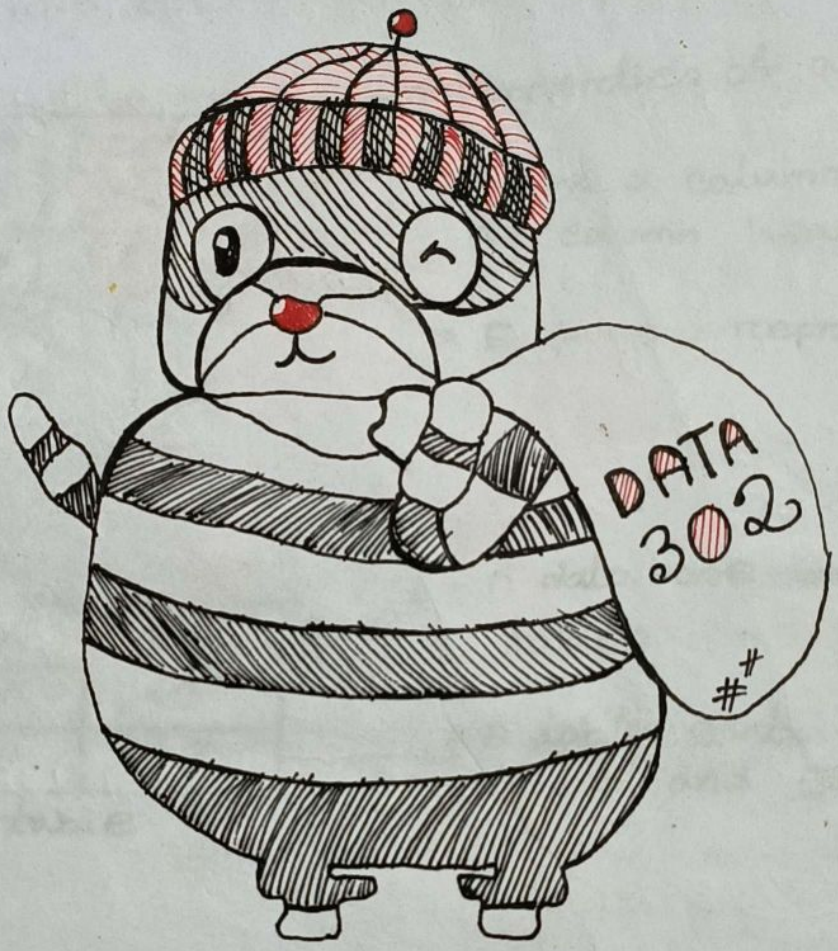
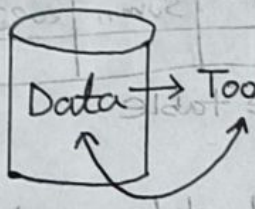


Database System

Sunday / Ch-1

Database: A place to store data.

Database system: A collection of interrelated data and a set of tools/applications to access those data.



→ MySQL (My Sequel)
→ Oracle [Oracle XE]
→ Microsoft SQL server
→ Postgresql (Open-source)

ID	Name	Dept	Blood
2021-01	Alve	CSE	B+
2021-02	Alice	EEE	A+

Student table

→ Properties of a student
→ Forms a column which is known as column header
→ Each row represents a student

Dept	Ext	Office Room
CSE	206	640
EEE	100	443

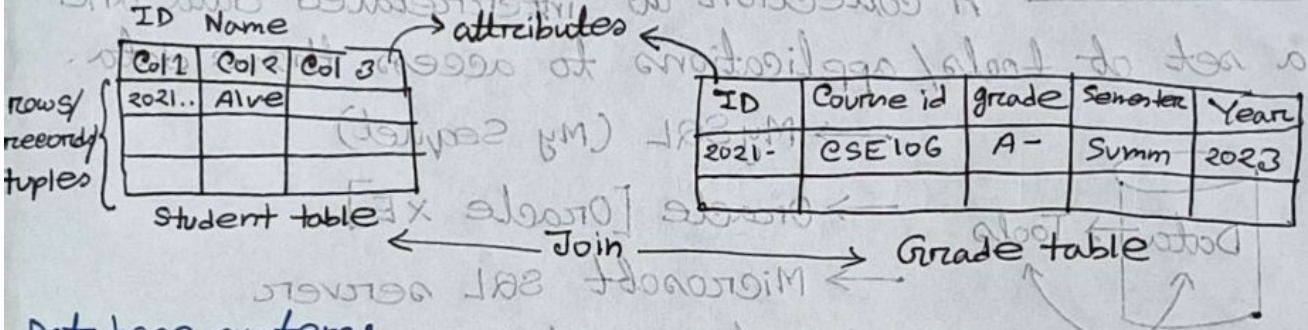
Dept table

- A database consist of multiple tables
- A table contains several rows and columns.

Sunday

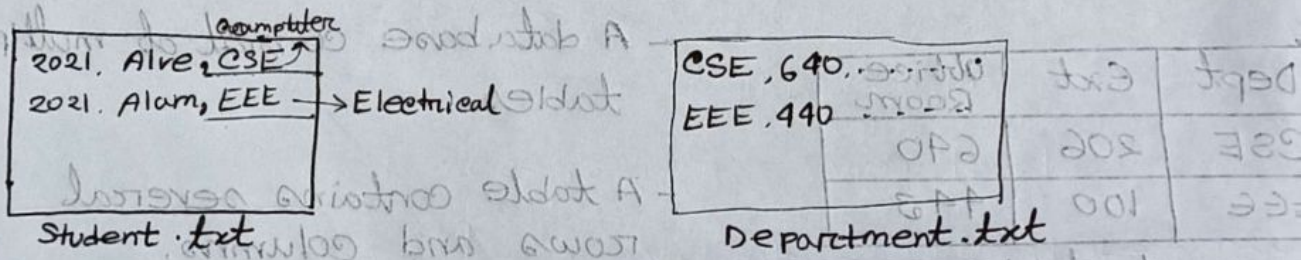
Chapter 01: Intro to Database systems.

1970 → Database invented by E.F. Codd who proposed relational data model



Database system: A collection of interrelated data and a set of programs/tools to access and possibly manipulate those data.

→ Data redundancy and Inconsistency
→ Difficulty in accessing data.
→ Data Isolation
→ Integrity problems



Integrity problem:

```
CREATE TABLE Emp(
    emp_Id  Number,
    empName  VARCHAR2(10),
    empSalary  NUMBER,
    PRIMARY KEY (emp_Id),
CHECK (empSalary >=0)
);
```

→ Condition check করবে যে Salary 0 থেকে বড় / সমান কিনা।

↳ CHECK constraints

→ Atomicity of updates. → comes from atom

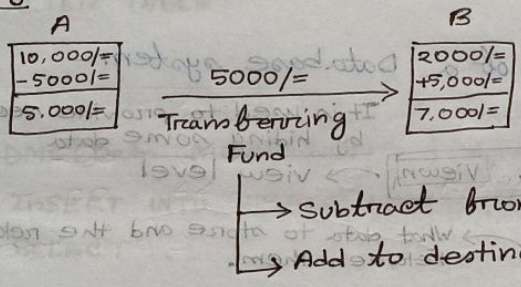
→ Concurrent access by multiple users (Parallel / Simultaneous)

Concurrency control protocols

↳ This protocol controls concurrent access so that the database remains consistent.

→ Security problems.

Atomicity:



** Database always provides "Consistency" even if there is a failure.

Fund
→ Subtract from source
→ Add to destination.

"Either all operations associated to a transaction will be executed or none of them will be executed."

* security:

ID	Name	Salary

Instructor table

Views: A virtual table. Multiple views can be created on a single table.

```
CREATE VIEW V AS
(SELECT id, name From instructor);
```

ID	Nam

10.10.23 / CSE 302
Tuesday / Ch-3

View of Data:

Abstract architecture of a Database system

It is used to provide security by hiding some data

view 1 | view 2 | ... | view n → view level

Logical Level → What data to store and the relationship between them.

Select * From student →

Physical data independence:

Physical level → How to store the data

* file system (NTFS, FAT32, Ext 2, Ext 3)
* Data structure (Linked list)
* Quota on disk storage (space allocation)

Physical schema:

design at physical level.

- Database schema: The overall database design at the logical level.
- Relational schema: The design/structure of a relation.
  Student (id, name, dept, total_credit)

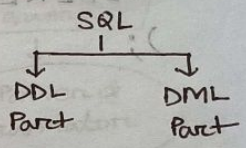
→ Database Instance: The content of the database at a particular point in time.

→ Relational Instance: The content of a relation at a particular point in time.

* Instances can be changed over time.

