Week 1 DBMS

Disadvantages of Physical data storage

- Durability: Physical damage to these registers is a possibility due to rodents, humidity, wear and tear
- Scalability: Very difficult to maintain for many years, some shops have numerous registers spanning over years
- Security: Susceptible to tampering by outsiders
- Retrieval: Time consuming process to search for a previous entry
- Consistency: Prone to human errors

Excel

Advantages of spreadsheet files

Google sheets

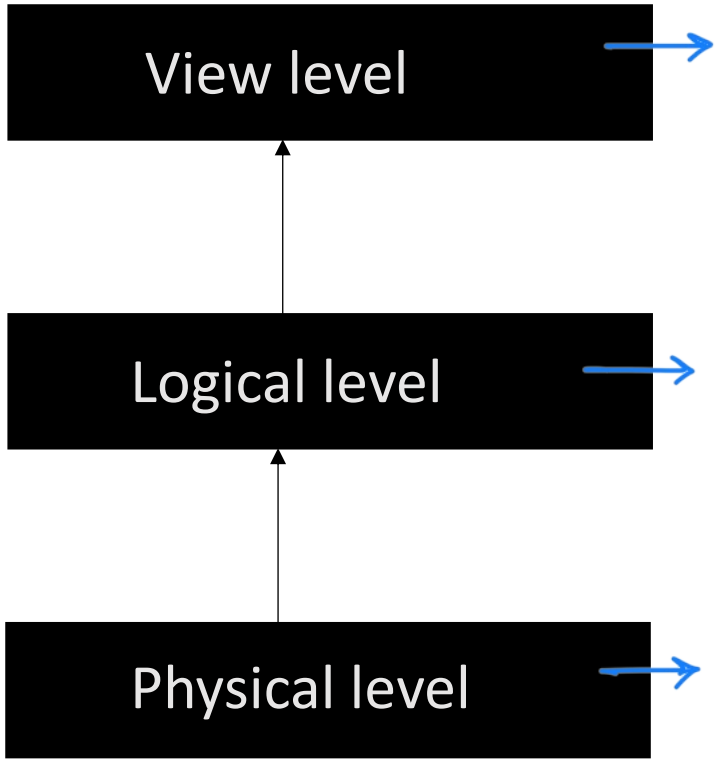
- Durability: These are computer applications and hence data is less prone to physical damage.
- Scalability: Easier to search, insert and modify records as compared to book ledgers
- Security: Can be password-protected
- Ease of Use: Computer applications are used to search and manipulate records in the spreadsheets leading to reduction in manpower
- Consistency: Not guaranteed but spreadsheets are less prone to mistakes than registers.

Comparison

Parameter	File Handling via Python	DBMS
Scalability with respect to amount of data	Very difficult to handle insert, update and querying of records	In-built features to provide high scalability for a large number of records
Scalability with respect to changes in structure	Extremely difficult to change the structure of records as in the case of adding or removing attributes	Adding or removing attributes can be done seamlessly using simple SQL queries
Time of execution	In seconds	In milliseconds
Persistence	Data processed using temporary data structures have to be manually updated to the file	Data persistence is ensured via automatic, system induced mechanisms
Robustness	Ensuring robustness of data has to be done manually	Backup, recovery and restore need minimum manual intervention
Security	Difficult to implement in Python (Security at OS level)	User-specific access at database level
Programmer's productivity	Most file access operations involve extensive coding to ensure persistence, robustness and security of data	Standard and simple built-in queries reduce the effort involved in coding thereby increasing a programmer's throughput
Arithmetic operations	Easy to do arithmetic computations	Limited set of arithmetic operations are available
Costs	Low costs for hardware, software and human resources	High costs for hardware, software and human resources

except these two DBMS is a better choice in every other aspect.

Level of abstraction



represents how the user sees the data

represents the relational model and conceptual schema of the data

deals with the actual storage structure of the data internally in the form of trees and other data structures.

In other words, change in Physical level of DBMS should not affect the View level or the logical level.

NOTE

- Physical Data Independence – the ability to modify the physical schema without changing the logical schema
- Logical Data Independence- the ability to modify the logical level without changing the view level.

Schema

schema is the way data will be organized, instance is the actual value of that

- Similar to type of a variable and value of the variable at run-time in programming languages
- **Schema**
  - **Logical Schema** – the overall logical structure of the database
    - Analogous to type information of a variable in a program
    - Example: The database consists of information about a set of customers and accounts in a bank and the relationship between them
    - Customer Schema

Name	Customer ID	Account #	Aadhaar ID	Mobile #

    - Account Schema

Account #	Account Type	Interest Rate	Min. Bal.	Balance

  - **Physical Schema**– the overall physical structure of the database

Instance

- **Instance**
  - The actual content of the database at a particular point in time
  - Analogous to the value of a variable
  - Customer Instance

Name	Customer ID	Account #	Aadhaar ID	Mobile #
Pavan Laha	6728	917322	182719289372	9830100291
Lata Kala	8912	827183	918291204829	7189203928
Nand Prabhu	6617	372912	127837291021	8892021892

  - Account Instance

Account #	Account Type	Interest Rate	Min. Bal.	Balance
917322	Savings	4.0%	5000	7812
372912	Current	0.0%	0	291820
827183	Term Deposit	6.75%	10000	100000

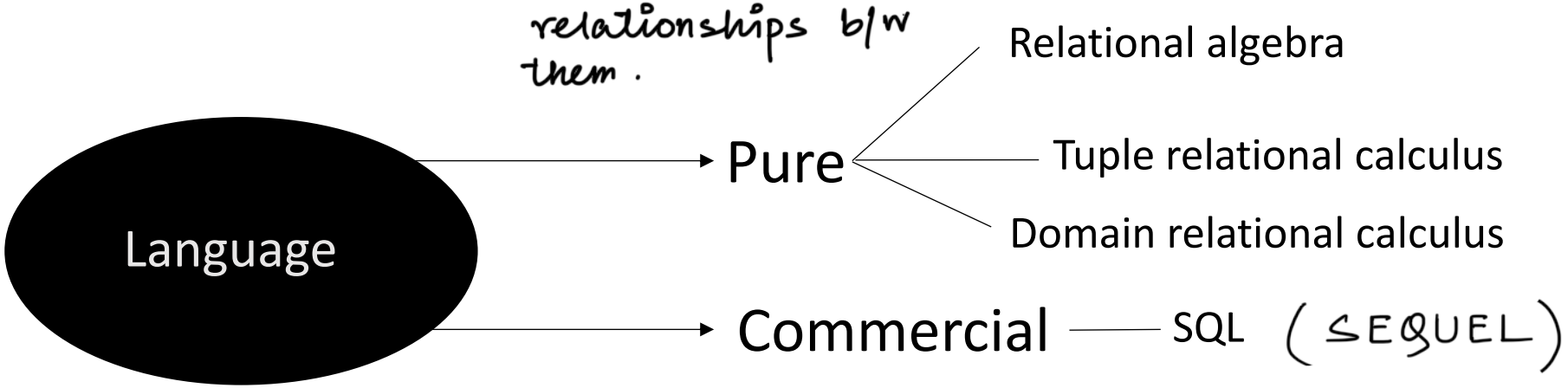
instance may be added/removed and the schema remains the same, vice versa is NOT true

Data Models

- A collection of tools for describing
  - Data
  - Data relationships
  - Data semantics
  - Data constraints

used during the planning and designing phase of a Database System as it's used to define a high level view of the data entities and the relationships b/w them.

- Relational model (data stored in various tables)
- Entity-Relationship data model
- Object-based data models (flat, atomic values)
- Network model
- Hierarchical model
- Recent models for Semi-structured or Unstructured data



DDL(data definition language)

- Specification notation to define the database schema
- Ex- create table, drop table, alter table
- It generates a set of table templates stored in a data dictionary

DML (data manipulation language)

- Language to manipulate and access data
- Ex- insert, update, delete
- It is also known as query language

SQL(Structured query language)

- Most widely used commercial language
- It is not a turning machine equivalent language (it means all C programs cannot run in SQL but vice versa is true.)

```
insert into table_name values ('x', y, 'z') -- default order.
insert into table_name (A2, A1, A3) values (y, 'x', 'z') -- specifying the order.
```

Logical design

- What attributes to record?
- What relation schemas should we have?
- How should the attributes be distributes?

Physical design

- Decides the physical layout of the database

XML(Extensible Markup Language):

- defined by www consortium(W3C).
- Has ability to specify new tags and create nested tag structures.
- used for sharing data over different systems over the web.

Database engine

Storage management

- interface between the low-level data stored in the database and the application programs and queries submitted to the system.
- Interacts with OS file manager.
- Stores, retrieves and updates data.

Query processing

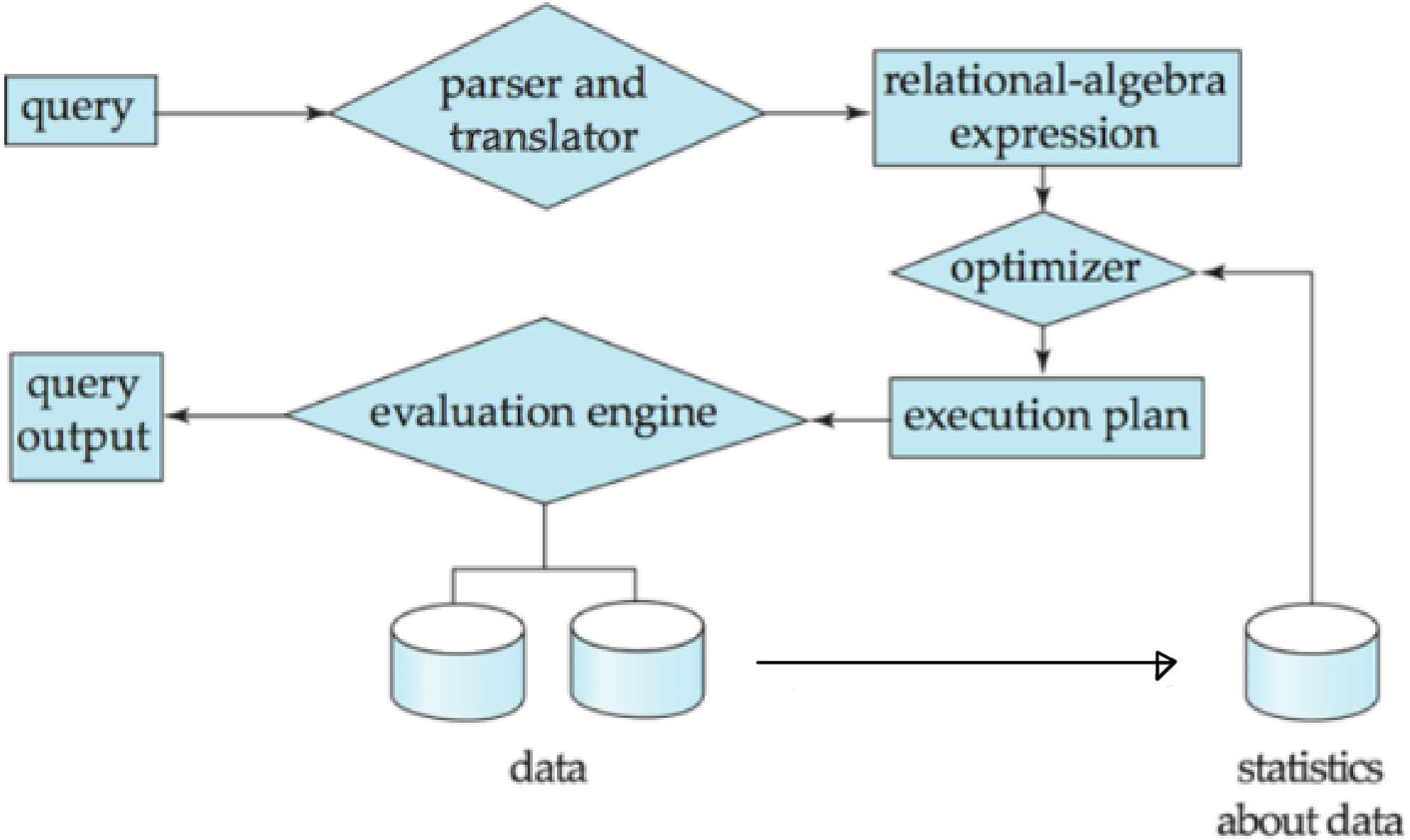
- Parsing and translation
- Optimization
- Evaluation

It helps in estimating the cost of operation.

Transaction management

A transaction is a collection of operations that performs a single logical function in a database application

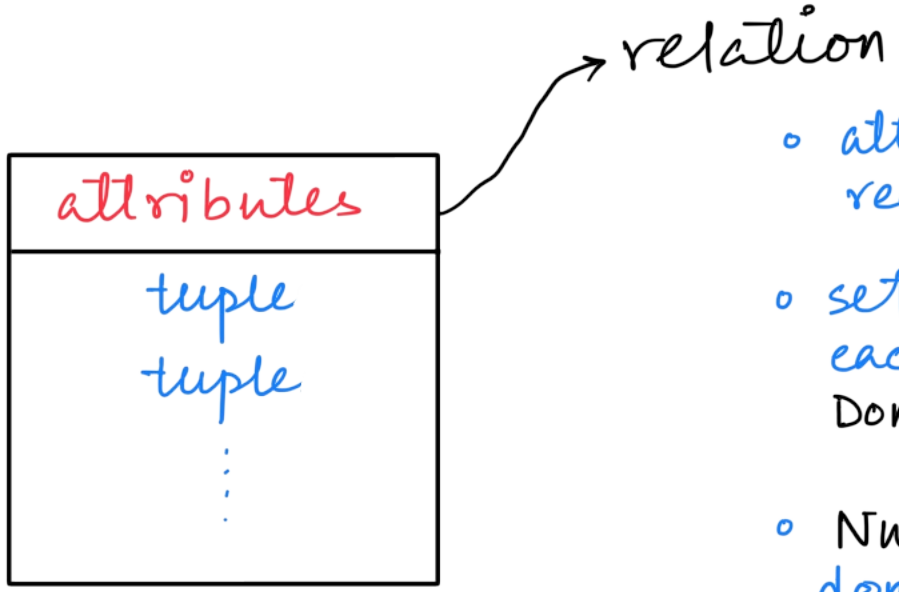
- Transaction-management component: ensures that the database remains in a consistent state despite system failures and transaction failures.
- Concurrency-control manager: controls the interaction among the concurrent transactions, to ensure the consistency of the database.


query
parser and translator
relational-algebra expression
optimizer
query output
evaluation engine
execution plan
data
statistics about data

WEEK 2 DBMS

- Domain Types of Attributes :
  - char(n) → fixed memory space
  - varchar(n) → variable memory space
  - date
  - int
  - float(n)
  - numeric(p, d)
  - ⋮

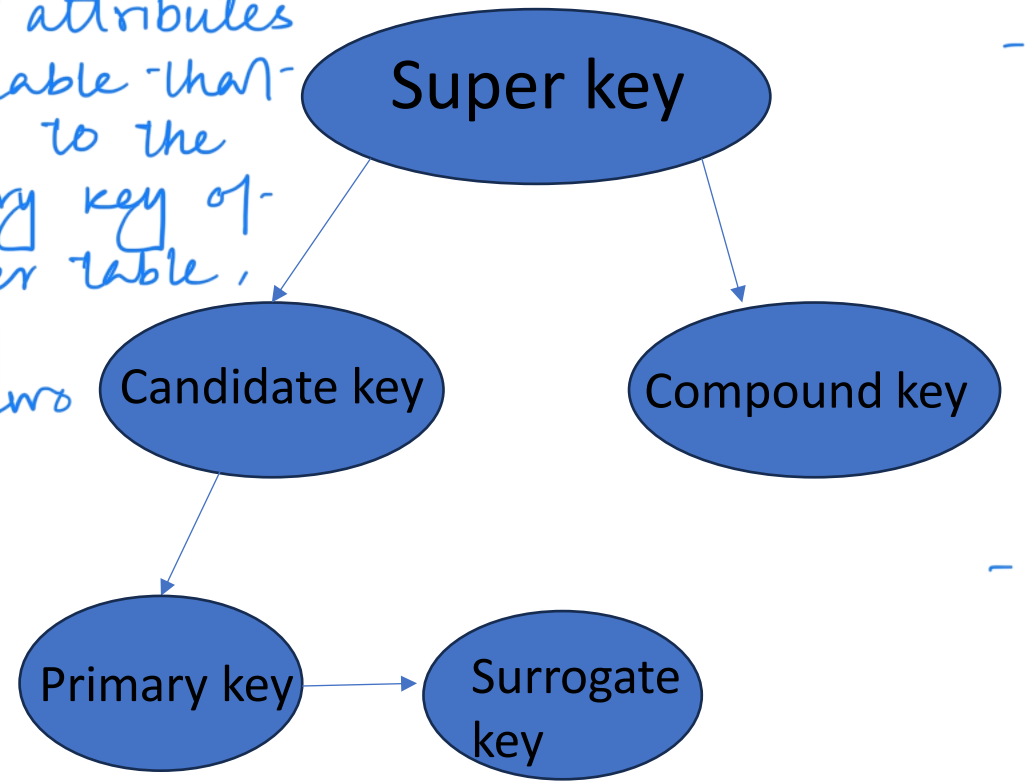
- Constraints :
  - Null / Not null
  - Unique
  - Check
  - Primary key (Not null + Unique)
  - Foreign key
    - referencing to different attributes.



