DBMS

4-1-24 Criteria to define database:

1. Collection of interrelated data
2. Collection of programs that allows access to the stored data

DBMS → 1. Construct Database
2. Store data
3. Retrive data
4. Manipulate / Modify the data

Purpose of DBMS (Drawbacks of using file management system)

1. Data Redundancy & inconsistency: Ineffective use of storage and inconsistent view of data
2. Data Isolation: Multiple files and different format
3. Data Integrity:

DB1: ☐ Instructor (___, dept_name)

DB2: ☐ Departments (dept_name, ___)

Instructor should have valid dept_name which is present in the departments database. Integrity constraints become buried in code rather than being stated explicitly. It is also hard to add new or modify current ones.

4. Atomicity: Failures may leave the database in an inconsistent state with partial updates carried out. If a task is initiated it should occur completly or must be as though the task never occurred.
5. Concurrent Anomalies: If multiple users are given access to data, the database might be left in a state where some transactions never occured.
6. Security: Should show users only the authorized data.

'relation'

Instructor Table

ID	Name	Salary	Dept
123	ABC	xx	EEE

'tuple'

columns are also referred to as 'attributes'

n-tuples

instance: data in table at any given time

schema: name of relation and set of attributes in relation: r(R)

⇒ Instructor(ID, Name, Salary, Dept)

domain: set of all attributes' possible values.

eg: domain of ID = {123}

null ← unknown (at time of insertion left blank)

the value doesn't exist

↳ default value

Attribute values → let relation have n attributes

R = {A1, ... An}

Super key is the set of one or more collective attributes that can recognise a unique tuple in the Database.

No two tuples can have same value for all attributes in a database

In Instructor,

SK1 = {ID}

SK2 = {name, dept name}

assuming no two instructors have same name in a dept.

* Any superset of a superkey will also be a superkey.

But it is required to have a minimal superkey which is designated as the candidate key.

If none of the proper subset of a superkey is also a superkey, then only it can be a candidate key.

Say, $SK_3 = \{ID, Salary\}$

then it cannot be a candidate key since $\{ID\}$ is also a superkey.

Out of candidate keys any can be chosen as primary key as per requirement

$CK = \{SK_1, SK_2\}$

Say, $\{ID\}$ is chosen as primary key then when schema is written, ID is written first.

Instructor(ID, name, dept_name, salary)

Student (ID, name, dept_name, total_creds)

department (dept_name, location, budget)

takes (ID, courseID, sectionID, semester, year, grade)

course (courseID, Title, dept_name, credits)

section (courseID, sectionID, semester, year, building, room, timeslot)

prerequisite (courseID, prerequisite_ID)

teaches (ID, courseID, sectionID, semester, year)

[---- is primary key]

18-1-24

No two tuples can have same value for set of primary keys.

Foreign key: instructor(ID, name, dept_name, salary) referencing referenced department(dept_name, location, budget)

dept_name of instructor is primary key of department hence dept_name of instructor should be foreign key from instructor to department.

Any attribute 'a' of a relation 'r' is said to be a foreign key if it appears as a primary key in another relation 's'.

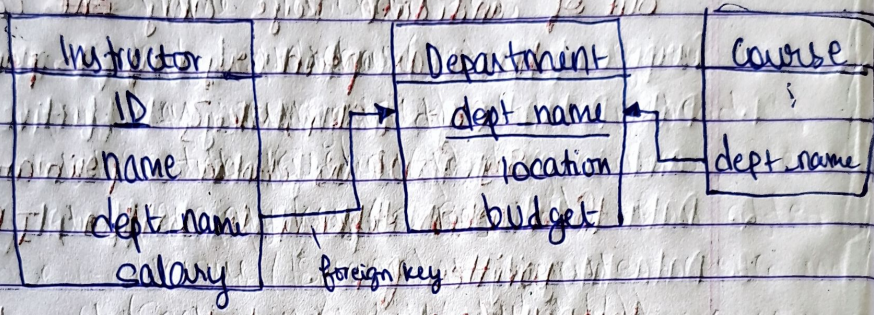
Course (course_ID, title, dept_name, credits)
Again here dept_name is foreign key from course to department.

r(A, B, C)
s(A, B)

* Important to create referenced relation first

here 'A' in 's' is not foreign key as 's' does not have other primary key 'C'

Schema Diagram



SQL

DDL → DML

Data Definition Language
- Define schemas
- Delete relations
- modify schemas

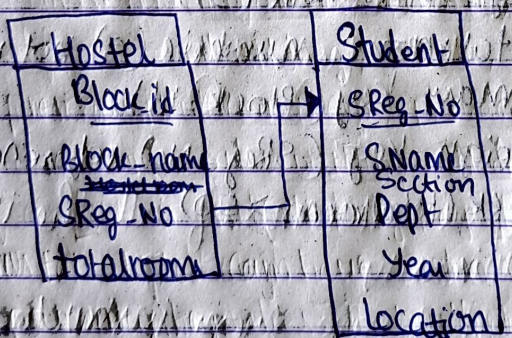
Data Manipulation Language
- insert records
- retrieve records
- Modify records/delete

Integrity Constraints
rules which database adheres to.
eg: ID of instructor must be non-NULL & distinct
budget of department has to be non-negative
dept_name of instructor must be a valid dept_name present in department table.

Create table <table_name> (A1 D1, A2 D2 ...,
<IC1>, <IC2>);

* Primary key Integrity constraint
primary key ($A_1 A_2 \ldots A_n$)
→ values of $A_1, A_2 \ldots A_n$ will be non-NULL & distinct
* Foreign key
foreign key ($A_1, A_2 \ldots A_i$) references S
eg: foreign key (dept_name) references department
→ attribute values of this relation must be in the attribute
value of the referenced schema.
* Candidate key
unique ($A_1, A_2 \ldots A_i$)
values will be distinct
* Not null
A_i is not null
→ values will be non-null
* Check
check (check (budget > 0))
check (semester in ('spring', 'fall'))

Q) Consider the schema diagram for the two relations as given below:



→ Consider the following integrity constraints for Hostel
Block_id is primary key, SReg_No is foreign key referencing Student, Block_name & totalrooms cannot have null values. Total rooms must have value > 50.

→ IC for Student
'SReg_No' is primary key
Sname is candidate key
Section must be 'a', 'b', 'c', 'd'

→ Enforce these IC with help of suitable DDL commands

A) create table ~~Student~~ Student (
~~Block_id~~ SReg_No numeric(3),
Sname varchar(30),
dept varchar(30),
year numeric(4),
Section varchar(1),
primary key (S_Reg_No),
unique (Sname),
Check(Section in ('a', 'b', 'c', 'd'))

ii)
create table Hostel (
Block_id numeric(3),
BlockName varchar(30) not null,
SReg_No numeric(3),
totalrooms numeric(3) not null,
primary key (Block_id),
foreign key (SReg_No) references Student,
Check (totalrooms > 50)
);

9-1-24 ① create table } 2 ways to define IC
② alter table

alter table <table_name> add constraint
primary key (...); ~~foreign~~ references S(...)
check (...); unique (...);

only one constraint can be added at a time.

alter table <table_name> modify <col_name>
(<domain>, primary key; | reference s; |
unique; | not null; | default <value>;

or

alter table <table_name> drop constraint <constraint_name>;
(cant change constraint; should drop & add new constraint)

① alter table instructor add constraint primkey-instr primary key(ID);

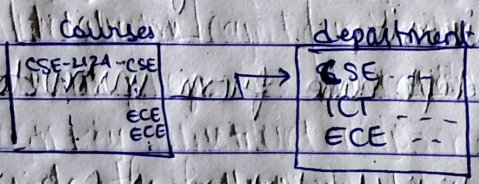
② alter table instructor drop constraint primkey instr;

③ alter table instructor add constraint pk_instr primary key(ID, Name);

(or)

alter table instructor modify ID numeric(3) primary key;

for checking foreign key constraint check if values present in table is present in referenced table



cascading delay is the delay when deleting foreign key constraint between referencing and referenced table

foreign key (dept_name) references ... on delete cascade

11-1-24

foreign key ( ) references ... on delete set null

[define in course table]

if record is deleted in department, corresponding courses values are set as NULL

child records

Foreign key can have null value. If say a department is being renamed the instead of deleting all the corresponding courses, we set their dept_name as null, then after we create new department computer science and set dept_name of all null dept_name courses to computer science

DML Commands

Retrieving the records:

select - from - where

③ ① ②

Q) find the department names of all the instructors in the university

A) select dept_name from instructors;

select distinct (dept_name) from instructors;

* remove duplicates

Q) Find the name of all instructors from CSE dept earning salary > 50 000.

<> → means not equal to

A)
```
Select instr_name
from instructors
where dept_name = CSE and
      Salary > 50000;
```

→ Say salary between 10000 & 50000

```
Select instr_name
from instructors
where dept_name = CSE and
      salary between 10000 and 50000;
```

→ salary is null;

Q) Provid the details of instructors if they were given a raise in salary by 1 %.

A)
```
Select *ID, instr_name, dept_name, salary * 1.
from instructor;
```

Q) Find name, dept_name, location of all the instructors

A)
```
Select name, dept_name, location, [illegible]
from instructor, department
where instructor.dept_name
      = department.dept_name;
```

Lab 14th Q

Q) Find the details of all the instructors who have taught some course at the university.

```
Select i.ID, iname, t.courseID
from instructor i, teaches t
where [illegible].ID = [illegible].ID
```

12-1-24

Q) Find details of all instructors who earn more than lowest paid instructor of biology department.

A)
```
select i1.ID, i1.name, i1.dept_name, i1.salary
from instructor i1, instructor i2
where i2.dept_name = 'Biology' and
      i1.salary > i2.salary
```

Q) Find the details of instructors who dept_name contain the string 'sci'

```
select *
from instructor
where dept_name like '%sci%';
```

--- sci → three char before sci
--- % → atleast three char
lower(dept_name)
upper(_)

Ordering of tuples

Q) List all instructors of CSE dept in alphabetical order of their names.

A)
```
select *
from instructor
where deptname = 'CSE' order by
      name;
```

Q) Sort all instructor in descending order of their salary: if any two have same salary then sort them in alphabetical order of their name

A)
```
select *
from instructor
order by salary desc name asc;
```

Set Operation ($\cup_{\text{union}}$, $\cap_{\text{intersect}}$, $-_{\text{except}}$)

Q) Find the course id of all the courses taught in Fall 2017 and Fall 2010.

A)
```
(select course_id
 from section
 where semester = 'Fall' and year = 2017)
union
(select course_id
 from section
 where semester = 'Fall' and year = 2010)
```

Intersect operator will remove the duplicates however intersect all is going to retain the duplicates and the ~~minimum~~ number of duplicates ~~of~~ in the result is going to be minimum of duplicates of input relation.

Union operator will remove duplicates however union all is going to retain duplicates such that there are all the duplicates present in the input relation.

