

19/9/24/Thu.

DBMS

GAYATHRI B NAIR
AM.EN.U4AIE22117

Different categories of DBMS commands?

1. DDL

- Create, Alter, Drop, Truncate
- mainly deals with structure

2. DML - Database Manipulation Language

- Insert, Update, Delete

- mainly for dealing or manipulating data

3. DCL - Database Control Language

- Grant, Revoke

- By administrators to grant or revoke permissions.

4. TCS - Transaction Control Statement

- commit, rollback, savepoint.

- to make all transactions

- commit - to commit what we have done.

- rollback - Undo.

- savepoint

* CREATE - not just used to create a table.

- we can create any object eg: cursor.

Composite primary key:

- Multiple attributes together forms a primary key.

- primary key: unique & not null values.

- only constraint which cannot be specified in column level is composite primary key.

- create table Branch (ba-name varchar,
bank-name varchar,

.....
.....

constraint con1 primarykey (ba-name,
bank-name));

constraint name.

This is how we create a composite primary key constraint. This says that the PK is a combination of ba-name and bank-name.

- It is not required to name a constraint but it is better to do so as it is required if we need to drop a constraint later.

* Foreign key - at table level and at column level constraint
(Refer ppt)

References.

20/9/24/Friday.

- If we try to delete rows from a master table which have references to some other table, it will not work.

We will first have to delete all the dependencies with reference values to master table then only delete.

To avoid this big process we can use:

```
create table Emp(eno ... ,  
ename ... ,  
dno varchar references  
Dept(deptno) on delete cascade);
```

This automatically deletes the dependent rows whenever we delete the rows in master table.

Instead if we put on delete set null,
it will become null.

Alter:

- To alter the structure
- Syntax:
 1. Alter table tab-name add column col-name datatype;
 2. Alter table tab-name drop column col-name;
 3. Alter table tab-name rename ^{column} col-name to new_colname;
 4. Alter table tab-name alter column col-name set not null / drop not null;
 5. Alter table tab-name alter column col-name set default value;
 6. Alter table tab-name add constraint con-name keyword;
 7. Alter table tab-name rename to new-tab-name;

→ used for adding ^{all} constraints except not null and default constraints. The syntax for those are written on 4 and 5.

Drop & Truncate:

- for dropping columns - drop
- drop - drops all values including structure, tuples etc.
- truncate - values deleted but structure is maintained

DML - Insert, Update, Delete

- Insert into tab_name values (val1, val2, ...)
- update tab_name set col_name = value [where condition]
condition can be specified in delete until the
- truncate where we delete all rows.
- delete from tab_name [where condition]
- select [all / distinct] col1, col2, ... from tab_name [where condition] [group by groupby-column] [having condition] [order by col-name]

Q. Consider the following relation?

↳ Accounts (acc_no, cust_no, ac_type,
br_name, bal_amt);

Q1. Retrieve all account balance?

Q2. Find account type for customer C1?

Q3. Find all accounts having balance amount
greater than 50 thousand?

Q4. Find all customer numbers which have
balance in the range 50k to 1 lakh.

Q5. Find all account numbers with ac_type
as savings or checking?

Q6. Find the cust_no who belong to Kollam
branch and having amount < 25k

A:

1. select * from bal_amt from Accounts;
2. select ac_type from Accounts where cust_no = 'C1';
3. select * from Accounts where bal_amt > 50000;
4. select cust_no from Accounts where (bal_amt > 50000
and bal_amt < 100000);
5. select ac_no from Accounts where (ac_type = 'savings'
or ac_type = 'checking');
6. select cust_no from Accounts where (br_name = 'Kollam'
and bal_amt < 25000);

A:

1. select * from bal-amt from Accounts;
2. select ac_type from Accounts where cust_no = 'C1';
3. select * from Accounts where bal_amt > 50000;
4. select cust_no from Accounts where (bal_amt > 50000
and bal_amt < 100000);
5. select ac_no from Accounts where (ac_type = 'saving'
or ac_type = 'checking');
6. select cust_no from Accounts where (br_name = 'Kollam')
and bal_amt ~~<~~ < 25000);

23/9/24/Mon.

'like' operation:

- to match patterns

- _ - single unknown character

- % - unknown number of unknown characters

Q. From Emp(eno, ename, sal, dno), find all employee names starting with 'A'?

A: select ename from Emp where ename like 'A%';

* '-a%' - eg: Parvathy, Salicylic

'%a%' - eg: Pickaxe

Sorting:

- select * from Emp by sal desc;

sorted in descending order of salary.

- select * from Emp by ename, sal desc;

First sorted in ascending order of ename
If ename is same then ordered by descending order of salary.

Group / Aggregate Functions

- built-in functions.

- max(), min(), avg(), sum(), count(), count(*)

All avoid null values

↳ takes null values.

e.g.:

select count(eno) from Emp where dno = 'D1';
(counts the no: of employees in dept D1.)

Sorting:

- Select * from Emp by sal desc;
sorted in descending order of salary.
- Select * from Emp by ename, sal desc;
first sorted in ascending order of ename
if ename is same then ordered by descending
order of salary.

Group / Aggregate Functions

- built-in functions
- max(), min(), avg(), sum(), count(), count(*)
All avoid null values ↗ states null values.

Eg:
select count(eno) from Emp where dno = 'D1';
(counts the no. of employees in dept D1.)

3/10/24 Thursday.

Count

Consider this relation: Emp(eno, ename, sal, dno, mgr_id)

1. find the no. of employees who are assigned to a manager?

ON EEEd

- A: $\text{count}(\text{select count(mgr_id)} \text{ from Emp);}$
OR
 $\text{select count(ename)} \text{ from Emp where mgr_id is not null;}$
2. Find the department number and the no. of employees in each department.
- A: select count
 $\text{select dno, count(eno) from Emp group by dno;}$
the first and foremost, the grouping operation happens only after that does the select operation occur for each group.
3. What if in this we need to check whether there are more than 10 employees in the department?
- A: $\text{select dno, count(eno) from Emp group by dno having count(eno) > 10;}$

We cannot use 'where' clause along with group by. We must use 'having' whenever group by is used.

Q. Consider the relations:

Account (ac-no, cno, ac-type, bal-amt, veri-emp-no)
Customer (cno, cname, ctype)

1. Find the no. of accounts verified by each employee.
2. Find the customer numbers having more than one account in the bank.

A:
1. select count(veri-emp-no) from Account;
2. select ~~cno~~ (cno, ^{count(ac-no)}) from Account group by cno having count(ac-no) > 1;

Set Operations:

1. Union → the queries must be union compatible.
2. Intersect
3. Except.
↳ minus' in Oracle.

Union Compatibility Rules:

1. The no. of attributes or columns in both the queries must be the same.
2. The datatype of corresponding attributes must be the same.

If these rules are followed then union b/w the queries is possible

Q. Consider the relations:

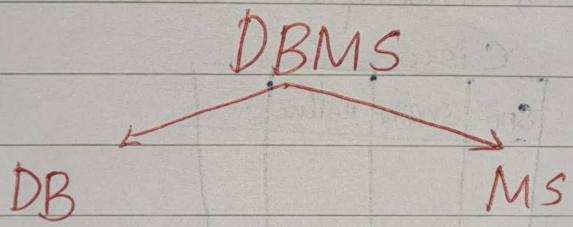
Customer-FD (cno, fd-no, fd-amt)

Customer-Loan (cno, ln-no, int-rate, ln-amt)

1. Find the customer numbers of those customers having a loan or fd at the bank.
 2. Find those customer numbers having both loan and fd.
 3. Find customer numbers having an fd but not a loan at the bank.
- A:
1. select cno from Customer-FD ~~union~~ select cno from Customer-Loan;
2. select cno from Customer-FD ~~intersect~~ select cno from Customer-Loan;
3. select cno from Customer-FD ~~except~~ select cno from Customer-Loan;

Customer - Loan;

4/10/24 | Friday.



- more structured storage.
- set of programs to manage data.
- data can be interrelated.
- provide security
- data sharing.

Goals Of DBMS:

- Storage of data
- Manipulation of data
- Security
- Sharing of data

Drawbacks of storing data within file system in OS:

1. Data redundancy & inconsistency
2. Difficulty in accessing data.
3. Data isolation
4. Integration problems
5. Concurrent access anomaly.
6. Atomicity problem
7. Security

1. Data redundancy & inconsistency.

Say for example in a university management system, a student can join multiple courses.