- relation is a set, ordering of tuples is inconsequential and all tuples must be distinct.

- attribute values are normally required to be atomic.
- set of allowed values for each attribute is called the Domain of the attribute.
- Null is a member of every domain, indicates that the value is unknown.

- Foreign key → set of attributes in a table that refers to the primary key of another table, linking these two tables.



- values present inside the foreign key column must present inside the primary key column to which the foreign key is referencing to.

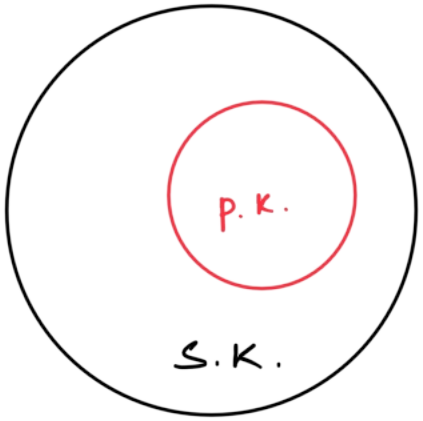
- f.k. column can contain null as well as duplicate values unless there is a constraint on the f.k.

Composite key — Any key which has more than one attribute. Example- {orderId, productId} is a primary composite key. ***two** primary keys needed to uniquely identify*

simple key — Any key which identifies a tuple uniquely with just a single attribute.

Secondary key

Super key: set of one or more attributes which can uniquely identify a tuple. It should have the primary key and then it can include the keys which doesn't determine uniqueness. Example: {empID}, {empID,empName},{empID,empName,empPhone}



Candidate key: minimal super key. Example: if {empID} and {empEmail} is individually a candidate key, then {empID, empEmail} can't be candidate key.

Primary key: a candidate key which isn't null. {empID}

Secondary key: non unique key which forms indices.
ex: empName

Compound key: set of multiple attributes which can individually be a candidate key of other tables.

Surrogate key: any key which can be system generated. Example: {rollno}

Relational Operators

- σ - Select → tuples (or rows)
- π - Project → attributes (or columns)
- $\neg$ - Negation (not)
- $\wedge$ - AND
- $\vee$ - OR
- $\cup$ - Union → relations must have same no. of attributes and corresponding attributes have the same domain.
- $\cap$ - Intersection → 'Union compatibility' (must).
- $\times$ - Cartestion Product
- $-$ - Set Difference → 'Union compatibility' (must).
- $\bowtie$ - Natural Join

- Relation r

A	B	C	D
α	α	1	7
α	β	5	7
β	β	12	3
β	β	23	10

Selects **rows** where A=B **and** D is greater than 5

- $\sigma_{A=B \wedge D>5}(r)$

(Relational Algebra)

A	B	C	D
α	α	1	7
β	β	23	10

- Relation r

A	B	C
α	10	1
α	20	1
β	30	1
β	40	2

Selects only A and C **columns** and remove duplicates

- $\pi_{A,C}(r)$

(Relational Algebra)

A	C
α	1
α	1
β	1
β	2

=

A	C
α	1
β	1
β	2

after removing the duplicates

UNION- takes all the distinct values of both the tables especially when the tables have similar structures.

SET DIFFERENCE- example r-s takes the tuples of r which are not present in s, when they are union compatible.

SET INTERSECTION- takes the common tuples, when they are union compatible.

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

CARTESIAN PRODUCT- example r x s

Cartesian Product

Table One

X	A
1	a
4	d
2	b

3 rows

Table two

X	B
2	x
3	y
5	v

3 rows

Result Set

X	A	X	B
1	a	2	x
1	a	3	y
1	a	5	v
4	d	2	x
4	d	3	y
4	d	5	v
2	b	2	x
2	b	3	y
2	b	5	v

NATURAL JOIN → removes duplicate attributes.

- Relations r, s:

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

r

B	D	E
1	a	α
3	a	β
1	a	γ
2	b	δ
3	b	$\in$

s

- $r \bowtie s$

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

DDL

Schema
Domain
Integrity constraints

A language which is used to design the schema of the database and also modifies it.

- It is a part of SQL

- **Integrity constraints**: are rules that maintains the consistency of the data by enforcing some conditions on data values.
  ex - ID varchar(25) not null .
  attribute
  domain
  integrity constraint .
- DDL enforces or specifies the integrity constraints while defining or modifying the schema.
- **Data dictionary**: contains data about data(metadata) stored in the database. It stores the data generated by DDL commands along with the integrity constraints.

-- we can add or drop attributes using ALTER .
ALTER TABLE takes add phone_num int-
ALTER TABLE takes drop phone_num

Create a Table

```
CREATE TABLE takes (
    ID varchar(5),
    course_id varchar(8),
    sec_id varchar(8),
    semester varchar(8),
    year_ numeric(4, 0),
    grade varchar(2),
    primary key (ID, course_id, sec_id, semester, year_),
    foreign key (ID) references student,
    foreign key (course_id, sec_id, semester, year_)references section
)
```

DROP

```
DROP TABLE takes   -- removes the table
```

(σ)

* in CAPITAL / in bold → SQL keyword.

DML

page - 9

SELECT Clause

(DQL command)

- DISTINCT - Selects all the distinct values
- * - Selects all the attributes
- as - Renames the attribute
- TOP <n> - selects top n tuples

FROM Clause

(Π)

Specifies the table name.

From instructor, course will result in a cross-join between two tables

```
-- selecting all attributes w/o * .
select all
from relation r
-- renaming can be done w/o AS.
```

WHERE Clause

Specifies conditions to retrieve data

WHERE i.course_id=c.course_id and i.dept_name='Biology' and salary>40000

and ; or ; not.

STRING OPERATION

- LIKE - Uses the pattern that are described in like condition and matches with the attribute
- % - matches any substring

%abc%
%abc
abc%

- '_ _ _' matches any string of exactly three characters
- '_ _ _%' matches any string of at least three characters

```
select * 
from relation
where attribute like '% abc %'
```

WHERE Clause Predicates

`BETWEEN a and b`

```
select name
from instructor
where salary between 90000 and 100000
```

- Select the name of the instructor whose salaries between 90000 and 100000 (both are inclusive)

`IN` → allows us to specify multiple values in a where clause.

- Acts like shorthand operator for `OR`

```
select name
from instructor
where dept_name in ('Comp Sci', 'Biology')
```

Set Operations → 'Union compatibility' (must).

- `UNION` and `UNION ALL` (or)
- `INTERSECT` and `INTERSECT ALL` (and)
- `EXCEPT` and `EXCEPT ALL` (bwt. / not in)
  - `EXCEPT` → set difference

Note

- `UNION ALL`, `INTERSECT ALL` and `EXCEPT ALL` retains the duplicate

* Union, Intersect, Except removes duplicate.

Group By

Aggregate function

attributes in select clause outside of aggregate functions must appear in GROUP BY list.

```
SELECT dept_name, avg(salary)
from instructor
group by dept_name
```

Having

```
select dept_name, avg(salary)
from instructor
group by dept_name
having avg(salary) > 42000;
```

Note - the Order of SQL queries

→ DESC
→ ASC (default)

SELECT, FROM, WHERE, GROUP BY, HAVING, ORDER BY , DISTINCT, LIMIT/OFFSET .

V I II III IV VII VI VIII

- Null values →
  "unknown".
  - any comparison w/ null returns null.

```
select name, dept_name, total_cred
from student
ORDER BY dept_name ASC, total_cred DESC
LIMIT 5
OFFSET 1
```

DBMS WEEK 3

Select distinct

building	*room_number*	*capacity*
Packard	101	500
Painter	514	10
Taylor	3128	70
Watson	100	30
Watson	120	50

Figure: *classroom* relation

- Query:

  select distinct *building*
  from *classroom*
  where *capacity* $<$ 100;

- Output :

building
Painter
Taylor
Watson

Select all

building	*room_number*	*capacity*
Packard	101	500
Painter	514	10
Taylor	3128	70
Watson	100	30
Watson	120	50

Figure: *classroom* relation

- Query:

  select all *building*
  from *classroom*
  where *capacity* $<$ 100;

- Output:

building
Painter
Taylor
Watson
Watson

Select all and select will give the same result

AS operation

select *S.name* **as** *studentname*, *budget* **as** *deptbudget*
from *student* **as** *S*, *department* **as** *D*
where *S.dept_name* = *D.dept_name* **and** *budget* < 100000;

studentname	*deptbudget*
Brandt	50000.00
Peltier	70000.00
Levy	70000.00
Sanchez	80000.00
Snow	70000.00
Aoi	85000.00
Bourikas	85000.00
Tanaka	90000.00

Renames student relation as S and relation department as D

Nested Subquery

A subquery is a select-from-where expression that is nested within another query.

Some Clause (≡ any)

- 5 > some(0, 5, 6) - True
- 5 = some(0, 5, 6) - True

All Clause

- 7 > all(0, 5, 6) - True
- 5 = all(0, 5, 6) - False

IN operator

ID	course_id	sec_id	semester	year
10101	CS-101	1	Fall	2017
10101	CS-315	1	Spring	2018
10101	CS-347	1	Fall	2017
12121	FIN-201	1	Spring	2018
15151	MU-199	1	Spring	2018
22222	PHY-101	1	Fall	2017
32343	HIS-351	1	Spring	2018
45565	CS-101	1	Spring	2018
45565	CS-319	1	Spring	2018
76766	BIO-101	1	Summer	2017
76766	BIO-301	1	Summer	2018
83821	CS-190	1	Spring	2017
83821	CS-190	2	Spring	2017
83821	CS-319	2	Spring	2018
98345	EE-181	1	Spring	2017

Figure: *teaches* relation

- Query:

```
select course_id
from teaches
where semester in ('Fall','Spring')
    and year=2018;
```

- Output:

course_id
CS-315
FIN-201
MU-199
HIS-351
CS-101
CS-319
CS-319

IN also acts like or operator

Except

will remove all the occurrences.

* Except all will remove only the occurrence(s) mentioned.

ID	name	dept_name	salary
10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	60000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
58583	Califieri	History	62000
76543	Singh	Finance	80000
76766	Crick	Biology	72000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000

Figure: *instructor* relation

- Query:

```
select name
from instructor
where dept_name in ('Comp. Sci.','Finance')
except
select name
from instructor
where salary < 90000 and salary > 70000;
```

- Output:

name
Srinivasan
Brandt
Wu

```
select name
from instructor
where dept_name in ('Comp. sci.', 'Finance')
and (salary >= 90000 or salary <= 70000)
```

1 AVG aggregate function

building	*room_number*	*capacity*
Packard	101	500
Painter	514	10
Taylor	3128	70
Watson	100	30
Watson	120	50

Figure: *classroom* relation

- Query:

  select *building*, **avg** (*capacity*)
  from *classroom*
  group by *building*
  having avg (*capacity*) $>$ 25;

- Output:

building	*avg*
Taylor	70.00
Packard	500.00
Watson	40.00

② MIN aggregate function

ID	*name*	*dept_name*	*salary*
10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	60000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
58583	Califieri	History	62000
76543	Singh	Finance	80000
76766	Crick	Biology	72000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000

Figure: *instructor* relation

- Query:

  select min(*salary*) **as** *least_salary*
  from *instructor*;

- Output:

least_salary
40000.00

3 MAX aggregate function

ID	*name*	*dept_name*	*tot_cred*
00128	Zhang	Comp. Sci.	102
12345	Shankar	Comp. Sci.	32
19991	Brandt	History	80
23121	Chavez	Finance	110
44553	Peltier	Physics	56
45678	Levy	Physics	46
54321	Williams	Comp. Sci.	54
55739	Sanchez	Music	38
70557	Snow	Physics	0
76543	Brown	Comp. Sci.	58
76653	Aoi	Elec. Eng.	60
98765	Bourikas	Elec. Eng.	98
98988	Tanaka	Biology	120

Figure: *student* relation

- Query:

  select max(*tot_cred*) **as** *max_credits*
  from *student*;

- Output:

max_credits
120

(9) COUNT aggregate function

* all the aggregate functions except count ignore tuples w/ null values.

course_id	*sec_id*	*semester*	*year*	*building*	*room_number*	*time_slot_id*
BIO-101	1	Summer	2017	Painter	514	B
BIO-301	1	Summer	2018	Painter	514	A
CS-101	1	Fall	2017	Packard	101	H
CS-101	1	Spring	2018	Packard	101	F
CS-190	1	Spring	2017	Taylor	3128	E
CS-190	2	Spring	2017	Taylor	3128	A
CS-315	1	Spring	2018	Watson	120	D
CS-319	1	Spring	2018	Watson	100	B
CS-319	2	Spring	2018	Taylor	3128	C
CS-347	1	Fall	2017	Taylor	3128	A
EE-181	1	Spring	2017	Taylor	3128	C
FIN-201	1	Spring	2018	Packard	101	B
HIS-351	1	Spring	2018	Painter	514	C
MU-199	1	Spring	2018	Packard	101	D
PHY-101	1	Fall	2017	Watson	100	A

Figure: *section* relation

- Query:

```
select building,
       count(course_id) as course_count
from section
group by building;
```

- Output:

building	*course_count*
Taylor	5
Packard	4
Painter	3
Watson	3

5 SUM aggregate function

course_id	*title*	*dept_name*	*credits*
BIO-101	Intro. to Biology	Biology	4
BIO-301	Genetics	Biology	4
BIO-399	Computational Biology	Biology	3
CS-101	Intro. to Computer Science	Comp. Sci.	4
CS-190	Game Design	Comp. Sci.	4
CS-315	Robotics	Comp. Sci.	3
CS-319	Image Processing	Comp. Sci.	3
CS-347	Database System Concepts	Comp. Sci.	3
EE-181	Intro. to Digital Systems	Elec. Eng.	3
FIN-201	Investment Banking	Finance	3
HIS-351	World History	History	3
MU-199	Music Video Production	Music	3
PHY-101	Physical Principles	Physics	4

Figure: *course* relation

- Query:

```
select dept_name,
    sum(credits) as sum_credits
from course
group by dept_name;
```

- Output:

dept_name	*sum_credits*
Finance	3
History	3
Physics	4
Music	3
Comp. Sci.	17
Biology	11
Elec. Eng.	3

EXISTS clause

```
select course_id
from section as S
where semester = 'Fall' and year = 2009 and
                exists (select *
                        from section as T
                        where semester = 'Spring' and year = 2010
                        and S.course_id = T.course_id);
```

This should exist(be true) for the above query to execute

Will show all courses which were taught in 2009 and 2010 in fall and spring semester.