- SQL (Structured Query Language)

• DDL (Data Definition Language) [Deals with Schema]
- CREATE TABLE . . .
- CREATE VIEW . . .
- ALTER TABLE



• DML (Data Manipulation Language) [Deals with Instance]
- INSERT INTO, UPDATE, DELETE From
- SELECT . . .

• Integrity Constraints -
- PRIMARY KEY
- FOREIGN KEY
- UNIQUE constraints
- CHECK constraints
- NOT NULL (Mandatory field)
  * name VARCHAR2 (20) NOT NULL

(Candidate keys)
If there are multiple attributes which are unique → the rest

* CREATE TABLE Student (
  PRIMARY KEY (ID), Alternate keys
  UNIQUE (NID),); Attr1

child table → Student (id, name, dept_name, creadit)

(Parent table) department (dept_name, building, budget)

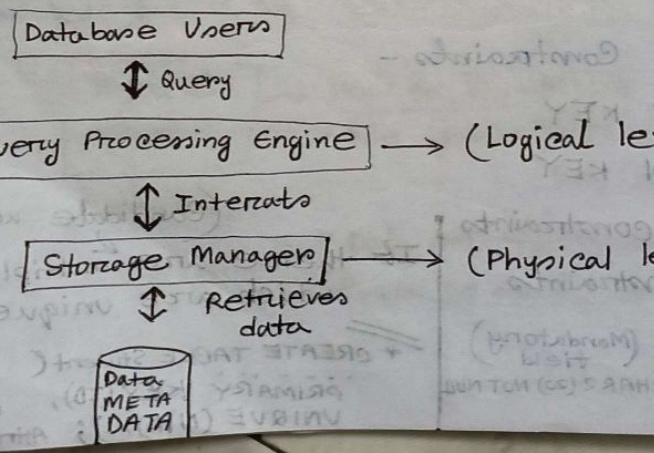
~~CREATE TABLE~~

* **Foreign key:** If an attribute A is primary key of a relation S and it also appears in relation R. The A is foreign key in R.

```
CREATE TABLE student (
    . . . . . .
    primary key (id),
    FOREIGN KEY (dept_name)
    REFERENCES department
);
```

* **Referential Integrity:** The attribute A in R must have values which also appear in S.

Components of a Database system:



* **Storage Manager:** A module that provides an interface between users executing queries and the data residing in disk storage.

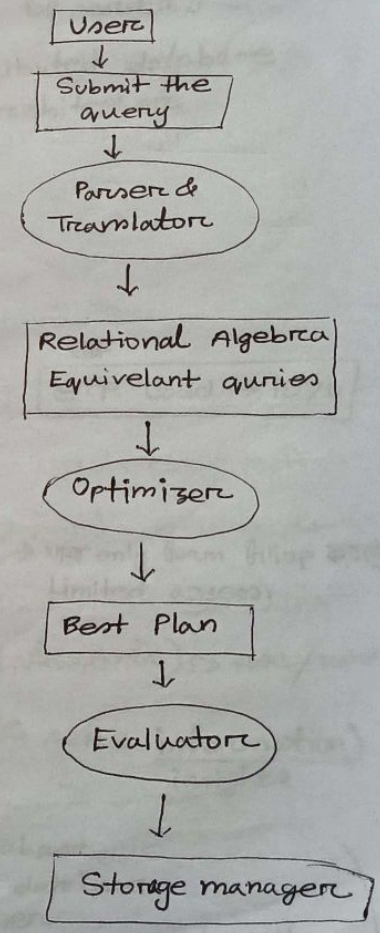
→ Authentication manager and Integrity.

→ Transaction manager (Atom

→ File manager

→ Buffer manager

* **Query Processing Engine:**



16.10.23 / CSE 302
Sunday / Ch-3

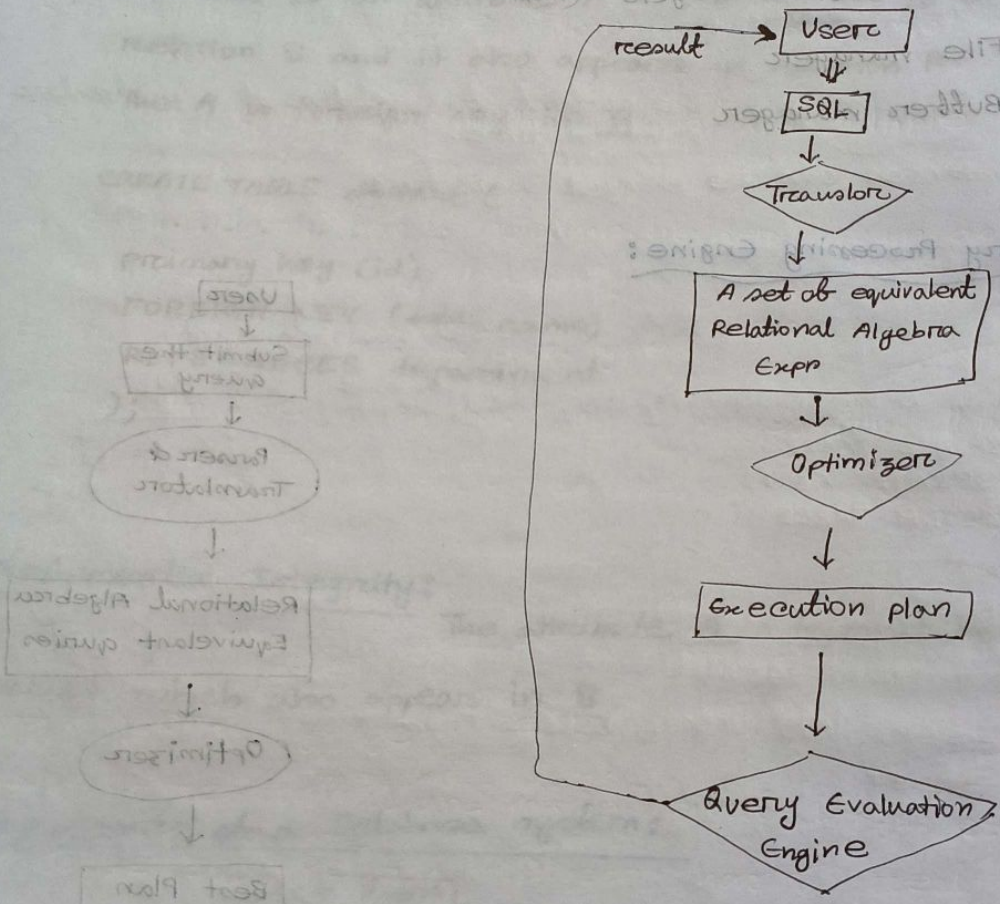
SELECT TABLE_NAME
FROM USER_TABLES;

↓

User এর create করা সব Table এর নাম Show করবে।

Part of data dictionary

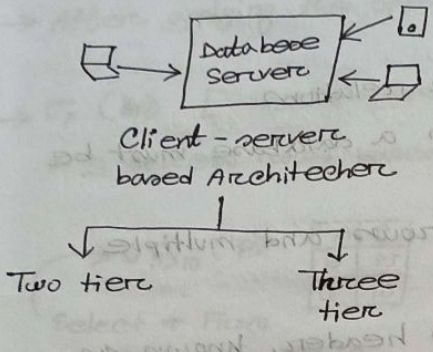
data dictionary
contains meta data
(data about data)



Query processor diagram

Database Architecture:

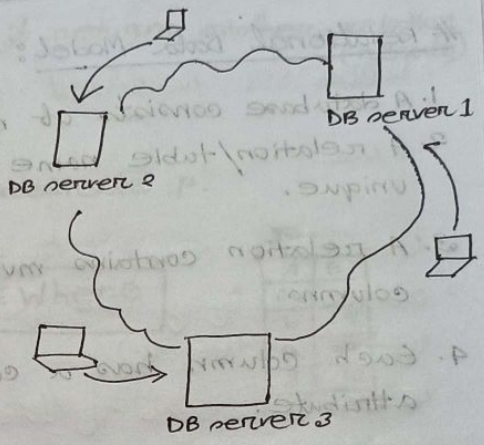
* Two tier and three tier architectures



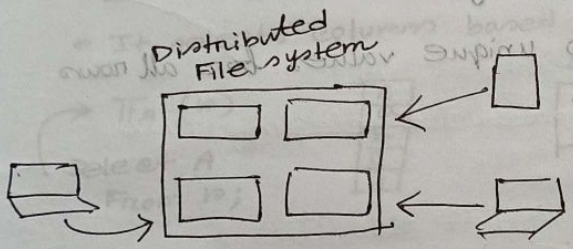
Client - server based Architecture

Two tier

Three tier



Distributed database Architecture



E.F. Codd → 1970

Database Users:

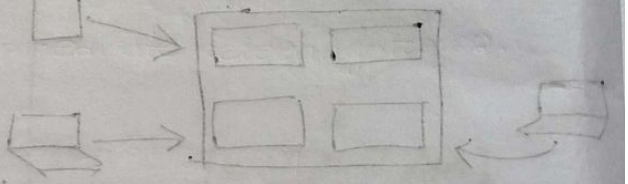
1. Naive Users: (Data entry operators) → যারা only form fillup করবে। Limited access.
2. Application Programmers: (Write SQL statements) → JDBC / coders
3. Data analyst: (Analyze the data to generate some information) insights
4. Database administrators: (DBA) (main role)
   - → Create the database
   - → Maintain the database
   - → Create the user
   - → Create user access control policy

Chapter - 2

Intro to Relational Model

Relational Data Model:

1. A database consists of multiple relations.
2. A relation/table name within a database must be unique.
3. A relation contains multiple rows and multiple columns.
4. Each column has a coloumn header, known as attribute.
5. An attribute has a data type.
6. The attribute which has unique values for all rows known as primary key.