BTech		Cisco	
Sno	Sname	Sno	Sname
..

here same data will have to be repeated in both \Rightarrow redundancy.

Now if one value is changed or updated in one table but not done in the other, the same person will have two addresses causing data inconsistency.

2. Difficulty in accessing data.

Writing application programs for accessing data which are stored in different formats is difficult.

Say for example there can be files of extensions .csv, .doc etc. Files can be stored in various formats and this could make it difficult to access.

3. Data isolation.

It is difficult to write application programs to retrieve data from different files.

4. Integrity Problems.

Integrity constraints such as primary key, foreign key, salary not less than incentive etc are rules incorporated by organizations.

There are no way of adding these in file based system. Enforcing these integrity constraints in file based system is very difficult as we'll have to update the app: pgm each time.

5. Concurrent Access Anomaly:

UMS:

Cno	...	vacant seat
c ₁		2
c ₂		1

Here, say 2 users access the data at the same time and then update it at the same time. This could cause problems.

This problem which occurs due to concurrent access and updation of data is called

6. Atomicity Problem:

$$A \xrightarrow{1000} B$$

read(A) → making a copy.

$$A := A - 1000$$

write(A) → writing it in disk memory

$$\text{read}(B)$$

$$B := B + 1000$$

$$\text{write}(B)$$

This process should occur in the order. Say an issue happened at the time of write(A).

Then what happens is 1000 Re is gone from A but it is not added to B.

Atomicity: performing all the steps or none even in case of interruptions. This is not ensured in file based system.

7. Security:

Anyone who has access to the file can see everything in the file which should not be happening.

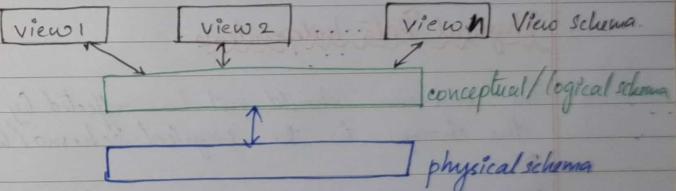
Eg: Salary of one person should not be visible to another person but anyone who has access to the salary file can see everyone's salary.

Data Abstraction:

- Complexities is hidden from the end users.

- There are different levels of data abstraction:

3 Layer Architecture:



Physical Schema:

- What implementations are done.
- What database ^{etc} is used.
- Access methods etc.

Conceptual Schema:

- What data is stored
- Is there any relation bw the data. ~~in the~~

View Schema:

- There can be multiple view schemas.
- This is where the end users interact.
- There are diff: views because for different users, the requirements are diff.
- eg: in UMS, a librarian and a student should not have the same view as their requirements are diff..

Physical Data Independence:

The end users should not be affected by the changes in the physical schema/level
OR.

When there is a change in physical level, the users in other levels ~~need~~ should not be affected.

what is contained in
the database at a particular
time. → overall design

Instance of Schema:

Instance: content of database at a particular time.

Schema: Overall design.

are done.

used.

te.

in bw the data. ~~in the~~

ple view schemas.

users interact -

ws because for

requirements are diff:

b'arian and a student
some view on them

dence:-

not be affected by
physical schema/level.

in physical level, the
should not be affected.

what is contained in
the database at a particular
time. → overall design

Instance of Schema:

Instance: content of database at a particular time.

Schema: Overall design.

7/10/24 | Mon.

Data Model

- underlying structure of a database system.
- allows us to specify constraints, relationships, semantics etc.
- 4 categories

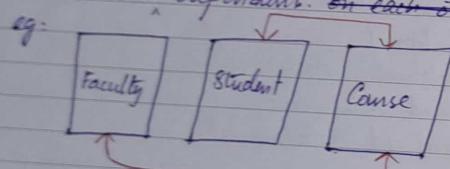
1. Relational Model.
2. Entity Relationship (ER) Model.
3. Object based data model.
4. Semi Structured data model.

Relational Model:

- describes data in the form of small relations.
- table format models.
- The whole system is defined in the form of small relations.
- **RDBMS:** any DBMS which follows the relational model. e.g. Oracle, Postgres etc.

2. Entity Relationship Model:

- Data is defined in terms of entities.
- We identify entities from the working of the system.
- Widely used in design phase.
- Eg: In University Management System: in this Students, Faculty, Courses etc are entities.
- Anything about which we want to store data in the database is called an entity.
- Entities are independent on each other.



• So we check relationship b/w entities in this model.

3. Object based data:

- How to store object directly in data?
- Extension of ER model but supports objects.
- They also support certain object paradigm

- **ORDBMS** - supports Object based models and Relational model.
- Eg: PostgreSQL.

4. Semi Structured Model:

- All other models are structured.

Eg:

Emp		
eno	ename	sal
100	John	1000

for every employee the same values are stored.

- In semi structured model, not all employees would have same values.

Eg: There might be an employee who has an attribute 'qualification' which is not present for the other employees.

- Eg: XML

Entity Relationship Model:

1. Entities

→ Regular / Strong entities: exist independently

→ Weak entities: depend on other entities to exist.

2. Attributes:

- characteristics of an entity

- Faculty (fid, fname, ~~dno~~ ...)

- Dept (dno, dname)

'dno' is not a direct attribute of faculty as it describes dept and not the faculty. So 'dno' is Dept's attribute and not the attribute of Dept Faculty.

Emp
eno

ename

sal

addr

phone

email

dob

age

1. Simple & Composite Attribute

- attributes which can be further divided: composite.

- eg: ename (^{fname}
^{middle name}
^{last name}), addr. etc.

- which can't be further divided: simple. eg: eno.

2. Single valued & Multivalued Attribute

- single valued: only one value can be assigned for a single entity.

- eg: eno, ename etc.

- multi valued: multiple values for single entity.

- eg: addr, phone, email etc.

3. Null attributes.

- Can be zero.

- eg: sal, email etc.

4. Derived attributes

- can be derived from another attribute
- eg: age (it can be derived from dob)
- ~~No~~ No derived attributes are put in the database at implementation phase. (It may be present in the design phase.)

ulty
faculty.
d not

4. Derived attributes

- can be derived from another attribute
- e.g.: age (it can be derived from dob)
- ~~No~~ **Only** derived attributes are put in the database at implementation phase. It may be present in the design phase.

14/10/24 | Monday.

Relationship sets

- association b/w entities in a system.

- Two concepts:

1. Cardinality ratio (mapping cardinality)

say two entity sets: entity & entity

$$E_1 \text{ --- } R \text{ --- } E_2$$

how many entities from E_2 can be associated to one entity in E_1 .

There can be: one in E_1 to one in E_2 & vice versa.

1. one to one

2. one to many

3. many to one

4. many to many

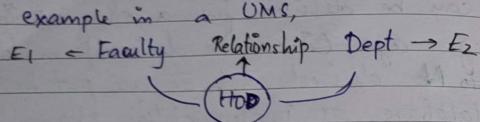
one from E_1 ↘ associated to many in E_2

but ~~only~~ ^{Page No.} one in E_2 is associated

ONLY to ONE in E_1 .

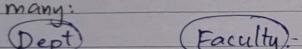
One to one:

example in a UMS,



one HOD (one faculty) to one dept.

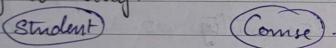
One to many:



one dept has many faculties.

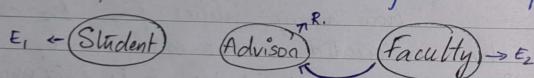
Reverse for many to one.

Many to many:



Many student in many courses.

2. Participation constraint: entity & relationship



Total participation \leftarrow Every student must have an advisor.
But every faculty need not participate in being advisor. \rightarrow Partial participation

Descriptive Attributes:

If an attribute is equally important ~~between~~ to the entities then we assign the attribute to the relationship.

Attributes that define or are associated with the relationship are called descriptive attributes.

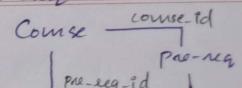
* degree of a relation: no. of columns / attributes
cardinality of a relation: no. of rows / tuples.

Cardinality of a relationship: no. of entities participating in a relationship.