NOT EXISTS clause

```
select distinct S.ID, S.name
from student as S
where not exists ( (select course_id
                    from course
                    where dept_name = 'Biology')
                    except
                    (select T.course_id
                            from takes as T
                            where S.ID = T.ID));
```

WITH clause : provides a way of defining a temporary relation whose definition is available only to the query in which the with clause occurs.

```
WITH AvgSalary AS ( SELECT AVG(salary) AS
avg_salary
 FROM Salaries )

SELECT e.employee_name, s.salary
FROM Employees e JOIN Salaries s ON
e.employee_id = s.employee_id WHERE s.salary >
(SELECT avg_salary FROM AvgSalary);
```

Shows the names of employees and their salaries if they get more than average salary.

INSERTION (DML command)

- Add a new tuple to course

  insert into *course*
  values ('CS-437', 'Database Systems', 'Comp. Sci.', 4);

- or equivalently:

  insert into *course (course_id, title, dept_name, credits)*
  values ('CS-437', 'Database Systems', 'Comp. Sci.', 4);

- Add a new tuple to student with *tot_creds* set to null

  insert into *student*
  values ('3003', 'Green', 'Finance', *null*);

Deletion (DML command)

a)
```
delete from instructor
-- delete all instructors
```

b)
```
delete from instructor
where dept_name = 'Finance'
-- delete all instructors
-- from the Finance department
```

UPDATES (DML command)

Updates the salary by 3% or 5% based on given condition

```
update instructor
        set salary = salary * 1.03
        where salary > 100000;
update instructor
        set salary = salary * 1.05
        where salary <= 100000;
```

* ON DELETE CASCADE : used to specify that when a row is deleted from the parent table, all rows in the child table that reference the deleted row should also be deleted.

* If not mentioned, we can't delete from the parent table.

JOINS

cross join: cartesian product of rows.
Inner join: intersection (the duplicate column is present)
Natural join: intersection (the duplicate column is not present)
Full outer join: union (uses null values) (keys are explicitly defined)
Left outer join: set A + intersection of set A and B
Right outer join: set B + intersection of set A and B
Natural full outer join: union (keys are not explicitly defined)

self Join: table is joined w/ itself.

- Students:

StudentID	Name
1	Alice
2	Bob
3	Charlie
4	Dave

- Courses:

CourseID	StudentID	CourseName
101	1	Math
102	2	Science
103	5	History

SELECT Students.Name, Courses.CourseName FROM Students **INNER JOIN** Courses ON Students.StudentID = Courses.StudentID;

Name	CourseName
Alice	Math
Bob	Science

Select Students.Name, Courses.CourseName from Students **LEFT JOIN** Courses ON Students.StudentID = Courses.StudentID;

Name	CourseName
Alice	Math
Bob	Science
Charlie	NULL
Dave	NULL

SELECT Students.Name, Courses.CourseName FROM Students **RIGHT JOIN** Courses ON Students.StudentID = Courses.StudentID;

Name	CourseName
Alice	Math
Bob	Science
NULL	History

SELECT Students.Name, Courses.CourseName FROM Students FULL OUTER JOIN Courses ON Students.StudentID = Courses.StudentID;

Name	CourseName
Alice	Math
Bob	Science
Charlie	NULL
Dave	NULL
NULL	History

SELECT Students.StudentID, Students.Name, Courses.CourseID, Courses.CourseName FROM Students NATURAL JOIN Courses;

StudentID	Name	CourseID	CourseName
1	Alice	101	Math
2	Bob	102	Science

VIEWS

- A view provides a mechanism to hide certain data from the view of certain users
- Any relation that is not of the conceptual model but is made visible to a user as a “virtual relation” is called a view.

- A view of instructors without their salary

```
create view faculty as
        select ID, name, dept_name
        from instructor
```

- Find all instructors in the Biology department

```
select name
        from faculty
        where dept_name = 'Biology'
```

- Create a view of department salary totals

```
create view departments_total_salary(dept_name, total_salary) as
        select dept_name, sum (salary)
        from instructor
        group by dept_name;
```

A view relation v1 is said to depend directly on a view relation v2 if v2 is used in the expression defining v1.

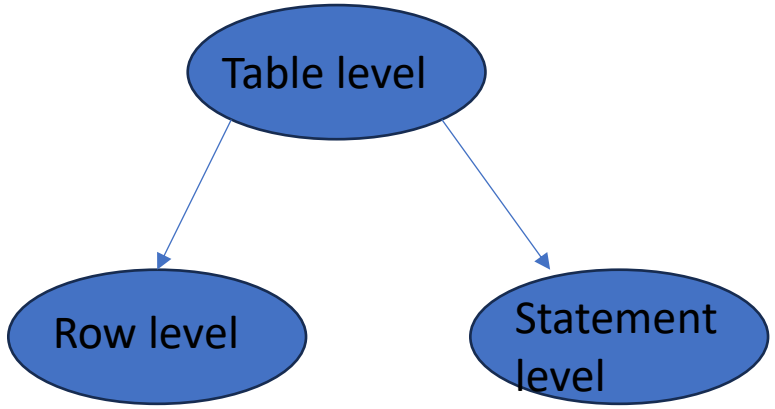
A view relation v is said to be recursive if it depends on itself.

A view relation v1 is said to depend on view relation v2 if either v1 depends directly to v2 or there is a path of dependencies from v1 to v2.

TRIGGERS

- Automatic execution of specific action that are performed in response to an insert, update or delete operation on a specified table.
- Executed before or after an action .

- Row level trigger: executed for each row
- Statement level trigger: executes for all rows only once.



- $\wedge$, $\vee$ part;
- joining conditions;
- what we're going to project (projection)

for checking TRC and DRC expressions

DBMS Week 4

Division Operation

$X \div Y$ or X / Y

can be applied iff :

- attributes of Y ⊂ attributes of X.
- the relation returned will have attributes = (All attributes of X − All attributes of Y).
- The relation returned will return those tuples from X which are associated to every Y's tuples.

- Relations *r*, *s*:

A	B
α	1
α	2
α	3
β	1
γ	1
δ	1
δ	3
δ	4
ϵ	6
ϵ	1
β	2

r

B
1
2

s

A
α
β

$r \div s$

Relational Algebra

→ output of RA expression is a set.

It's a procedural query language

- σ - Select → rows (where clause)
- π - Project → columns (select distinct)
- $\neg$ - Negation (not)
- $\cup$ - Union
- $\cap$ - Intersection
- $\times$ - Cartestion Product
- $-$ - Set Difference (Except)
- $\bowtie$ - Natural Join

Example

1. Find all the names of students whose age is greater than 25, or who are enrolled in Maths

$$\pi_{Name}(\sigma_{Age<25 \lor Subject='Maths'}(Students))$$

2. Find the name and sports of the student whose age is less than 25 and awards is greater than 3

$$\pi_{Name,Sports}(\sigma_{Age<25 \land Awards>3}(Students \bowtie Activity))$$

- Procedural → [how-to-do]

(Row)

Tuple Relational Calculus

- TRC is a non-procedural (declarative) query language, where each query is of the form [what-to-do]

$$\{t \mid P(t)\}$$

where **t** = resulting tuples,

P(t) = known as predicate and these are the conditions that are used to fetch t.

- Quantifiers:

$\exists$ → "there exists"

$\forall$ → "for all"

Example

1. Find the name of the students whose age is 21

$\{t.name \mid students(t) \wedge t.age = 21\}$

or, $\{t \mid \exists\, s \in students(t.name = s.name \wedge s.age = 21)\}$

$\{t.name \mid t \in students \wedge t.age = 21\}$

2. Find the name of the employees who works in department manufacturing

employee(*id*, name, salary)

department(*id*, d_id, name, building)

$$\{M \mid \exists\, E \in employee\ \exists\, D \in department(E.id = D.id \wedge D.name =' Manufacturing' \wedge M.name = E.name)\}$$

(Column)

Domain Relational Calculus

declaritive

- A non-procedural query language equivalent in power to the tuple relational calculus
- Each query is an expression of the form:

$$\{< x_1, x_2, \ldots, x_n > \mid P(x_1, x_2, \ldots, x_n)\}$$

- $x_1, x_2, \ldots, x_n$ represent domain variables
- P represents a formula similar to that of the predicate calculus

* the domain variables should be mentioned in the same order as given in the table.

Example

1. Find the name of the students whose age is 21

student(name, age, marks)

$$\{< a > \mid \exists\, b\, (< a, b, c > \in students \ \wedge \ b = 21)\}$$

2. Find the name of the employees who works in department manufacturing

employee(*id*, name, salary)
department(*id*, d_id, name, building)

$$\{< b > \mid \exists\, a, c, d\, (< a, b, c > \in employee) \ \wedge \ \exists\, y\, (< a, x, y, z > \in department \ \wedge y =' Manufacturing')\}$$

Entity Sets

- An **entity** is an object that exists and is distinguishable from object.
- An **entity set** is a set of entities of the same type that share the same properties

Strong Entity set

- A strong entity set is an entity set that contains sufficient attributes to uniquely identify all its entities.
- A primary key exists for a strong entity sets

Weak Entity Set

- A weak entity set is an entity set that does not contain sufficient attributes to uniquely identify its entities.
- A primary does not exists for a weak entity set
- However, it contains a partial key called as a **discriminator**
- **Discriminator** represented by underlining with a dashed line.

Weak Entity set (continued)

- Weak entity set cannot exist independently since it doesn't have primary key
- It features in the model in relationship with a strong entity set. This is called the **identifying relationship**
- **Primary key of weak entity set = Discriminator + Primary key of Strong entity set**
- It must have **total participation** and **identifying relationship**

Attributes

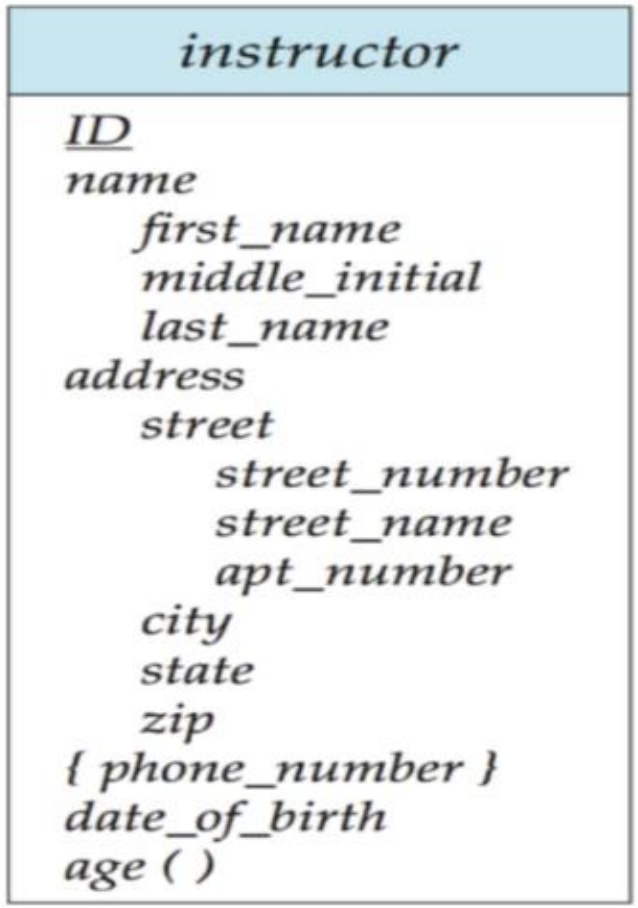
- An attribute is a property associated entity set.

Types of attributes

- Simple attribute
- Composite attribute - **Eg** - fname, mname, lname can consits in a name

- Multivalued attribute - **Eg** - **{phone_numbers}**
- Derived attribute - **Eg** - **age()** from date of birth

Example



ID → Primary key (underlined)

name → Composite attribute

address → Composite attribute

phone_number → Multivalued attribute

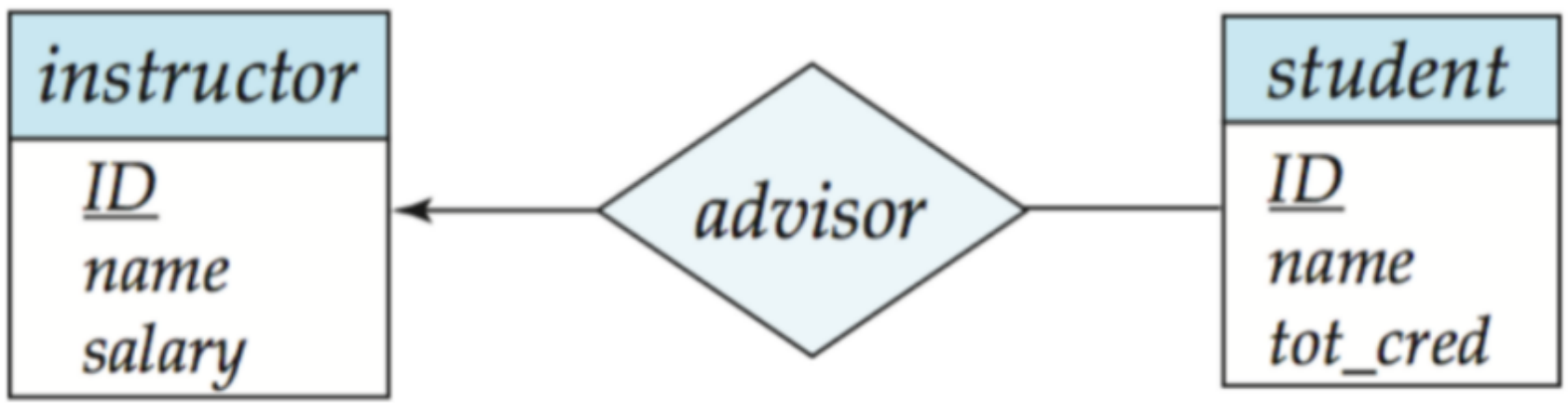
age → Derived attribute

One to One relationship

- Each instructor has atmost one student
- Each student has atmost one instructor

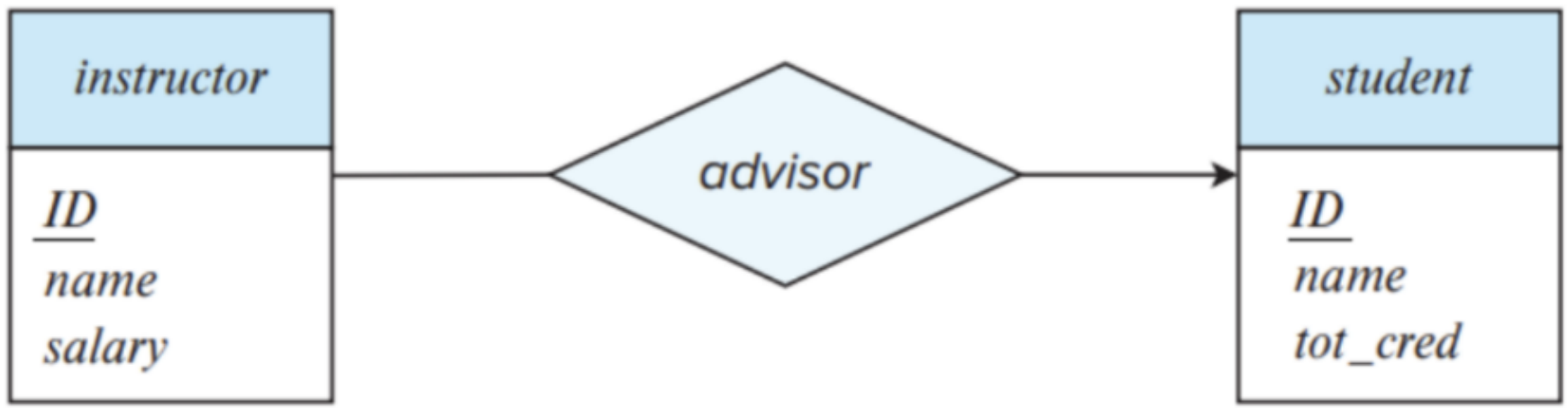
One to Many relationship

- Each instructor has one or many students
- Each student has at most one instructor