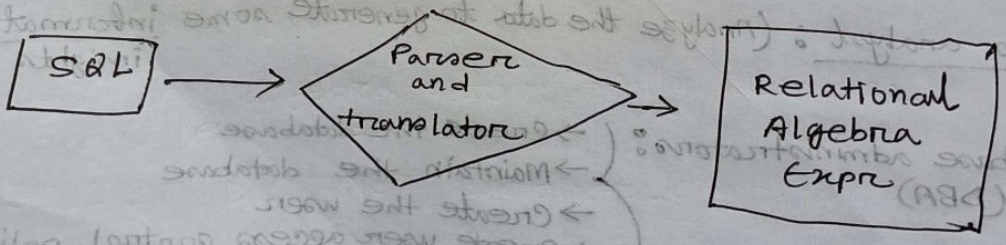


Relational Algebra:

→ A query language.

→ A more formal and mathematical language.

→ Unlike SQL, Relational Algebra has no real implementation



Relational Algebra operators:

→ Operators are applied over relations.

→ After applying the operators, it returns another relation (result)

→ $\sigma_p(r)$ [selection operator]

* It selects rows based on predicate P.

→ $\sigma_{B>10}(r)$

Select * From r where B>10

A	B
b	15
c	13

** σ ≅ Where

r

A	B
a	10
b	15
c	13
d	10

→ $\Pi_L(r)$ [Projection]

* It selects columns based on list L

→ $\Pi_A(r)$

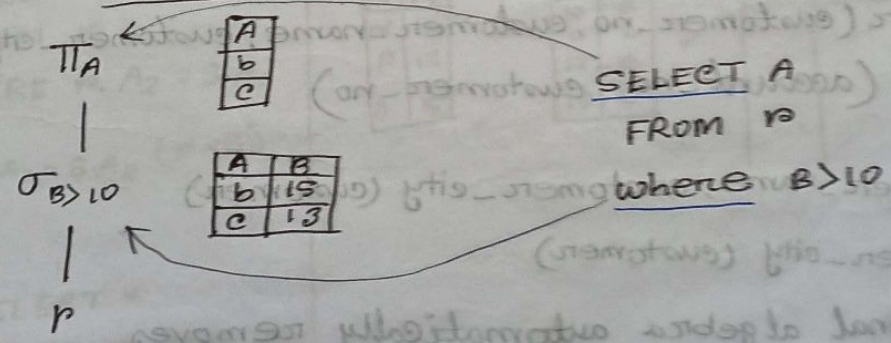
Select A From r;

A
a
b
c
d

** Π ≅ SELECT

$\Pi_A(\sigma_{B>10}(r))$

Expression Tree



17.10.23 / Tuesday / CSE302 Cls-4

Relational Algebra Expression:

→ A query language.
→ A more formal and mathematical language.
→ Unlike SQL, Relational algebra has no real implemetation.

SQL → Parsers or translators → Relational Algebra Expression

→ Operators are applied over relations.
→ After applying the operators, it return another relation (result).
→ σ_p (r) It is used by the query processing engine.
→ Relational Algebra Expr : SQL vice versa.

1. σ (Select / Selection) operator ≅ WHERE
   $\sigma_p(r)$ where p is the predicate selecting rows.

2. π (Project / Projection) Operator ≅ SELECT
   $\pi_L(r)$ where L is the list of selecting columns attribute.

Account (account_no, balance)
customer (customer_no, customer_name, customer_city)
Dipositor (account_no, customer_no)

1. $\pi_{customer\_name, customer\_city}$ (customer)
2. $\pi_{customer\_city}$ (customer)
   → Relational algebra automatically removes duplicate values.

```
SELECT DISTINCT customer_city
FROM customer;
```

3. $\pi_{account\_number}(\sigma_{balance>7000}(Account))$

$\pi_{account\_number}$

|
Act_no
A-101

$\sigma_{balance>7000}$

|

A-10ϕ	12000

Account

A-101	12000
A-102	6000
A-103	2500

4. $\pi_{customer\_name,\ customer\_no.}(\sigma_{customer\_city = 'Khulna'}(customer))$

5. $\pi_{customer\_no\ customer\_name}(\sigma_{customer\_city\ !='Dhaka'}(customer))$

Operation 3 (JOIN):

$r \times s$ (r cross product s)

```
SELECT *
FROM r, s
WHERE r.A2 = s.A2
```

$\sigma_{r.A_2 = s.A_2}(r \times s)$

or,

```
SELECT *
FROM r JOIN s
ON r.A2 = s.A2
```

r

A_1	A_2
α	10
β	12
δ	15

x s

A_2	A_3
10	a
11	b
12	a
15	c

=

r.A_1	r.A_2	s.A_2	s.A_3

vi) $\Pi_{customer\_name,\ customer\_city}(\sigma_{account.account\_no = depositor.account\_no\ AND\ depositor.customer\_no = customer.customer\_no\ AND\ balance > 7000}(account \times depositor \times customer))$

```
SELECT customer_name, customer_city
FROM (account JOIN depositor
ON Account.account_no = depositor.account_no)
JOIN customer
ON customer.customer_no = depositor.customer_no
WHERE balance > 7000;
```

#4th operator: Natural JOIN (⋈)

→ a variant of cross product / JOIN

$r \bowtie s \equiv \sigma_{r.A2 = s.A2}(r \times s)$

→ the join condition is IMPLICIT

$\Pi_{customer\_name,\ customer\_city}(\sigma_{balance>7000}(account \bowtie depositor \bowtie customer))$

```
SELECT
FROM (account NATURAL JOIN depositor)
NATURAL JOIN customer
WHERE balance >7000;
```

SET Operation :

5. SET UNION ($r \cup s$) $= \{t \mid t \in r \text{ or } t \in s\}$
6. SET Intersection ($r \cap s$) → INTERSECT
7. SET difference ($r - s$) → MINUS

Two conditions -

1. Both relations must have the same number of columns (arity).
   → number of rows ≈ cardinality
2. For ith column of both relations, the data type must be compatible.

Write an expression that lists account_no and customer_number.

$\Pi_{account\_no}(account) \cup \Pi_{customer\_no}(customer)$

A-101
A-102
A-103

=

C-101
C-102

A-101
A-102
A-103
C-101
C-102

SQL

```
SELECT account_no
FROM account
UNION
SELECT customer_no
FROM customer;
```

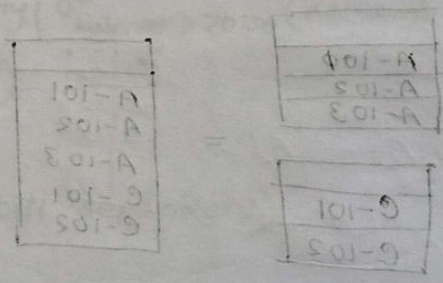
Find instructor id and name who taught in both fall 2009 and spring 2010 semester.

$$\Pi_{id}\left(\sigma_{\substack{semester='Fall' \\ year=2009}} \text{and } (teaches)\right)$$

$$\Pi_{id}\left(\sigma_{\substack{semester='Spring' \\ year=2010}} \text{and } (teaches)\right)$$

SQL:

```
SELECT id,
FROM teaches
WHERE semester = 'Fall' AND year = 2009
INTERSECT
SELECT Id
FROM Teaches
WHERE semester = 'Spring' AND year = 2010;
```



