch_name = 'Perryridge' and (branch_name, cust_name) in (select branch_name, cust_name from depositor, account where depositor.account_no = account.account_no)

- Find branches for which avg. bal is greater than or equal to all avg. bal. / Find the branch which has highest avg bal.

(Can't write max(avg(...)) so)

Select branch_name from account group by branch_name having avg(bal) >= all (select avg(bal) from account gr. by branch_name)

- greater than at least ⇒ > some (. . .)

. = some is equivalent to in

• <> all is equivalent to not in

Kartik 3rd chapter

Consider employee db

emp (emp-name, street, city)
works (emp-name, comp-name, sal)
company (comp-name, city)
manages (emp-name, manager-name)

b. Find names, street, cities of all emps. who work for First Bank Corporation & earn more than $10,000

c. Find emps. in db who don't work for First Bank Corp.

d. Find emps. in db who earn more than each emp of small bank corp.

e) ~~Assume~~ Find comps. located in every city in which small bank corp. located.

f. Find company that has most employees.

g. ~~Find.~~

j. Find those comps. whose emps earn a higher sal, on avg, than the avg. sal at 1st bank corp.

person (driver_id, name, addr.)
car (license, model, yr)
accident (report_no, date, loc)
owns (driver_id, licence)
participated (driver_id, car, report_no, damage_amount)

i. Find the total no. of people who owned cars that were involved in accidents in 1989

Transaction

- Transaction refers to a collection of operations that form a single logical unit of work. eg. transfer of money from one account to another is a transaction consisting of 2 updates.
  → appears to db users as a single unit of work.

T1: read(A); //read(X) → transfers data item X from disk to local buffer
A := A-50;
write(A); //write(X) → transfers data X from local buffer to db.
read(B); always not immediately update data in disk.
B := B+50;
write(B).

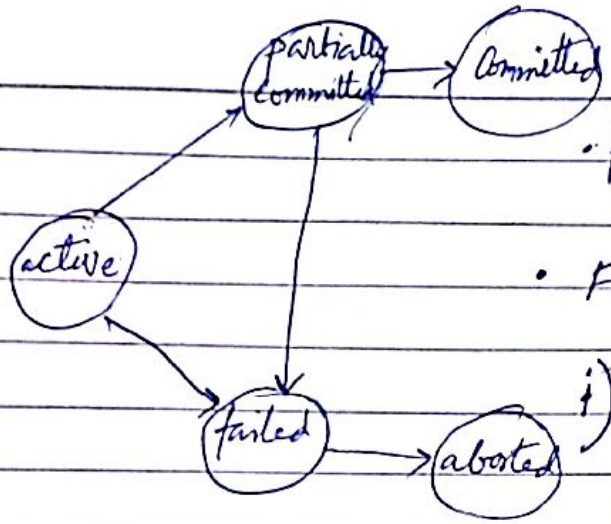
Atomicity: either all actions of transaction executed completely or none of it does; in some failure, partial execution of transaction must be undone.

Once transaction successfully executed, user notifies that transfer of fund taken place, data must persist in db, no failure will result in loss of data corresponding to this transfer of funds (Hard disk not failed) ⇒ Durability

consistency : sum of A & B is unchanged before & after execution of transaction. → consistent before & after transaction ..

result in inconsistent state.

Transaction state:



- **Active**: initial state; transaction in this state while executing
- **Partially committed** - after final stmt. has been executed
- **Failed** - normal execution no longer proceed.
  - i) aborted transaction
  - ii) kill trans..
- **aborted** - trans. rolled back & db has been restored to its prior state to restart of trans.
- **committed** - successful completion.
- terminated → committed / aborted.

- **schedule** - execution seq. of instructions. It represent chronological order in which instructions are executed in system. schedule for a set of consist of sequence
- **serial / non serial schedule** — of instructions from various transactions where instructions ~~from~~ belonging to one transaction appear together in that schedule. single

- **Serializable** = A schedule of n transactions is serializable if it is equivalent to some serial schedule of same n transaction.
under concurrent execution, make sure that, any schedule that is executed has the same effect as a schedule of equivalent effect of a serial one. serial.

• conflicting operations :- 2 operations in schedule conflict if they satisfy all 3 cond^{n} :- 1) belong to diff. transactions 2) access same data X. and 3) at least one of the operations is write(X).

conflict equivalent :- // 2 schedules are conflict equivalent if the order of any 2 conflicting operations is same in both schedules // If S can be transformed to S' by a series of swaps of nonconflicting instructions, then S & S' conflict equivalent.

conflict serializable :

S is conflict serializable if S is equivalent to some serial schedule S'_{ins} (In this case, we reorder nonconflicting operations, until we form equivalent serial schedule S'_{ins}).