- $E_1 - R$: unary
- $E_1 - R - E_2$: binary
- $E_1 - R - E_2 - E_3$: ternary
- $E_1 - R - E_2 - E_3 - E_4$: quaternary

Unary Relationship:

- Must include the roles just above the connection lines.
- eg: a course is related to another course through a pre-req.



Regular & Weak Entities:

Weak Entity: eg: Bank has Branch

-
-
-
-
-

branch depends on bank for its existence.
So branch is a weak entity.

Entities which depend on other entities for its existence: weak entity.

Weak entities do not have a primary key on their own. They have an attribute from ~~Bank~~ ^{strong entity} and then an attribute from weak called partial key / discriminator.

Attribute to distinguish b/w diff. instances of a particular strong entity - partial key.

e.g. branchname is a discriminator/partial key
as if we have two ~~books~~ instances of bank A, one could be in Vallikava & other can be in TVM (which is written as branch name).

Identifying relationship:

The relationship which connects a dependent weak entity to its dependent entity.

Here Bank → dependent
Branch → weak.

Identifying relationship:

The relationship which connects an dependent weak entity to its dependent entity.

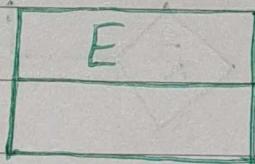
Here Bank → dependent
Branch → weak.

15/10/24 | Tuesday.

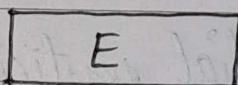
ERD Notations:

ER Diagram: pictorial representation of ER model.

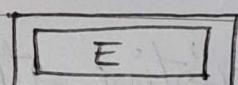
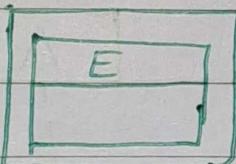
Entity set.



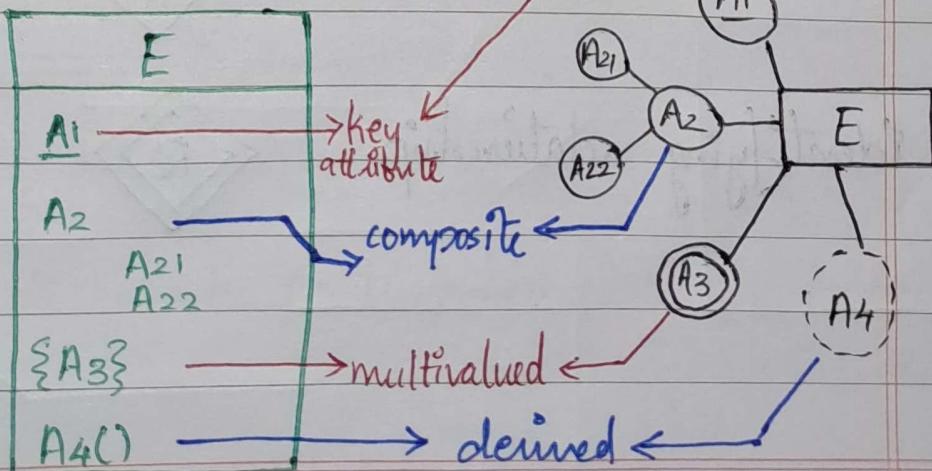
Notation 2.

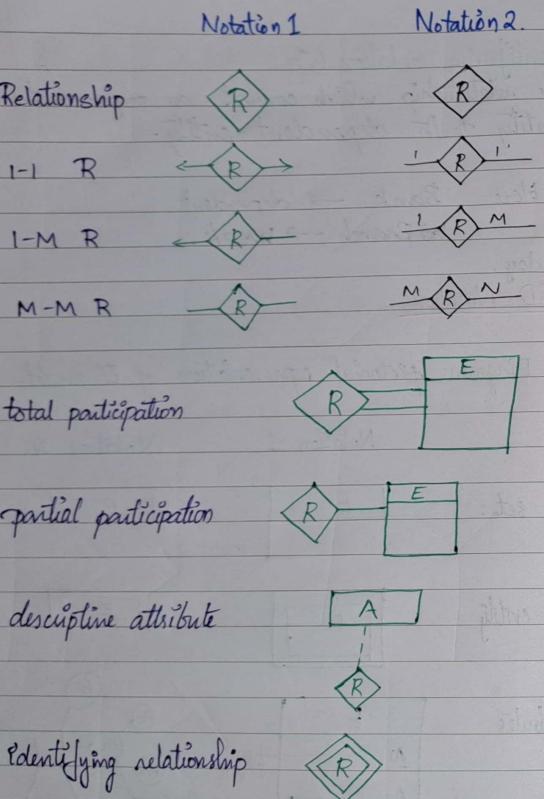


Weak entity



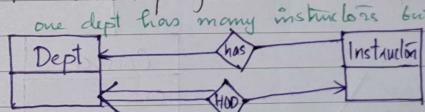
Attributes



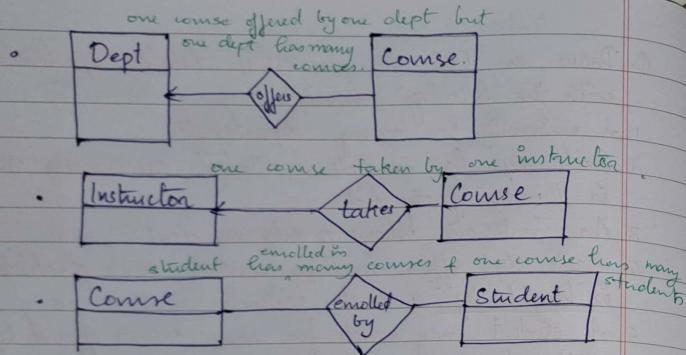


- Q: Draw an ERD for university database application with the following description
1. A university has many departments
 2. Each dept has multiple instructors, one among them is the HOD of the dept.
 3. An instructor belongs to only one dept.
 4. Each dept offers multiple courses, each of which is taught by a single instructor.
 5. A student may enroll for many courses offered by different dept.
- A: Entities present & their attributes:
1. Dept - dno, dname, dloc
 2. Instructor - ino, irname, sal, qual
 3. Course - cno, cname, credit
 4. Student - sno, sname, saddr.

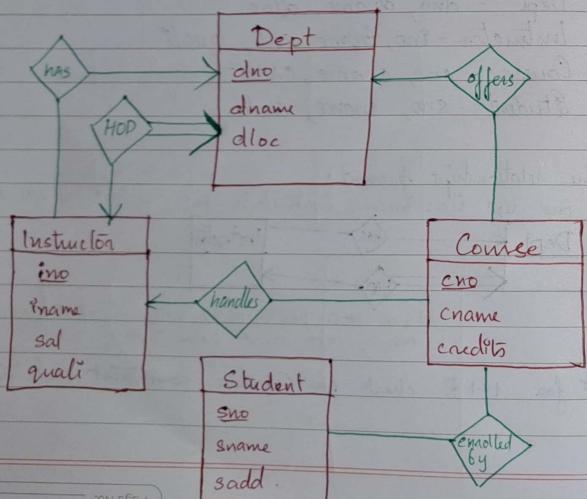
The relationships present.



* for 1-1 R check participation constraint each time.



Now make all this into single ERD,



- Q. Consider a banking scenario given below & draw an ERD for the same.
- Assume in a city there are multiple banks & each bank has many branches.
 - Each branch has multiple customers.
 - Customers have various types of accounts.
 - Some customers have also taken diff. types of loans from these bank branches.
 - One customer can have multiple accounts & loans.

- A: Entities & attributes:
- Bank - bname
 - Branch
 - Customer
 - Account
 - Loan

Subquery:
17/10/24/Thursday.

Emp (eno, ename, sal, dno)
Dept (dept_no, dname, loc)

Q. Find name of dept in which emp E1 works.

Select dname from Dept where (dept_no = (

Dept no
not true within

select dno from Emp where
eno = 'E1';
Query for this.

Thus a query inside a query, is called subquery.
is used.

→ select dname from Dept where dept_no = (select dno
from Emp where eno = 'E1');

The subquery within the innermost () is
executed first.