Except operator removes duplicates whereas except all removes same number of duplicates

DBS
DBS except all DBS ⇒ DBS
ES (2-1) ES

Aggregate Operations

Whenever any operation has to be performed on any collection of attribute values, the following aggregate fnc of SQL can be used:

- min
- max
- avg
- sum
- count

Q) Find the avg salary of instr of 'CSE' dept. Rename the resulting attribute as AVG_SAL.

A)

```
select
avg (salary) as AVG_SAL
from ~~department~~ instructor
where dept_name = 'CSE';
```

Q) Find total no. of instructors who taught a course in the year 2018 of spring semester

A)

```
select count (distinct ID)
from teaches
where year = 2018 and
semester = 'spring';
```

15-1-21

Q) Find the enrollments for each section offered in fall 2017.

A)

```
slect count (t.ID)
from takes t
where sem = 'Fall' and year = 2017
```

group by course_id, section_id

- **Having Clause:**
  is used to state a condition that applies to a group rather than individual tuples.

Q) For each course section offered in 2017 find the average credits of all the students enrolled if the section had atleast two students.

A)
```
select avg(s.total_credits)
from takes t, student s
where t.ID = s.ID and year = 2017
group by courseID, sectionID, semester
having (count(t.ID) >= 2)
```

Q) Find the departments where instructors average salary exceeds 50000.

A)
```
select dept_name, avg(i.salary)
from instructors i
group by dept_name
having (avg(i.salary) > 50000)
```

* When a group by clause exists, the attributes that must exist in a selected clause can only be grouping attributes and/or attributes being aggregated.

Ex: Above we cannot select i.ID.

Nested Subqueries

select
from (s-p-w)
where (s-p-w) ~~set membership~~ set cardinality comparision

① Set membership
in, not in

③ Set comparison
some, all
>, <, <=, >=, =, <>

Q) Find course ID of all courses offered in both fall 2017 & spring 2018.

A)
```
(select courseID
 from section
 where semester = 'Fall' and
       year = 2017)
and courseID in
 ( select courseID
   from section
   where semester = 'Spring' and
         year = 2018)
```

can use = some or intersect

— has to be same

Q) Find the names & dept of all instr. who neither belong to physics nor finance.

A)
```
select name, dept_name
from instructors
where dept_name not in
('Physics', 'Finance');
```
set enumerated

Q) Find the names of all instructors who earn the salary greater than atleast one instructor of biology department

A)
```
select
from instructor
where salary > some
(select salary
 from instructor
 where dept_name = biology)
```

in = intersect ; = some
not in = except ; <> all

16-1-24 ② Set cardinality

- test for empty relations — exists / not exists
- test for absence of duplicate value — unique / not unique

Q) Find the id & names of all the instructors who have never taught a course in the university.

A)
```
select i.ID, i.name
from instructor i
where not exists (
    select * from teaches t
    where t.ID = i.ID);
```

The exists clause yields a value 'true' when the nested subquery yields a non-empty relation.
Similarly not exists clause returns 'True' when the nested subquery yields an empty relation.

not exists (∅) = true

Q) Find the no. of students who have taken the courses taught by the instr. whose id is '10101'.

A)
```
select count(distinct t1.ID)
from takes t1
where exists (select *
from teaches t2
where t2.ID = '10101'
```

and t2.courseID = t1.CourseID
and t2.sectionID = t1.sectionID
and t2.Semester = t1.Semester
and t2.year = t1.year);

Q) Find the ID & name of all the students who have taken all the courses offered by the biology department.

A)
```
Select S.ID, S.name
from student s
where not exists
(
(Select C.courseID
from ~~takes t,~~ course C
where ~~not exists~~ dept_name
='Biology')
except
(select t.courseID
from takes t
where t.ID = S.ID)
);
```

Unique construct returns a value true if the argument nested subquery does not contain and duplicates

Q) Find the ID and courses that were offered atmost once in 2017.

A)
```
select c.courseID, c.title
from course c
where unique(
Select S.courseID
from Section S
where S.year = 2017 and
S.Course_id = c.courseid);
```

~~where unique~~

Other method

```
where 1 >= (
    select count(s.courseId)
    ;
    ;
    );
```

19-1-24 **Subqueries in the from clause**

```
select A1, A2 ... An
from R1, R2 ... Rn
where P;
```

```
select A1, A2 ... An
from R1, R2 ... (s-f-w) Ri ... Rn
where P;
```

(s-f-w) Ri: returns a relation; Ri: name of new relation; temporary

Q) Find the departments where average instructors salary exceeds 50000.

A)
```
select DEPTAVGSAL.AVG_SAL
from (select avg(salary) AVG_SAL
      from instructor
      group by dept_name) DEPTAVGSAL
where DEPTAVGSAL.AVG_SAL > 50000;
```

Q) Find the maximum across all the departments, the sum of salaries of instructors.

A)
```
select max([redacted] SUM_SAL [redacted])
from (select [redacted] sum(salary) [redacted] SUM_SAL
      from instructor
      group by dept_name) INSTR_SUM_SALARY
[redacted]
```

with R1(V1, V2) as
(s
f
w),
R2(V1, V2...) as (s
f
w)

Temporary relation using with clause

'With' clause allows creation of temp relations such that this definition is available to the subsequent query. It is possible to use this relation any number of times in the outer query.

Q) Find all the departments that have the maximum budget.

A)
```
with DEPT_MAX_BUDGET(val) as
    (select max(budget)
    from department)
select d1.dept_name, d2.val
from department d1, DEPT_MAX_BUDGET d2
where d1.budget = d2.val;
```

Q) Find all the dept. that have total instr. sal atleast average of total instr. sal. for each dept.

A)
```
with SUMSAL(V1, dept_name) as
    (select sum(salary), dept_name
    from instructor
    group by dept_name),
AVGSAL(V2) as (
    select avg(V1)
    from SUMSAL
    group by dept_name)

select S.dept_name, S.V1
from SUMSAL S, AVGSAL a
where S.V1 >= a.V2;
```

W/o using temp relations:

```
select sum(salary), i.dept_name
from instructor i
group by dept_name
having (sum(salary)) >= (
```

```
select avg( TOTSAL)
from (select sum(salary)
      from instructor
      group by dept_name) TOTSAL
);
```

22-1-24

Q) Find all the dptms that have the least per department maximum salary.

A)
```
with MAX_SAL(V1) as
(select max(salary), dept_name
 from instructor
 groupby dept_name)
select ~~min(V1)~~ dept , val
from ~~department~~, MAXSAL
where ~~V1~~ = ( select Min(V1)
                 from MAX_SAL);
```

```
with MAX_SAL ( V1 , dept) as
( select max (salary) , dept_name
  from instructor
  group by dept_name) select dept, V1
from MAX_SAL
  where V1 = (select Min(V1) from MAX_SAL);
```

Q) Find all the course sections that have the max enrollments in fall 2010.

A)
```
with COUNT( C, S, Sem, year, V1) as
( select count (ID)
  from takes
  where semester = 'Fall' and year = 2010
  groupby courseID, sectionID, semester, year)
select C, S, Sem, year
from COUNT
where val = ( MAX(V1 from COUNT))
```

Scalar Subqueries

Queries that return a single tuple with a single attribute, this value can be utilized in any 'select from where' expression where a single value is expected

Q) And the dept names along with the count of instructors in each department.

A)

```
Select dept_name, (select count(ID)
    from instructor i where i.dept_
    name = d.dept_name)
  from department d
```

Q) Consider a relation grade_points (grade, points) with grade as primary key which provides a conversion from letter grade to an integer value. Using this list the ID & GPA for each student.

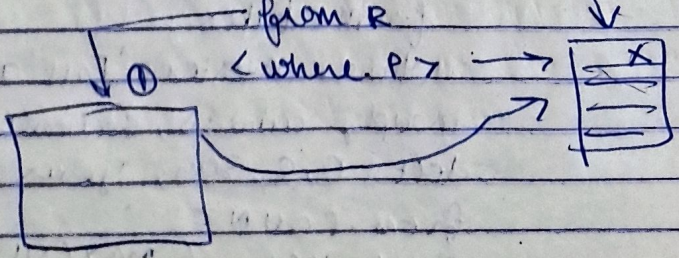
A)

```
                                    / sum(c.credits)
                                    GPA
select ID, (select sum(c.credits * gp.points)
    from takes t, course c, grade_points gp
    where t.ID = c.ID, t.courseID
      = c.courseID, t.grade = gp.grade)
    from student s.
```

Modifications to the DB

* delete

delete
from R
<where P>



Q) delete all the instr records associated with watson building.

A)
```
delete
from instructor
where dept_name in (
  select dept_name
  from department
  where location = 'watson');
```

Q) Delete all the courses that have never been offered.
```
delete
from course
where course_ID not in (
  select s.course_ID
  from section s
)
```

Update

```
update R
set A = val
< where P >
```

```
case
when P1 then res1
when P2 then res2
  :
else rem
end
```

Q) Update the salary of instructors such that instructors who earn salary above 10k recieve a 3% raise whereas other recieve 5% raise.

A)
```
update instructor
set salary = ( case
when salary > 100000 then
  salary * 1.03
else salary * 1.05
  end)
```

Q) Update the total credits attribute for each student to sum of the credits of the courses which he has completed.

A)

```
update
  student
set total_credits = (
  select sum(credits)
  from takes t, course c
  where t.ID = student.ID and
    t.grade <> 'F' and t.grade
  is not null and t.course_id =
  c.course_id)
```

23-1-24 Insert

↳ insert a tuple,
Set of tuples - query.

Q) For each student of the music department who has obtained more than 100 credit hours insert a record into the instructor table with a salary of 15,000.

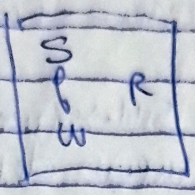
A)

```
insert into instructor(
  select ID, name, dept_name, 15000
  from student
  where dept_name = 'Music'
  and total_credits > 100);
```

Views → Virtual Relations

⇒ with R(a,b) as (

)



Views are virtual relations that are created apart from the relations defined as part of the conceptual model.

Advantages:

① Restrict/Hide certain information/attributes from the table.

Example: a view can be defined to contain only ID, name of instr. thus preventing certain users from accessing the salary information from the ~~student~~ instructor relation.

② Views are not precomputed.

Def: A view in SQL contains 'name of the view', 'query', which is referred to as the view definition

⇒ Create view v as (query);

Q) Create a view with the name physics fall 2010 that contains all the course sections offered by the physics dept for fall 2010.

A)

```
create view Physics-Fall-2010 as
( select C.course_id, S.sec_id, S.building, S.room_number
  from course C, sections S
  where C.dept_name = 'Physics' and
  C.course_id = S.course_id and
  S.Semester = 'Fall' and S.year = 2010
```

Only when the subsequent queries access the view, the view def. shall be fetched and the corresponding query or view def" shall be executed by SQL

SQL ~~stores~~ will only store the view def" not compute the view def. defined as part of the view.

Q) Use the above view to find all the course sections of Watson Building.

