19/9/24/Thu.

DBMS

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 form a primary key.
- primary key: unique & notnull values.
- only constraint which cannot be specified in column level is composite primary key.
- create table Branch (br_name varchar,
  bank_name varchar,
  . . . . . .
  . . . . . .
  constraint con1 primarykey (br_name, bank_name));
  ↓
  constraint name.

This is how we create a composite primary key constraint. This says that the PK is a combination of br_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~~ later.

★ Foreign key - at table level and at column level constraint
↓
References.
(Refer ppt)

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 ~~dependies~~ dependencies with reference values to master table then only delete.

To avoid this big process we can use:
create table Emp(eno ...,
ename ...,
dno vchar 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 as 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 unlike 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 conditions] [order by col_name]

Q. Consider the following relation:

Accounts (acc_no, cust_no, ac_type, bs_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 ~~are~~ 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 ~~*~~ 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 (bs_name = 'Kollam' and bal_amt < 25000);

A:

1. select 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' operator:

- 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: Pakvathy, 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.

eg:

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 ↳ takes null values.

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

3/10/24 Thursday

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

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

A: ~~count(~~ select count(mgr_id) from Emp;

OR

select count(ename) from Emp where mgr_id is not null;

2. Find the department number and the no: of employees in each department.

A: ~~select count~~
select dno, count(eno) from Emp group by dno;
here first and foremost, the groupby 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: select dno, count(eno) from Emp group by dno having count(eno) > 10;

We cannot use 'where' clause in along with group by. We must use 'having' wherever 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 ~~ac_no~~ 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.

~~Only~~ If these rules are followed then union blw 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 ~~un~~ 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.

DBMS

DB

- more structured storage.
- data can be interrelated.

MS

- set of programs to manage data.
- ~~provide security~~
- ~~data sharing~~.

Goals Of DBMS:

- Structure for 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

Before proper DBMS setups were done, data was stored within the File System in OS. ← The drawbacks were

1. Data redundancy & inconsistency.

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

BTech

SID	sname	addr	...

Cisco

SID	sname	address	...

here same data will have to be repeated in both ⇒ 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, ~~sal~~ salary not less than incentive etc are rules incorporated by organizations.

~~There are no way of adding those integ~~ 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 seats
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
read(B)
B := B + 1000
write(B)

This process should occur in the order.
Say an issue happened at the time of write(A).
Then what happens is 1000 Rs is gone from A but it is not added to B.

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

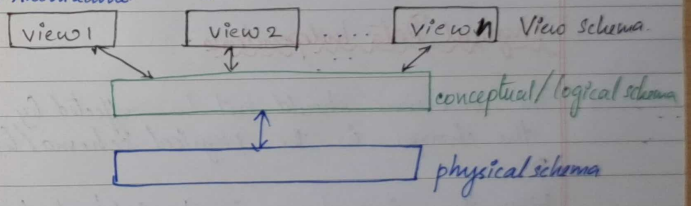
7. Security.

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 is used.
- Access methods etc.

Conceptual Schema:

- What data is stored
- Is there any relation btw 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.

are done.
used.
te.

n btw the data. ~~in the~~

ple view schemas.
d users interact-
ws because for
requirements are diff:
librarian and a student
same view as their
..

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 & Schema:

Instance: content of database at a particular time.

Schema: Overall design.

Page No.
Date
FABER-CASTELL

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

Instance & 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.

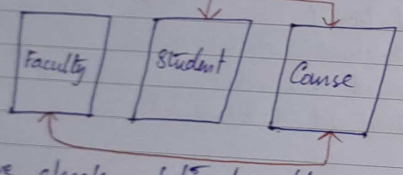
1. Relational Model:

- 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. eg: 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: an 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 not independent. ~~on each other~~

eg:



- So we check relationship blw entities in this model.

3. Object based data

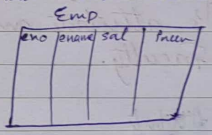
- How to store object directly in data?
- extension of ER model but supports objects.
- They also support certain object paradigms

- ORDBMS - supports Object based models and Relational model.
  - eg: Postgres.