- Q. Find the name of all employees who work in
'sales' department?
A: select ename from Emp where dno = (select dno
dept_no from Dept where dname = 'sales');
- Q. Find those employees (names) getting a salary greater than that
of employee 'E1'?
A: select ename from Emp where sal > (select sal
from Emp where eno = 'E1');
- Q. Consider the given schema
Customer (cno, cname, ctype)
Accounts (acc_no, cno, ac_type)
Customer_Loan (cno, ln_no, ln_amt)
Customer_FD (cno, fd_no, fd_amt, int_rate).

1. Find the names of all customers who have taken a loan
of amount > 50k?
2. Find those customer numbers who have same ac_type
as customer 'C1'
3. Find those customer names who have not taken any loan.
4. Find those customer names who have a fixed deposit
at the bank.

- A: 1. select cname from Customer where cno \in (select cno from Customer-Loan where In-amt \geq 50000);
2. select cno from Accounts where ac-type = (select ac-type from Accounts where cno = 'C1');
3. select cname from ~~Customer~~ where cno $\not\in$ (select cno from Customer-Loan);
4. select cname from Customer where cno \in (select cno from Customer-FD);

We can only use '=' operator if value is one. If more values then use 'in'.

^{on inner queries}
Subqueries which can work independently of outer query without needing anything called from outer query is called Independent subquery.

↳ opposite
Correlated subquery.

↳ needs value from outer query.

18/10/24 | Friday.

Q. Consider the given relation.

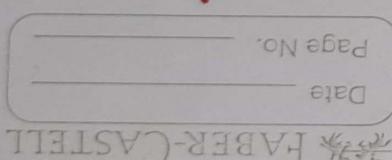
Emp (eno, ename, sal, dno)

Emp-dependent (eno, dep-no, dep-name, gender)

Emp-Proj (eno, pno, work-hrs)

Proj (pno, pname)

1. Find the ename of all employees getting salary greater than that of all employees in dno 'D1'.



1: select ename from EMP where sal > ALL (select sal from EMP where dno = 'D1')

~~If we have~~

If we have an inner query which returns multiple values & we need to compare those with a single value; we use 'ALL'.

OR
→ select ename from EMP where sal > (select max(sal) from EMP where dno = 'D1')

2. List the name of employees getting sal greater than that of any employee in dept 'D1'.

1: select ename from EMP where sal > ANY (select sal from EMP where dno = 'D1')

or

select ename from EMP where sal > (select min(sal) from EMP where dno = 'D1').

~~If we~~

ANY: for each tuple the salary will be taken & compare it with the returned value of inner query & if it is greater than it will break there f won't look into the others.

Q. Same relation as previous qn except:
EMP(eno, ename, sal)

- A:
1. Find the name of employee getting highest sal?
 2. List the ename having female dependents.
 3. Find the ename of all emps who worked in 'Order Entry' project for more than 10 hrs.
 4. Find ename getting sal more than avg. sal of all employees.

1. select ename from EMP where sal = (select max(sal) from EMP);
2. select ename from EMP where eno = (select eno from Emp_dependent where gender = 'F');
3. select ename from EMP, ~~where eno = (select eno of from Emp_Proj where pno = (select pno from Proj where pname = 'Order Entry'));~~
4. select ename from EMP where sal > (select ~~sal~~ avg(sal) from EMP);

5. Find the names of emp who is not working in any proj.

A: select ename from EMP where eno not in (select eno from Emp_Proj);

A: 3. select ename from EMP where eno in (select eno from Emp_Proj where work_hrs > 10) and pno = (select pno from Proj where pname = 'Order Entry');

Q. Consider these relations

EMP(eno, ename, sal, dno, job_id)
Accounts(ac_no, cur_bal, br_no)
Job(job_id, jname)

1. Find all dept getting ~~sal~~ min sal > dept 'D1'?
2. Find eno & name of those emp who are not IT programmer & getting sal > Any IT programmer.

A:

1. select dno, min(sal) from Emp Group by dno having min(sal) > (select min(sal) from EMP where dno = 'D1');

2. select eno, ename from EMP where job_id > (select job_id from Job where jname = 'IT programmer') and sal > (select sal from EMP where job_id = (select job_id from Job where jname = 'IT programmer'));

Correlated subquery:

- Inner queries that require some sort of reference from outer subquery.
- The inner query will be executed as many times as the no. of tuples in the outer query for a correlated subquery whereas for an independent subquery it executes only once.

Q. Customer (cno, cname)

Cust-FD (cno, fd-no, fd-amt)

Cust-Loan (cno, ln-no, ln-amt, int-rate)

1. List all customers who have a fixed deposit of amount less than sum of all loans taken by that customer.

A:- select cno from Cust-FD F where
fd-amt < (select sum(ln-amt) from Cust-Loan
where cno = F.cno);

Cust-FD F → temporary name.

C. Here F refers to the 'alias' name of the Cust-FD, so that whenever we get / use 'F' from here after then it means that table.

B. $cno = F.cno$

cno that we have selected from F.
Here we are saying that in the inner query, the 'cno' should be equal to that of 'F.cno'.

If in the inner query we write as:

(... where cno = cno) The system would return an ambiguity error.

2. List all cust names who have both FD + Loan

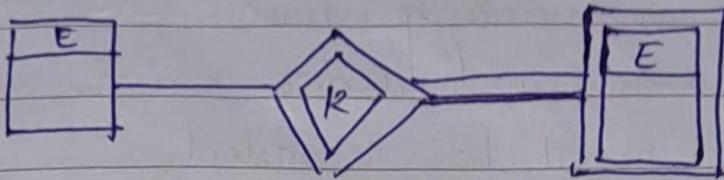
A:- select cname from Customer F where cno in (select cno from Cust-FD where cno = F.cno) and cno in (select cno from Cust-Loan where cno = F.cno);
3. List accounts along with cur-bal and bn-no having a balance more than the avg bal of the branch to which the account belongs.

Accounts (ac-no, cur-bal, bn-no).

A:- select ac-no from Accounts A where
cur-bal > (select avg(cur-bal) from Accounts
where bn-no = A.bn-no);

21/10/24/Mon.

continued here from 07/10/24(Thu)



every weak entity
must have a total
identifying relation
(Identifying relation) with
dependent relation.

(solution put in AUMS)

Q. Draw an ERD for the following company database

1. The company is organised into departments.
2. Each dept has a unique no., name and a particular employee who manages the dept.
3. To keep track of the starting date, that particular employee started managing the dept.
4. A dept may have several locations.
5. A dept controls a no. of projects each of which has a unique number, name and a single location.
6. We store each employees' id, name, addr, salary, gender and birth date.
7. An employee is assigned to only one dept, but may work on several projects which are controlled by different departments.
8. We also keep track of the no. of hours an employee works on each project.
9. There is a supervisor for each employee who is another employee.

employee

- 10. We want to keep track of the dependence of each employee for insurance purpose.
- 11. The dependent's name, birth date and the relationship with employee must be recorded.

A: Entities & attributes.

~~Department~~

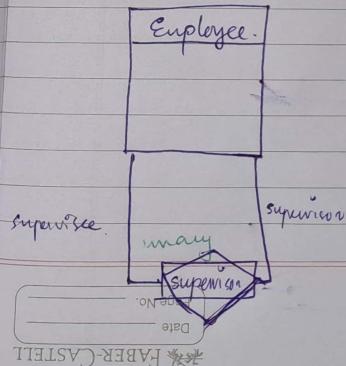
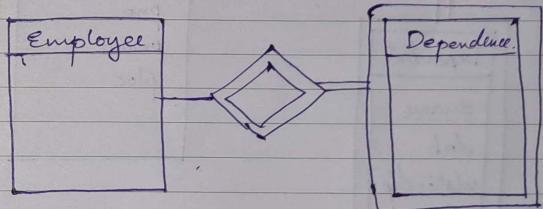
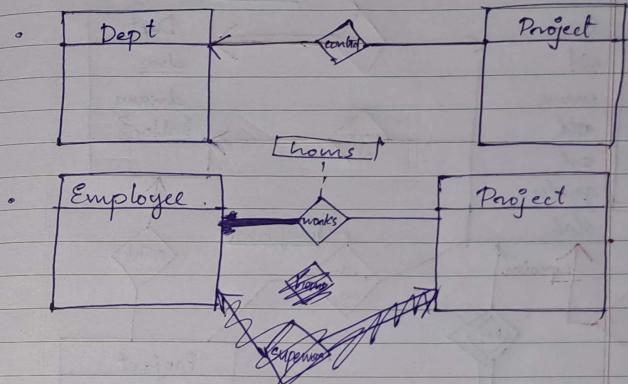
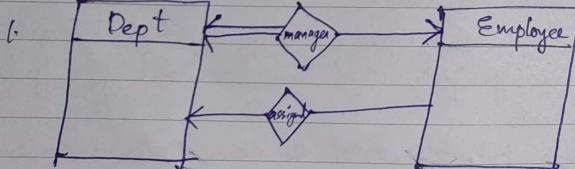
- Dept - dno, dname, ~~start manager~~, ~~start date~~,
- Employee - eid, ~~dept~~, ~~addr~~,