Many to one relationship

- Each instructor has at most one student
- Each student has one or many instructors

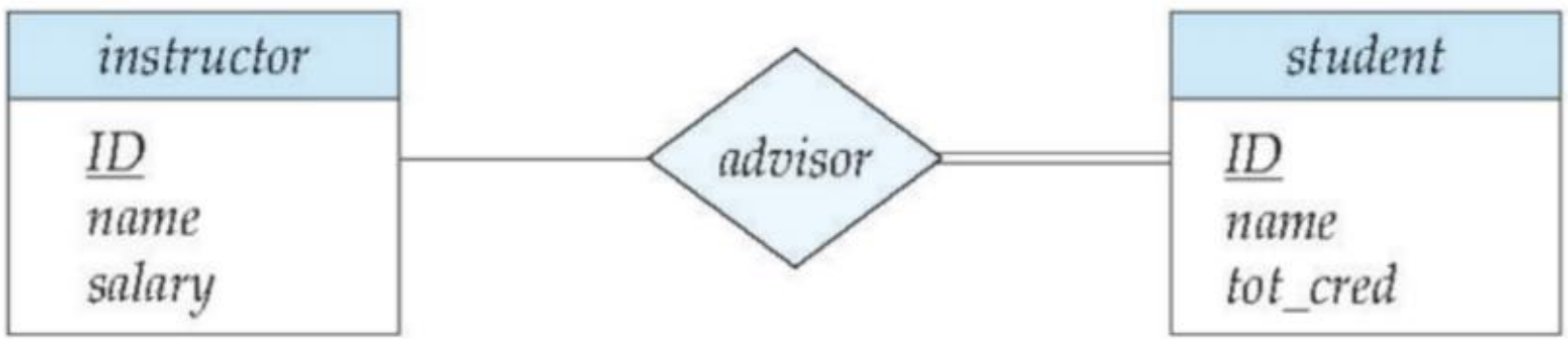


Many to Many relationship

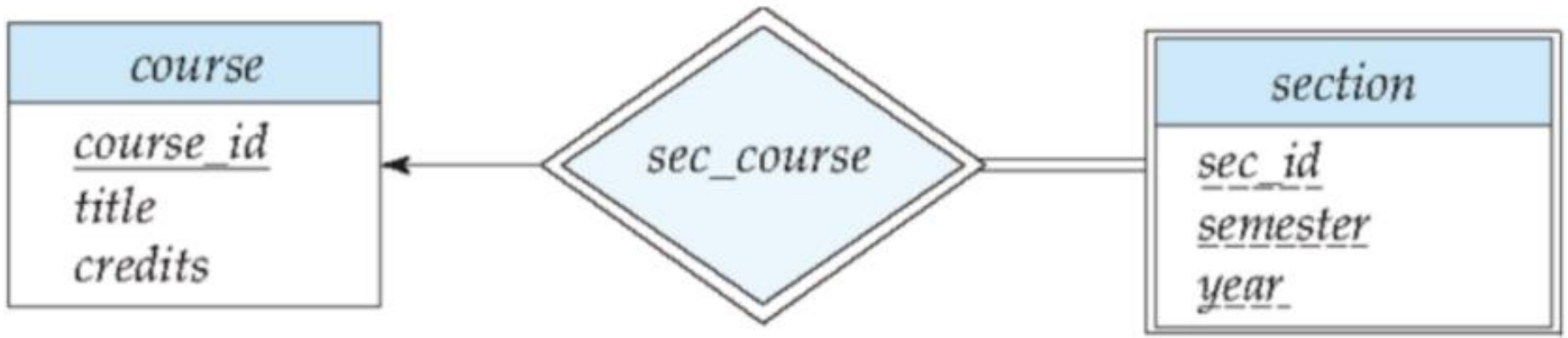
- Each instructor has one or many students
- Each student has one or many instructors

Total and Partial Participation

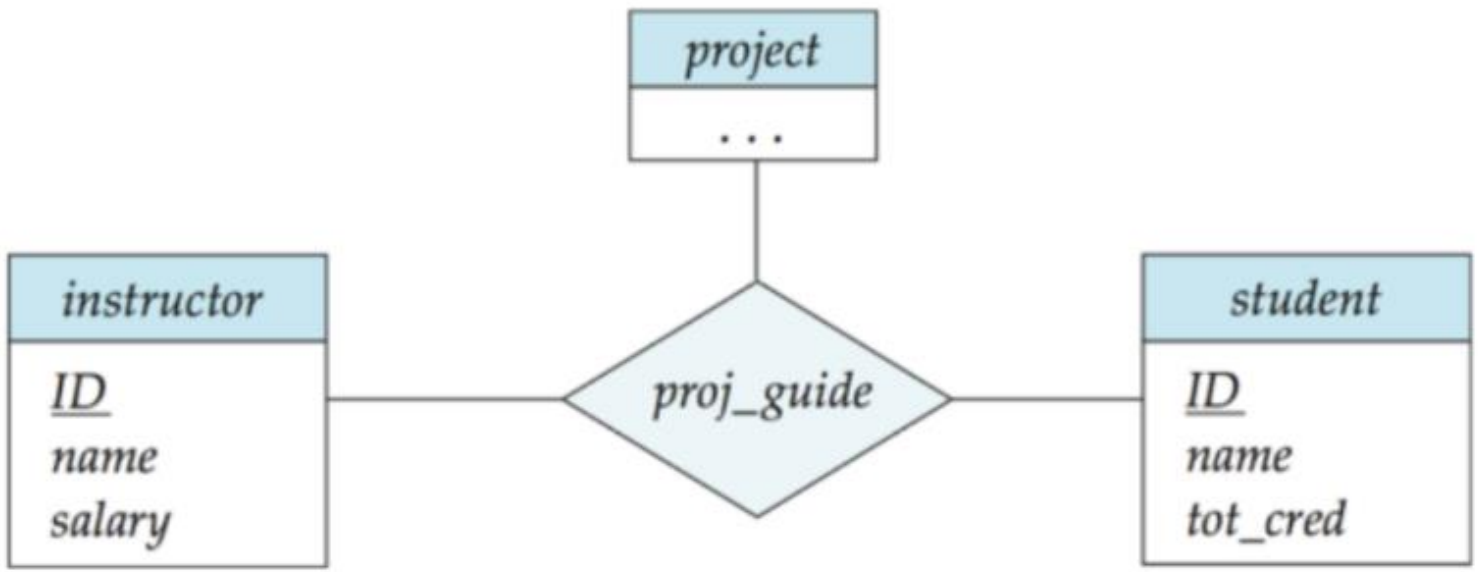
- Every student must have an instructor
- Instructor may or may not have a student



Expressing weak entity set



Ternary Relationship



Aggregation

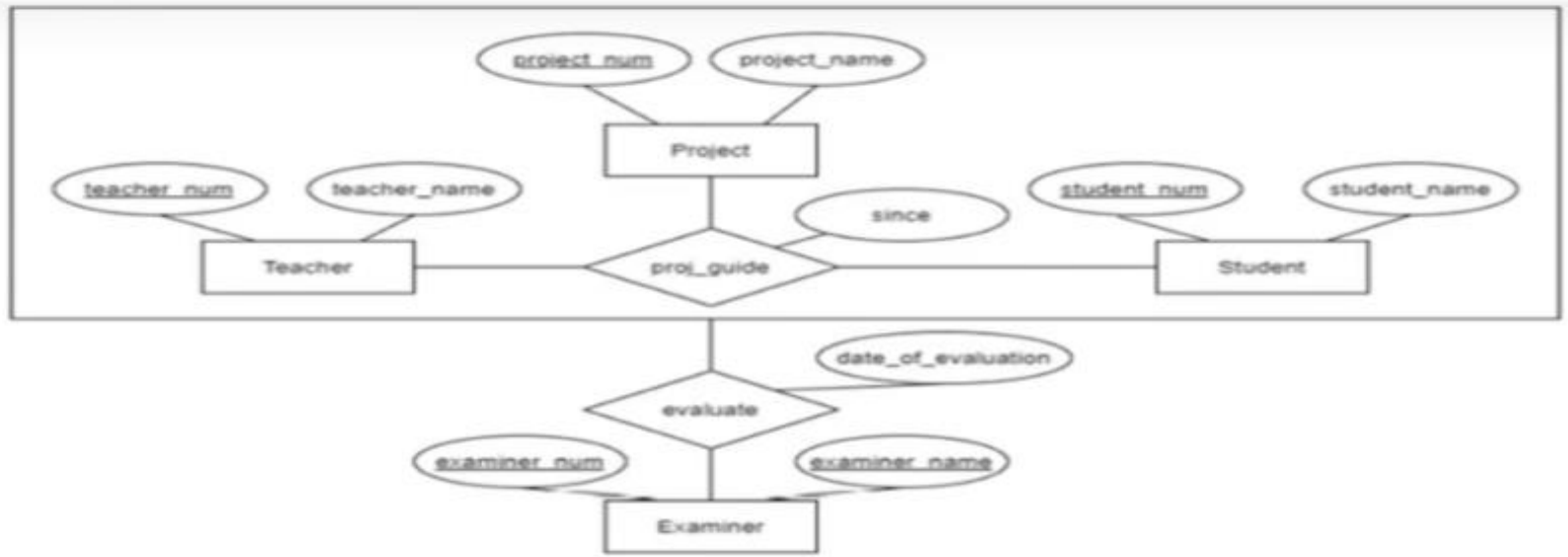


Figure: Example of Aggregation

- **Teacher**{*teacher_num*, *teacher_name* }
- **Student**{*student_num*, *student_name* }
- **Project**{*project_num*, *project_name* }
- **Examiner**{*examiner_num*, *examiner_name* }
- **proj_guide**{*teacher_num*, *student_num*, *project_num*, *since* }
- **evaluate**{*teacher_num*, *student_num*, *project_num*, *examiner_num*, *date_of_evaluation* }

Overlapping and Partial

- A person can be either student or faculty or both.
- There may be some persons who are just persons not belongs to faculty and students

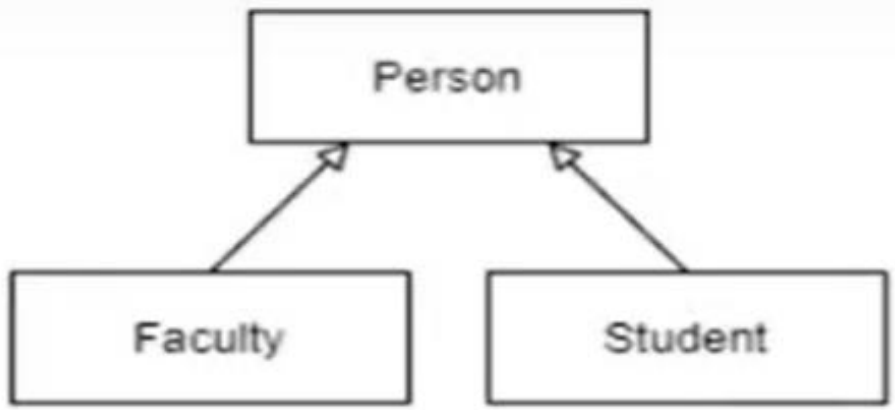


Figure: Overlapping and Partial

Disjoint and Partial

- A Student can be either UG students or PG students but they cannnot be both UG and PG students
- There may be students who are just students not belongs both UG and PG students