In korth eg:

T_1	T_2
r(A)	
A := A − 50	
W(A)	
	r(A)
	temp2 := A * 0.1
	A := A − temp
	write(A)
r(B)	
B := B + 50	
W(B)	
	r(B)
	B := B + temp
	W(B)

transactions consists of all inst. of those transactions & preserve the order in which they appear in the trans.

• conflicting operations :- 2 operations in schedule conflict if they satisfy all 3 cond^{n} :- 1) belong to diff. transactions 2) access same data X. and 3) at least one of the operations is write(X).

conflict equivalent :- // 2 schedules are conflict equivalent if the order of any 2 conflicting operations is same in both schedules // If S can be transformed to S' by a series of swaps of nonconflicting instructions, then S & S' conflict equivalent.

conflict serializable :

S is conflict serializable if S is equivalent to some serial schedule S'_{ins} (In this case, we reorder nonconflicting operations, until we form equivalent serial schedule S'_{ins}).

In both eg.

T_1	T_2
r(A)	
A := A - 50	
W(A)	
r(B)	
B := B + 50	
W(B)	
	r(A)
	temp := A * 0.1
	A := A - temp
	write(A)
	r(B)
	B := B + temp
	W(B)

transactions consists of all inst. of those transactions & preserve the order in which they appear in the trans.

Conflict serializability:

I_i, I_j consecutive instructions of T_i, T_j ($i \neq j$). If I_i, I_j refer to diff. data item, then I_i, I_j can be swapped without affecting results of any instr. in schedule. Now, if I_i, I_j refer to same data item, then order of 2 instructions matter in following way. →

1) I_i = read(Q), I_j = read(Q). order of I_i, I_j doesn't matter as same value of Q is read by T_i & T_j.

2) I_i = read(Q), I_j = write(Q). If I_i comes before I_j, then T_i doesn't read the value written by T_j in T_j. If T_j comes first, then T_i reads the Q value written by I_j.

3) I_i = write(Q), I_j = read(Q). Similar as previous.

4) I_i = write(Q), I_j = write(Q). Here next read(Q) instr. is affected as result of only the latter of 2 write instr. is preserved in db. // If no other write operation after I_i & I_j in any schedule, then the order of I_i, I_j directly affects the final value that results from S.

eg: In S-3, write(A) of T_1 conflicts with read(A) of T_2. but write(A) of T_2 doesn't conflict with read(B) of T_1.

Now, if I_i, I_j don't conflict, then we can swap the order of I_i, I_j to produce a new schedule S'. Now, S, S' equivalent as all instr. appear in same order in both schedule except I_i, I_j whose order doesn't matter.

S-1 is conflict ~~serializable~~ equivalent with S3 as read(B), write(B) of T_1 can be swapped with read(A), write(A) of T_2.

Now, since write(B) of T_2 in S3 doesn't conflict with read(B) of T_1, they can be swapped to form next equivalent schedule S5.
Now, we continue to swap nonconflicting instr.

- read(B) of T_1 with read(A) of T_2
- write(B) of T_1 with write(A) of T_2
- write(B) of T_1 with read(A) of T_2.