4. Semi structured model:

- all other models are structured.

eg:



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
~~pno~~
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.
- multivalued: 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.

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.

14/10/24 | Monday.

Relationship sets.

- association blw entities in a system.
- Two concepts:
  1. Cardinality ratio (mapping cardinality)

say two entity sets: entity & entity

E_1 —— R —— 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.

reverse

one from E_1 associated to many in E_2

but only one in E_2 is associated ONLY to ONE in E_1.

one to one:

example in a UMS,

E_1 ← Faculty — Relationship: HOD — Dept → E_2

one HOD (one faculty) to one dept.

one to many:

Dept — Faculty

one dept has many faculties.

Reverse for many to one.

many to many:

Student — Course.

Many student in many courses.

2. Participation constraint: entity & relationship.

E_1 ← Student — Advisor (R) — Faculty → E_2

Total participation ← Every student must have an advisor.
But every faculty need not participate in being advisor. → 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
- $E_1 - R - E_2$ (with E_3 below R) : tertiary.
- $E_2 - R - E_3$ (with E_1 above R, E_4 below R) : quaternary.

Unary Relationship:

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

Course — course_id — pre-req — pre-req_id

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 blw diff: instances of a particular strong entity — partial key.

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

Identifying relationship:
The relationship which connects an ~~dependent~~ weak entity to its dependent entity.

Here Bank → dependent
~~bb~~ Branch → weak.

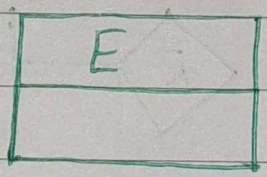
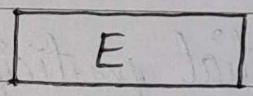
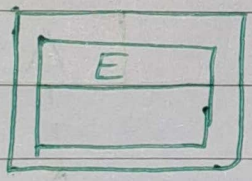
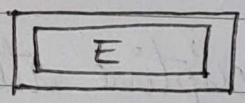
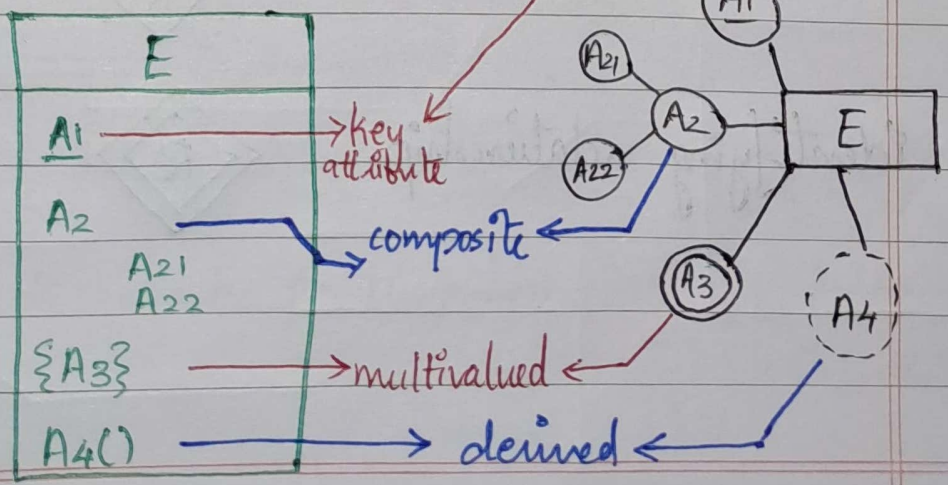
Identifying relationship:
The relationship which connects a ~~dependent~~ weak entity to its dependent entity.

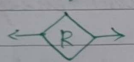
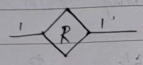
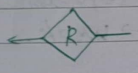
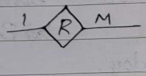
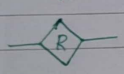
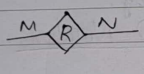
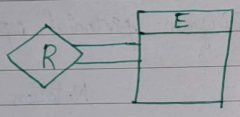
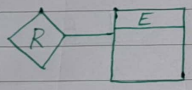
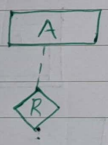
Here Bank → dependent
~~Bank~~ Branch → weak.