A)

~~Select C.course_id, S.sec_id, S.building, S.room_number~~

```
Select
from Physics-Fall-2010
where building = 'watson'
```

~~Viii~~ Materialized Views

Sometimes it is important to persist the view def" so as to prevent the computation of the view.

25-1-21

~~Consider a view department~~

Materialized View Maintenance

Consider a view dept_instr_sal which computes the sum of salary of instructors of each department. Whenever the salaries of instructors gets updated, this materialized view has to be recomputed so that it provides the updated information. This process of updating a mat. view is known as materialized view maintenance.

There are three strategies used for this:

① Immediate maintenance: as soon as any underlying relations are updated the materialized view is immediately updated

② Lazy Maintenance: Update the mat^er view whenever there is some query that tries to refer to this view.

③ Periodical maintenance: The materialized views are updated at regular intervals in the database.

View Updates

Q) Consider a view instructor_info which is created to store the id, name and location of the instructor. Write query for this view.

A)
```
create view instructor_info as
    ( select ID, name, building
      from instructor i, department d
      where i.dept_name = d.dept_name
    );
```

Q) Consider inserting a tuple into the above view where the tuple is (12345, 'ABC', 'Watson')

A) If there is Whenever the above record is inserted a tuple (12345, 'ABC', null, null) has to be inserted into the instructor table along with (null, 'watson', null) into the department table.

The above insert operations are going to be successful only in the instructor table provided there is not null constraint on salary attribute. However the second tuple can never be inserted into the department table as a null value is being provided for primary key. Hence sql does not provide any mechanism to update the view.

Only under the following circumstances sql allows view updates. Whenever ① only a single relation is part of ~~from~~ the from clause, ② The select clause ~~does not~~ does not have attr. that ~~do not~~ have not null IC and ③ the attrib must not be primary key.

The select clause does not contain aggregation operation or distinct specification on the attributes.

Join Operations

Q) Find all the students who have taken some course at the university.

A)
```
select s.name, t.course_id
from student s, takes t
where s.ID = t.ID;
```

Most often whenever there are multiple rel's, the where clause equates the attributes that are common to the specified relations.

The above query can be rewritten as:

```
select name, course_id
from student natural join takes
```

Natural join operation returns a single relation whose tuples are having same value on all the attributes that are common to both the relations

~~Find~~

Q) Find the names of students along with title of courses that they have taken.

A)
```
select name, title
from (student natural join takes) ~~natural~~
~~join~~ , course
where takes.course_id = course.course_id;
```

29-1-24

Q) Find the name, course_id and title of courses taken by the student.

A)
```
select student.name, takes.course_id, c.title
from (student natural join takes), course
where c.course_id = takes.course_id;
```

forms a relation but we cannot use alias for this

↓

can use alias for individual relations

* Whenever natural join operator is SQL does not permit to use alias for the relation that is output from the natural join.

* Although indiv. tables (input of natural join operator) can have table aliases, it is only

possible to refer to non-joining attributes using these.

From Q1 only the students who have taken one or more courses will be output because for the student who has not taken any course there will be no match from the takes table.

join ... using

The using clause with the join operator is a ~~more~~ specific case of natural join that matches only the attributes specified in the using clause.

Q2 can be written as:

```
select student_names, course_id, title
from (student natural join takes) join
course using (course_id);
```

* Any attri. that is specified in the using clause must be common to both relations.
* It is not possible to use a alias for the joining attribute.
* The attributes listed in using clause will appearing before all the other attributes in the output schema.

Q) Rewrite query1 using 'join... using' operator

A)
```
select ID, name, course_id
from student join takes using(ID);
```

Join ... on

The 'on' clause with the join operator shall ~~equate~~ allow any general predicate to be specified unlike 'using' clause which requires only common joining attributes

Query 1 can be rewritten as

```
select *
student s join takes t on s.ID = t.ID
```

* With the help of 'on' clause, it is possible to seperate the joining conditions from the rest of the predicates.

* It is possible to use the on clause to match the required attributes even when they do not have the same identifier/names.

Outer joins

generally tuples will be lost whenever natural join operators are used.

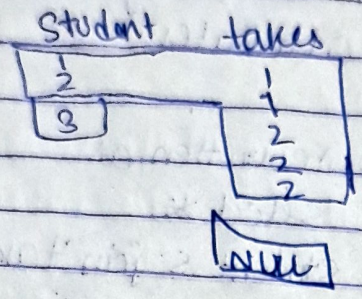
However at times it is required to preserve the tuples from either of the relations or both the relations being joined. In this case 3 diff versions of outer join operator may be used.

→ 1. Left Outer join - preserves the tuples from the input rel^n to the left of join operatr.

Q1 rewritten as:

Select *
from student natural left outer join takes;

After the natural join computation, the attr. shall have null value for the attr. of second input rel^n



30-1-24

→ 2. Right outer join preserves all the tuples in the relation that appears to the right of join operator.
The above query is now:

select * takes
from ~~student~~ natural right outer join student;

→ 3. Full outer join preserves all the tuples present in ~~both~~ either of the relations.

Q) Display the details of courses that are taken by all the students of CSE dept. for Spring 2017. Ensure all the course sections of Spring 2017 is displayed in the result.

A) Students of CSE 30 course.

```
select *
from
  (select *
   from student
   where dept_name = 'CSE')
  natural full outer join
  ( select *
    from takes
    where semester = 'Spring' and
          year = 2017 ));
```

The first input relation results in all students of comp. sci. dept, second results in courses taken by students in spring 2017.
A natural join ensures the tuples of two relations are joined using the common attr. ID.
The output schema shall have all attrib. of student & takes relations.

Q) Display ID & the no. of sections taught by the instructor using
① outer join operator
② scalar subquery.

A) ①
```
select count (section_id)
from (instructor natural left outer
  join teaches)
groupby ID;
```

②
```
select ID, (select count (section
_id) from teaches t
  where t.ID = i.ID)
from instructor i;
```

Q) Display the dept name along with the count of instructors in each dept. Using outer join operation & scalar subquery.

A)

```
select ID, count(sec_ID)
from department left outer join
    Instructor
    group by ID
```

```
Select ID,
(select count
    (sec ID)
from instr
where t.ID
= i.ID) from
department
d;
```

5-2-24

Assertions

Q) At any given point of time the total_creds attr of student table must be equal to sum of the credits of the courses which the student has completed successfully.

A)

```
Create assertion stu_tot_credits_const as (
    not exists
    (select
        from student
        where tot_credits <>
        ( select sum(credits)
            from takes natural join course
            where student.ID = takes.ID
            and takes.grade <> 'F' and
            takes grade is not null
    ))
```

Whenever the above assertion is created the database tests for its validity. For any other future modifications of the table if the tuples do not satisfy the above assertion, the tuple will not be inserted

due to violation.

Type Conversion:
The cast function converts an expression e to the specified type t. For example:

Q) Sort all the instr. in ascending order of ID value interpreted numerically.

A):
```
cast (ID as numeric(12)) ID_num
from instructor
order by ID;
```

User defined datatypes
There are two diff ways in which UDD can be created using sql.

(I) Create type

```
Create type INR as numeric(10,2);
```

This will create a UDD INR whose datatype is numeric. Once created, it can be used in place of any valid datatype.

```
Create table Department (
        budget INR
)
```

(II) Create domain.

```
Create domain Yearly_Salary
numeric(10,2) not null,
```

The UDD created using domain will also allow the specification of integrity constraints such as not null as defined for yearly_salary type.

(ii) Create a domain Sem_type whose values are ~~1, 2, 3, 4~~ I, II, III, IV

create domain sem_type varchar(5)
constraint sem-check check (value in ('I', 'II', 'III', 'IV'))

Create Table Like

(i) To temporarily create the schema of an existing table, create table like extension is used.

Create table table_ins like instructor;

The above statement copies the schema of instructor into a temporary schema identified as temp_instr.

(ii) It is also possible to copy the results of a query into a temp relation using this extension.

create table with data extension.
The query is specified after create table clause.

create table temp_instr
(select * from instructor)
with data;

The relation returned by the query is copied into a temp relation identified

as temp_instr. The attributes & their datatypes will be inferred from those of the query result.

DATABASE DESIGN

(I) User requirements
(II) Conceptual - analysis

data model (relation)

ER dia
→ incompleteness
→ redundancy

(III) db implementation
① Logical design

8/2/24 **DATA ABSTRACTION**

Hiding the complexity of the data from different database users, making it easier to interact with the database system.

db admins

① Physical level describes how the data is stored in the dbs.
It describes how the data is stored, data str. & file organization in detail

application programmers

② Conceptual level: descr. what data is stored, and rel^n that exist among those data. It describes the db through comparatively simpler structures than those at the physical level.
Ex: schema can be used to represent the information in the entire db.

users

③ View level: All the info in the db need not be provided to all the diff users of the system. In this regard, only certain parts of the db or tables can be created

and provided to the users.

Components of DBS

* Storage manager
* Query Processing engine
* Transaction Manager

Storage Manager:
It is responsible for converting DML commands into low level file system commands. It provides an interface between the low level data and queries that are submitted to the db. The diff subcomponent are:
① Authorization & Integrity Manager: responsible for testing whether IC are satisfied & whether the users are authorized to access the data.
② File manager: responsible for managing space allocated for the db on the disk.
③ Buffer manager: responsible for movement of data between disk & main memory

DS used by SM include data files, - actual data stored on the disk, data dict/- stores the metadata (data about data) about data stored in files.

Query Proce Engine:
① DDL interpreter - responsible for interpreting the DDL commands and storing it in data dictionary.
② DML compiler - resp for translating

DML commands into multiple evaluation plans for execution by query evaluation engine.
A query optimizer selects one of the eval plans that has least evaluation cost.
Query eval engine executes the eval plan submitted to it.

Transaction Manager:

① Recovery manager - resp for ensuring rollback operations are performed on the event of transaction failure. This will leave the db in a consistent state.

② Concurrency & control manager: is resp for resolving the conflicts due to access by multi users concurrently.

Database users

- Administrator - sophisticated users
- general users
- application programmers

General users → To use the UI provided by app^n programmer to retrieve or modify the data.

App^n progr^{mrs} → develop the application using any high level progg^n lang.

Admin → resp^n for allocat^n of space on the disk, maintaing data dict. & granting authorization to diff category of users

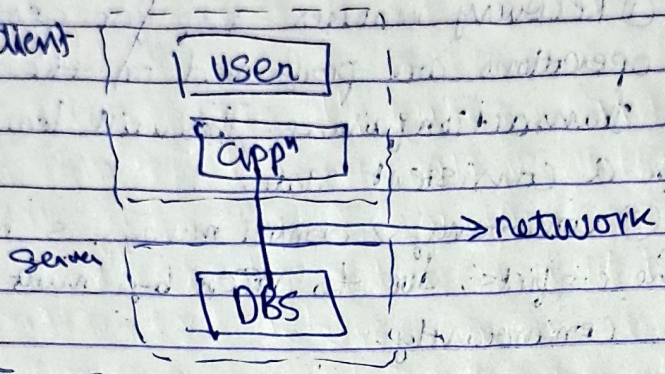
Sophisticated users → Unlike app^n Progmmers they directly interact w/ db through command line interfaces provided by commercial db vendors

DB Architecture

generally a db is installed on a sys & accessed by diff users remotely over a network.