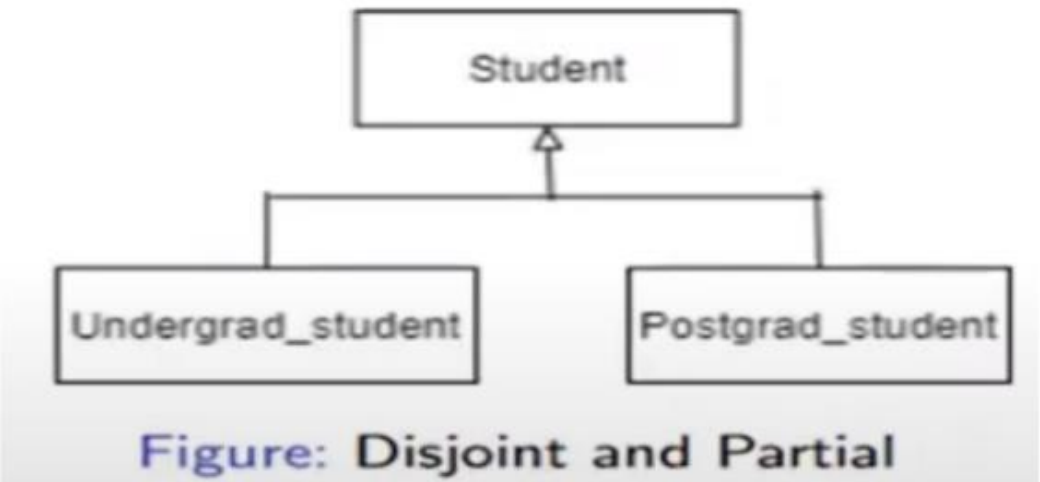


Figure: Disjoint and Partial

Disjoint and Complete

- Every Part must be present in either Purchased part or Manufactured part

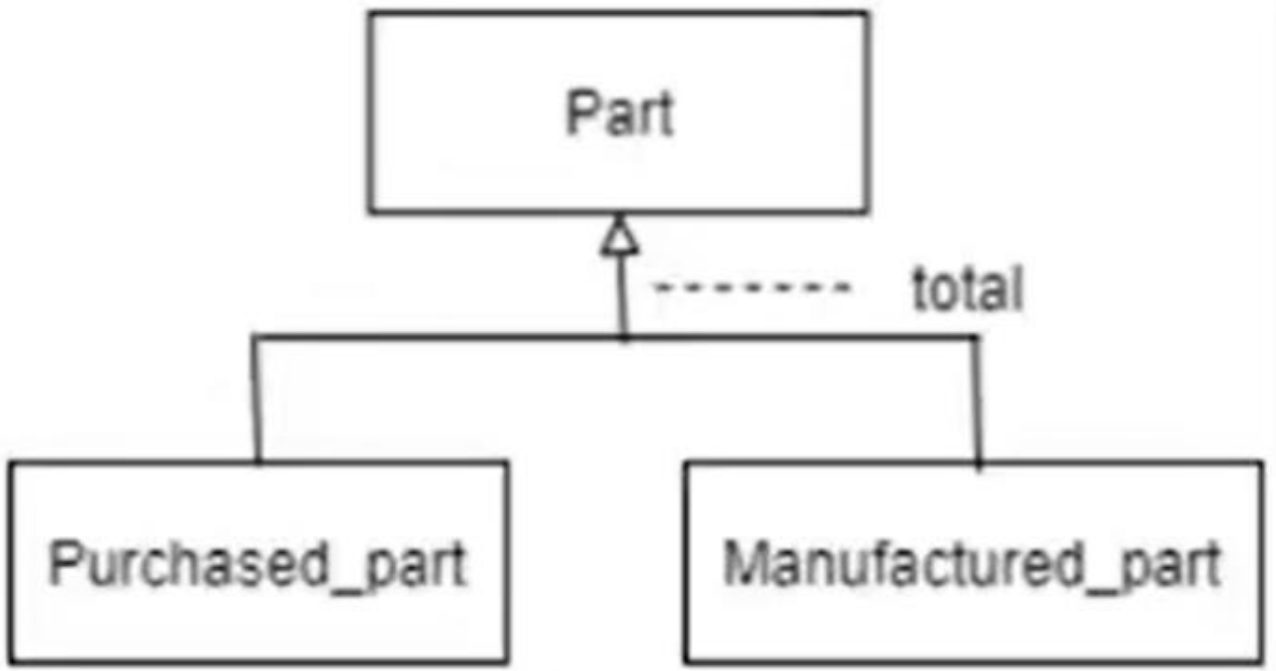


Figure: Disjoint and Complete

Strong Entity Set

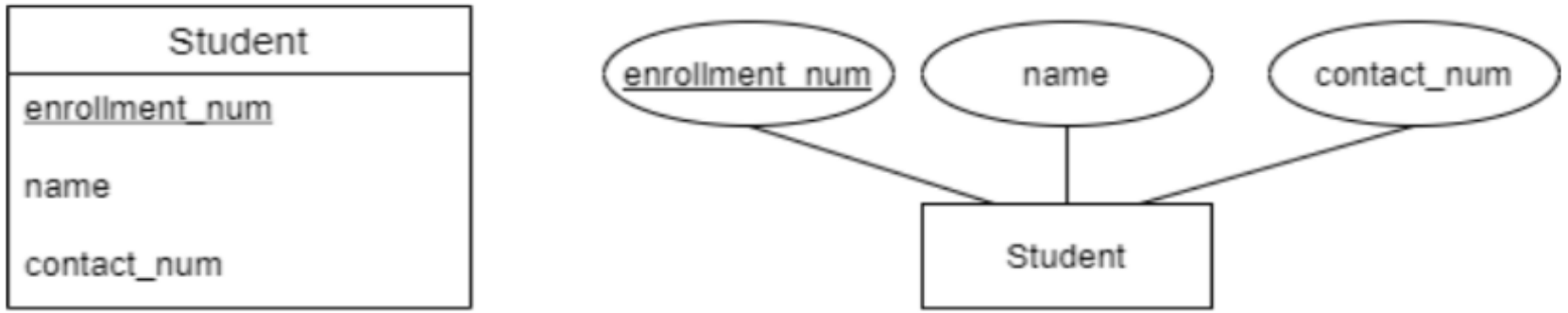


Figure: Strong entity with simple attributes

Student{*enrollment_num*, *name*, *contact_num*}

enrollment_num	name	contact_num
101	RAJ KUMAR MISHRA	222-222
102	SANAT K ROY	333-333

Students table

Strong entity set with composite key

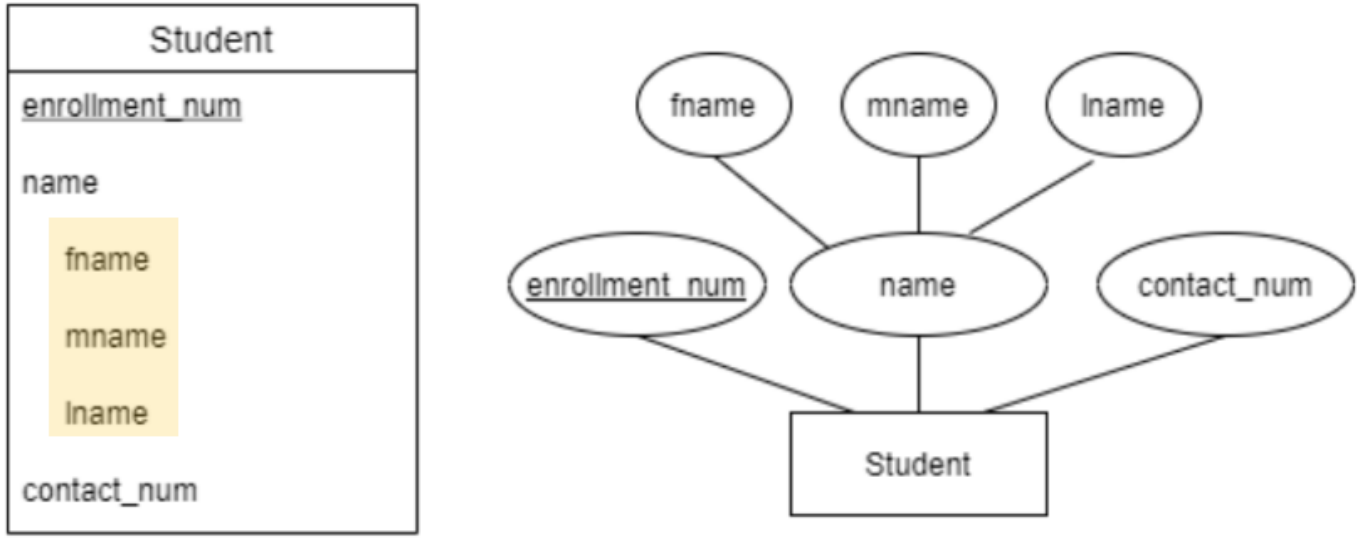


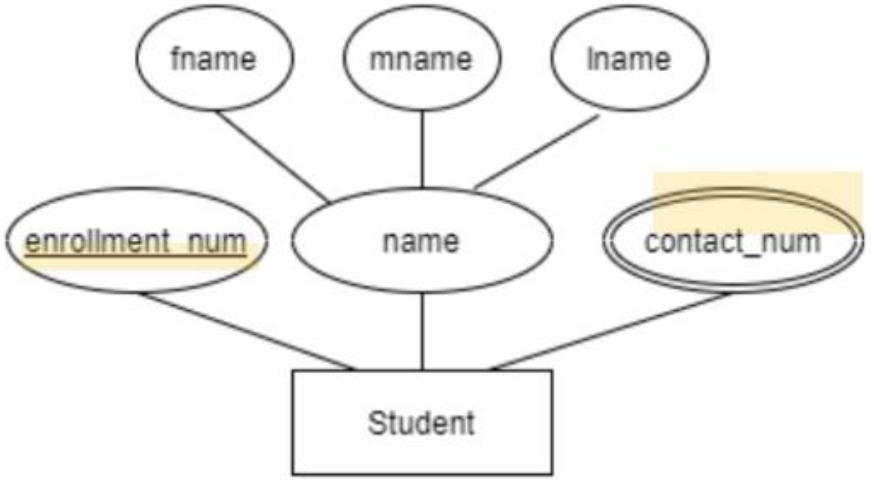
Figure: Entity set **Student** with simple and composite attributes

Student{*enrollment_num*, *fname*, *mname*, *lname*, *contact_num*}

enrollment_num	fname	mname	lname	contact_num
101	RAJ	KUMAR	MISHRA	222-222
102	SANAT	K	ROY	333-333

Students table

Strong entity set with composite key and multivalued attribute



Student{*enrollment_num*, *fname*, *mname*, *lname*, *contact_num*}

enrollment_num	fname	mname	lname	contact_num
101	RAJ	KUMAR	MISHRA	222-222, 777-777
102	SANAT	K	ROY	333-333, 999-999, 666-666

Figure: Table **Student**

It is better to decompose these kinds of tables into multiple tables to avoid redundancy and inconsistency

Student{*enrollment_num*, *fname*, *mname*, *lname*, *contact_num*}

enrollment_num	fname	mname	lname	contact_num
101	RAJ	KUMAR	MISHRA	222-222
101	RAJ	KUMAR	MISHRA	777-777
102	SANAT	K	ROY	333-333
102	SANAT	K	ROY	999-999
102	SANAT	K	ROY	666-666

Figure: Table **Student**

OR

Student{*enrollment_num*, *fname*, *mname*, *lname* }

enrollment_num	fname	mname	lname
101	RAJ	KUMAR	MISHRA
102	SANAT	K	ROY

Figure: Table **Student**

Contacts{*enrollment_num*, *contact_num*}

enrollment_num	contact_num
101	222-222
101	777-777
102	333-333
102	999-999
102	666-666

Figure: Table **Student**

Strong entity set with composite key, multivalued attribute and derived attribute

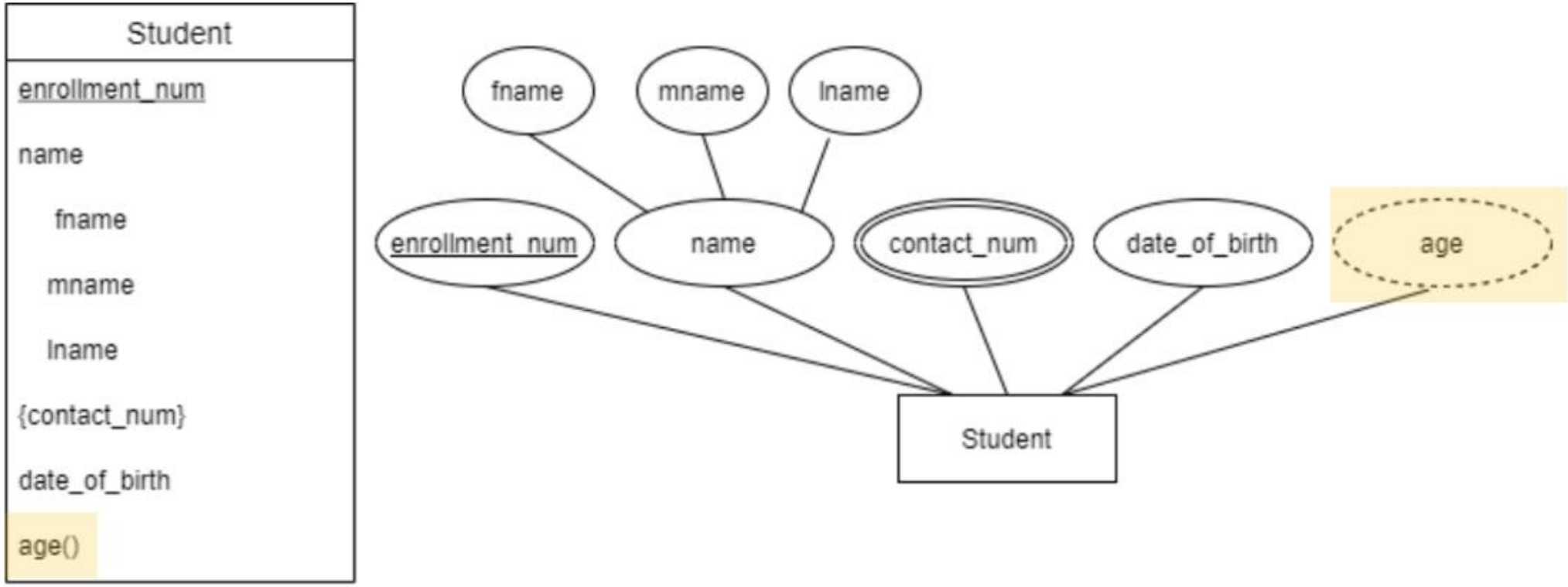


Figure: Entity set **Student** with simple, composite, multivalued attributes, and derived attribute

Student{*enrollment_num*, *fname*, *mname*, *lname*, *data_of_birth* }

Contacts{*enrollment_num*, *contact_num*}

Relationship: Cardinality Constraint (many-to-many)

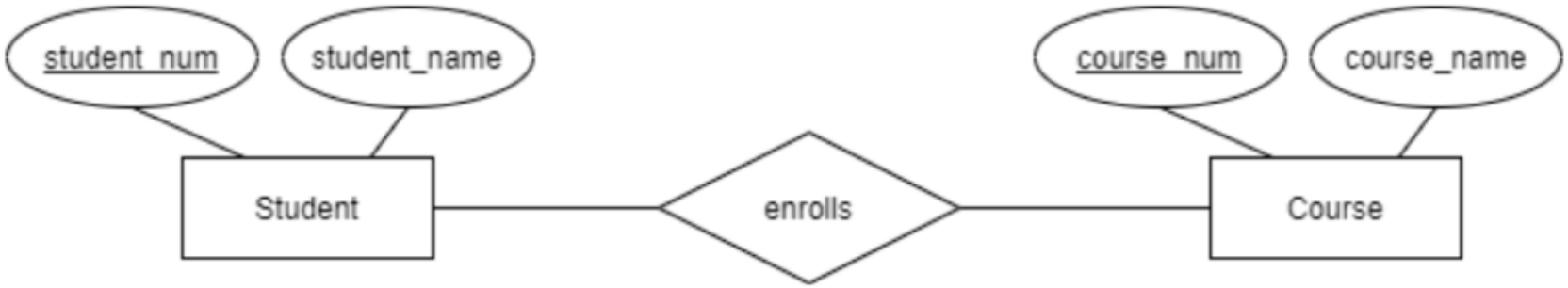


Figure: A many-to-many relationship between **Student** and **Course**

- **Student**{*student_num*, *student_name*}
- **Course**{*course_num*, *course_name*}
- **enrolls**{ *student_num*, *course_num* }

student_num	student_name
101	RAJ
102	SANAT

Student

student_num	course_num
101	CS101
102	CS101
102	MT110

enrolls

course_num	course_name
CS101	Computer Science
MT110	Mathematics

Course

Figure: Table: **Student**, **Course** and **enrolls**

Relationship: Cardinality Constraint (many-to-many) with Descriptive Attributes

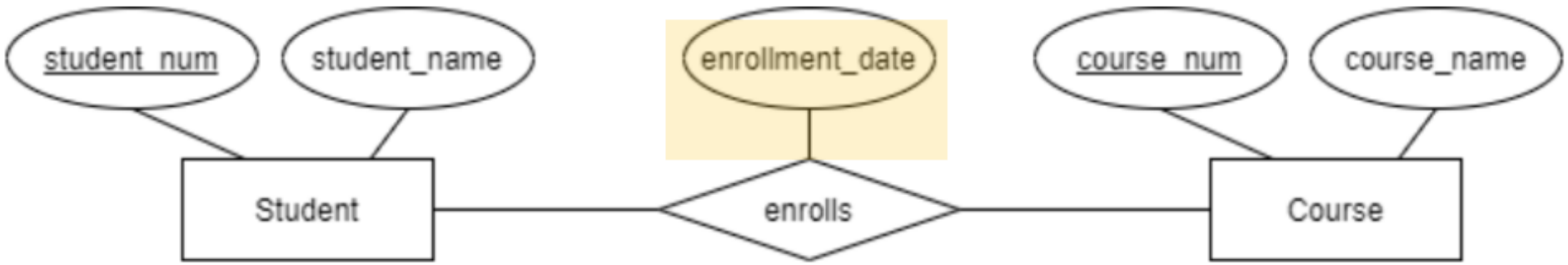


Figure: A many-to-many relationship between **Student** and **Course**

- **Student**{*student_num*, *student_name*}
- **Course**{*course_num*, *course_name*}
- **enrolls**{ *student_num*, *course_num*, *enrollment_date* }

Relationship: Cardinality Constraint (many-to-one)

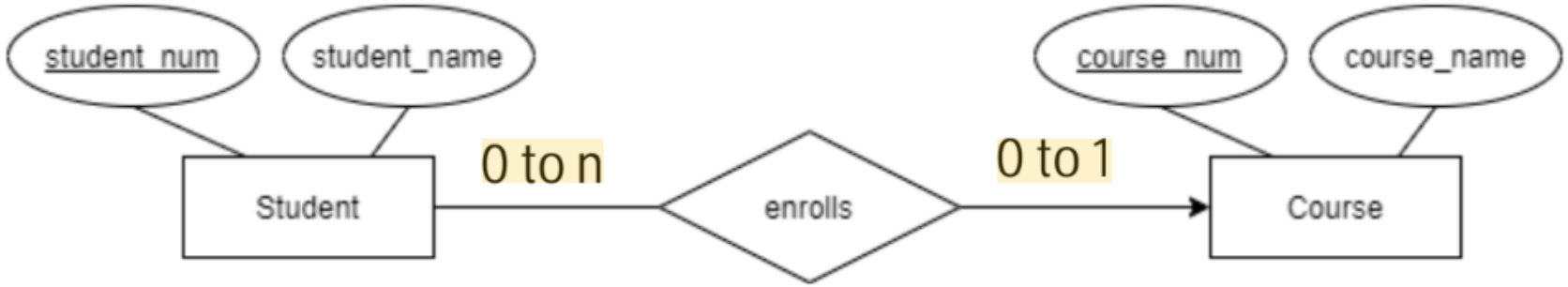


Figure: An many-to-one relationship between **Student** and **Course**

- **Student**{*student_num*, *student_name*, *course_num*}
- **Course**{*course_num*, *course_name*}

must be nullable

The primary key of one becomes a attribute of many.