15/10/24 Tuesday.

ERD Notations:

ER Diagram: pictorial representation of ER model.

	Notation 1	Notation 2.
Entity set.		
Weak entity		
Attributes		

	Notation 1	Notation 2.
Relationship		
1-1 R		
1-M R		
M-M R		
total participation		
partial participation		
descriptive attribute		
Identifying relationship		

Q. Draw an ERD for university database application with the following description

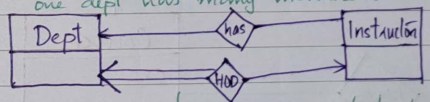
1. A university has many departments
2. Each dept has multiple instructors, one among them is the HOD of the dept.
3. An instructor belongs to only one dept
4. Each dept offers multiple courses, each of which is taught by a single instructor
5. A student may enroll for many courses offered by different dept

A: Entities present & their attributes:

1. Dept - dno, dname, dloc
2. Instructor - ino, iname, sal, quali
3. Course - cno, cname, credit
4. Student - sno, sname, sadd.

The relationships present:

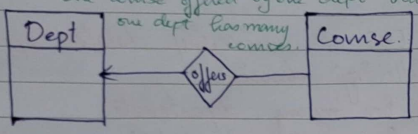
one dept has many instructors but one inst has one dept



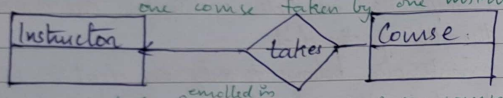
↳ one HOD per dept & only one inst becomes HOD. every dept must have HOD.

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

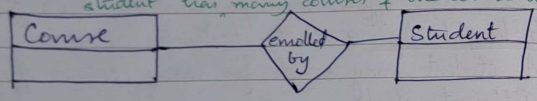
- one course offered by one dept but one dept has many courses.



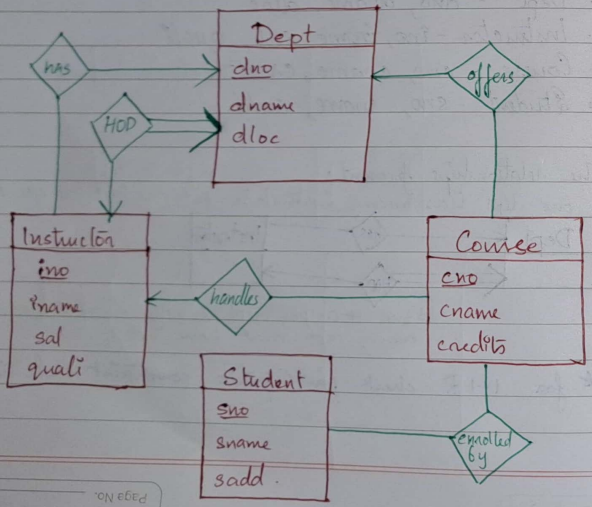
- one course taken by one instructor.



- student enrolled in has many courses & one course has many students.



Now make all this into single ERD,



Q. Consider a banking scenario given below & draw an ERD for the same.

1. Assume in a city there are multiple banks & each bank has many branches.
2. Each branch has multiple customers.
3. Customers have various types of accounts.
4. Some customers have also taken diff types of loans from these bank branches.
5. One customer can have multiple accounts & loans.

A: Entities & attributes:

- Bank - bname
- Branch
- Customer
- Account
- Loan.

17/10/24/Thursday.

Subquery:

Emp (eno, ename, sal, dno)

Dept (dept_no, dname, dloc)

Q. Find name of dept in which emp E_1 works.

select dname from Dept where dept_no = ?

select dno from Emp where eno = 'E1';

(we do not know this 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 from Dept where dname = 'sales');
dept_no

Q. Find those employees (names) 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 (ac_no, cno, ac_type)
Customer_Loan (cno, ln_no, ln_amt)
Customer_FD (cno, fd_no, fd_amt, int_rate)

1. Find the name 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 ln_amt > 50000);

2. select cno from Accounts where ac_type = (select ac_type from Accounts where cno = 'C1');

3. select cname from ~~Accounts~~ 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'.