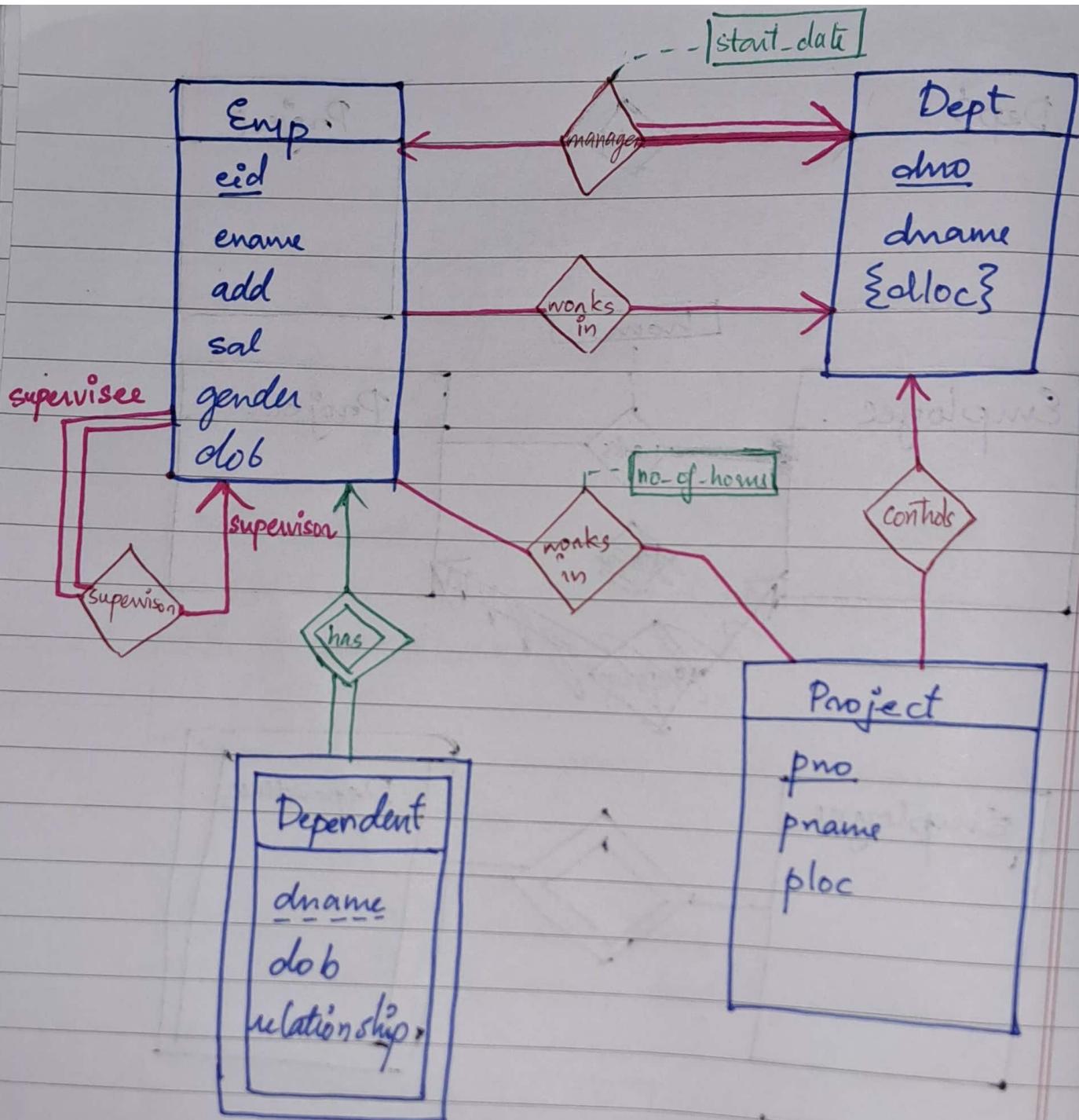
Employee - eid, empname, add1, salary, gender, dob, ~~addr~~

Project - pno, pname, place.

Dependence - depname, ~~dob~~, relation.

Relationships: [starting date]

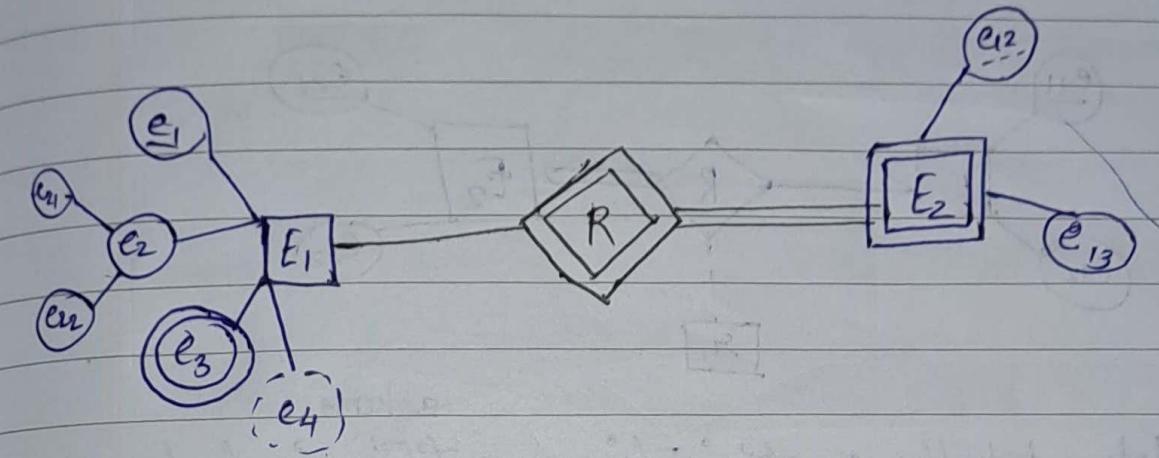




24/10/84/ Thursday.

ER to Relational Mapping:

Regular & Weak Entities



- ▷ No derived attributes included into schema.
- ▷ Components of composite attribute are added to schema.
- ★▷ Include primary key of dependent entity as a foreign key of weak entity ^{components of composite attribute e2.}

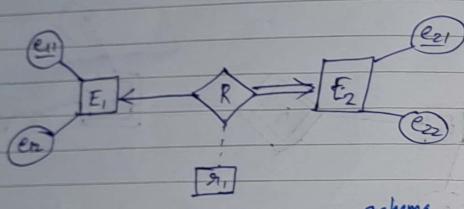
E1 (e_1, e_{21}, e_{22})

E2 (e_1, e_{12}, e_{13})

↳ primary key of E_1 added as foreign key of E_2 .

2. Mapping 1-1 Relationship:

2. Mapping 1-1 Relationship:



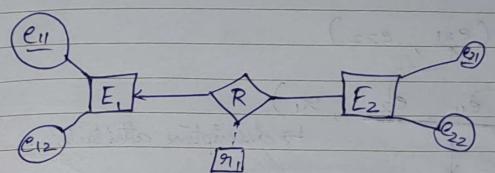
► Into totally participating entity's include primary key of partially participating entity as a foreign key.

► The descriptive attributes, if present, should be added to ~~partially participating entity's~~ schema.

$E_1(e_{11}, e_{12}) \rightarrow$ partially participating entity.

$E_2(e_{21}, e_{22}, e_{11}, g_1) \rightarrow$ total participation entity
↓
descriptive attribute.