Course_num(primary key of one) becomes attribute of student(many)

Relationship: Cardinality Constraint (one-to-one)

Figure: An one-to-one relationship between **Department** and **Manager**

- **Department**{*dept_num*, *dept_name*}
- **Manager**{*mgr_num*, *mgr_name*, *dept_num*}

OR

- **Department**{*dept_num*, *dept_name*, *mgr_num*}
- **Manager**{*mgr_num*, *mgr_name* }

Relationship: Cardinality Constraint (many-to-one) with Participation Constraint

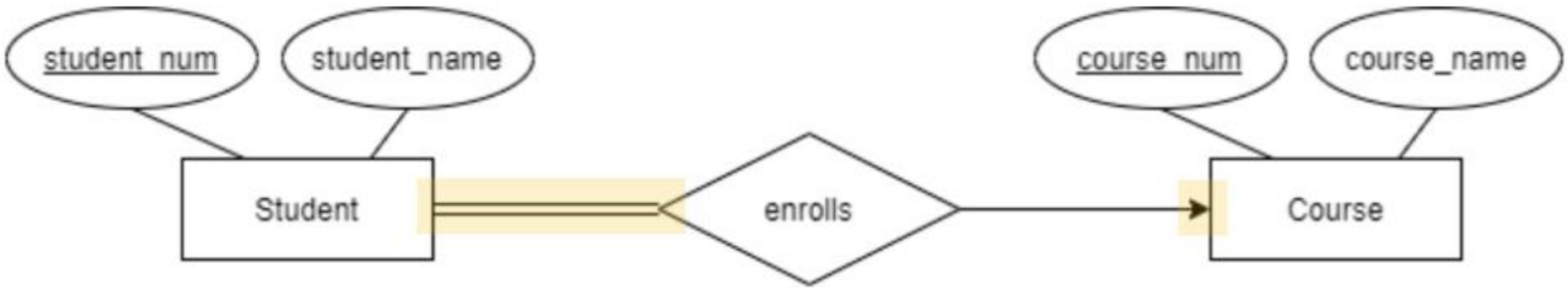


Figure: An many-to-one relationship between **Student** and **Course**

not null

- **Student**{*student_num*, *student_name*, *course_num*}
- **Course**{*course_num*, *course_name*}

Relationship: Cardinality Constraint (one-to-one) with Participation Constraint

Figure: A one-to-one relationship between **Manager** and **Department**

- **Mgr_Dept**{*mgr_num*, *dept_num*, *mgr_name*, *dept_name* }

Weak Entity Set

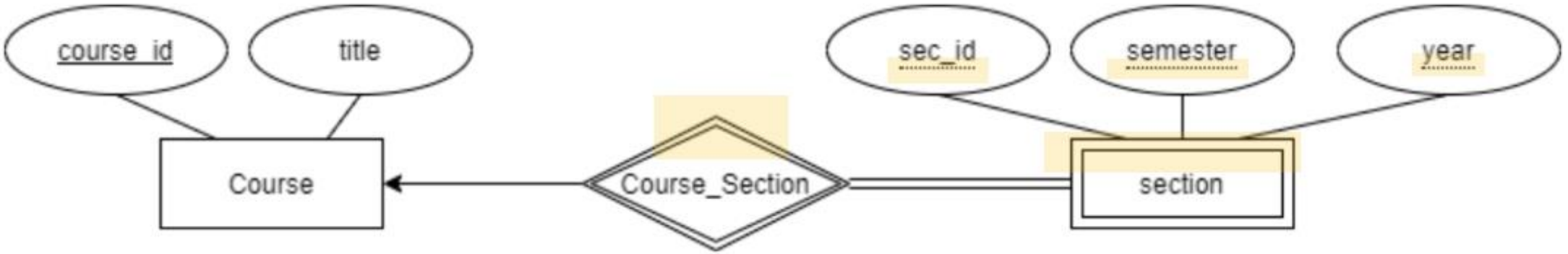


Figure: A relationship between **Course** and **Section**

- **Course**{*course_id*, *title* }
- **Section**{*course_id*, *sec_id*, *semester*, *year* }

Ternary relationship

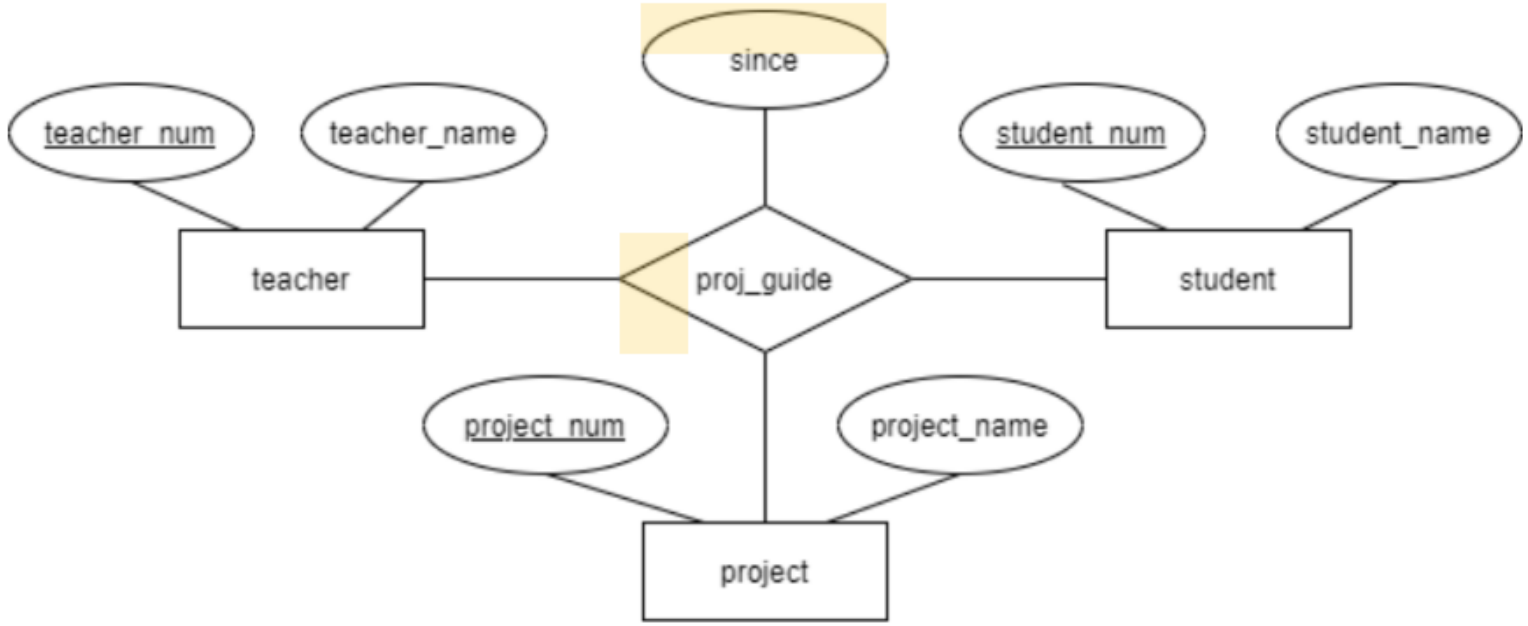


Figure: Example of ternary relationship

- **teacher**{*teacher_num*, *teacher_name* }
- **student**{*student_num*, *student_name* }
- **project**{*project_num*, *project_name* }
- **proj_guide**{*teacher_num*, *student_num*, *project_num*, *since* }

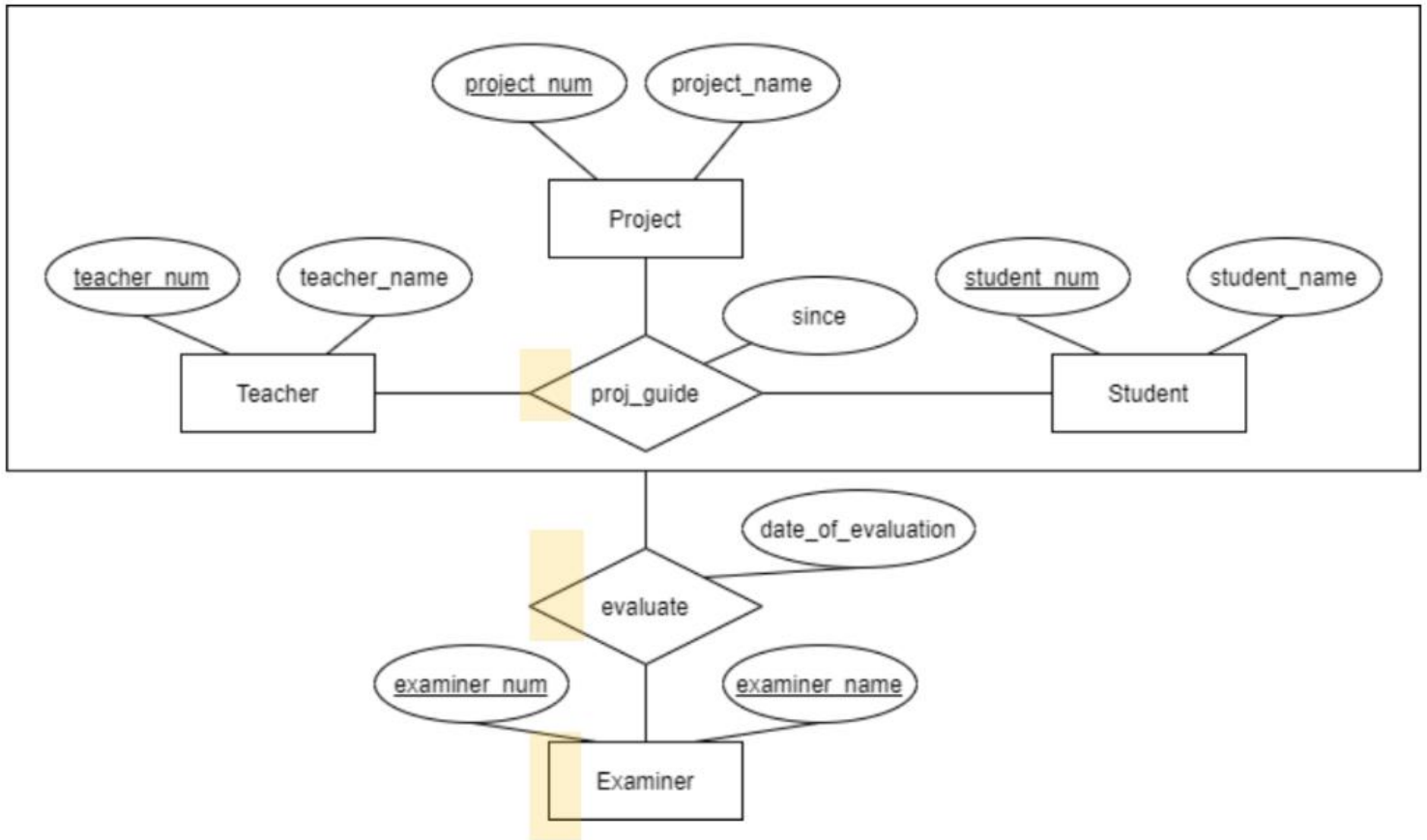
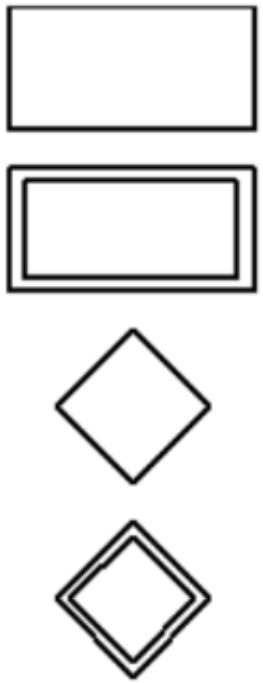


Figure: Example of Aggregation

- **proj_guide**{*teacher_num*, *student_num*, *project_num*, *since* }
- **Examiner**{*examiner_num*, *examiner_name* }
- **evaluate**{*teacher_num*, *student_num*, *project_num*, *examiner_num*, *date_of_evaluation* }

E-R diagrams symbols

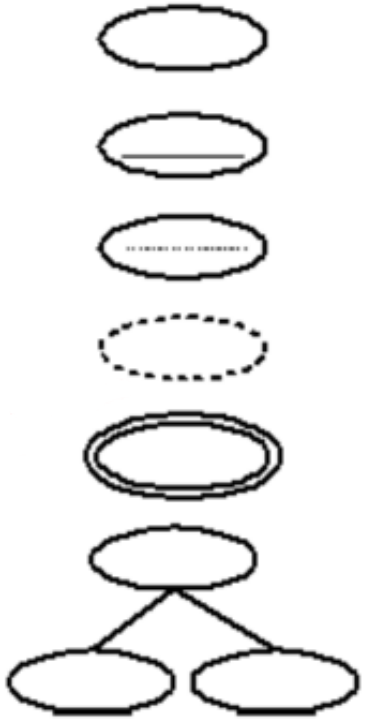


entity class

weak entity class

relationship type

identifying relationship type



attribute

key attribute

discriminator (partial key) attribute

derived attribute

multivalued attribute

composite attribute

DBMS Week 5

Redundancy

- Having multiple copies of same data in the database
- It leads to **anomalies**

Anomaly

- Inconsistencies that can arise due to data changes in a database with insertion, deletion, and update

Types of Anomalies

- Insertions Anomaly
- Deletion Anomaly
- Update Anomaly

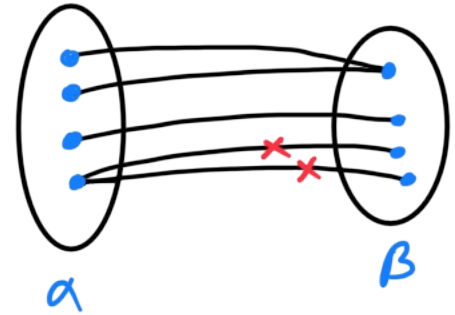
Redundancy and Anomaly

- Redundancy $\Longrightarrow$ Anomaly
- Dependency $\Longrightarrow$ Redundancy
- Good Decomposition $\Longrightarrow$ Minimization of Dependency
- Normalization $\Longrightarrow$ Good Decomposition

Decomposition → process of breaking down the database into smaller tables.