22.10.23 / CSE302 Ch-5
Sunday

*Cartesian Product / Natural Join —

→ r × s

r.A_1	r.A_2	s.A_2	s.A_3
α_1	β_1	β_1	γ_1
α_1	β_1	β_2	γ_1
α_1	β_1	β_3	γ_2
α_1	β_1	β_4	γ_2
α_2	β_2	β_1	γ_1
α_2	β_2	β_2	γ_1
α_2	β_2	β_3	γ_2
α_2	β_2	β_4	γ_2
α_3	β_1	β_1	γ_1
α_3	β_1	β_2	γ_1
α_3	β_1	β_3	γ_2
α_3	β_1	β_4	γ_2

```
Select *
From r, s;
```

```
Select *
From r JOIN s;
```

But এগুলোতে r এবং s এর mismatch থাকে।

$\sigma_{r.A_2 = s.A_2}(r \times s)$

```
SELECT *
FROM r, s
WHERE r.A2 = s.A2;
```

→ r ⋈ s

A_1	A_2	A_3
α_1	β_1	γ_1
α_2	β_2	γ_1
α_3	β_1	γ_1

r

A_1	A_2
α_1	β_1
α_2	β_2
α_3	β_1

s

A_2	A_3
β_1	γ_1
β_2	γ_1
β_3	γ_2
β_4	γ_2

```
SELECT *
FROM r NATURAL JOIN s;
```

(Join condition is implicit.)

SET:

r ∪ s ⟶
```
SELECT * FROM r
UNION
SELECT * FROM s;
```

r ∩ s ⟶
```
SELECT * FROM r
INTERSECT
SELECT * FROM s;
```

r − s ⟶
```
SELECT * FROM r MINUS
SELECT * FROM s
```

Aggregate Queries:

avg()
sum()
min()
max()
count()
median()

avg:

** Calculate average credits completed by all students.

SQL
```
SELECT avg (tot_cred) AS Avg_credit
FROM student;
```

$A_1, A_2, A_3\ \mathcal{G}\ F_1(A_1), F_2(A_2) \ldots F_n(A_n)\ (r)$

Here,
F_1, F_2, A_3 → aggregate function
A_1, A_2, A_3 → attributes of r
→ Grouping attribute

Relational: $\mathcal{G}_{avg(tot\_cred)\ as\ avg\_credit}\ (student)$

* Calculate avg credits completed by students bor each department.

→
```
Select dept_name, avg (tot_cred)
FROM student
GROUP BY dept_name;
```

$dept\_name\ \mathcal{G}_{avg(tot\_cred)}\ (Student)$

* Calculate average credits completed by students ~~from~~ bor each department except law and English.

→
```
SELECT dept_name, avg (tot_cred)
FROM student
WHERE dept_name != 'LAW' OR dept_name != 'Eng'  // dept_name NOT IN ('LAW', 'Eng')
GROUP BY dept_name;
```

$$\Gamma_{dept\_name,\ avg(tot\_cred)}\left(\sigma_{dept\_name\ !='LAW'\ OR\ dept\_name\ !='Eng'}(student)\right)$$

* Calculate average credits completed by 'CSE' students.

```
SELECT dept_name, avg(tot_cred)
FROM student
WHERE dept_name = 'CSE';
```

$$\Gamma_{avg(tot\_cred)}\left(\sigma_{dept\_name='CSE'}(student)\right)$$

01.11.23 / CSE302 / Tuesday / Cls-6

Attributes -

descriptive properties of an object

→ Each attribute has a set of allowed values. It is known as domain of the attribute

cgpa	dob
0.0 – 4.0	any valid domain

name: any combination of 40 chars

→ Each attribute has a datatype too.

→ Attribute values are atomic.

↓

Can not be divided into multiple parts.

telephone_no

8802123456

18799101 11

Country_code	area_code	tel-no
880	2	12 3456
1	878	91011

NULL

Each condition has two truth values.

→ Any comparison involving NULL values returns Unknown truth value.

NULL value

student (id, name, dept_name, tot_cred)

** → Find student id and name where who have NULL in their tot_cred field

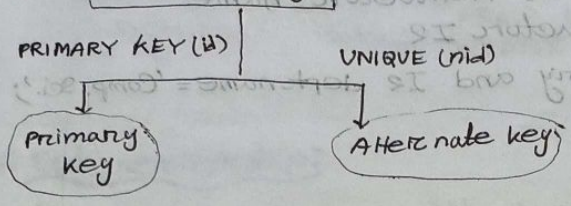
```
SELECT id, name
FROM student
WHERE tot_cred IS NULL;
      IS NOT NULL
```

keys

Primary key: An attribute or a set of attributes that is used to uniquely identify tuples within a relations.

Candidate key:

A relation can have multiple attributes which can serve as a primary key. In this case both are known as Candidate keys.



Super key:

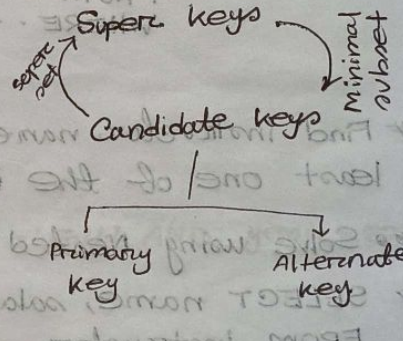
Super set of a candidate key are the super keys.

$k_1 = \{id\}$ $k_2 = \{nid\}$

$\therefore S_1 = \{id, \text{name}\}$ $S_2 = \{nid, dob\}$

S_1 is super set of k_1

S_1 → Super key.



05.11.23 / CSE302 / Sunday / Ch-7

The select, where and from clause

* Table rename করলে as লাগবে না, Attribute করতে হলে as লাগে

⇒
```
SELECT distrinct I1.name as instructor_name
FROM instructor I1, instructor I2
WHERE I1.salary > I2.salary and I2 dept_name = 'Comp. Sci.';
```

Nested Subquery :

A query (inner) within another query (outer)

Inner query can appear -

1. Within WHERE clause
2. Within FROM clause
3. Within SELECT clause.

* Nested sub query within WHERE clause,

```
SELECT...
FROM....
WHERE...(Select.....
         FROM.....
         WHERE...);
```
} inner

Find instructor name who gets salary more than at least one of the instructor from Comp. Sci dept

•• Solve using Nested subquery and Some key

⇒
```
SELECT name, salary
FROM instructor
WHERE salary > SOME (SELECT salary
                     FROM instructor
                     Where dept_name = 'Comp.Sci');
```

Some → At least একটি element এর সাথে compare এ ব্যবহৃত হয়।

All → সবগুলো এর এর সাথে Compare হবে।

Find instructor name and salary who gets salary than all instructors of Comp. Sci department.

```
=> select name, salary
FROM instructor
WHERE salary > ALL (SELECT salary
                    FROM instructor
                    WHERE dept_name = 'Comp. Sci');
```

Set membership:

'in' check the presence of a value in a relation.

Find instructor name and dept name who taught in Fall 2009 semester.

(without in)

```
SELECT name, dept_name
FROM Teaches NATURAL JOIN instructor
WHERE semester = 'Fall' AND year = 2009;
```

(using in)

```
Select name, dept_name
FROM instructor
WHERE id IN (SELECT id
             FROM teaches
             WHERE semester = 'Fall' AND year = 2009);
```

Test for empty relations –

Exists returns true when inner query has result.
Not Exists returns true when inner query has no result.

- Exists / Not exists must have a correlated condition.

```
SELECT name, dept_name
FROM instructor I
WHERE EXISTS (SELECT *
              FROM teaches T
              WHERE semester = 'Fall'
              AND year = 2009
              AND I.id = T.id);
```

07.11.23 / CSE302
Tuesday / Cb-8

Scalar subquery:

3. within SELECT clause –

→ A query that is contained within another query

or

→ SELECT ....., (SELECT .....
                 FROM .....
                 WHERE .....)

FROM .....
WHERE .....;

* Find number of instructors in each department

```
SELECT dept_name, count(*)
FROM instructor
GROUP BY dept_name;
```

* Find number of instructors in each department. Also, include the department without no instructors.

department

dept_name	...	...
CSE		
EEE		
PHRM		

instructor

id	.....	dept_name
1		CSE
2		CSE
3		EEE
4		EEE
5		CSE
6		EEE

CSE 3
EEE 3
PHRM 0

```
SELECT dept_name, (SELECT count(*)
                   FROM instructor
                   WHERE department.dept_name = instructor.dept_name
FROM department;
```

Outer Joins:

→ Extension of Natural Join

(Limitation: Natural Join loses some data from the relations)

Outer Joins include information / tuples even when the tuples has no matching value with another tuple from the other relation

1. Left outer JOIN
2. Right " "
3. Full " "

Left

r ⟕ s

A	B	C
a_1	b_1	c_1
a_2	b_1	c_1
a_3	b_2	c_1
a_4	b_4	NULL

Right

r ⟖ s

A	B	C
a_1	b_1	c_1
a_2	b_1	c_1
a_3	b_2	c_1
NULL	b_3	c_2

FULL

r ⟗ s

A	B	C
a_1	b_1	c_1
a_2	b_1	c_1
a_3	b_2	c_1
a_4	b_4	NULL
NULL	b_3	c_2

Queries using outer Join –

Find number of instructors in each department. Also include the dept with no instructor.