Subqueries or inner queries which can work independently ~~of~~ without needing anything ~~is called~~ from outer query is called Independent subquery.

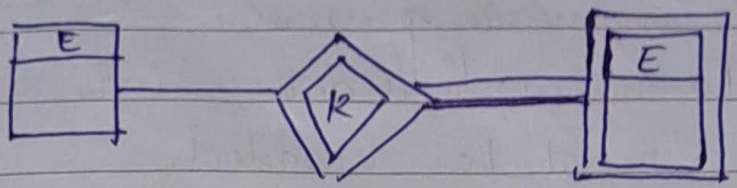
↓opposite

Correlated subquery.

↳needs value from outer query.

21/10/24/Mon.

continued here from (17/10/24 (Thu)



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

(solution put in NUMS)

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. We 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.

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~~

relation ← attribute

• Dept – dno, dname, ~~dhead~~ ~~dmanager~~, ~~startdate~~, dloca,

~~Employee~~

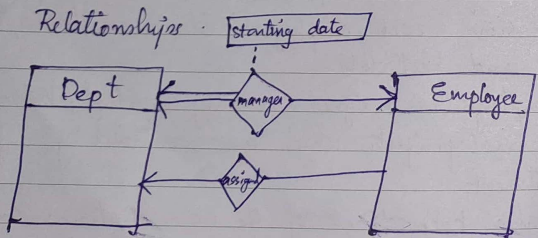
• Employee – eid, empname, addr, salary, gender, dob, ~~dno~~

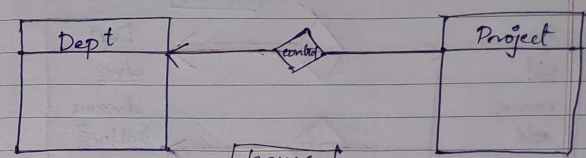
• Project – pno, pname, ploc.

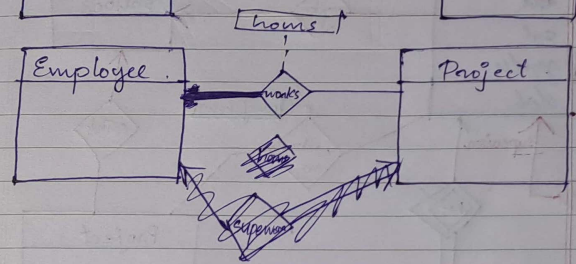
• Dependence – depname, dob, relation.

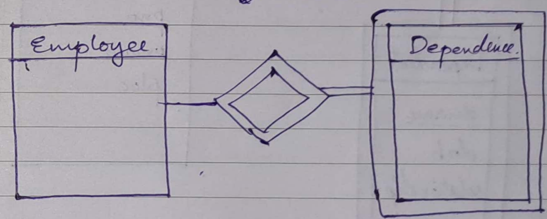
Relationships ·

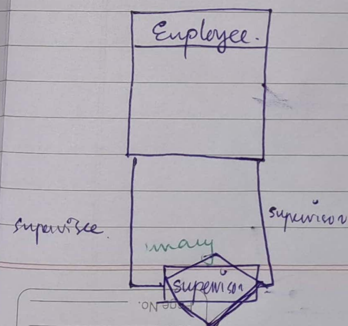
1.