Normalization → systemic approach to organize data in the database.

Functional Dependencies



- It is a relationship between two sets of attributes.
- Require that the value for a certain set of attributes determines uniquely the value for another set of attributes
- Let R be a relational schema, $\alpha \subseteq R$ and $\beta \subseteq R$
- The functional dependencies or **FD** is $\alpha \rightarrow \beta$ holds on R if and only if for any legal relations $r(R)$, whenever any two tuples t1 and t2 of r agree on the attributes α, they also agree on the attributes β.

$$t1[\alpha] = t2[\alpha] \implies t1[\beta] = t2[\beta]$$

Example

StudentID	Semester	Lecture	TA
1234	6	Numerical methods	John
1221	4	Numerical methods	Smith
1234	6	Visual computing	Bob
1201	2	Numerical methods	Peter
1201	2	Physics II	Simon

- StudentID $\rightarrow$ Semester ✓
- {StudentID, Lecture} $\rightarrow$ TA ✓
- {StudentID, Lecture} $\rightarrow$ {TA, Semester} ✓

Trivial Dependencies

- A functional dependency is trivial if it satisfies below condition
- $\alpha \rightarrow \beta$ **is trivial if** $\beta \subseteq \alpha$

Example

- ID, name $\rightarrow$ ID
- name $\rightarrow$ name

Note

- Relationship between a set functional dependencies F and its closure F^+
- $F \subseteq F^+$

Armstrong's Axioms

- **Reflexivity:** if $\beta \subseteq \alpha$, then $\alpha \rightarrow \beta$ Trivial
- **Augmentation:** if $\alpha \rightarrow \beta$, then $\gamma\alpha \rightarrow \gamma\beta$
- **Transitivity:** if $\alpha \rightarrow \beta$ and $\beta \rightarrow \gamma$, then $\alpha \rightarrow \gamma$

o Additional Derived Rules →

- **Union:** if $\alpha \rightarrow \beta$ holds and $\alpha \rightarrow \gamma$ holds, then $\alpha \rightarrow \beta\gamma$ holds
- **Decomposition:** if $\alpha \rightarrow \beta\gamma$ holds, then $\alpha \rightarrow \beta$ holds and $\alpha \rightarrow \gamma$ holds

} RHS

- **Pseudotransitivity:** if $\alpha \rightarrow \beta$ holds and $\gamma\beta \rightarrow \delta$ holds, then $\alpha\gamma \rightarrow \delta$ holds

- Augmentation → $\gamma\alpha \rightarrow \gamma\beta$ (A)
- Transitivity $\alpha\gamma \rightarrow \gamma\beta$; $\gamma\beta \rightarrow \delta$

$\alpha\gamma \rightarrow \delta$ (T)

Closure of Attribute set

- The set of all those attributes which can be functionally determined from an attribute set is called as a closure of that attribute set.

Example

Consider a $R(A, B, C, G, H, I)$ and $FD = \{A \rightarrow B, G \rightarrow CH, BC \rightarrow GI\}$

- $(AC)^+ = AC$
- $(AC)^+ = ACB$
- $(AC)^+ = ACBGI$
- $(AC)^+ = ACBGIH$

Candidate Key

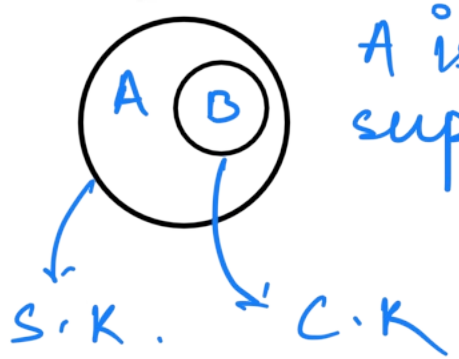
{roll no., name} ← prime attributes
or, {first-name, last-name} etc.

- A set of minimal attributes that uniquely identifies a tuple/row

• $(key)^+ = R$

Super Key

- Super set of any candidate key is Super Key



A is the superset of B.

Prime Attribute

- Attributes that are present in the candidate key is prime attribute

• C.K.?
attribute not part of the RHS of FDs, must be part of C.K.

Note

- Every Candidate key is a Super Key, $C.K \subseteq S.K$
- If closure of any attribute covers all the attributes in a relation, then it is a SuperKey and if it is minimal, then it is called Candidate key

attributes that are not part of C.K. → 'Non-Prime Attribute'

Formula's to find maximum number of Super Keys

- N attributes, only 1 candidate key(by single attribute) = 2^{N-1}
- N attributes,1 candidate key formed *k* by attributes = 2^{N-k}
- N attributes, each attribute is a candidate key = $2^N - 1$

- N attributes, 2 different candidate key(formed by one or many attribute)
  - Maximum number of super keys = (S.K with A1) + (S.K with A2) - (S.K with both A1 and A2)
  - Max no of Super Keys = $2^{N-m} + 2^{N-n} - 2^{N-(m+n)}$

Example

studentInfo{enrollment_num, class, section, roll, name}.
{enrollment_num} and *{class, section, roll}* are two possible candidate keys.

What is the maximum number of possible superkeys of studentInfo?

- Max no of Super Keys = $2^{N-m} + 2^{N-n} - 2^{N-(m+n)}$

$$= 2^{5-1} + 2^{5-3} - 2^{5-(1+3)}$$
$$= 2^4 + 2^2 - 2^1$$
$$= 16 + 2 - 2$$
$$= 18$$

Extraneous attribute

(Redundant)

Example for 1st condition (LHS)

- $F = \{A \rightarrow B, AB \rightarrow C\}$

Let's check B for is an extraneous attribute or not

Now, $F' = \{A \rightarrow B, A \rightarrow C\}$

$(AB - B)^+ = A^+ = ABC$

We got B attribute after taking an closure.

$\therefore$ B is an extraneous attribute

Example for 2nd condition (RHS)

- $F = \{A \rightarrow BC, AB \rightarrow CD\}$

Let's check C for is an extraneous attribute or not

Now, $F' = \{A \rightarrow BC, AB \rightarrow D\}$

$(AB)^+$ = AB^+ = $ABCD$

We got C attribute after taking an closure.

$\therefore$ C is an extraneous attribute

Canonical Cover / Minimal Cover

- A canonical cover is a set of functional dependencies that is equivalent to a given set of functional dependencies but is minimal in terms of the number of dependencies

Steps to find an canonical cover

1. Write all single attributes on right hand side Decomposition
2. Remove redundant functional dependencies by Armstrong axioms
3. Remove extraneous attributes

Example

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

$A \to B, A \to C$

~~A-->B~~ (by redundancy)

$B \to C$

~~A-->BC~~ (by redundancy, extraneous attribute)

Canonical cover is $A \to B, B \to C$

Equivalence of Functional Dependency

1. F covers G - $G \subseteq F^+$
    - All relation should satisfy on G while taking the closure of F
2. G covers F - $F \subseteq G^+$
    - All relation should satisfy on F while taking the closure of G

- If it satisfies both F is equivalent to G

Example

$F1 = \{P \to Q, Q \to R, R \to P\}$
$F2 = \{P \to R, Q \to R, R \to Q\}$

Let's check F1 covers F2

Taking Closure on F1

- $P^+ = PQR$
- $Q^+ = QRP$
- $R^+ = RPQ$

F1 covers F2 because by taking the closure on F1, all the relation satisfies on F2

Example (continued)

$F1 = \{P \to Q, Q \to R, R \to P\}$
$F2 = \{P \to R, Q \to R, R \to Q\}$

Let's check F2 covers F1

Taking Closure on F2

- $P^+ = PQR$
- $Q^+ = QR$
- $R^+ = RQ$

F2 not covers F1 because by taking the closure on R, it's not able satisfy the dependency $R \to P$ on F1

- # tuples and # columns in R_1, R_2, and R must be same.

Loseless Join Decomposition

- Original relation decomposed into smaller relations in such a way that it can be reconstructed through natural joins without any loss of information

*

- $R_1 \bowtie R_2 = R$
- $R_1 \cap R_2 \neq \phi$
- $R_1 \cup R_2 = R$
- $R_1 \cap R_2 \rightarrow R_1 \text{ or } R_2$

→ There must be some common attribute.

→ That particular attribute should give me R_1 or R_2.

→ must be a key of either R_1 or R_2.

- If all the above conditions satisfies, then it's a loseless decompostion

(opp. lossy)

Example

A relation $R(A, B, C, D)$ with functional dependencies
$FD = \{A \to B, B \to C, C \to D\}$ is decomposed into three relations as follows:

$R1(A, B), R2(B, C), R3(C, D)$

- $R_1 \cup R_2 \cup R_3 = R$

- $R_1 \cap R_2 = B$
- $B^+ = BCD$ (R_2 relation)

- Now, $R_{12}(A, B, C)$
- $R_{12} \cap R_3 = C$
- $C^+ = CD$ (R_3 relation)

Dependency Preservation

- When a relation is decomposed, it is important to ensure that functional dependencies that held in original relation still maintained.
- It helps to prevent anomalies and ensures data integrity
- $F \subseteq F^+$, F^+ should preserve all dependency in F

Note

- Loseless Decompostion and Dependency preservation are independent to each other.

Example

A relation $R(A, B, C, D)$ with functional dependencies $FD = \{A \rightarrow B, B \rightarrow C, C \rightarrow D, D \rightarrow A\}$ is decomposed into three relations as follows:

$R1(A, B), R2(B, C), R3(C, D)$

$F_1 = \{A \rightarrow B, B \rightarrow A\}$ based on R_1
$F_2 = \{B \rightarrow C, C \rightarrow B\}$ based on R_2
$F_3 = \{C \rightarrow D, D \rightarrow C\}$ based on R_3

Example (Continued)

$$F^+ = F_1 \cup F_2 \cup F_3$$

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

Now, $D \to A$ is missing F^+

Let's take D^+ closure on F^+

$$D^+ = DCBA$$

$\therefore D \to A$ dependency is preserved after decomposing it.

lossless + dependency preserving → 3NF

DBMS Week 6

- To check Normal Forms -
  1. Find out C.K.s .
  2. Find out P.A.s & N.P.A.s .

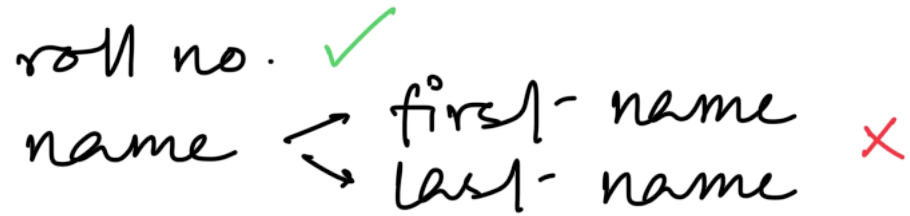
Normalization and Normal Forms

- Normalization or Schema Refinement is a technique of organizing the data in the database
- Normalization is used for mainly two purpose:
    - Eliminating redundant (useless) data
    - Ensuring data dependencies

- A normal form specifies a set of conditions that the relational schema must satisfy in terms of its constraints

1NF (First Normal Form)

- All Domains should only have Atomic Values
- No Multivalued attributes



Example

SID	Sname	Cname
S1	A	C, C++
S2	B	C++, DB
S3	C	DB

- *Cname* is a multivalued attribute. So, the above table is not in *1NF*

2NF (Second Normal Form)

Relation R is in Second Normal Form (2NF) only if

P.A. → N.P.A. ✗

- R is in 1NF and
- R contains no Partial Dependency

(full dependency)

proper subset of C.K → N.P.A. ✗ (non-unique)

Partial Dependency

$A \rightarrow B$ is a Partial dependency only if

- A - Proper subset of Candidate Key $A \subset C.Key$
- B - Non Prime Attribute $B \notin C.Key$

Example

CHECK WHETHER IT IS IN 2NF OR NOT

1 N F

Consider a relation $\mathbf{R}(A,B,C,D,E,F,G,H)$, where each attribute is atomic and the following functional dependencies hold:

$\mathcal{F}=\{AB \rightarrow CDE, D \rightarrow F, F \rightarrow GH, E \rightarrow AB\}$

C. K.

$(AB)^+ = ABCDEFGH$

↳ no partial dependency

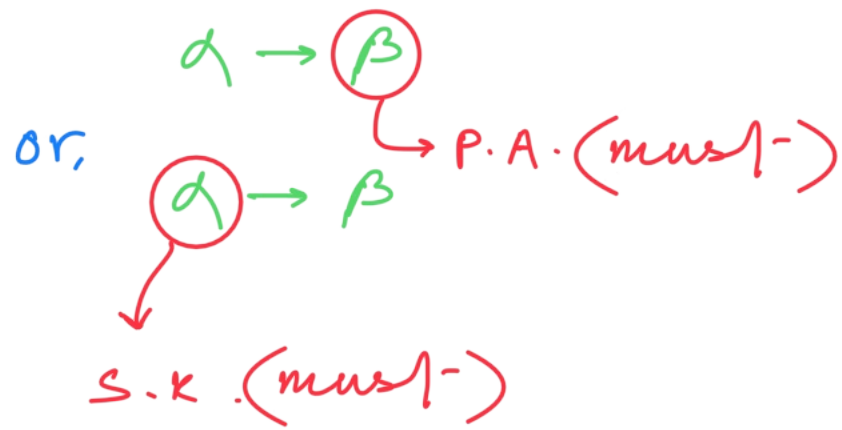
No transitive dependency

NPA → NPA ×

3NF (Third Normal Form)

A relational schema R is in 3NF if for every FD $A \rightarrow B$ associated with R either

- R is in 2NF and
- $B \subseteq A$ (Trivial FD) (or)
- A is a superkey of R (or)
- B is a prime attribute



Note:

- 3NF preserves both dependency preservation and loseless join decomposition

Example

Consider a relational schema **Contacts**($aadhaarNo, name, mobileNo, address$).Assume that all the attributes have atomic values. Which of the following functional dependencies is/are example(s) of the third normal form?

$\mathcal{F} = \{aadhaarNo \rightarrow (name, address),$
$mobileNo \rightarrow aadhaarNo,$
$(name, address) \rightarrow mobileNo\}$

$\mathcal{F} = \{aadhaarNo \rightarrow name,$
$mobileNo \rightarrow aadhaarNo,$
$(name, address) \rightarrow aadhaarNo\}$

$\mathcal{F} = \{(aadhaarNo, name) \rightarrow address,$
$mobileNo \rightarrow name,$
$(name, aadhaarNo) \rightarrow mobileNo\}$

$\mathcal{F} = \{(aadhaarNo, name) \rightarrow (address, mobileNo),$
$mobileNo \rightarrow aadhaarNo,$
$name \rightarrow address\}$

BCNF (Boyce – Codd Normal Form)

A relational schema R is in BCNF if for every FD $A \rightarrow B$ associated with R either

- R is in 3NF and
- A must be a superkey (or)
- $B \subseteq A$ (Trivial FD)

(super key)$^+$ → N.P.A. / P.A.

Note:

- BCNF decomposition is loseless
- BCNF decomposition is may or may not be dependency preserving

Example

Consider the relational schema $\mathbf{R}(U, V, W, X, Y, Z)$ where the domain of every attribute consists of atomic values. The set of functional dependencies for the relation $\mathbf{R}$ is given as follows:

$$\mathcal{F} = \{UV \rightarrow W, W \rightarrow X, X \rightarrow VY, Y \rightarrow Z, Z \rightarrow U\ \}$$

What is the highest normal form of the given relation $\mathbf{R}$?