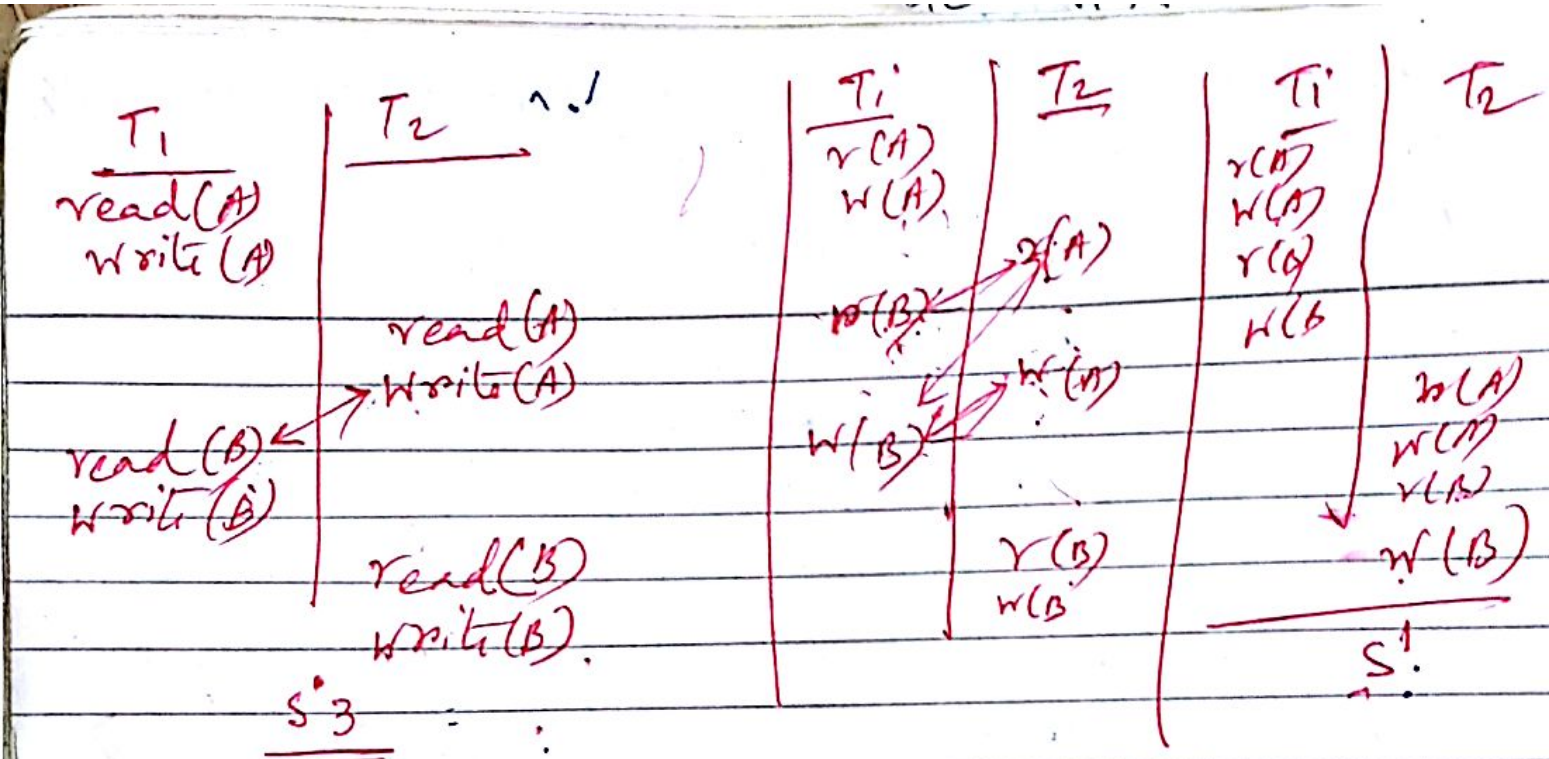
result → serial schedule S6 (S1). So, S3 is conflict serializable. // schedule 7 is not conflict serializable as it is not equivalent to serial schedule $\langle T_3, T_4 \rangle$ or $\langle T_4, T_3 \rangle$.

View serializability:- - less stringent than conflict equi.
same set of transactions participates in S & S'. S, S' are view equivalent if 3 cond^n^s met.

1. for each data item Q, if trans T_i reads initial value in S, it should read initial value in S' in both schedules.
2. In S, write(Q) by T_j, that value read by T_i, in S' also this must happen in similar way (order).
3. In both schedule final write must be done by same trans.

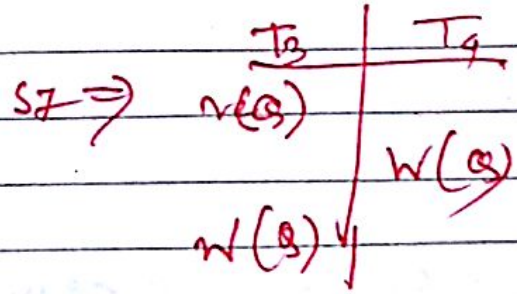
eg:- S1, S2 not view equi. but S1, S3 are view equ.
(reason is given in book)

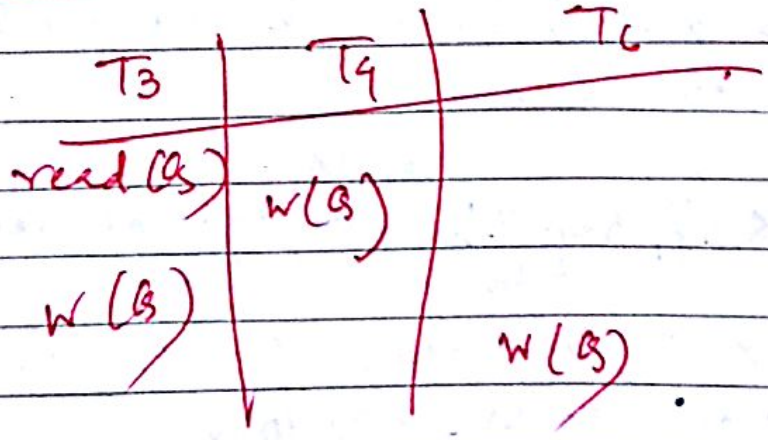
* every conflict serializable is view serializable, but reverse is not true.