⇒

department ⟕ instructor

CSE
EEE
PHRM

```
SELECT dept_name, count (*) (id)
FROM department LEFT JOIN instructor
ON department.dept_name = instructor.dept_name
GROUP BY dept_name;
```

* কোনো info বাদ দিতে না চাইলে Outer JOIN use করতে হবে।
* Count (*) যদি Row পায় তাহলে count করবে যদিও Row টা null হলেও

Case → থাকবে না

Update the salary of 'CSE' dept instructor by 10%, 'EEE' 'EEE' dept " " 5%, 'PHRM' " " " 8% and the rest of the instructors will not get any increment.

```
UPDATE instructor
SET
salary = CASE
WHEN dept_name = 'CSE' Then salary + salary × 0.1
WHEN dept_name = 'EEE' Then salary + salary × 0.05
WHEN dept_name = 'PHRM' Then salary + salary × 0.03
ELSE salary
END;
```

Delete Vs Truncate

Delete

(DML command)

Log File

Log entry

- (It deletes data according rows the AND there is a WHERE clause)
- (Speed slow)
- (Can restore/undo)

Truncate

```
TRUNCATE TABLE instructor;
```

(Truncate use করলে table এর structure থাকবে কিন্তু ভিতরের সব Data delete হয়ে যাবে)

- (DDL command)
- (No where clause)
- (Speed much faster)
- (Can not restore/undo)

deallocate (memory)

free(ptr)

delete(ptr)

Chapter-1 → Theory Questions

Chapter-2 → Relational Algebra Expre
Theory on Relational mod
Schema diagram.

Chapter-3/4 → SQL Queries

Views:

Final

12.11.23 / CSE 302 / Ch-9 / Sunday

Views are used to hide details of a database by creating a view over a relation, it is possible to hide the val

* Instructor (Id, Name, Dept_name, Salary)

* Dept Secretary (a database user) needs access to the id, name and dept_name of the instructors. He/she does not need to see the salary values.

⇒
```
CREATE VIEW faculty AS
(SELECT id, name, dept_name
 FROM instructor);
```

* A view is a virtual relation. It act like a relati however, the tuples / record within a view has no physical existence. The tuples within a view are not permanently stored in a database.

```
SELECT *
FROM faculty;
```

=

```
CREATE VIEW V1 AS
(SELECT dept_name, sum(salary) as tot_salary
 FROM instructor
 GROUP BY dept_name);

CREATE VIEW V2 AS
(SELECT dept_name, total_salary
 FROM V1
 WHERE total_salary > 100000);
```

Materialized Views –

```
CREATE MATERIALIZED VIEW faculty_mv
    BUILD IMMEDIATE
    REFRESH FAST ON COMMIT
    AS
    (SELECT id, name, dept_name
     FROM instructor);
```

```
# CREATE MATERIALIZED VIEW faculty_mv
    NOLOGGING
    CACHE
    BUILD IMMEDIATE
    AS
    (SELECT id, name, dept_name
     FROM instructor);
```

```
BEGIN
dbms_mview.refresh('faculty_mv');
END;
/
```

Update of a VIEW –

```
INSERT INTO Faculty Values ('11100', 'Bob', 'Elec.Eng.');
```

যে view.কে update করা যায় তাদেরকে simple view/ Update views.

Authorization:

* GRANT [Privilege] ON [owner_name].[view_name]
  TO [user_name];

  GRANT SELECT ON CSE302.FACULTY TO secretary;

* REVOKE [Privilege] ON [owner_name].[view_name]
  FROM [user_name];

  REVOKE SELECT ON CSE302.FACULTY FROM secretary;

Create View Supreme As
SELECT Name
FROM PizzaOrder
WHERE Pizza_Type = "Supreme";

03-12-23/CSE 302
Sunday Cs-10

Chapter-6 → Database Design Using the E-R Model

Design the database:

University database
↳ Student
↳ faculty

Entities:
An object that exists in the real life and it is relevant to your database.

Entities → Group of same type of objects.
Set

Relationship:

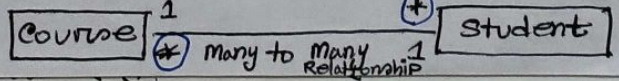
↳ Student A took Course C
↳ Customer Alve bought Rice } (Binary relationship)

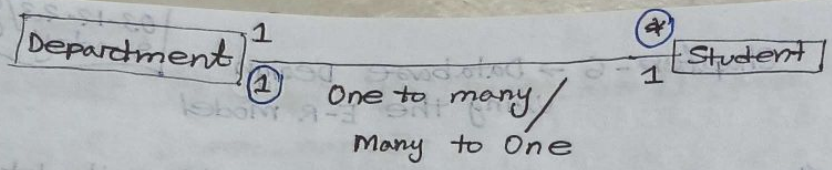
↳ * The association / connection / link between two or more entities.

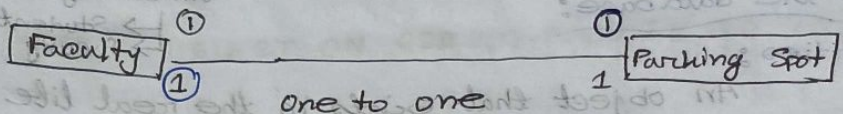
→ Student A is doing project P under the faculty P
(Ternary relationship)

Degree of relationship: The number of entities participate in a relationship.

* A scenario / case study will be provided.
  - ↳ Usually, nouns are representing Entities.
  - ↳ Usually, verbs are representing Relationships
  - ↳ Properties / Attributes: Properties or attributes are used to describe an entity. Attributes can describe an entity set.
  - ↳ Relationship Cardinalities: (Course - Student)







05-12-23/CSE302
Tuesday Ch-11

ER Modeling:

→ To design the database schema

→ Entities (Entity set) Relationships (Relationship Set)

Entity: An object that ~~not~~ exists in the real world and that can distinguishable from other objects

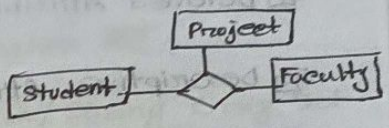
Each entity has properties/ attributes

Entity set: A group of same type of entities sharing common properties

2021-3-60 ——— CSE 302

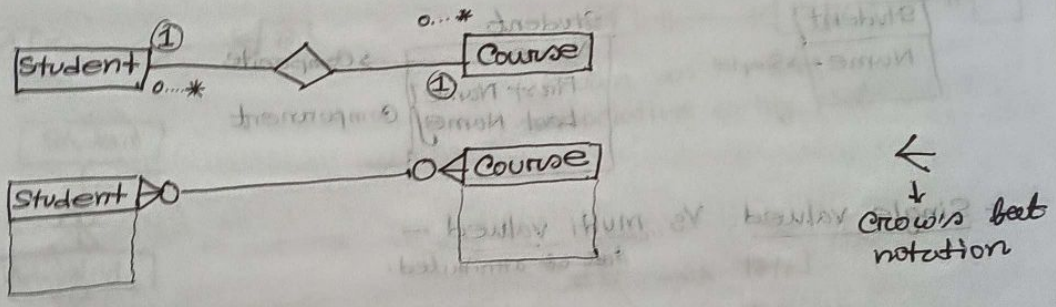
Degree of Relationship

- Binary relationship, degree = 2
- Ternary relationship, degree = 3

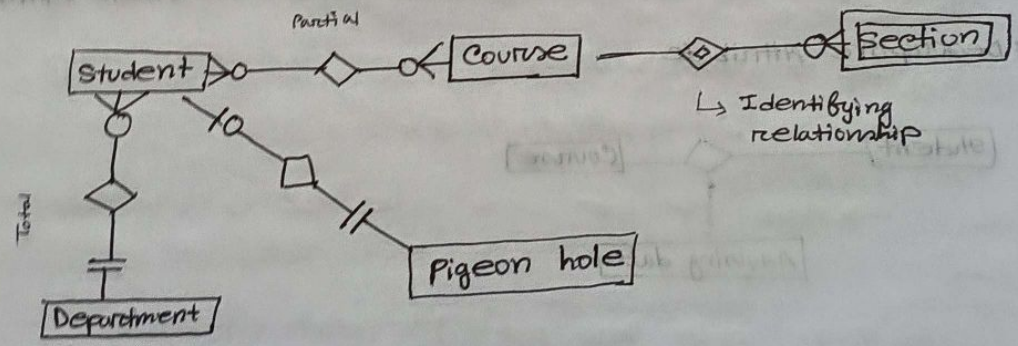


Relationship Cardinalities:

→ Many to many Relationship



→ One to many Relationship

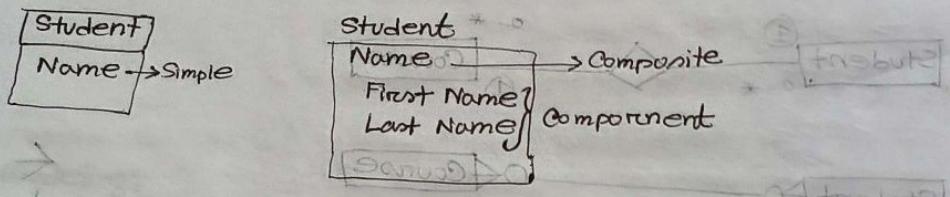


Types of Attributes:

- Simple Vs Composite
- Single valued Vs Multi valued
- Derived Attribute
- Descriptive Attribute

Each attribute has a domain.
↳ The set of permited / allowed values.

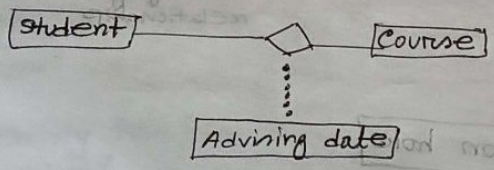
Simple Vs composite —



Single valued Vs Multi valued —
set of attributed.

Derived Attribute
↳ Which value can be calculated based on other attribute.

Descriptive Attribute



2021-3-60 CSE302
05.12.23

Participation of an Entity set in a Relationship set
- min. Cardinality

→ 1. Total (Mandatory) — 1

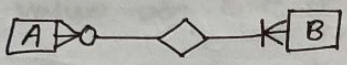
→ 2. Partial (Optional) — 0

Weak Entity Set:

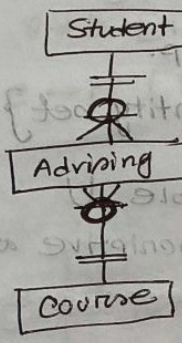
- এদের Always One to many Connection হবে।

→ Identifying Relationship বলা থাকলে Weak / Strong Entity set relation কে বোঝায়।

Associative Entity Set:



known as associative entity set



= → Total

→ → One

— → Many

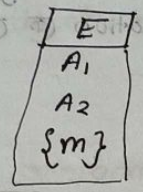
Transforming / Reducing ERmodel to Relational Schema.

1. Create a relation / table for each entity set

→ If a entity set E has a multi valued attribute M then

→ Create one table for E without M

→ Create another table for E' containing primary key of E and M



1. E (A1, A2)
2. E' (A1, M)

2. Rules for relationship sets

i) Create a table for M : N relationship.

PK of the table/relation = {PK of both entity set}

Attribute of the table = PK of the table ∪ any other descriptive attribute

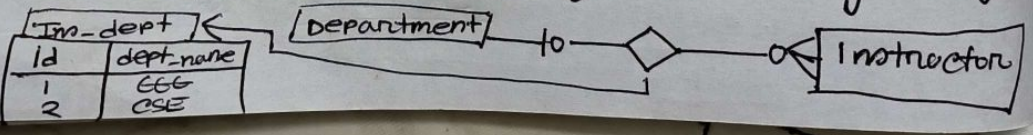
ii) a) One to Many / Many-to-one in which many side has "total" participation.

→ Add PK of one side as a FK of many side

b) "Many" side has "partial" participation

→ Create a new table

PK of new table = {PK of many side}



08.12.23 / CSE302
Friday / Ch-12

Relational Database Design.

Functional dependencies-

$X \rightarrow Y$ (x functionally determines Y)

A	B	C	D
A1	B1	C1	D1
A2	B1	C2	D2
A3	B2	C2	D2
A4	B2	C1	D1

Primary key {A}

$A \rightarrow BCD$

A functionally determines B, C, D

If we identify the value of A then we will be also able to identify the value of B, C, D

$B \rightarrow C$ [it is not a valid functional dependency]

$B \rightarrow D$ [it is not a valid functional dependency]

$C \rightarrow D$ [It is a valid functional dependency]

Anomaly:

- Deletion Anomaly
- Insertion Anomaly
- Update Anomaly

15.12.23 / CSE302
Friday / Ch-13

Data base Normalization-

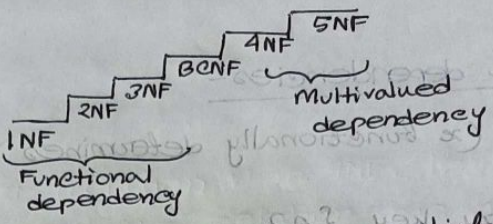
↳ The process of dividing a given relation into smaller relation.

Why: To remove the anomalies

- → Insertion
- → Deletion
- → Update

* Normal forms –

cummulitive
i) 1 NF (First NF)
ii) 2nd NF
iii) 3rd NF
iv) BCNF
v) 4th NF
vi) 5th NF



i) First NF (1NF) → All tables/relations satisfy 1 NF
↳ Each cell in the table must contain a single value.

ii) Second NF (2NF) → A relation is in 2NF, if it already satisfies 1NF and there is no partial key dependency.
↳ subset of key

R(A, B, C, D)
key attributes → all other attributes.
FD 1. (AB) → CD [default FD]
FD 2. (B) → D
↳ determinant

Since B ⊆ AB, FD2. is a partial key dependency
Therefore R is not in 2NF

R(A, B, C, D) → Lossless Join decomposition
↳ $R_1 \bowtie R_2 \supseteq R$
↳ If not satisfied, lossy decomposition.

R_1(A, B, C, D)
R_2(B, D)

** যদি Column PK হবে সেখানে Partial dependency থাকবে।

Name	DOB	Score	Age

Name, DOB → Score, Age
DOB → age

Name	DOB	Score

DOB	age

iii) Third NF (3NF) → A relation is in 3NF if it satisfies 2NF and there is no transitive dependency

ID	name	dept_name	CGPA	NID

FD 1. $A \rightarrow B$
FD 2. $B \rightarrow C$
$A \rightarrow C$

ID → name, dept_name, CGPA, NID { ID → NID, NID → Name

r_1 (ID, NID, dept_name, CGPA)
r_2 (NID, name)
> The given relation shows transitive dependency and therefore it is not in 3NF

* দুইটা unique id / একটা PK এবং একটা Associate key থাকলে table এ থাকবে transitive dependency থাকবে।

iv) Boyce Codd NF (BCNF) → A relation is in BCNF, if it satisfies 3NF and every determinant found in the relation is a candidate key.

Fund ID	Investment type	Manager
99	Common stock	Smith
99	Minicipal Bonds	Jones
33	Common stock	Green
22	Growth Stock	Brown
11	Common Stock	Smith

A	B
α_1	β_1
α_2	β_1
α_3	β_1
α_4	β_1
α_5	β_1
α_1	β_1

FD 1. (Fund ID, Investment type) → Manager
FD 2. (Fund ID, Manager) → Investment Type
FD 3 (Manager) → Investment

* Candidate key তে কোনো Repeatation থাকবে না।
In this case {manager} is not a candidate key
Therefore, given relation is not in BCNF.

r_1 (Fund ID, manager)

r_2 (manager, Investment type)

v) Forth NF (4NF) → A relation is in 4NF. If it satisfies BCNF and the relation has no multivalued dependency.

→ The relation must have at least 3 attributes.

→ Assume that the relation r(A, B, C)

→ For a single value of A multiple B value exist

→ For a single value of A multiple C value exist.