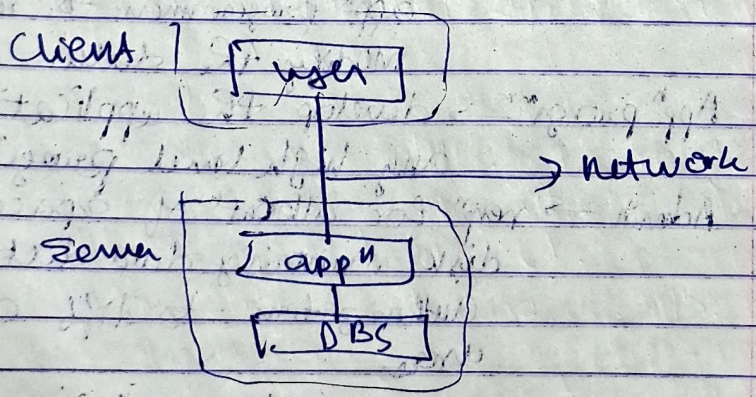
Two tier

Here the client directly invokes the db calls which is transferred to the server through a remote machine/network.



Three Tier

The client does not invoke db calls directly. The client only acts as a front end wherein a forms interface/GUI is provided to the user. This information is then passed over the network to the server that translates it to suitable db calls



10/2/21

Entity-Relationship Model

~~Three~~ ER Model facilitates db design process by allowing the specification of schema that represents the overall logical structure of the db.

The three important aspects of ER Model are

1) Entity set
2) Relationship sets
3) Attribute

Entity sets

Any object or concept present in the real world can be considered as an entity. A collection of entities of the same type is reffered to as entity set.

Ex: students, instr (objects)
course (concept)
are some of the entities in University mini-world.

Similarly in a company:
employee, dependent (objects)
works-on, project (concepts)

Relationship sets

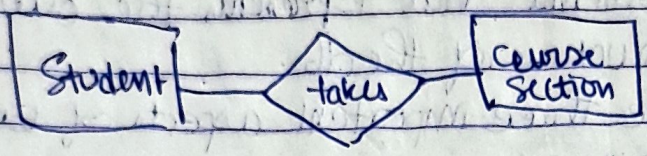
An association among several Entities is described as a relationship.

A student is associated with an instructor through an advisor relation. Similary a student & course section can be associated through an enrollment or takes relation.

The number of entities participating in a relation is reffered to as the degree of rel'ship Set.

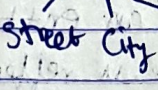
Above ex are of degree 2 and are called binary relationship sets

In ER Model an entity is represented using a rectangle and a relationship set using a diamond.
The following notation is used to represent example in ER model



Attribute sets

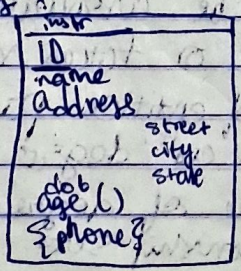
Categories of attribute: Simple, composite
→ A simple attr. cannot be further subdivided into parts. Ex: Instr. ID, Stu_ID.
Comp attr can be further subdivided into multiple parts. Ex: address



→ Single valued attr is an attr that can take a single value.

If an instr has multiple phone numbers, it is an ex of multivalued attr.
They are represented by enclosing the attr in {}.

Derived attr: if the value of an attribute can be inferred from an existing attr. A der. attr. will be appended w/ ()
An instr. entity w/ all these diff attr. can be represented as:



Mapping Cardinalities and Participation Constraints

The no. of entities associated with another entity set is reffered to as mapping Cardinality / Card. ratio.
Ex: an instr. can advice 0 or more Student s. Similarly a student can be advised by exactly one instructor.

One to Many

An entity of set A will be associated with 0 or more entities of set B.
An entity in B is assoc. w/ atmost 1 entity of set A.

One to one

An entity of set A is associated w/ 0,1 entity of set B.
An entity of set B is assoc. w/ atmost 1 entity of set A.

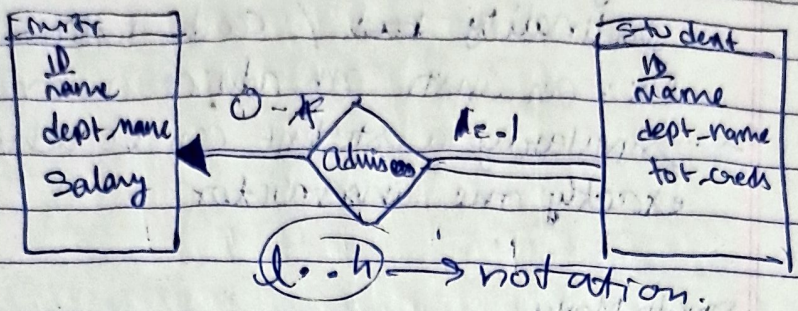
Many to one

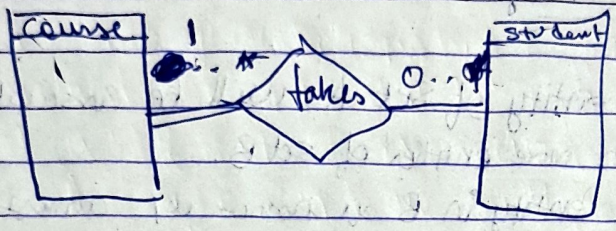
entity set A assoc. w/ atmost 1 entity of set B
an entity of set B is assoc. with 0 or more entities of set A.

Many to Many

An entity of set A is assoc. with 0 or more entities of set B
and vice-versa.

In the ER notation, an arrow is directed towards the entity that is at the 'one' end of the mapping cardinality.





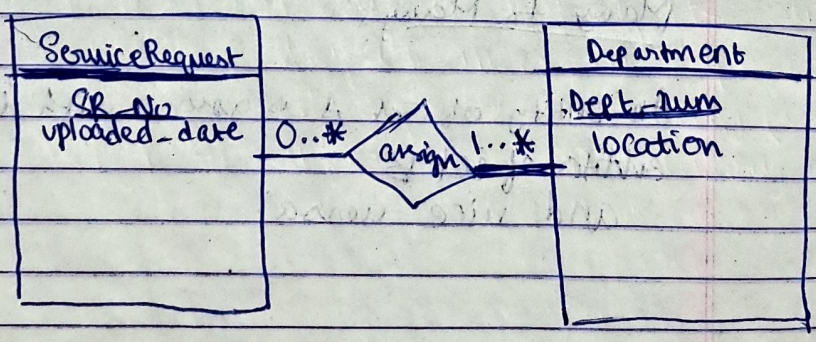
A servicerequest has attributes, service req no, the complaint, and the date on which it was uploaded. A service req need not be assigned to any dept but if assigned, it can be to multi depts.

A dept is characterized by unique dept no. and its location. There can be no dept which is not resolving any service request. ⑧ Identify the entities attr and the rel" shp sets & represent using ER notation.

Dept mapping cardinality and part" constraint.

Also indicate min & max cardinalities

12/02/24



Total Participation of an Entity (=)

an entity a has a total participation in the relationship set when every instance is participating in the relationship set.

The min cardinality then for such an entity is going to be 1.

Total participation is also denoted as a double line that emerges from the set and moves to the totally participating entity.

Also called existential dependency. for the following reasons:

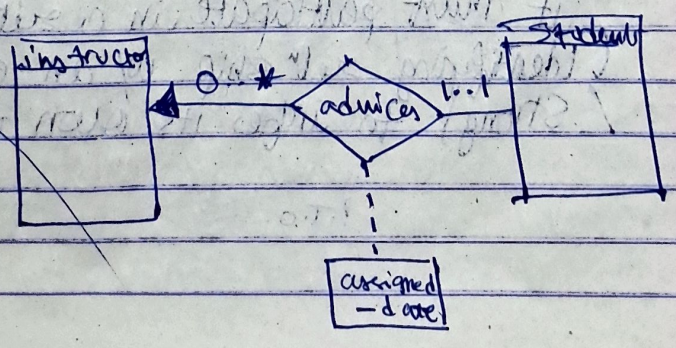
① A student can exist in this relationship set only if he has an advisor.

Descriptive Attributes

desc. attr. are the attr. which are part of the relationship set. They are enclosed in a rectangle and are connected to the rel"ship set via a dashed line. There can be multiple such attr. for a rel" ship set.

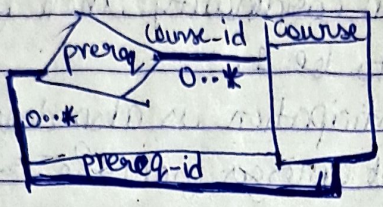
Example: If it is req to keep track on which date an instr started advising a part^c student, the attr assigned_date cannot be placed in instr. or student.

It can more appropriately be a descriptive attr and linked to advises rel" set.



Recursive Relationships

Consider the requirement, a course can have several prerequisite courses



In a binary relationship when the entities involved are not distinct such a relationship set is referred to as recursive rel^n ship set.

Ex: A course can have multiple prerequisite courses and a course can be a prerequisite for several other courses. It can observed that ① There is only a single participating entity course, w/ diff roles. Once as a course & once as a prereq.

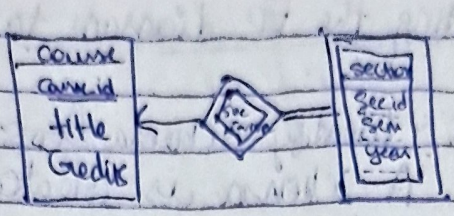
② The mapping cardinality is going to be many to many.

Weak Entities

An entity that doesnt have a key of its own is a weak entity.

It is represented with a double rectangle. As it does not have a key of its own, it must participate in a rel^{n}ship set (identifying rel^{n}ship w/ an entity (owner / strong) to infer its own primary key).