3. Mapping 1-M / M-1 Relationship:



Same as previous (1-1 relationship).

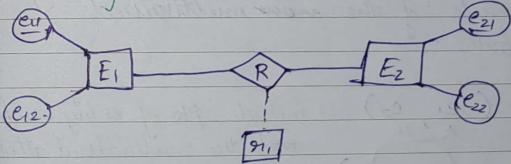
Total participation == Many side.

Partial participation == One side.

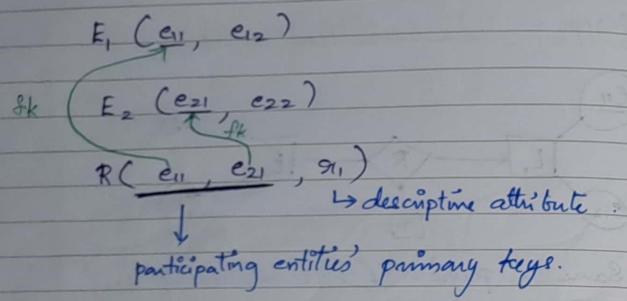
$E_1(e_{11}, e_{12}) \rightarrow$ one

$E_2(e_{21}, e_{22}, e_{11}, g_1) \rightarrow$ many
↓
descriptive attribute.

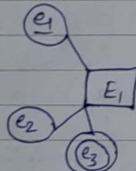
4. Mapping M-M relationship:



For each M-M relationships, add new table



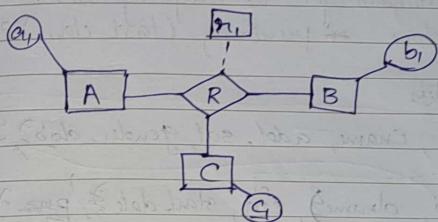
5. Multivalued Attributes



We create a new relation corresponding to each of the ~~new~~ multivalued attributes.

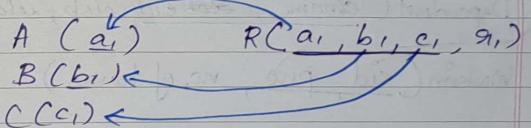
- $E_1 (e_1, e_2)$ include pk of entity as fk.
- $E_1 - e_3 (e_3, e_1)$ with pk becomes the pk of the relation

6. Relation's degree ≥ 2



Create new table for the relation with degree ≥ 2 .

Include pk's of all the participating entities as foreign key & merge them & use as pk.
Include descriptive attributes as well.



The output of ER to relational mapping is called schema diagram.

It should have attributes of entities + relations
(if fks used etc.)

Schema

- Q. Schema diagram for the company database
question done ~~not~~ previously (last class)?

A:

Regular entities

25/10/24 | Friday.

(Tables need not have same names as entities)

Emp (e_id , fname, lname, sal, dob, gender, dno, supervisor-id)

Dept (d_no , dname, mgr-eid, mgr-startdate)

Proj (p_no , pname, ploc, did)

DepAlloc (d_no , d_no)

Dependent (e_id , $dname$, dob, relationship)

Works-on (e_no , p_no , hours)

Procedure of selecting primary keys:

1. Select the super keys
2. From there select candidate keys
3. And thereby select primary keys.

eg. Emp (e_no , ename, email, age)

Super keys:

{ e_no }, {email}

{ e_no , ename}, { e_no , email}, { e_no , age}

{ename, email}, {email, age}

{ e_no , ename, email}, { e_no , email, age}

{ e_no , ename, email, age}

These combinations are capable of identifying the tuples. Hence they are super keys.

Candidate key:

If you take a combination super key & one of its subsets is a super key then it is not a candidate key.

Minimal super keys are candidate keys.

so:

None of the combinations are candidate keys (as they all contain eno or email or both which are super keys).

hence $\{eno\}$ and $\{\text{email}\}$ are the candidate keys.

Out of the candidate keys, the developer usually choose the key which is easy to remember or do not change often.

Here $\{eno\}$ is chosen.

Extended ER (ER) features:

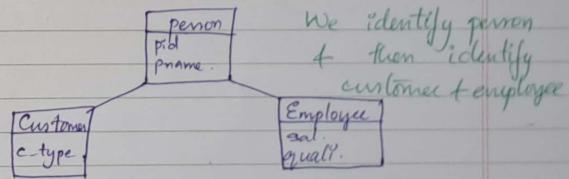
Certain features of OOPs such as inheritance, polymorphism etc cannot be displayed using our normal ERD. So we ~~can~~ included them in EER.

The features are:

1. Generalisation / Specialization
2. Super class / Sub class.
3. Attribute Inheritance
4. Aggregation.

1. Generalisation / Specialization

- Top down approach - Specialization.
- Identify major entity & then identify the sub entities.



We identify person
& then identify
customer & employee

- Bottom-up approach - Generalisation
- Same thing but we identify employee & customer first & then person.

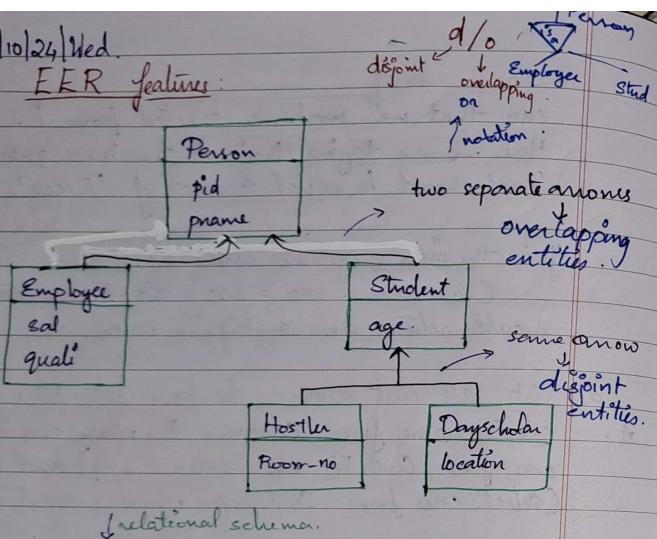
-  **Overlapping** - a customer can also be an employee.
- **Disjoint** - a student can either be a day scholar or a hostles.

Here the person → super class.
customer & employee → sub class.

Attributes of person are inherited by customer & employee.

30/10/24 Wed.

EER features:



↓ relational schema.

Person (`pid`, `pname`)
Employee (`eid`, `sal`, `quali`)
Student (`sid`, `age`)

- Create a table for the head entity (Person)
- Then create separate table for the sub entity with their specific attributes & then add the pk of the two super entity as a fk in the sub entity-tables.

Disadv:

To get the whole information related to a sub entity then you'll have a hard time.

↓ how to remove.

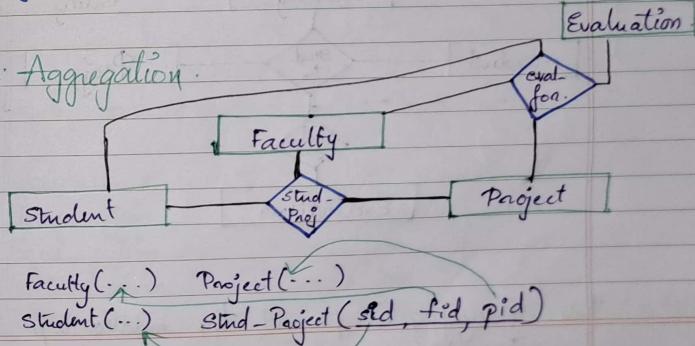
In the relational schema do not create a table for the higher level entity. Create only create the tables corresponding to the lower level entity.

(Disadv.)

In case of overlapping entities, there would be repetitions in the overlapping entities.

Hence we only use this kind of relational schema if all the entities are disjoint.

2. Aggregation.



eval-fac (sid, fid, pid, eval-id)
Stud-Proj (sid, fid, pid)

There is redundancy

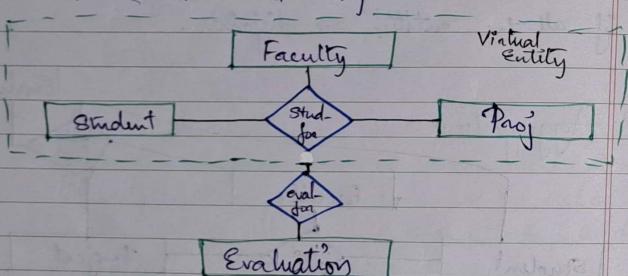
So can we remove Stud-Proj?