MVD (Multivalued Dependency)

Let R be a relation schema and $\alpha \subseteq R$ and $\beta \subseteq R$. The multivalued dependency

$\alpha \twoheadrightarrow \beta$

holds on R if in any legal relation $r(R)$, for all pairs for tuples $t1$ and $t2$ in r such that $t1[\alpha] = t2[\alpha]$, there exist tuples t3 and t4 in r such that:

- $t1[\alpha] = t2[\alpha] = t3[\alpha] = t4[\alpha]$
- $t1[\beta] = t3[\beta]$ and $t2[\beta] = t4[\beta]$
- $t2[R–\beta] = t3[R–\beta]$ and $t1[R–\beta] = t4[R–\beta]$

For MVD, ⦿ single value of A → more than one value of B exists.

⦿ Total number of attributes should be more than two .

⦿ If there exist 3 attributes, then 2 attributes must be independent of each other .

MVD Theory

Name	Rule
Complementation	If $X \twoheadrightarrow Y$, then $X \twoheadrightarrow (R - (X \cup Y))$
Augmentation	If $X \twoheadrightarrow Y$ and $W \supseteq Z$, then $WX \twoheadrightarrow YZ$
Transitivity	If $X \twoheadrightarrow Y$ and $Y \twoheadrightarrow Z$, then $X \twoheadrightarrow (Z - Y)$
Replication	If $X \to Y$, then $X \twoheadrightarrow Y$ but the reverse is not true
Coalescence	If $X \twoheadrightarrow Y$ and there is a W such that $W \cap Y$ is empty, $W \to Z$ and $Z \subseteq Y$, then $X \to Z$

Example

SId	Sname	Course	Instructor	Inst_Room
ME1001	David	Python	MK Singh	503
ME1001	David	Java	SN Joseph	505
ME1001	David	Python	SN Joseph	505
ME1001	David	Java	MK Singh	503

- {SId, Sname} $\twoheadrightarrow$ Course$
- SId $\twoheadrightarrow$ {Instructor, Inst_Room}

Example

$\alpha \twoheadrightarrow \beta$: i. keep the α same.
ii. interchange β value b/w any two tuples.

* resulting rows must be present in the table

↓

MVD exists.

course_name	instructor	book	edition
DBMS	Geeta	DBMS-Beginner	3
DBMS	Arjun	DBMS-Beginner	3
DBMS	Geeta	DBMS-Expert	2
DBMS	Arjun	DBMS-Expert	2
Java	Rahul	Java-Beginner	5
Java	Rahul	Java-Intermediate	3
Java	Rahul	Java-Expert	4
Java	Armaan	Java-Beginner	5
Java	Armaan	Java-Intermediate	3
Java	Armaan	Java-Expert	4

- course_name $\twoheadrightarrow$ instructor ✗
- course_name $\twoheadrightarrow$ {book, edition} ✓

Trivial MVD

A MVD $X \twoheadrightarrow Y$ in R is called a trivial MVD is

- Y is a subset of X $(Y \subseteq X)$ (or)
- $X \cup Y = R$

Example

- $AB \twoheadrightarrow B$ (trivial MVD)

4NF (Fourth Normal Form)

A relation schema R is in 4NF if and only if the following conditions are satisfied

- R is in BCNF and
- Should not have any multi-valued dependency

* R(A, B) → in 4 NF.
(relation w/ 2 attributes)

week-7 : Python DB Connection →

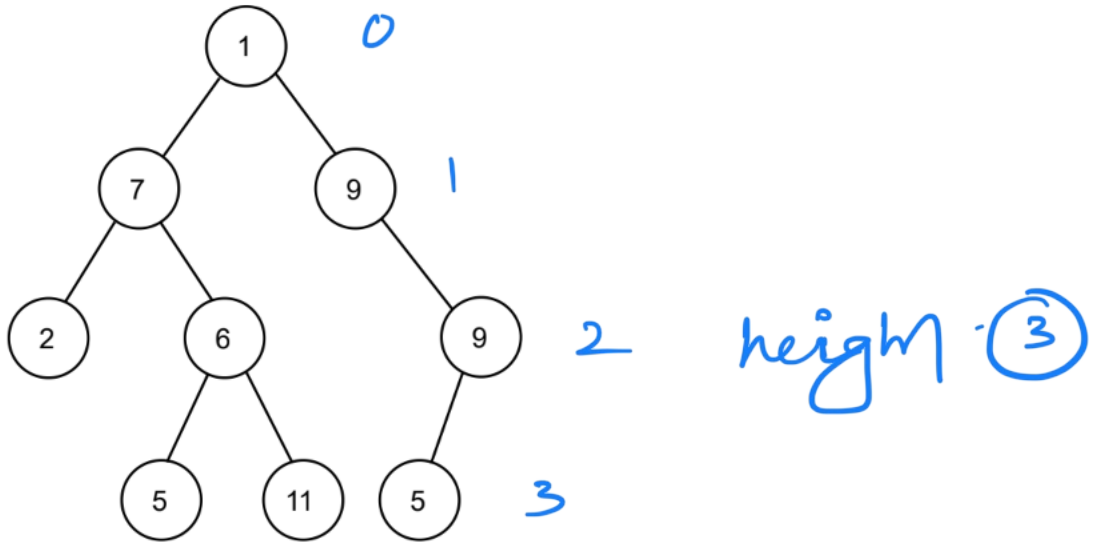
1. Create the connection
2. Create a cursor
3. Execute the query
4. Commit / Roll back
5. Close cursor
6. Close connection

Week 8 DBMS

i. cursor.fetchone() → 1 tuple
ii. cursor.fetchmany() → list of tuples → by default, takes 1 as the parameter
iii. cursor.fetchall() → list-of-tuples

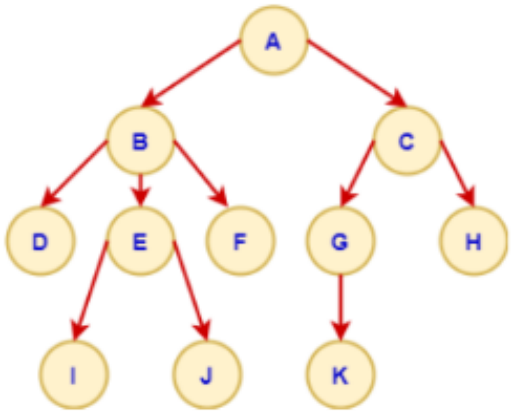
* [week-2 SQL]

Tree



- **Internal Nodes** - The node which has at least one child is called internal Node
- **Subtree** - Subtree represents the tree rooted at that node
- **Siblings** - Nodes having the same parents
- **Arity** - Number of children of a node
- **Height** - Maximum level in a tree

Arity of the tree = Maximum arity of a node



The root node is said to be at Level 0 and the children of the root node are at Level 1 until and unless root is specified to exist at level 1

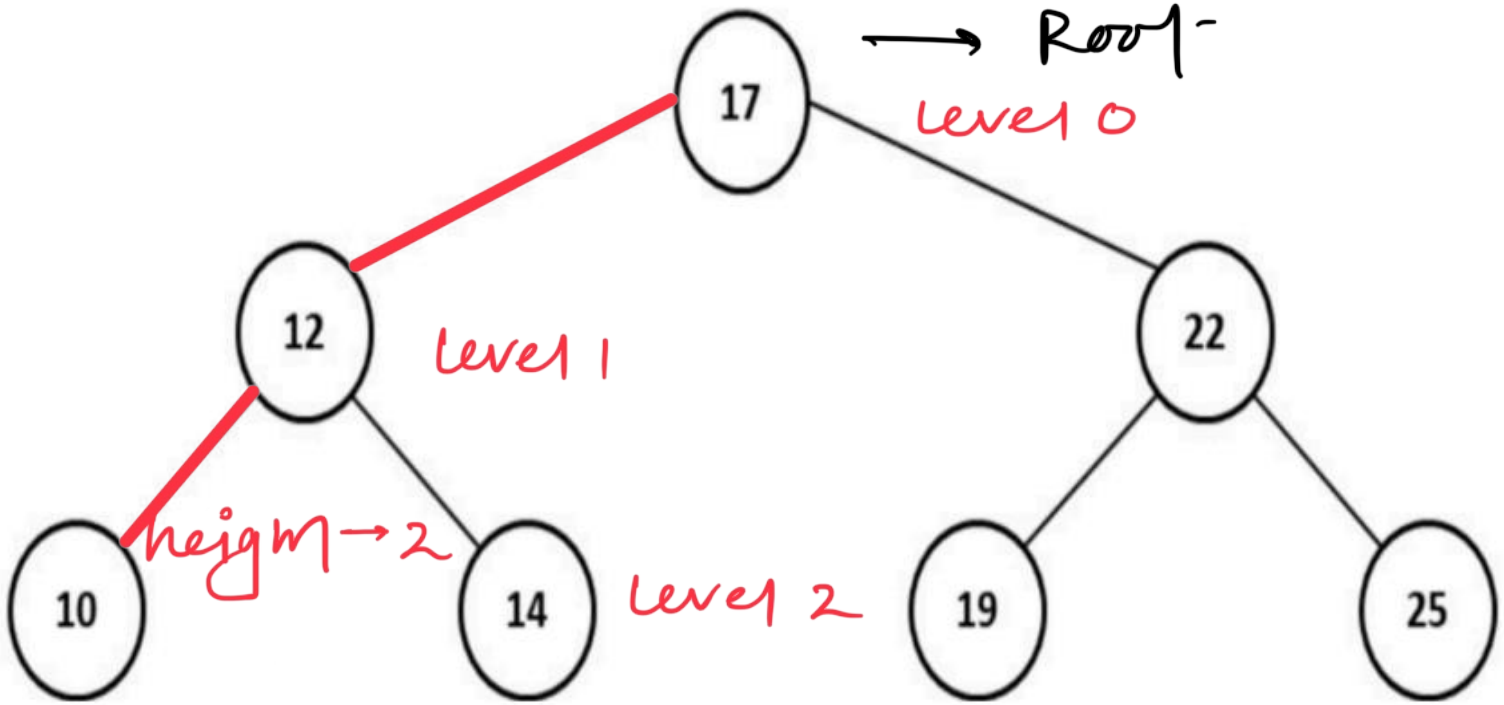
- **Fact 1**: A tree with n nodes has $n-1$ edges
- **Fact 2**: The maximum number of nodes at level l of a binary tree is 2^l.
- **Fact 3**: If h is the height of a binary tree of n nodes, then:
  - $h+1 \leq n \leq 2^{h+1}-1$
  - $\lceil \lg(n+1) \rceil - 1 \leq h \leq n-1$
  - $O(\lg n) \leq h \leq O(n)$
  - For a k-ary tree, $O(\lg_k n) \leq h \leq O(n)$

Binary Search Tree

- left subtree < Root < right subtree

Consider a data: 17, 22, 12, 10, 14, 25, 19

- height = most no. of edges in any Path



Total no. of elements $= 2^{height+1} - 1$

Storage Heirarchy

not relevant except the image

Primary Storage

- Volatile
- Very fast

Secondary Storage

- Non-volatile
- Moderately fast

Tertiary Storage

- Non-volatile
- Very slow

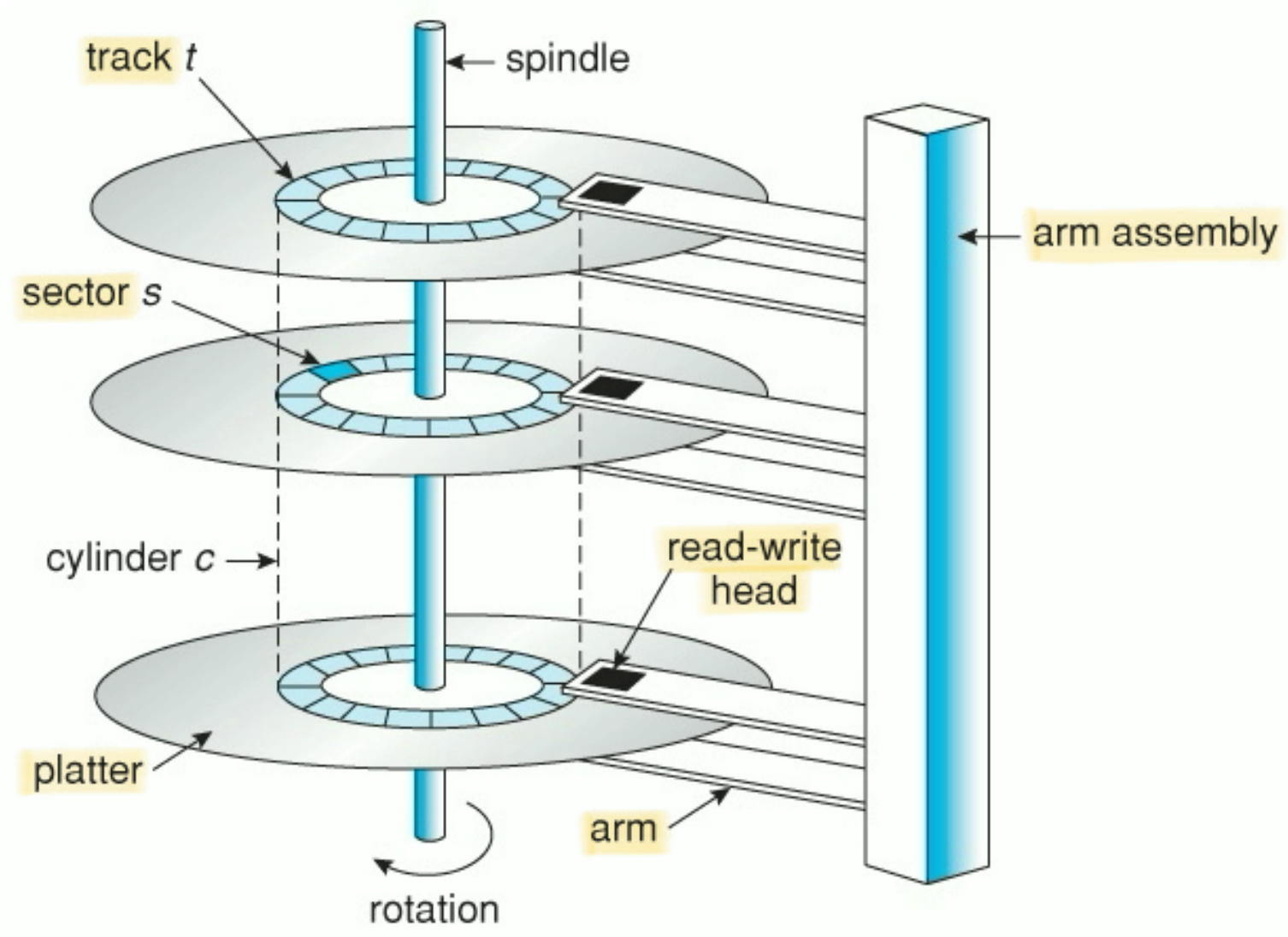


Fig - Magnetic disk

Magnetic Disk

- **Access Time:** time from a read or write request issue to start of data transfer:
- **Seek Time:** time to reposition the arm over the correct track
- **Rotational Latency:** time for the sector to be accessed to appear under the head
- **Data-transfer Rate:** the rate at which data can be retrieved from or stored to the disk

- Access time = seek time + Rotational latency
- $Access\ time = seek\ time + \frac{1}{2} \times time\ period$

↳ time required for a complete revolution.

Formula's for Numerical Problems

- Capacity of disk = Total no of sectors x sector size
- $Time\ period = \frac{1\ minute}{Rotational\ speed}$
- $Transfer\ rate = \frac{\text{bytes of one track}}{time\ period}$
- $Transfer\ time = \frac{File\ size}{transfer\ rate}$
- Number of cylinders = Number of tracks/surface
- Min no of bits required to address all sectors = $\lceil log_2(no\ of\ sectors) \rceil$

'Buffer Replacement- Policy (LRU)'

Least-Recently Used