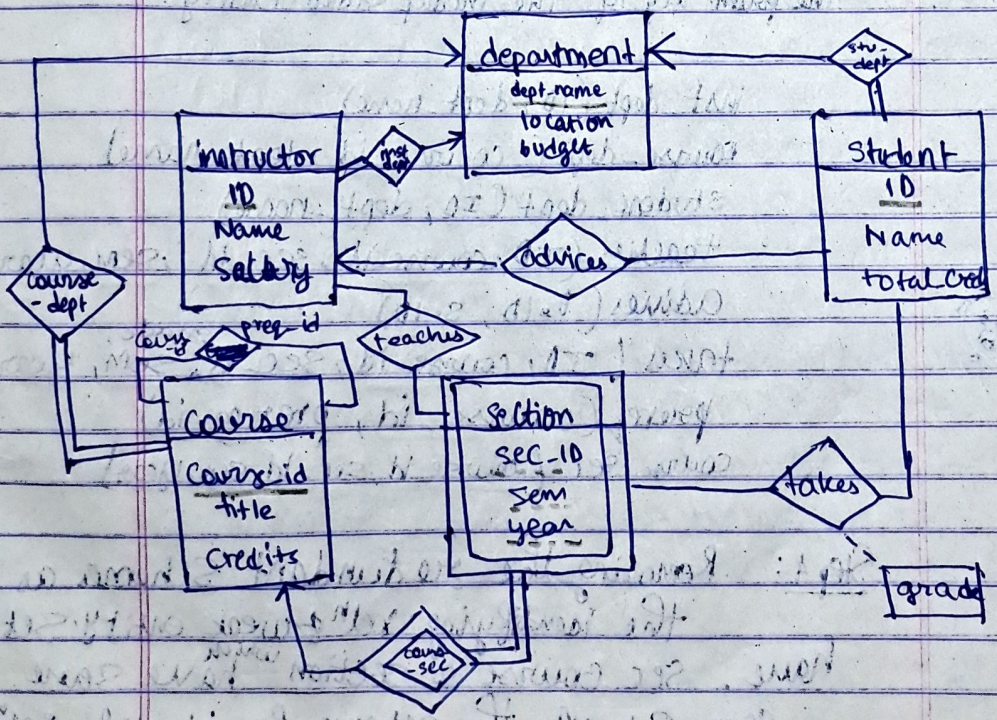
P.T.O.



The participation of weak entity must be a total participation w/ the identifying relⁿ-ship set. Here the attr set (sec-id, sem, year) which are the partial primary key are referred to as discriminant attr. and they are indicated with a dashed underline inside a weak entity.
Thus the primary key of section is the prim-ary key of course union discriminant attr.

Prim key(section) = Prim key(course) + discriminant attr.

13/2/24 **ER Diagram**



Converting the ER diagram to Schema

Step 1: Identify schema for strong entity set
A schema is created for each strong entity such that its attr are part of the corresponding schema.

department (dept_name, location, budget)
instructor (ID, name, salary)
course (course_id, title, credits)
student (ID, name, tot_credits)

Step 2: Identify schema for weak entity set

section (courseid, sec_id, sem, year)

Step 3: For any rel^{n}ship set 'R', the attr. are determined as the union of primary key participating entities. The primary key of the created schema will be the prim. key of the many sided entity.

inst_dept (ID, dept_name)
course_dept (course_id, dept_name)
student_dept (ID, dept_name)
teaches (ID, course_id, sec_id, sem, year)
advises (i.ID, s.ID)
takes (ID, course_id, sec_id, sem, year, grade)
prereq (course_id, prereq_id)
course_sec (course_id, sec_id, sem, year)

Step 4: Removing the redundant schema as the identifying rel^n & weak entity set have, sec_course & section which have same schema, the schema for identify Rel^n

ship set can be eliminated. (Preserve schema of weak entity)

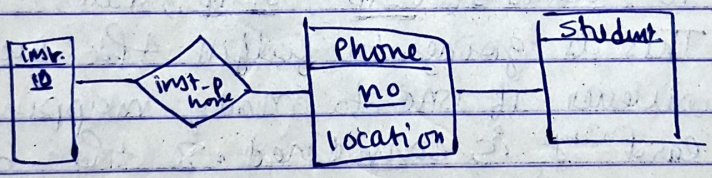
Step 5: (Merging the schemas) A many to one rel^n ship set with total participation of many sided entity can be merged w/ the schema of many sided entity.

inst_dept (ID, dept_name)
instructor (ID, name, salary, dept_name)
course_dept (—)
course (course_id, title, credits, dept_name)
Student (ID, name, tot_credits, dept_name)

* Many sided rel^n ship set must have total participation.

Design Issues

① Placement of Multivalued attr.: Consider a multi valued attr. phone no. for the entity instr.

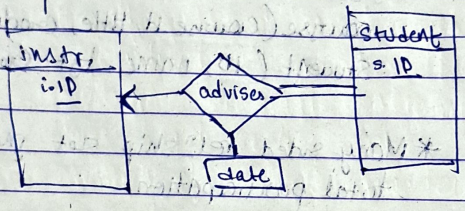


With the above design, there are following limitations:
1> In a rel^n instance, there can be several instructors who may not have any phone, in this case there will several null values for the multivalued attr.
2> If any add^n info such as loc. (office, personal/home) has to be maintained

it is not possible to capture this, as an attr. in the instr. entity. The design can be modified as ~~follows~~ above.

inst-phone (ID, phoneno)

(2) Placement of rel"ship attr: Consider the advises rel"ship set. It has a descriptive attr. date which keeps track of the date on which an instr started advising a particular student.

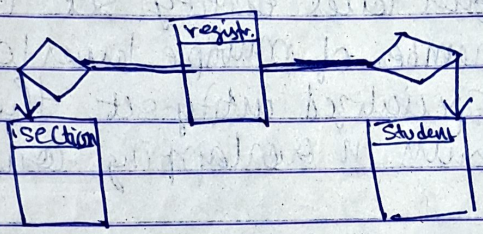


advises (i_ID, s_ID, date)

When the mapping cardinalities change to many to many, the schema changes to advises (i_ID, s_ID, date). This is going to fulfill the req. however if one to many mapping cardinality is considered & the date attr is part of student entity then the schema for student will be student (ID, tot_credits, name, date) This still fulfills the req. however it fails when mapping changes to many to many. As per the prev. design the schema for advises will be advises (i_ID, s_ID, date)

Thus it can be seen that whenever date is captured as a descriptive attr. it fulfills the req. in both the cases. Hence it is more appropriate to identify the date (which only has relevance when instr. advises student) as a descriptive attr.

(3) Identification of entity set vs Rel^{n}ship set : The enrollment of a student into a course section is identified as takes rel^{n}ship set. If the req are such that for a course taken by each student a registration record has to be maintained, then



The reg. entity set then has a total part. w/ both student & section entities. It is still going to fulfill the req as done by the takes ~~entity~~ rel^{n}ship set.

Extended ER features

Specization / Generalzn

The process of designating sub-groups within an entity set is referred to as specialization. Example - Consider person entity set, there can be entities within this entity set

who earn a salary or there can be another subgroup who are Student. Although students / employees have some attr in common, they are specialized as follows:

Person + salary (employee)
Person + tot_credits (student)

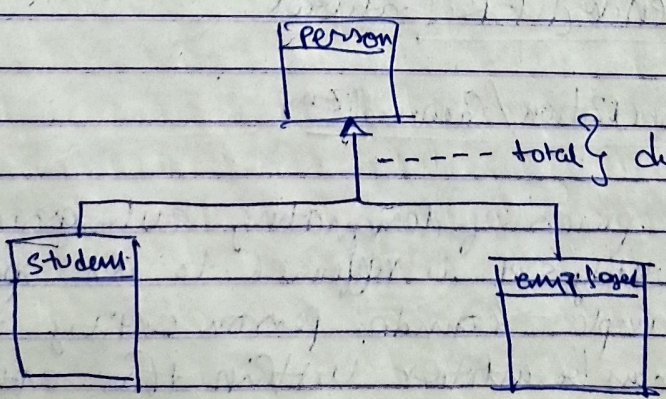
Constraints on specialization:

① Overlapping vs Disjoint:

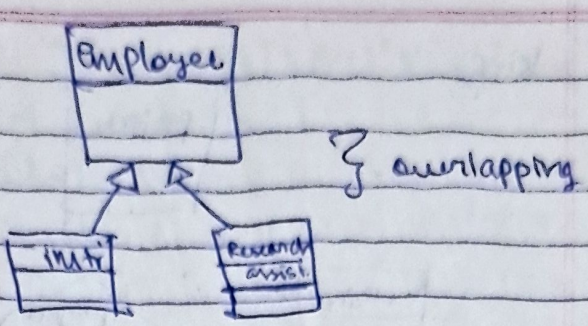
If a higher level entity set is a member of atmost one lower level or specialized entity set, then it results in a disjoint constraint. If a higher level entity set is a member of multiple lower level / specialized entity set, then it results in overlapping constraint.

② Total vs Partial:

~~If~~ a h.e.s is a part of any one of the specialized e.e. then it is total participation however in partial part[c] there can be es which dont necessarily participate in specialization.



卐



* Not specifying means partial participation.

Reduction to relation schemas:

(I) A schema is created for higher level entity set. A schema is also created for each of the lower level entity set however the attr. will only be the prim. key of the higher entity set along w/ any local attr.

person (ID, name, address)
employee (ID, salary)
student (ID, tot_credits)

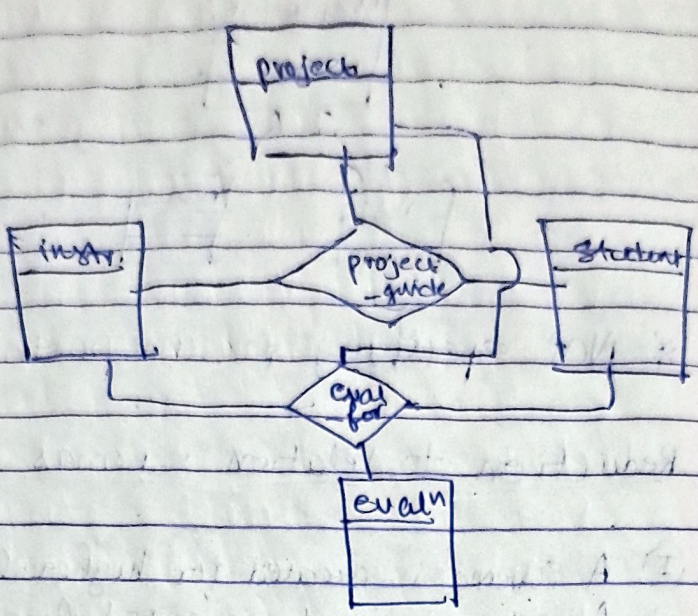
(II) Schema is created for each of the lower level ES in the following manner only if it's a total disjoint part c.

employee (ID, salary, name, address)
student (ID, tot_credits, " , ")

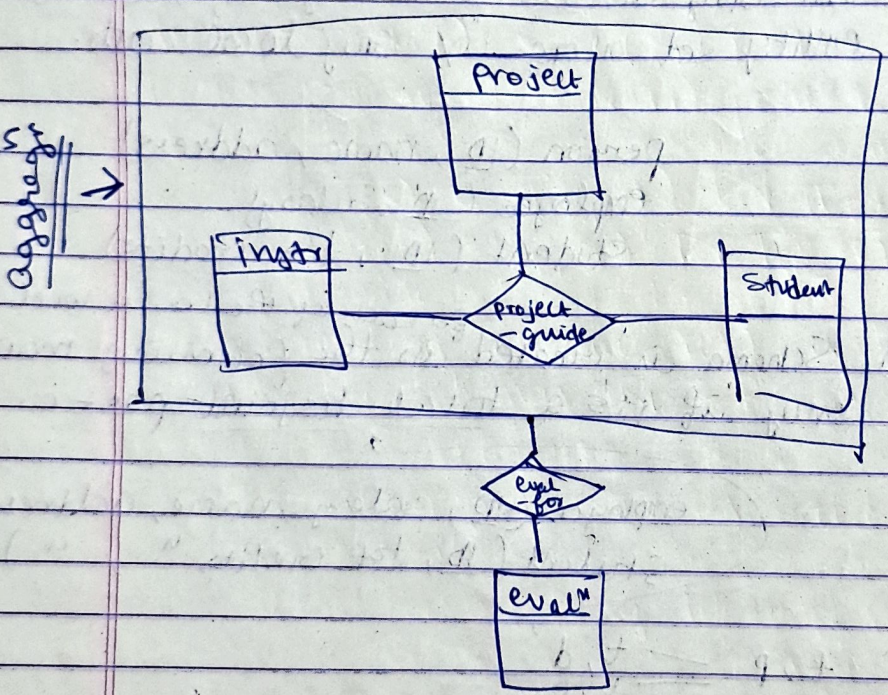
t, p — t, d
t, o
o, d — p, d
p, o

Aggregat"
Consider a ternary rel" ship set project-guide that associates each instr. with a project where multiple students are wor-

- ring.



or



If the req^n need to keep track of the day and the status of eval^n of each instr, proj, stud combin^t then an evaluation entity has to be created. It also has to be associated w/ the entities instr, project, student

Hence eval^n for is a rel^nship set w/ degree 4.