(A) Student ID	(B) Major	(C) Fav_sports
1	ACT	Football
1	HR	Cricket
1	ACT	Cricket
1	HR	Football

$A \twoheadrightarrow B$

$A \twoheadrightarrow C$

→ B and C are independent

The given relation shows/ has multivalued dependency

r_1 (Student ID, major)

r_2 (Student ID, Fav_sports)

vi) Fifth NF (5NF) → A relation is in 5NF. If it already satisfies 4NF and has no join dependency
(extention of 4NF)

→ Usually, we have to add one more relation comprising B and C

→ This is how, we can avoid join dependency

Student ID	Major	Fav_sport
1	ACT	Football
1	ACT	Cricket
1	HR	Football
1	HR	Cricket
2	Finance	Volleyball
2	IR	Basketball
2	Finance	Basketball
2	IR	Volleyball
3	IR	Football

r_1 (Student ID, major)

r_2 (Student ID, Fav_sports)

r_3 (Major, Fav_sports)

r_1

1	ACT
1	HR
2	Finance
2	IR

r_2

1	Football
1	Cricket
2	Volleyball
2	Basketball

r_3

ACT	Football
ACT	Cricket
HR	Football
HR	Cricket
Finance	Volleyball
Finance	Basketball
IR	Volleyball
IR	Basketball
IR	Football

Pros: Removing join dependency

Cons: Sometimes join operation generates extra tuples.

* book (Accession no, ISBN, title, publisher)
book_authors (ISBN, author)
User (User id, name, dept id), dept)

book

2NF: (No partial key)
Accession no → ISBN, title, publisher
(in 2NF)

3NF:
book_1 (Accession no, ISBN)
book_2 (ISBN, title, publisher)

User: 1NF, 2NF satisfied

3NF:
User_1 (User id, name, dept id)
User_2 (dept id, dept name)

**

Movie

1NF

R_1

Full name	Phy Add	Salution
Janet	First	MS
Robert	3rd street	Mrc
Robert	5th "	Mrc

Partial key dependency
Full name → Salutation

R_2

Full name	Phy Add	Salutation
Janet	First street	Pirates
Janet	" "	Clash of

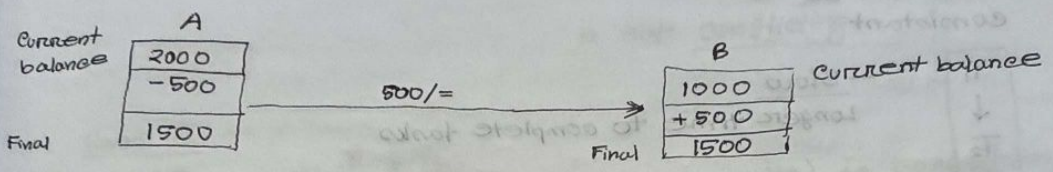
2NF
(Full name, salutation)
(Full name, Physical Address)

Chapter-17
Transactions

17.12.23 / CSE 302 / Sunday / Ch-14

→ A transaction consists of a series of operations to perform a task.

→ Read / write



→ Task: Transferring fund A to B
- → Subtracting the amount from A.
- → Adding the same amount to B

→ A transaction is considered as a unit of execution.

⇒ Properties of transactions:
ACID properties

1. Atomicity: Either all or none.

if 2000+1000 = 3000

1500+1000 = 2500

Database is corrupted

2. Consistency:

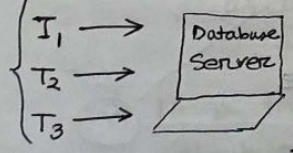
Before transaction ↑

Database Instance

After executing transaction ↑

3. Isolation: Transaction are executed in isolation. T_i will be executed in away that it is not aware of T_j and vice versa.

Concurrent Transactions



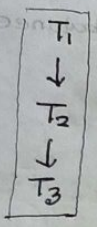
* T_1 would be executed in a way that it is not aware of T_2 and T_3

* T_2 would . . . .

aware of T_1 and T_3

4. Durability: If a transaction commits the data items updated by the transaction must hold their values and there values must be permanently stored in the database.

Data base will be consistant



Serial execution

- Slow
  Longer time to complete tasks
- Low throughput point out

Serializability Theory:

Schedule: An order of operations of the transactions which are to be executed.

Types: i) Serial schedule
ii) Concurrent schedule.

→ Some concurrent schedule behave like a serial schedule.

Out come of a concurrent ≅ Out come of a serial schedule.

* Serializable Schedule: It is a concurrent schedule equivalent to a serial schedule.

Two types: i) Conblict - Serializable
ii) View - serializable

→ Conflict - Serializable :

→ Conflict - equivalent instructions.

Case I :

	T_i	T_j
A=100	R(A)	
		R(A)

	T_i	T_j
	R(A)	
		R(A)

(read - read) instruction pair is not conflicting.

Case II :

A=100	T_i	T_j
	read (A)	
		write (A)
A=200		

	T_i	T_j
		write (A)
	Read (A)	

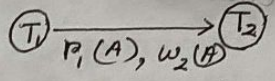
(read - write) is conflicting
(write - read) is conflicting
(write - write) is conflicting

* Finding conflict serializable schedules :

1. Prepare a Precedence graph.

V = { set of transactions }

E = { set of conflicting instruction from T_i to T_j }



$W_1(A)$, $R_2(A)$
$W_1(A)$, $W_2(A)$
$R_1(B)$, $W_2(B)$
$W_1(B)$, $R_2(B)$
$W_1(B)$, $W_2(B)$

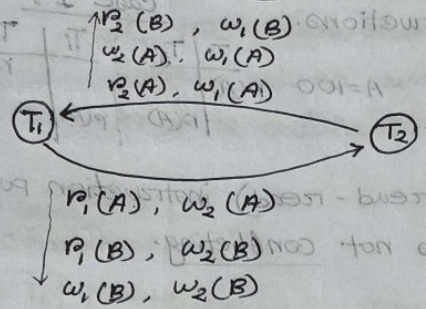
conflict serializable

The schedule is equivalent to $T_1 \rightarrow T_2$

2. If a the precedence graph contains a cycle, the schedule is not conflict serializable.

otherwise, the schedule is conflict - serializable.

Problem from slide

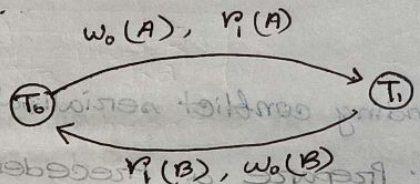


Is not conflict serializable.

It is not equivalent to anything.

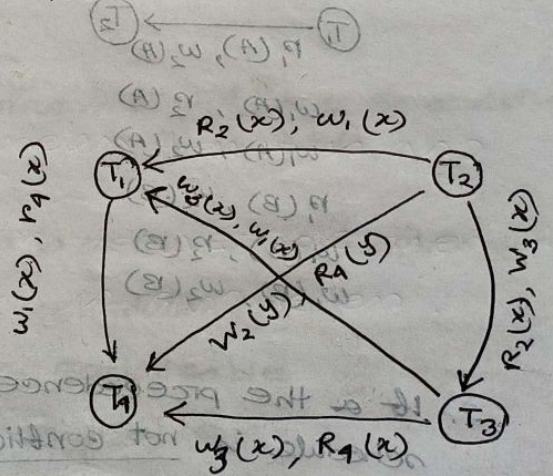
Last class

T_0	T_1
$r_0(A)$	
$w_0(A)$	
	$r_1(A)$
	$r_1(B)$
	C_1
$r_0(B)$	
$w_0(B)$	
C_0	

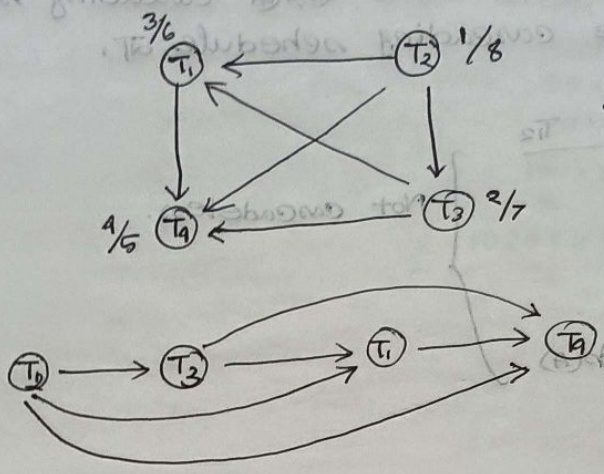


The precedence graph contains a cycle. So, it is not conflict - serializable schedule.

T_1	T_2	T_3	T_4
	$R(x)$		
		$W(x)$	
		Commit	
$W(x)$			
Commit			
	$W(Y)$		
	$R(Z)$		
	commit		
			$R(x)$
			$R(Y)$
			commit



The precedence graph does not contain a cycle.
So, it is a conflict-serializable schedule.