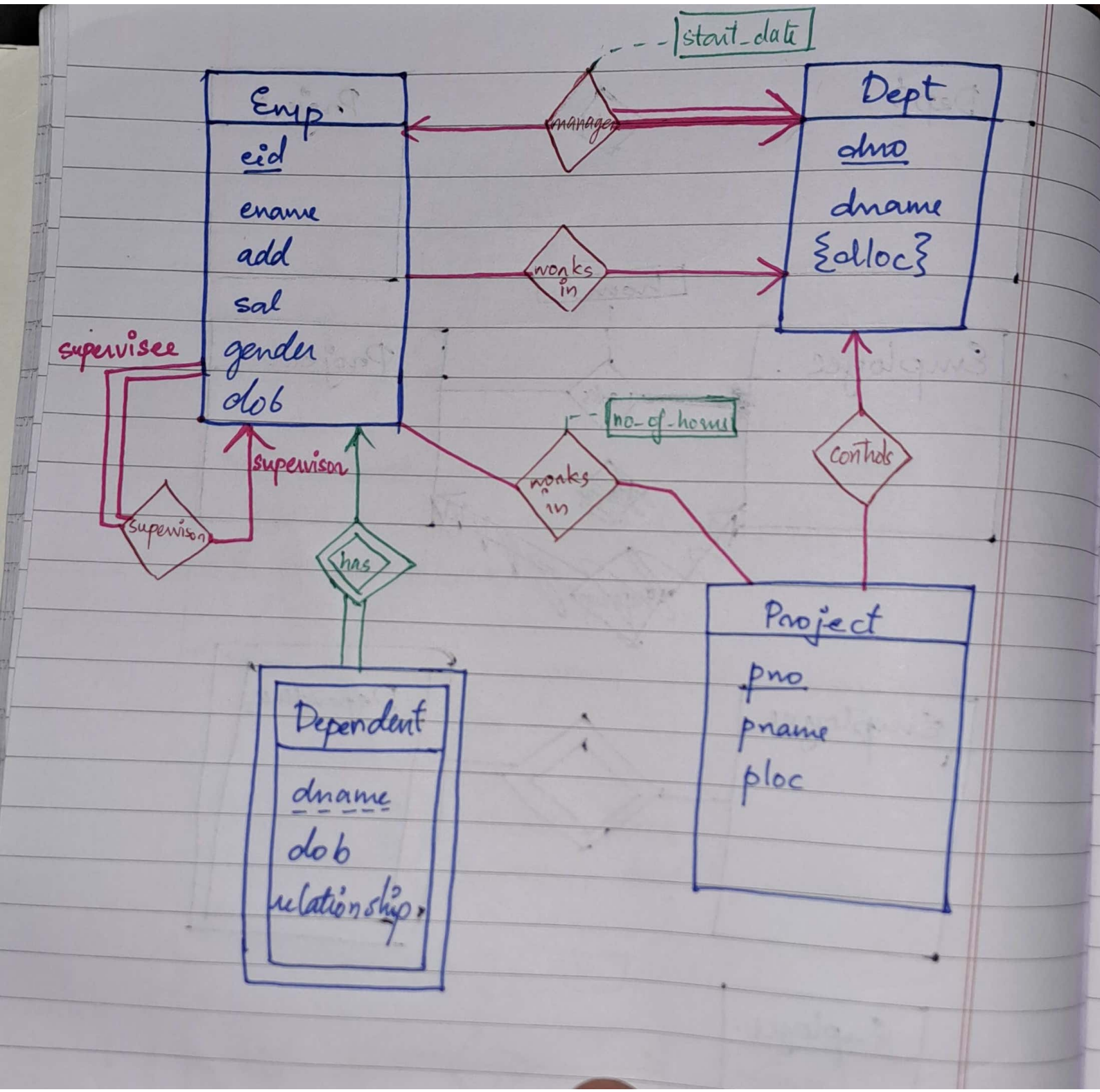








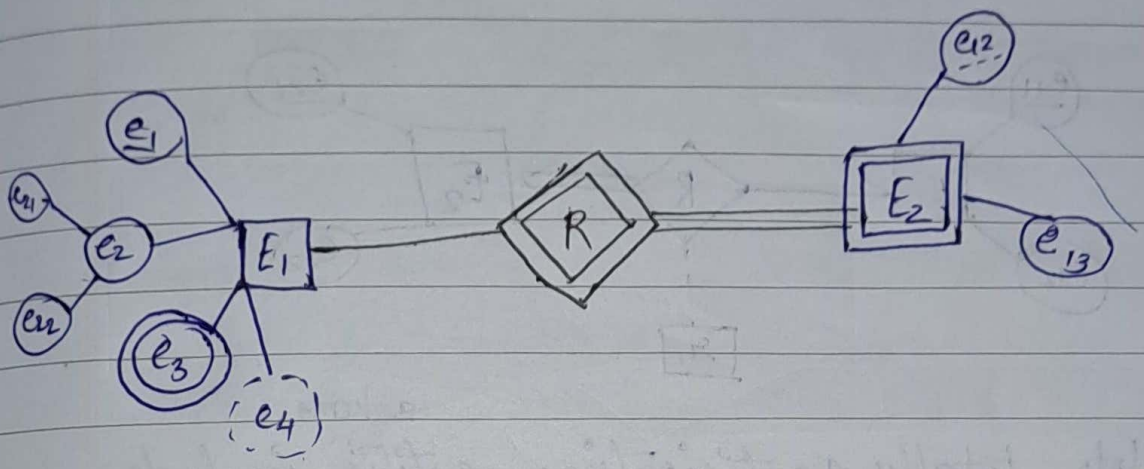



start_date
Emp
eid
ename
add
sal
gender
dob
manages
Dept
dno
dname
{dloc}
works in
supervisee
supervison
supervison
no_of_hours
works in
controls
has
Project
pno
pname
ploc
Dependent
dname
dob
relationship

24/10/24 | Thursday.

ER to Relational Mapping:

1. 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

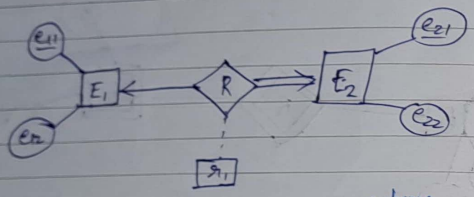
components of composite attribute e_2.

E_1 ($\underline{e_1}$, e_{21}, e_{22})

E_2 ($\underline{e_1}$, e_{12}, e_{13})

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

2. Mapping 1-1 Relationship:



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

▷ The descriptive attributes, if present, should be added to totally participating entity's schema.

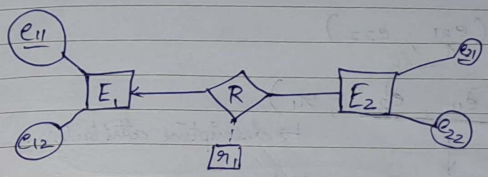
$E_1(\underline{e_{11}}, e_{12})$ → partially participating entity.

$E_2(\underline{e_{21}}, e_{22}, e_{11}, r_1)$ → total participation entity.