→ No, we cannot. Some projects may have just started & not yet had its evaluation.

We can remove repetition using aggregation.

In aggregation,

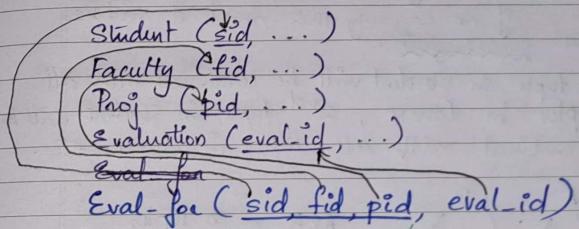
We will consider Stud-Proj connected with other 3 entities as an abstract entity. And then eval-fac will be connected to that abstract/virtual entity.



Rule: If aggregation present, the relationship

considered as a virtual entity will not have a table ~~connected~~ for it. Instead we will only create a table for the other ^{relationship} entity eval-fac & the components of virtual key with entity will be the pk and the pk from the entity (Evaluation) will just be treated as a fk.

Thus the tables would be:



the pk for the Stud-Proj

Going back to sql...

Joins:

1. Inner join
2. Equi join
3. Natural join
4. Outer join
5. Self join
6. Cross join

Joins are used to get info from more than one table at a time

1. Inner Join & Cross join & Natural join :

Student (sid, sname, age, cno)

Course (course_no, cname, credits)

Q. How to find info. of student details AND course details?

We use joins.

→ select * from Student, Course

If we say only this much it will be a cross join.

1st tuple in Student will be combined with all tuples in Course, 2nd tuple in Student will be combined with all tuples in Course etc.

$$\begin{array}{cc} A & B \\ () & () \end{array} \quad A \times B = 50 \text{ tuples}$$

10 tuples in A 5 tuples in B

↳ cross join.

This is not what we want. Somewhat.

Internally:

The system will do cross join & then apply conditions for the other joins onto the cross join output & get the output for other joins.

actual answer:

▷ select * from Student, Course where cno = course_no;

or

▷ select * from Student natural join Course;

In natural join, whichever column is common for both tables, it will consider that as the condition and give output. Common col \Rightarrow there must be reference

▷ select * from Student inner join Course on cno = course_no;

Natural join only works if the two tables have ^{some} columns of that refer each other.

actual answer:

- ▷ select * from Student, Course where
cno = course_no ; → This is also equijoin
because we use equality operator
- ▷ select * from Student natural join Course;

In natural join, whichever column is common
for both tables, it will consider that as the condition
and give output. Common col \Rightarrow there must be reference

- ▷ select * from
Student inner join Course on
cno = course_no;

Natural join only works if the two ~~columns~~
have ^{same} columns & that refer each other.

12/11/24/Tuesday

2. Equijoin:

Inner joins with equality operator condition.

eg: select sname, cname from Student, Course
where cno = course_no;

Q. Emp(cno, ename, city, sal)

Customer (cno, cname, cltype, city, sal)

Q. Find the name of customers getting salary $>$ that of employees

1. select ename from Customer where sal > ANY

select cname from Customer C, Emp E where
C.sal > E.sal.

↓ This is an inner join. Because we
are combining two tables who have no referencing
between each other. ~~that's not~~

2. Find employee names of customer names who belong
to the same city?

1. select ename, cname from Customer C, Emp E
where C.city = E.city;

OR

select ename, cname from Emp E inner join
Customer C on E.city = C.city;

Self Join:

Emp (eno, ename, sal, mgr_no)			
E ₁	a	-	E ₁
E ₂	b	-	E ₁
E ₃	c	-	E ₂
E ₄	d	-	E ₁

A join which joins a table to itself.

1. Find employee names with corresponding manager
names?

1. The answer we need:
ename mgr-name
a a
b a
c b
d a

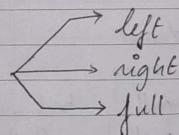
We will consider one more copy of the same table.
Thus:

E₁ (eno, ename, sal, mgr_no)
E₂ (eno, ename, sal, mgr_no).

The query becomes:

select E₁.ename, E₂.ename from Emp E₁, Emp E₂
where E₁.mgr_no = E₂.eno;

Outer Join



Customer (cno, cname, cadd)

Cust FD (cno, fd-no, fd-amt)

A join in which it will print matching tuples
and non-matching tuples.

1. Find customer details with corresponding fd details
if any?
↳ We have to print all customer details as well as
the fd details if they have fd.
Thus we have to use outer join.

select * from Customer C **left outer join**
Cust-FD CFD on C.cno = CFD.cno;

left outer join:

all tuples from the table of left side is
printed in the output and only the matching
condition values in right table is printed other
values are written as null.

cno	cname	cadd	cno	fd-no	fd-amt
C ₁	a	x	a	123	120000
C ₂	b	y	null	null	null
C ₃	c	z	null	null	null

right outer join:

Same as left join that instead of left table
it will be the right table.

Full outer join:

From both tables, all
matching and unmatched
tuples are printed out.

right outer join:

Same as left just that instead of left table it will be the right table.

Full outer join: all

From both tables, matching and unmatched tuples are pointed out.

14/11/24 Thursday.

Normalization:

Stud-Course-Grade (sno, sname, cno, cname, credit, mark, grade)

S ₁	a	C ₁	OS	3	85	A
S ₁	a	C ₂	WT	4	90	A+
S ₂	b	C ₁	OS	3	35	F
S ₃	c	C ₁	OS	3	70	B
S ₄	d	C ₃	CCNA	2	85	A

We do normalization if anomalies are present.

These are 3 types of anomalies that can occur:

→ Insert

Assume a new course has been introduced but no student has joined the course. sno + cno form PK of the schema so we cannot insert the new course because we do not have any student in it.

Thus this is an insert anomaly.

→ Update

Say the student S1 joins another course and we insert / update the schema. When there is repetition of information, whenever we want to update on the repeated data, we need to make changes in multiple rows. Thus creates an update anomaly.

→ Delete

Assume student S4 withdraws from the course. Then we will need to remove the whole row and also the course because that course is only taken by S4. Hence this is delete anomaly.

Functional Dependency:

$R(x, y, z)$

determinant $\hookrightarrow x \rightarrow y \Rightarrow x$ determines y or y is conditionally dependent on x .

For a particular value of x , we must always get a unique value of y attribute. Then x determines y .

$$x_1 \rightarrow y_1$$

$$x_2 \rightarrow y_1$$

$$x_1 \rightarrow y_2$$

x_1 can only have y_1 .

Thus in the given schema, we find functional dependency of every non-key attribute

$$sno \rightarrow sname$$

$$cno \rightarrow cname$$

$$cno \rightarrow credit$$

$$sno, cno \rightarrow mark$$

$$sno, cno \rightarrow grade$$

$$mark \rightarrow grade$$

} all non-key attributes.

1. Partial Functional Dependency:

If a non key attribute is functionally dependent on a part of the primary key then it is partial FD.

e.g.: sname, cname, credit

2. Full Functional Dependency:

For the non key attribute determinant is the full primary key.

e.g.: mark.

3. Transitive FD:

We derive the FD from other functional dependencies.

$$\text{eg: } \begin{array}{ll} \text{mark} \rightarrow \text{grade} & -\textcircled{1} \\ \text{sno, cno} \rightarrow \text{mark} & -\textcircled{2} \end{array}$$

from $\textcircled{1} + \textcircled{2}$:

$$\text{sno, cno} \rightarrow \text{grade}.$$

Normal Forms:

Normalization is based on certain normal forms.

1. First Normal Form (1NF)

A relation is in 1NF if all the attributes should be atomic in nature.

↳ not composite

not multivalued.

In the given schema (Stud_Course_Grade), all attributes are atomic and hence is in 1NF _{one NF (pronunciation)}

2. Second Normal Form (2NF)

Conditions:

1. Must be in 1NF

2. No partial dependency b/w non-key attribute & key attribute.

3. Third Normal Form (3NF):

Conditions:

1. Must be in 2NF
2. No transitive dependency between non-key attributes and key attributes.

In the given schema, there are partial dependencies hence it is not in 2NF.