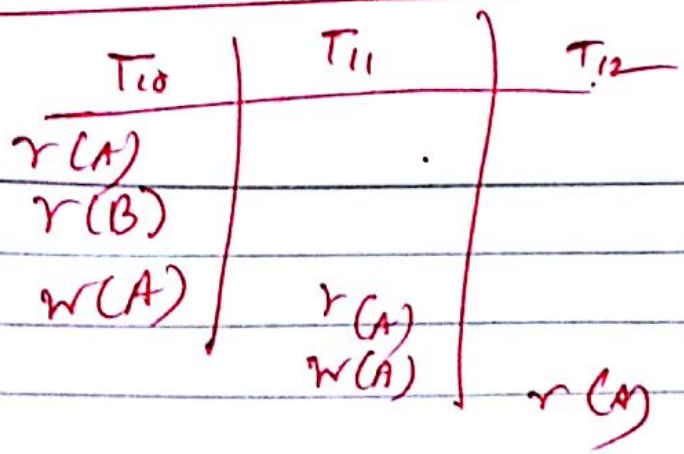
S'3

S1 ⇒





cascading rollback



Lock based Protocols.

Shared mode lock

exclusive mode - lock.

compatibility matrix.

	S	X
S	T	F
X	F	F

T1: lock-X(B)
read(B)
B = B-50
W(B)
unlock(B)
lock-X(A);
r(A)
A = A+60
W(A)
unlock(A)

T2: lock-S(A)
read(A)
unlock(A)
lock-S(B)
r(B)
unlock(B)
display(A+B).

T_1	T_2	~~T_3~~
lock-X(B)		grant
r(B)		
B = B-50;		
W(B)		
unlock(B)		
	lock-S(A)	
	r(A)	
	unlock(A)	
	lock-S(B)	
	r(B)	
	unlock(B)	
	disp(A+B)	
lock-X(A)		
r(A)		
A = A+50		
W(A)		
unlock(A)		

early unlocking will lead to undesirable condⁿ.

T_3: T_1 → with all unlocks later.

T_4: T_2 → with all unlocks later.

locking can lead to deadlock also.

T_3	T_4
lock-X(B)	
r(B)	
B = B-50	
W(B)	
	lock-S(A)
	r(A)
	lock-S(B)
lock-X(A)	

Schedule ↑

T_3:
lock-X(B)
r(B)
B = B - 50
W(B)
lock-X(A)
r(A)
A = A + 50
W(A)
unlock(B);
unlock(A);

T_4:
lock-S(A)
r(A)
lock-S(B)
r(B)
display(A+B)
unlock(A);
unlock(B);

2-Phase locking protocol

To ensure more serializability

Growing phase – In this phase, a transaction may obtain lock, but can't release locks

Shrinking phase –

end of growing phase → lock pt.

In 2 phase locking, there are also deadlock (T_3 & T_4) and cascading rollback (T_5, T_6, T_7)

Strict two phase - lock → avoid cascading rollback

T_5	T_6	T_7
lock-x(A)		
r(A)		
lock-s(B)		
r(B)		
w(A)		
unlock(A)		
unlock(B)		
	lock-x(A)	
	r(A)	
	w(A)	
	unlock(A)	
		lock-s(A)
		r(A)

'rigorous 2-P locking'

hold Exclusive locks until trans. commits. it ensures that any data written by an uncommitted transaction are locked in exclusive mode until trans. commits preventing others from reading data

---

lock conversion ensures more concurrency.

T8	T9
lock-s(a1)	
	lock-s(a1)
lock-s(a2)	
	lock-s(a2)
lock-s(a3)	
lock-s(a4)	
	unlock(a1)
	unlock(a2)
upgrade(a1)	

T8:
r(a1)
r(a2)
r(a3)
w(a1)

T9:
r(a1)
r(a2)
display(a1+2)

---

upgrade → in growing phase.
downgrade - in shrinking phase.

---

1. if read - lock-s(a)
   if write - upgrade.
   all locks must be unlock after commit or abort

Relational Calculus

Tuple Relational:-

- Select * from Emp where Sal > 5000

$\{t \mid EMP(t) \text{ AND } t.sal > 5000\}$

$\{t.Fname, t.Lname \mid EMP(t) \text{ AND } sal > 5000\}$

Domain Relational:- list DOB, add of emp whose Fname is SMITH & Sal > 2000

$\{uv \mid (\exists r)(\exists t)\ Emp(qrstuvwx) \text{ AND } r = \text{'smith' AND } t \geq 1000\}$

Emp = {qrstuvwx}

Fname Sal dno

these are domain variables.

Find name, add of emps. who work for Rsrch dept.

$\{ q.S \mid (\exists x)(\exists l)(\exists m)\ Emp(qrstuvwx)\ AND\ Dept(lmno)\ AND\ l = 'Rsrch'\ AND\ m = x \}$