↓

descriptive attribute.

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



Same a previous (1-1 relationship).

Total participation == Many side.

Partial participation == One side.

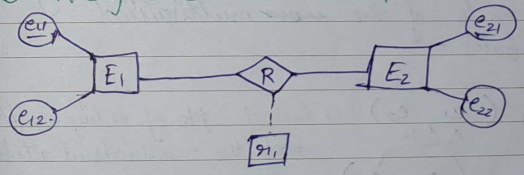
$E_1(\underline{e_{11}}, e_{12})$ → one.

$E_2(\underline{e_{21}}, e_{22}, e_{11}, r_1)$ → many.

↓

descriptive attribute.

4. Mapping M-M relationship:



For each M-M relationships, add new table schema.

E_1 (e_{11}, e_{12})

fk E_2 (e_{21}, e_{22})

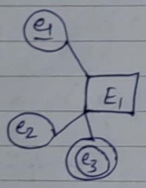
R (e_{11}, e_{21}, a_1)

↳ descriptive attribute.

↓

participating entities' primary keys.

5. Multivalued Attributes

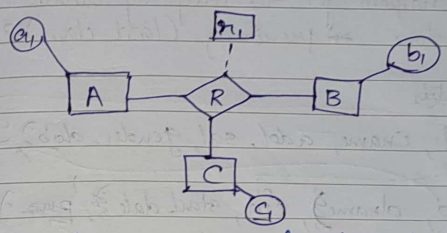


We create a new relation corresponding to each of the ~~new~~ multivalued attributes.

E_1 (e_1, e_2) ◦ include pk of entity as fk.

$E_1\_e_3$ (e_3, e_1) ◦ the multivalued attribute with pk becomes the pk of the relation

6. Relation's degree > 2.



create new table for the relation with degree > 2.