In extended ER Notation it is possible to treat project_guide (and associated entities) as a single higher level ES. This abstraction is called aggreg^n.

Now, it is possible for this aggregation or HES to part^c in eval_for rel^nship with the eval^n entity. The degree is 2.

The schema for the HES is same as schema for proj_guide.

Design Alt

(3) Larger Schemas

in_dept (ID, name, salary, dept_name, bldg, budget)

(1) Redundancy
↳ Inconsistent updates

(2) Incompleteness
↳ Cannot have dept w/o instr.

(II) Decompon^n

employee (ID, name, salary, address)

e1 (ID, name)

e2 (name, salary, address)

R: 123 ABC x XYZ
234 BCD y AXY

R1: 123 ABC
234 BCD

R2: ABC x XYZ
BCD y AXY

~~If we not join~~

join

$R \subset (e_1 \bowtie e_2)$

Lossy decomposition → other superset (cartesian product)

$R = (e_1 \bowtie e_2)$

lossless decomp

↑ criteria for

$R_1 \cup R_2 = R$

$r(R)$

$r_1(R_1)$

$r_2(R_2)$

lossy → $r(R) \subset r_1(R_1)$ join $r_2(R_2)$

lossless → $r(R) = r_1(R_1)$ join $r_2(R_2)$

$r(R) \subset \Pi_{R_1}(R) \bowtie \Pi_{R_2}(R)$

$r(R) = \Pi_{R_1}(R) \bowtie \Pi_{R_2}(R)$

if a relation schema $r(R)$ is decomposed into the schemas R_1 & R_2 such that

$R_1 \cup R_2 = R$

the decomposition is said to be lossless if

Also for a lossy decomposition

Normal Forms

First NF (1NF)

A relation schema R is said to be in first normal form if the domain of all the attr are atomic

Any real world constraint that the database application must satisfy is referred to as a funct" dependency.

$\alpha \subseteq R, \beta \subseteq R$

(1) $\alpha \to \beta$ holds on $r(R)$ (whenever ∀ pairs of tuples t_1, t_2 if $t_1[\alpha] = t_2[\alpha]$, it is also the case that $t_1[\beta] = t_2[\beta]$

antecedent → consequent

(2) if $\alpha \to \beta$ holds on all instances $r(R)$ then $\alpha \to \beta$ satisfies the entire schema

Q) Consider a rel" instance as given below, verify whether the func dependency $A \to C$, $C \to A$ holds.

A	B	C	D
a_1	b_1	c_1	d_1
a_1	b_2	c_1	d_2
a_2	b_2	c_2	d_2
a_2	b_3	c_2	d_3
a_3	b_3	c_2	d_4

A) (1) To check $A \to C$:

for all pairs of tuples t_1, t_2 where $t_1[A] = t_2[A]$, $t_1[C]$ is also $= t_2[C]$.

for the first pair it can be seen that when $t_1[A] = t_2[A]$ it is also the case that $t_1[C] = t_2[C]$. Also for the second pair.

Thus $A \to C$ holds on the instance R

(2) $C \to A$:

funct dependency does not match

By contradiction it is observed that $t_4[C] = t_5[C]$, $t_4[A] \neq t_5[A]$ and so $C \rightarrow A$ does not hold on the instance R.

Trivial Func^c Dependency

any FD of the form $\alpha \rightarrow \beta$ is a trivial FD if $\beta \subseteq \alpha$.

Ex: $A \rightarrow A$
$AB \rightarrow A$

Closure of FD

For a rel^n schema R given the set of FDies F it is possible to infer more functional dependencies from this set F. Such a set which contains all the FD inferred from F is closure of F. (F^+)

Boyce-Codd NF

A rel^n schema R w/ a set FD F is said to be in BCNF if $\forall\ \alpha \rightarrow \beta \in F^+$ atleast one of the following condition holds

1. $\alpha \rightarrow \beta$ is a trivial FD
2. α is a super key of R

For a set of attr K, ($K \subseteq R$) K is a super key of R iff $K \to R$

in_dept (ID, name, salary, dept_name, bldg, budget)

$F = \{$ dept_name $\to$ bldg, budget $\}$

Assume $F^+ = F$

because the given func^c dependency violates the conditions for boyce-codd NF (since it is a non-trivial func^c dependency and the antecedent (α) is not a super key)

$R_1 = \alpha \cup \beta$

$R_2 = R - (\beta - \alpha)$

$R_1 = \{$ dept_name, bldg, budget$\}$

$R_2 = \{$ ID, name, salary, dept_name$\}$

To check whether R_1 & R_2 are in boyce-codd NF.

w.r.t F, R_1 is in boyce-codd NF because the FD is of the form $K \to R$.

w.r.t F, R_2 is not applicable for the test, because the FD cannot be enforced on R_2.

If it is possible to verify, whether a func^c dependency can be enforced on a given schema R w/o any overhead then we say that the design is dependency preserving. It is important to ensure that the relⁿ is dependency preserving wrt the decomposition along with being in the BCNF.

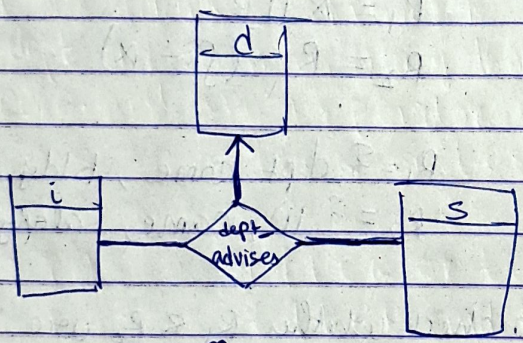
Q) Consider the req, a student can be advised by multiple instructors w/ at most one instructor from a given dept.

① Model the above req using an ER diagram and derive the schema for the assoc b/w the entity sets.

② Det F and check whether the above rel set is in BCNF. & justify.

③ Decompose, if the relation is not in BCNF and check whether it is dependency preserving. Justify your observations.

A)



R = dept_advises (i.ID, dept_name, s.ID)

F = { {s.ID, dept_name → i.ID}, {i.ID → dept_name} }

Assume $F^+ = F$,

Consider FD_1,

FD_1 is of the form $K \rightarrow R$

Consider FD_2,

FD_2 is not trivial & the antecedent is not K.

$R_1 = \alpha \cup \beta = \{i.ID, dept\_name\}$

$R_2 = R - (\beta - \alpha)$

$= \{i.ID, s.ID\}$

R_2 - Not dependency preserving.
Although R_1 is dep^n preserving wrt FD_2 R_2 is not.
If we consider both of them, then for FD_1 both are not dep^n preserving
Overall design is not dependency preserving.

Relational Algebra

RA is a formal model that provides theoretical background for rel^n query langes such as SQL. It defines the following six fundamental operators:

1. select → (σ)
2. project → (π)
3. cartesian product → ($\times$)
4. rename → (ρ)
5. set difference → ($-$)
6. set union → ($\cup$)

Using the above F.O., it is possible to define the following 3 operators:

1. Natural join → ($\bowtie$)
2. Assignment → ($\leftarrow$)
3. Intersection → ($\cap$)

Select (σ)

This works similar to the where clause of SQL that is it selects those tuples that satisfies a given predicate. The general form is $\sigma_p(r)$

$= \{t \mid t \in r \text{ and } P(t) \text{ is true}\}$

The predicates can be formed by combining the logical connectives: and ($\wedge$), or ($\vee$) and $\neg$ (not).

Project (Π)

This operator only outputs the tuple values for the specified attr.

The general form is $\Pi_{A_1, A_2 \ldots A_i}(r)$

Out of n diff attr in r, only the i attr are displayed.

Q) Provide the monthly salary along with ID and name of instr. who belong to phy dep.

$$\Pi_{ID, name, salary/12}\left(\sigma_{dept\_name = "Physics"}(instructor)\right)$$

Q) Project the details of instructors who belong to the CSE dept and earn a salary that exceeds 80000.

Set union operator for two rel^n

$r \& s = \{ t \mid t \in R \text{ or } t \in S \}$

$r \cup s = \{ t \mid t \in R \text{ or } t \in S \}$

$r - s = \{ t \mid t \in R \text{ and } t \notin S \}$

$r \cap s = r - (r - s)$

Q) Find all the dept names that do not have any instr.

$$\Pi_{dept\_name}(department) - \Pi_{dept\_name}(instructor)$$

~~Natural Join (⋈)~~ Cartesian Product (x)

For any two rel^n r & s

r ~~⋈~~ x s can be computed as:

$r \text{ x } s = \{ t_r t_s \mid t_r \in r \text{ and } t_s \in s \}$

Q) Find the names of all instr from phy dept along w/ the course id of each course taught.

A) $\Pi_{name, course\text{-}id}\left(\sigma_{ID.teaches = teaches.ID}\left(\sigma_{dept.name = "physics"}(i) \times teaches\right)\right)$

Rename (ρ)

It projects the result of rel^n oper. E as x with the attr. renamed to $A_1, A_2 \ldots A_n$.

$$\rho_{x(A_1 \ldots A_n)}(E)$$

Q) Find the largest instr. salary at the university.

A) all salaries $R1 \leftarrow \Pi_{salary}(instructor)$

not 1st or greatest $R2 \leftarrow \Pi_{salary}(\sigma_{instructor.salary < i.salary}(instructor \times \rho_{I}(inst)))$

$result = R_1 - R_2$

Q) Find the IDs and names of instructors who never taught a course.