↳ So now we divide relations so that it becomes 2NF.

3. Third Normal Form (3NF):

Conditions:

1. Must be in 2NF
2. No transitive dependency between non-key attributes and key attributes.

In the given schema, there are partial dependencies hence it is not in 2NF.

→ So now we subdivide relations so that it becomes 2NF.

15/11/24 | Friday.

The partial dependencies were:

sno → sname

cno → cname, credits

Now we split this,

Student (sno, sname)

Course (cno, cname, credits)

Now two more dependency left

sno, cno → mark

mark → grade.

This becomes:

Mark Grade (sno, cno, mark, grade)

Thus Student-Course-Grade has been divided into 3 relations: Student, Course & Mark-Grade.

How to check if we can stop normalization?
We check if any anomalies are left?

Here:

Student (sno, sname)

Course (cno, cname, credits)

Mark-Grade (sno, cno, mark, grade)

We can see that no more insert, delete or update anomalies are present in Student or Course.

For Mark-Grade:

Mark-Grade (sno, cno, mark, grade)

S ₁	C ₁	85	A
S ₁	C ₂	70	B-
S ₂	C ₂	85	B
S ₃	C ₁	80	B ⁺
S ₄	C ₂	85	A
-	-	-	-
-	-	-	-

Are we repeating any data?

→ No but check A grade and mark. A particular mark has particular grade is being repeated here. That is in some way an update anomaly.

is present. Say grade for 85 marks has changed to A-, we'll have to change values in multiple rows.

Any delete anomaly?

Assume student S₁ is deleted. Then 70 marks gives B-grade is also deleted. Thus delete anomaly occurs here.

Is insert anomaly present?

No.

But since no delete & update anomalies are present we go for next normal form.

3NF

Mark-Grade (sno, cno, mark, grade)
only this relation has anomalies so we only check this relation.

Mark-Grade → is in 2NF.
→ but has transitive FD.
→ so not in 3NF.

↓
split relation.

Thus table divided as:

Stud-Mark (sno, cno, mark)

Mark-Grade (mark, grade)

Any info repeated?

No info repeated in either tables. Thus no update anomaly.

Any delete anomaly?

No delete anomaly. marks & grade not affected when you delete a student info.

Thus no more anomalies. The relation becomes:

Student (sno, sname)

Course (cno, cname, credits)

Stud-Mark-Grade (sno, cno, mark)

Mark-Grade (mark, grade)

Here the attributes are:

Stud-Details $\leftarrow S_1$

Course-Details $\leftarrow C_1$

Mark-Details $\leftarrow M$

They are all composite attributes.

Thus this is NOT IN 1NF.

Thus we split it into:

Stud-Course-Grade (sno, sname, cno, cname, credits, mark, grade)

The remaining is done as we did previously.

Q. Assume it was given like the following & we have to normalise it.

Stud-Course-Grade

Stud-Details	Course-Details	Mark-Details
S ₁ a	C ₁ OS 3	85 A
S ₂ b	C ₁ DBMS 4	80 B+
S ₃ c	C ₂ WT 3	70 B-
S ₁ a	C ₂ WT 3	90 A+

Here the attributes are:

Stud-Details \leftarrow^{S_1}
a

Course-Details \leftarrow^{C_1}
os
3

Mark-Details \leftarrow^{M_1}
A

They are
all composite
attributes.

Thus this is

NOT IN 1NF.

Thus we split it into:

Stud-Course-Grade (sno, sname, cno, cname, credits,
marks, grade)

The remaining is done as we did previously.

18/11/24/Monday.

Q. Salesperson-Product (sl-no, sname, pno, pn-name,
unit-price, city, status, qty-sold).

Assumption: one prod. sold by many salesman. One
salesman sells ~~one~~ many products.

sl-no sname pno pn-name unit-price city status qty-sold.

Sample tuples:	S1	a	p1	aa	10	C1	A	100
	S1	g	p2	bb	15	C2	B	200
	S2	b	p1	aa	10	C2	B	150
	S2	bb	p2	bb	15	C3	C	500
	S3	d	p3	cc	20	C3	D	250
	:	:	:	:	:	:	:	:

Is there insert anomaly?

- ▷ Say a new salesman joined. We are not able to insert the value because they have not yet sold anything \Rightarrow Insert Anomaly.
- ▷ Say a new product has been introduced. It has not yet been sold. So again we cannot insert it \Rightarrow Insert Anomaly.

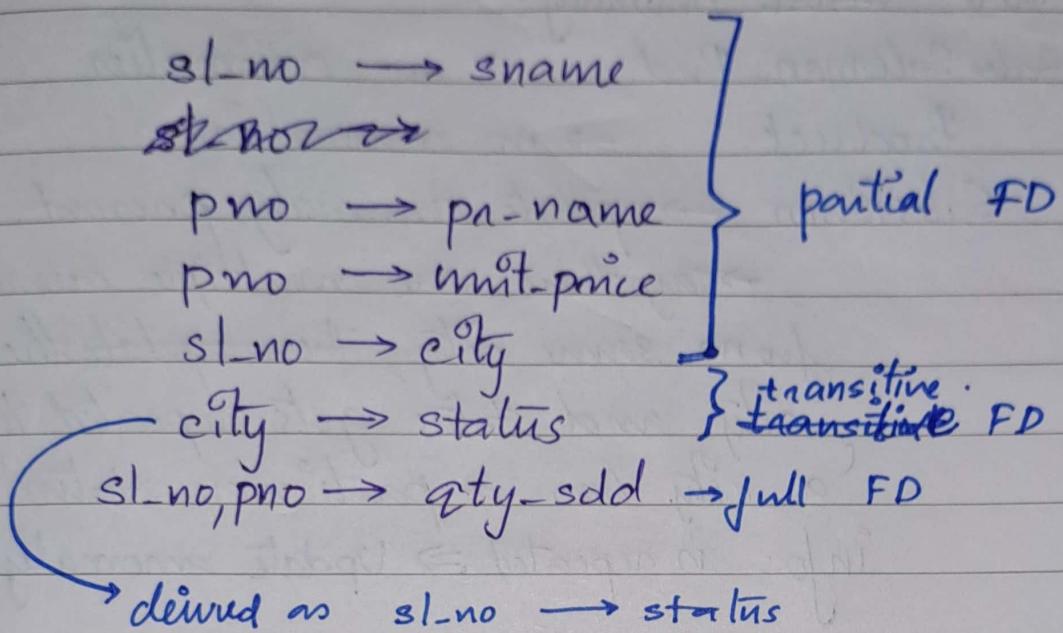
Is there update anomaly?

- ▷ Name of salesman and other info related to salesman is repeated.
- ▷ ~~Reo~~ If same product is sold by 2 different salesmen then again prod details are repeated.
- ▷ Thus ~~update anomaly~~ \Rightarrow update anomaly.

Is there delete anomaly?

- ▷ Only salesman ~~P3~~ S_3 has sold product P_3 . Say salesman S_3 leaves. If we delete salesman S_3 ~~its~~ info related to P_3 is also deleted \Rightarrow Delete anomaly.

Functional dependencies of the relation:



1NF

- all attributes are atomic in nature
- Relation is in 1NF.

2NF

- should be in 1NF
- no partial dependency.
- Has partial dependency so not in 2NF.

Now divide the relation. It should be lossless

↓
no attribute in original relation must be left out.

It should be

present in atleast one of the subdivide relations.

SalesPerson (sl-no, sname, city, status)

Product (pno, pn-name, unit-price)

Salesman_Prod (~~sk_no~~ sl-no, pno, qty-sold)

Now check:

Is there ~~any~~ Anomaly?

~~Sales~~ Salesman_Prod → no anomalies

Product → no anomalies.

SalesPerson → update anomaly present.

→ say there are multiple salesman

from same city, then update then city and status gets repeated. Thus a city has a particular status such

info: is repeated ⇒ Update anomaly.

3NF

- should be in 2NF
- no transitive FD.

SalesPerson (sl-no, sname, city)

CityStatus (city, status)

{ No more anomalies . }

Thus the relation has been divided into:

Salesman (sl-no, sname, city)

Product (pno, pname, unit-price)

Salesman_Prod (sl-no, pno, qty-sold)

CityStatus (city, status)