Include pks of all the participating entities as foreign key & merge them & use as pk.

Include descriptive attributes as well.

A (a_1)

B (b_1)

C (c_1)

R (a_1, b_1, c_1, r_1)

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

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

Schema

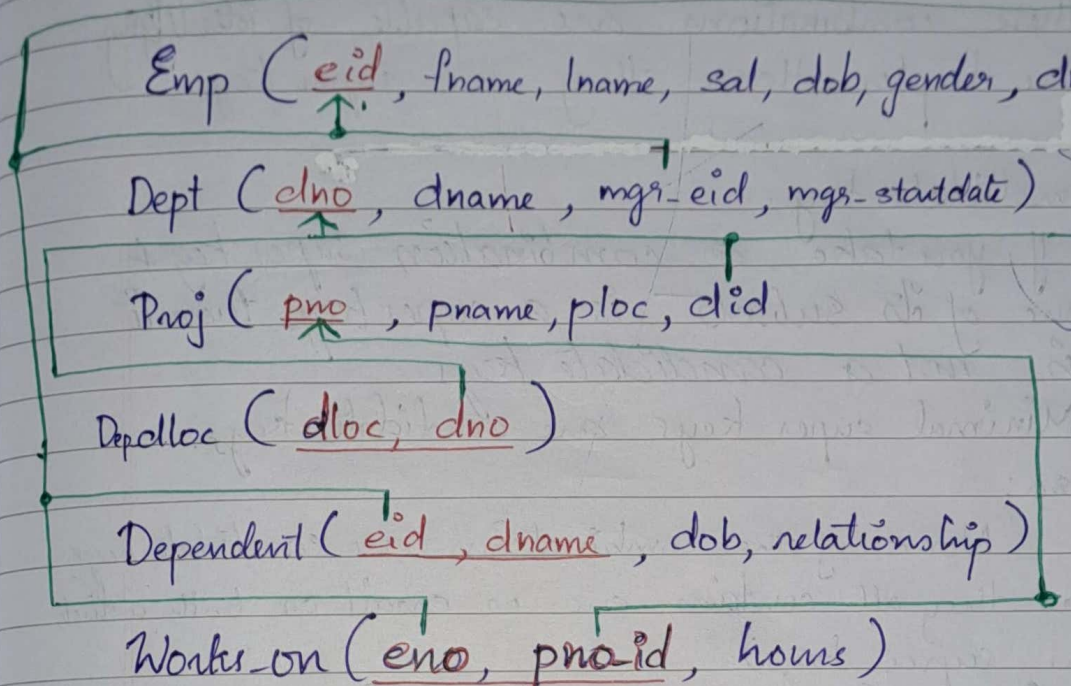
Q. Schema diagram for the company database question done previously (last class)?

A:

Regular entities

25/10/24/Friday.

(Tables need not have same names as entities)



Procedure of selecting primary keys:

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

eg: Emp (eno, ename, email, age)

Super keys:

{eno}, {email}

{eno, ename}, {eno, email}, {eno, age}

{ename, email}, {email, age}

{eno, ename, email}, {eno, email, age}

{eno, 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 keys.

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 {email} are the candidate keys.

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

Here {eno} is chosen.

Extended ER (EER) features:

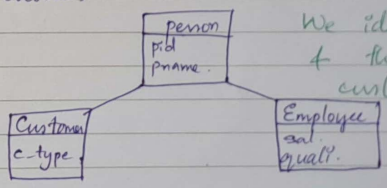
Certain features of ~~is~~ OOR such as inheritance, polymorphism etc cannot be displayed using our normal ERD. So we ~~extend~~ included them in EER.

The features are:

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

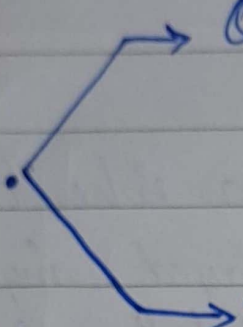
1. Generalisation / Specialization:

- 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 hostlers.

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

Attributes of person are inherited by customer & employee.