A) ① − ②

$\Pi_{ID, name}(instr.) - \Pi_{ID, name}(\sigma_{ID = t.ID}(instr. \times teaches\ t))$

Q) Find name & ID all the students who haven't taken a course in the year 2018.

A) $R_1 \leftarrow \Pi_{ID, name}(student)$

$R_2 \leftarrow \Pi_{ID, name}(\sigma_{year = 2018, i.ID = t.ID}(student \times takes))$

$R1 \leftarrow \Pi_{ID}(takes) - \Pi_{ID}(\sigma_{year=2018}(takes))$

$result = \Pi_{ID,name}(R1 \bowtie Student)$

Q) Consider the following schema: employee (personname, street, city)
company (comp_name, city)
manages (personname, managename)
works (personname, comp_name, salary)

Write RA queries for the following:
① Find all employees w/ salary more than every employee at SBC.
② Find all the employee who lives in the same city & street as do their managers.

Theta Join

$$r \bowtie_\theta s = \sigma_\theta(r \times s)$$

(g) Groupby → refer to TB

27/2/24 Third NF :-

For any given relⁿ R and set of FD F, R is in TNF if R,F ∀ $\alpha \rightarrow \beta \in F^+$ one of the following conditions must be satisfied:
* $\alpha \rightarrow \beta$ is a trivial FD
* α is superkey of R
* every attr a in $(\beta - \alpha)$ is

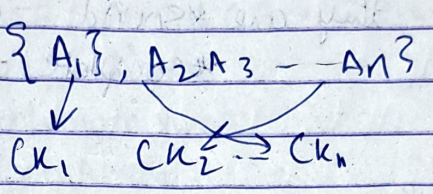
present in some candidate key of R.

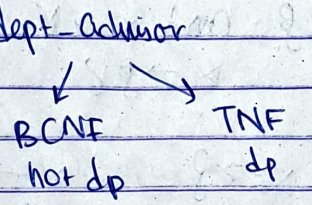
~~If a relⁿ R is in TNF then it is also in BCNF.~~

If a relⁿ R is in BCNF then it will also be in TNF

Consider the relⁿ dept_advisor (stu_id, dept_name) it was found to violate BCNF wrt to FD instr. ID → dept_name ~~is~~ with the FD.

dept_name, ~~difference~~ 1-1 ~~B~~ is seen to satisfy TNF as it is candidate key of ~~dept~~ advisor





Theory of FDs

logically implied FD - F { α1 → β1, α2 → β2 } (αk → βk) ← logically implied

using inference rules it is possible to arrive at F+

F+ = F ∪ { all the LI FD }

* Reflexivity rule - if α & β are sets of attr then both α, β ⊆ R and if β ⊆ α, α → β holds on R

* Augmentation
if $\alpha \to \beta$ holds on R & $\gamma \subseteq R$
the $\alpha\gamma$ determines $\beta\gamma$
$\alpha\gamma \to \beta\gamma$ holds

* Transivity Rule
If $\alpha \to \beta$, $\beta \to \gamma$ holds on R
then $\alpha \to \gamma$

There are armstrong's axioms

Two imp characteristics of AA are
they are *sound - correct
*complete - all the $\overset{LI}{FD}$ can be inferred.

* Union Rule
If $\alpha \to \beta$ and $\alpha \to \gamma$ then
$\alpha \to \beta\gamma$

$\alpha \to \beta$ $\quad$ $\alpha \to \gamma$
$\downarrow \alpha$ $\quad$ $\alpha\beta \to \beta\gamma$
$\alpha \to \alpha\beta$ $\nearrow$
$\Rightarrow \alpha \to \beta\gamma$

* Decomposition Rule
If $\alpha \to \beta\gamma$ holds on R then
$\alpha \to \beta$ & $\alpha \to \gamma$ holds on R

$\alpha \to \beta\gamma$
$\swarrow$ $\quad$ $\searrow$
$\alpha\beta \to \beta\gamma$ $\quad$ $\alpha \to \alpha\beta$
$\downarrow$ $\quad$ $\downarrow$
$\alpha \to \gamma$ $\quad$ $\alpha \to \beta$

$\alpha \to \beta\gamma$

$\Delta \to \alpha\beta\gamma$

$\alpha\beta\gamma \to \beta\gamma$

$\beta \subseteq \alpha\beta\gamma$

$\alpha\beta\gamma \to \beta$ } RR

$\alpha\beta\gamma \to \gamma$ }

* Psuedo transivity rule

If $\alpha \to \beta$ & $\beta\gamma \to \Delta$ then

$\alpha\gamma \to \Delta$ holds on R

21/2/24

Q) $R = \{A, B, C, G, H, I\}$

Let $F = \{ A \to B, A \to C, CG \to H, CG \to I, B \to H \}$

Find closure of F. (F^+).

A) using union rule,

from $A \to B$, $A \to C$:

$A \to BC$

from $CG \to H$, $CG \to I$:

$CG \to HI$

Using Transivity rule

from $A \to B$, $B \to H$,

$A \to H$

~~from $CG \to HI$~~

Using PsuedoTransivity rule,

from $A \to C$, $CG \to H$

$\alpha \to \beta$ $\beta\gamma \to \Delta$

$\alpha\gamma \to \Delta$

$AG \to H$

using union rule,

from $AG \to H$, $AG \to I$

$AG \to HI$

$F^+ = F \cup \{ A \to BC, CG \to HI, A \to H, AG \to H, AG \to I, AG \to HI, A \to BCH \}$

Closure of attr

An attr β is said to be func det by α if $\alpha \to \beta$ is true.

All the attr that are func det by α forms the closure of α known as α^+.

$$\alpha^+ = \{ \beta \mid \beta \in R \text{ and } \alpha \to \beta \}$$

If $\alpha^+ = \{R\}$ it implies that α func det all the attr of R. Hence α is a super key.

Applcn:

① To chek if $\alpha \to \beta$ is true; given R and F, compute α^+ if β is present in α^+ then $\alpha \to \beta$ is true.

② To Find $F^+ = \alpha^+ \cup \beta^+$ ——

For every attr $\alpha \in R$, α^+ is computed. For every attr S present in α^+, $\alpha \to S$ is added to F^+

Q) Determine $\{AG\}^+$ for prev question.

A) $F = \{ A \to B, A \to C, CG \to H, CG \to I, B \to H \}$

(I) $\{AG\}^+ = \{AG\}$ Since $AG \to AG$ is true

$A \to B$

$A \subseteq \{AG\}^+$

$AG \to A$

$\Rightarrow AG \to B$

$\{AG\}^+ = \{AG\} \cup \{B\}$

$\{AG\}^+ = \{ABG\}$

$A \to C$

$\{AG\}^+ = \{ABCG\} \quad \because C \subseteq \{AG\}^+$

$CG \to H$

$\{AG\}^+ = \{ABCGH\} \quad \because CG \subseteq \{AG\}^+$

$CG \to I$

$\{AG\}^+ = \{ABCGHI\} \quad \because CG \subseteq \{AG\}^+$

Extraneous Attr and Canonical Cover

To check extraneous attr A in α wrt the fnc dependency $\alpha \to \beta$:

Consider $\gamma = \alpha - A$. if $\gamma \to \beta$ is true given F, then A is extraneous in α.

If $\beta \subseteq \gamma^+$ then $\gamma \to \beta$ is true.

To check if A is extraneous in β wrt $\alpha \to \beta$:

Consider $F' = \{F - \{\alpha \to \beta\}\} \cup \{\alpha \to (\beta - A)\}$

Using F' it is req to check if $\alpha \to A$ is true in F'. If $A \subseteq \alpha^+_{\text{given } F'}$, then $\alpha \to A$ is true.

Q) Let $F = \{AB \to C, A \to C\}$. Check if B is extr. in $(AB \to C)$.

A) $\alpha = AB$

Consider $\gamma = \alpha - B$

$\gamma = AB - B$

$= A$

To check if $A \to C$ is true in F.

$A^+ = \{A\}$

$A \to C$

$A^+ = \{AC\}$ since $C \subseteq \{A\}^+$

B is extr in $AB \to C$

Q) Let $R = \{A, B, C, D, E\}$. Let $F = \{AB \to CD, A \to E, E \to C\}$
Check if C is extran. in $AB \to CD$.

A)

$$F' = \{A \to E, E \to C\} \cup \{AB \to D\}$$

to find $AB \to C$ is true in F'

$AB^+ = \{ABECD\}$

$\Rightarrow AB \to C$ is true in F'

therfore C is extr. in $AB \to CD$.

check if D is extr.

$$F' = \{A \to E, E \to C\} \cup \{AB \to C\}$$

check $AB \to D$ is true in F':

$AB^+ = \{ABEC\}$

$AB \to D$ is not true in F'

therfore D is not extr.

Canonical Cover F_c

$F \Leftrightarrow F_c$

① $F_c \to \alpha_1 \to \beta_1, \ \alpha_2 \to \beta_2$

$\alpha_1 \neq \alpha_2$ ← No two FD have same antecedant

② $\forall \ \alpha \to \beta \in F_c$

↖ All the FD are free from extr. attr both in antecedant & consequent

Q) let $r(A, B, C)$ satisfy the FD in $F = \{A \to BC, A \to B, B \to C, AB \to C\}$. Compute the canonical cover F_c.

A) ③ $F_c = F$

$A \to BC$ & $A \to B$ have same antecedent, apply union rule

⇒ $A \to BC$

$F_C = \{A \to BC, B \to C, AB \to C\}$

Consider $A \to BC$,

Checking if extr. attr present.

To check B is extr.

if $A \to B$ is true in

$F' = \{B \to C, AB \to C\} \cup \{A \to C\}$

$A^+ = \{AC\}$

$A \to B$ is not true,

B is not extr.

To check C is extr

if $A \to C$ is true in

$F' = \{B \to C, AB \to C\} \cup \{A \to B\}$

$A^+ = \{ABC\}$

$A \to C$ is true,

So C is extr.

So we eliminate C in $A \to BC$

$(A \to B)$

In $AB \to C$

To check if A is extr.

if $B \to C$ is true

$\{B^+\} = \{BC\}$ $B \to C \Rightarrow \{B^+\} = \{BC\}$

So A is extr.

To check if B is extr.

if $A \to C$ is true

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

B is not extr.

So we eliminate (A),

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

So

$F_C = \{A \to B, B \to C\}$

Q) Given $F = \{A \to BC, B \to AC, C \to AB\}$ on $r\{A, B, C\}$
Determine F_c

A) $A \to BC$

Check B

$F' = \{B \to AC, C \to AB\} \cup \{A \to C\}$

$\{A^+\} = \{ACB\}$

$A \to B$ is true

So B is extr. $F_c = \{A \to C, B \to AC, C \to AB\}$

Check C

$F' = \{B \to AC, C \to AB\} \cup \{A \to B\}$

$\{A^+\} = \{ABC\}$

$A \to C$ is true

C is extr.

$B \to AC$

Check A

$F' = \{A \to C, C \to AB\} \cup \{B \to C\}$

$\{B^+\} = \{BCA\}$

eliminate A

$F_c = \{B \to C, A \to C, C \to AB\}$

Check A

$F' = \{B \to C, A \to C\} \cup \{C \to B\}$

$\{C^+\} = \{CB\}$

not extr.

Check B

$F' = \{B \to C, A \to C\} \cup \{C \to A\}$

$\{C^+\} = \{CA\}$

not extr.

So $F_c = \{B \to C, A \to C, C \to AB\}$

Lossless Decompos^n using FDs

Let $r(R)$ be decomposed into two rel^n $r_1(R_1)$ and $r_2(R_2)$ where

$$\Pi_{R_1}(r_1) \bowtie \Pi_{R_2}(r_2) = r$$

$$R_1 \cup R_2 = R$$

Based on FD if any one of the following 'conditions' is satisfied decomp^{n}s R_1 & R_2 must be lossless:

1. $R_1 \cap R_2 \rightarrow R_1$
2. $R_1 \cap R_2 \rightarrow R_2$

If the attr. that are common across the decomp^n rel^n functionally det. the attr. of any one of the decomp. rel^n then the decomposition is lossless. or if the common attr. of the decomp. rel^n appear as super key in any one of the decomp rel^n. then the decomp^n is lossless.

inst_dept (ID, name, salary, dept_name, bldg, budget)

dept_name $\rightarrow$ {bldg, budget}

$R_1 = \{$dept_name, bldg, budget$\}$

$R_2 = \{ID, n, s, d\}$

$R_1 \cap R_2 = \{$dept_name$\}$ forms superkey of R_1 thus R_1 & R_2 not only satisfy BCNF but are also lossless decomp^{n}s.

$$t = (\text{result} \cap R_1)^+ \cap R_2$$
$$= \{D\}^+ \cap \{ADE\}$$
$$= \{D\}$$

As it is observed that $E \notin CD^+$, $CD \to E$ is not preserved in the decompositions. Hence the entire design is not dependency preserving.

① $\left(\bigcup_{i=1}^{n} F_i\right)^+ = F^+$

② $\alpha \to \beta$ is preserved in F'
if $\alpha^+ \cap R_i$
then $(\text{result} \cap R_i)^+ \cap R_i = t$
$\text{result} = \text{result} \cup t$

Decomposition Algorithms

(I) BCNF Decomp"

$\alpha \to \beta$ is not trivial in F^+
(Super key) $\alpha^+ \to R$ is true in F^+

$R_1 = (\alpha \cup \beta)$ $\alpha^+ \neq R$
$R_2 = R - (\beta - \alpha)$
$= R - \beta \mid (\alpha \cup \beta \phi)$

Q) Let $R\{A, B, C, D, E, G)$ w/ $F = \{AB \to CD$
$B \to D$, $DE \to B$, $DEG \to AB$, $AC \to DG\}$

① Check $AB \to CD$ violates conditions for BCNF.
② Provide decomp" of R starting w/ R_{BCNF}

$AB \to CD$, if violation is True.

A) ① $A \to C$ is non trivial
AB is not superkey. It violates

② $R_1 = (A, B, C, D)$ $R_2 = (A, B, E, G)$

Consider R_1

$(AB)^+ = \{A, B, C, DE\}$

$\Rightarrow$ AB is superkey of R_1

Not violating BCNF on R_1

~~Consider R_2~~

$B \to D$ $B^+ = \{BD\}$

$D \to D$ violates BCNF on R_1

$R_{11} = \{BD\}$

$R_{12} = \{ABC\}$

Consider R_{11}

$D^+ = \{D\}$

$R_{12}(ABC)$

$AB \to CD$

$AB^+ = \{ABCDE\}$

Not violating BCNF

Consider R_2

$AB \to CD$

$AB^+ = \{ABCDE\}$

$\Rightarrow AB \to E$

violating BCNF as it does not det. G.

$R_{21} = (ABE)$

$R_{22} = (ABG)$

result = {R};

while (result changes)

{ if ($\exists$ R_i | R_i not in BCNF)

if ($\alpha \to \beta$ is a non-trivial & $\alpha \cap \beta = \phi$)

result = (result − R_i) ∪ (α, β) ∪ ($R_i - \beta$)

break }

Every decomposition must be in BCNF.

Q) Given $R = \{A, B, C, D, E\}$ $F = \{A \to BCD, BC \to E, B \to D, D \to A\}$ Verify if R is in BCNF, if not decompose R into individual rel^ns all of which are in BCNF showing all the steps in detail.

A) Consider $A \to BCD$

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

$A \Rightarrow BCD$ violates BCNF

$R_1 = (A, B, C, D)$

$R_2 = (A, E, F)$

Consider $R_1 (A, B, C, D)$

$A \to BCD$ ⇒ not violating

$BC \to DE$

$BC^+ = \{B, C, D, E, A\}$ ⇒ not violating

$B \to D$ ⟹ not violating

$D \to A$ ⟹ not violating

Consider $R_2 (A, E, F)$

$A \to E$ violates BCNF on R_2

$R_{21} = (A, E)$

$R_{22} = (A, F)$

BCNF decomp^n are R_1, R_{21}, R_{22}

BCNF Decomposition Algorithm

① Compute F_c

② $i=0$ ∀ $\alpha \to \beta \in F_c$

$i += 1$

$R_i = (\alpha, \beta)$

③ if (∄ any R_i which has a CK of R)

$i += 1$

$R_i = CK$

④ if ($R_j \subset R_k$)

delete R_j

$F = \{A \to BCD, BC \to DE, B \to D, D \to A\}$

if B is extr

$A^+ = \{ACD\}$ $F' = \{BC \to DE, B \to D, D \to A, A \to CD\}$

B is not extr

if C is extr

C is not extr $A^+ = \{ABD\}$

if D is extr $F' = \{BC \to DE, B \to D, D \to A, A \to BC\}$

$A^+ = \{ABCDE\}$

D is extr.

eliminate.

$F_c = \{A \to BC, BC \to DE, B \to D, D \to A\}$

if B is extr in BC then —

B is not extr.

if C is extr in BC

$B^+ = \{BDAC\}$

C is.

E is not extr.

$F_E = \{ A \to BC, B \to DF, D \to A \}$

$R_1 = (ABC)$ $R_2 = (BDE)$
$R_3 = (DA)$

Multivalued dependencies

Instructor (ID, name, department_name, str, city)

ID → name, dept_name, str, city
ID → name
inst1 (ID, name)
ID →→ dept_name
ID →→ {str, city}
inst2 (ID, dept_name, str, city)

r(R) $\alpha \twoheadrightarrow \beta$

(1) $\forall t_1, t_2 \quad t_1[\alpha] = t_2[\alpha]$
$\exists t_3, t_4$
$t_1[\alpha] = t_2[\alpha] = t_3[\alpha] = t_4[\alpha]$
(2) $t_1[\beta] = t_3[\beta]$
$t_2[\beta] = t_4[\beta]$

(3) $t_1[R-\beta] = t_4[R-\beta]$
(4) $t_2[R-\beta] = t_3[R-\beta]$

ID →→ {str, city}

Fourth Normal form

$D \rightarrow (MVDs) \cup (FDs)$ D^+

Multi Valued D Pure D

$\forall \alpha \twoheadrightarrow \beta \in D^+$ R is 4NF

1. $\alpha \twoheadrightarrow \beta$ is trivial FD
2. α is superkey of R

Q) $R(A, B, C, G, H, I)$

$F = \{A \twoheadrightarrow B,\ B \twoheadrightarrow HI,\ CG \twoheadrightarrow H\}$

Provide 4NF decomp of R

A) $A \twoheadrightarrow B$

$A^+ = \{ABHI\}$

$R_1(A, B)$ ✓

$R_2(ACGHI)$

$A \twoheadrightarrow H$ violates 4NF on R_2

$R_{21}(A, H)$ ✓

$R_{22}(ACGI)$

$A \twoheadrightarrow B$ $B \twoheadrightarrow H$ $B \twoheadrightarrow I$

$A \twoheadrightarrow I$ violates 4NF on R_{22}

$R_{221}(AI)$ ✓

$R_{222}(ACG)$ ✓ → $A \not\rightarrow C$ or G

so we leave it.

alt: $CG \twoheadrightarrow H$

$R_{21} = (CGH)$ ✓

$R_{22} = (ACGI)$

$A \twoheadrightarrow I$

$R_{221} = (AI)$

$R_{222} = (ACG)$

Q) R(ABCDE) F = { A → BC ; CD → E ; B → D ; E → A }

1. Obtain all keys of R
2. Check if schema is in BCNF, if not provide BCNF decompositions.
3. Det if its in 3NF if not provide decom using canonical cover of F.

A) (1) ① Essential Attr: None
all engage in FD

② No attr det. itself

③ No non essent. attr

④ Identify middle ground attr:

(A) B C D (E)

~~AB~~ ~~BC~~ ~~AD~~ ~~AE~~ (BC) BD ~~BE~~ ...

candidate keys — (CD)

$A^+ = \{ABCDE\}$

$B^+ = \{BD\}$

$C^+ = \{C\}$

$D^+ = \{D\}$

$E^+ = \{EABCD\}$

$BC^+ = \{BCDEA\}$

$BD^+ = \{BD\}$

$CD^+ = \{CDEAAB\}$

Candidate keys are:

⇒ A, E, {BC}, {CD}

(3) $F_C = F$

Consider: A → BC

if A → B is true in

$F' = \{CD \to E, B \to D, E \to A\}$

$A \to C\}$

$A^+ = \{AC\}$

B is not extraneous

$A \to C$ is true in $F' = \{CD \to E, B \to D, E \to A, A \to B\}$

$A^+ = \{ABD\}$

C is not extr.

$CD \to E$

if C is extr $(CD - C) \to E$ must hold in F_c

$D^+ = \{D\}$

D is not extr.

Similarly C is not extr.

$F_c = F$.

Given set F was the canonical cover.

$R_1(A\ B\ C)$

$R_2(C\ D\ E)$

$R_3(B\ D)$ ✓

$R_4(E\ A)$

} contains CK.

(2) BCNF

$A \to BC$

$A^+ = \{ABCDE\}$

$B \to D$ violates BCNF

$R_1(B\ D)$ ✓

$R_2(A\ B\ C\ E)$ ✓

Q) R(A B C D E G) F = {AB → CD, A → B, B → C, C → E, BD → AG}

① Obtain CK of R

② Provide BCNF decomp of R

A) ① $G^+ = \{G\}$

E is non essential

MGAs

(AG) – BG – CG – DG

ACG (BDG) CDG

② AB → CD violates BCNF

R_1 (A B C D)

R_2 (A B E G)

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

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

R_{11} (B E) ✓

R_{12} (A B D) ✓

A → B

R_{21} (A B) ✓

R_{22} (A E G)

A → E violates

R_{221} (A E) ✓

R_{222} (A G) ✓