Recoverable schedule:

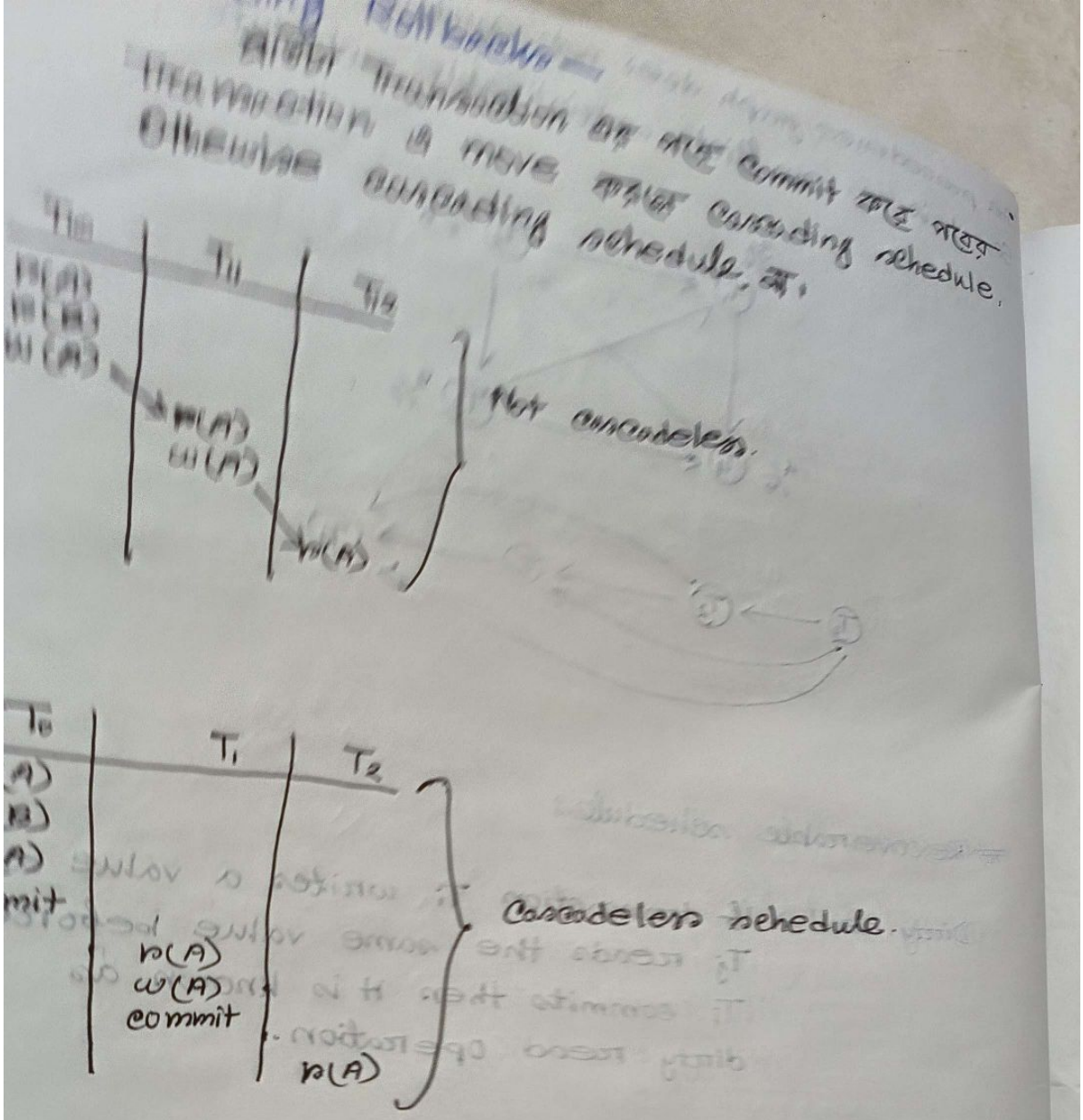
Dirty read → If transaction T_i writes a value and T_j reads the same value before T_i commits then it is known as dirty read operation..

T_8	T_9
r(A)	
w(A)	r(A)
	commit
r(B)	

} Irrecoverable Schedule.

এখানে T_8 , T_9 এর আগে commit করলে এটা recoverable schedule হতো।

* যদি T_i write করার পরে commit না করে এবং T_j ঐ সময় Read করে commit করে ফেলে then এইটা Irrecoverable Schedule.
But T_i যদি Write করে সাথে সাথে commit করে then T_j read করে then এইটা Recoverable schedule.



If a schedule is cascadeless then it is also recoverable

But if a schedule is recoverable then there is no garuntee that is cascadeless.

Billing

10890
10815

75 × 9.85 = 363.75
124 × 6.50 = 822.12
~~10815~~
99 × 6.95 = 688.05
99 × 7.39 = 726.66
199 × 11.51 = 2290.49
96
10294 × 13.26 = 136498.99

A99

Month / Date / Year
12/1

LAB-1

ORACLE APEX

→ ORACLE Database system

→ sqlplus ↙ → system → password

A command line interface (CLI) for communicating to oracle database.

Application Express

→ It is a platform for rapid application development.

→ Web based platform

Creating a table

```
CREATE TABLE Student (        ← table name
    Student_id VARCHAR2 (13),
    Student_name VARCHAR2 (20),

    PRIMARY KEY (Student_id)
);
```

Data type of SQL

* VARCHAR2 (n)
* NUMBER
* DATE [dd-MMM-YY]

PRIMARY KEY

The value of this attribute must be UNIQUE for all rows/records.

```
SELECT * FROM <table name> ;
```

SELECT

Select * From Department

Full table এর Data সহ Display

Select dept_name

Only row print হবে।

```
SELECT ␣ <column name> FROM <Table name>;
```

Drop

```
DROP TABLE student
cascade constraints;
```

desc ␣ table name

to display table.

Schema (Structure)

string, date এর '' use করতে হবে।

Data Insert

```
INSERT INTO <table name> VALUES
( , , , );
```

Where Find:

```
SELECT * FROM <Table name> Where budget > 5000;
```

Find two things.

SELECT * FROM Department WHERE budget > 5000 AND building = 'Main';

In:

SELECT name, population FROM world
WHERE name IN ('Sweden', 'Norway', 'Denmark');

↳ Opposite Not IN

COMMIT; → For saving all.

08.10.23

LAB ~ 2

* create user Alve identified by CSE302; ↷ new user creation
* connect Alve → Then pass:

Executing SQL script from SQLPlus tool –

@ "<path/location of the file>\<file name>.sql"

How to update existing rows ?

⇒ UPDATE student
SET Tot_cred = 112
Where ID = '00128';

Two update together –

CASE.... WHEN ..., THEN... END

```
UPDATE instructor
SET salary = CASE
        WHEN salary < 100000 THEN salary x 1.1
        ELSE salary x 1.05
        END;
```

CRUD
- CREATE
- READ
- Update
- DELETE

DELETE:

```
DELETE From student
  WHERE Id = '00128';
```

* WHERE ␣ Row ␣ NOT IN (' ', ' ', ' ');

* WHERE NOT (name = 'Alve' OR name = 'Alam');

String matching:

* Find students whose name starts with 'A' and have exactly four characters
* Find students who have 'Dhaka' in their address.

→ Patteron building → two special characters
(%) Percent: Can match to 0-n characters
(_) underscore: can match to exactly one character

```
SELECT * FROM students
  WHERE name LIKE 'A%'
        LIKE 'A___'

  WHERE name LIKE 'A%' and
        length (name) > 4;
```

16.10.23

LAB - 03

SQLBolt.com

Relational Algebra:

→ σ (selection) ≅ Where

→ Π (projection) ≅ SELECT

⇒ Find instructor id and name from 'Comp. Sci.' dept.

SQL

```
SELECT id, name
  FROM instructor
  WHERE dept_name = 'Comp. Sci.';
```

Relation Algebra

$\Pi_{id,name}(\sigma_{dept\_name = 'Comp\ Sci.'}(instructor))$

* Multi table query:

⇒ Find name of instructors having their department located in 'Watson' building.

JOIN operation →

Cartesian Product (Cross product)

```
SELECT *
  FROM r, s
  WHERE r.B = s.B;
```

(join condition)

```
SELECT *
  FROM r JOIN s
    ON r.B = s.B;
```

```
SELECT *
FROM instructor JOIN department
  on instructor.dept_name
    = department.dept_name;
```

Select instructor.id, name, dept_name
FROM instructor JOIN teaches
ON instructor.id = teaches.id
where semester = 'Spring' And year = '2009';

Sort data in descending order: —

```
SELECT name, salary
FROM instructor
WHERE dept_name = 'Comp. Sci.'
ORDER BY Salary desc;
```

Movies (Id, Title, director, Year, length)

Boxoffice (movie_id, Rating, domestic_sale, international_sale).

#DDL Commands (Data definition language):

- CREATE TABLE
- CREATE VIEW
- CREATE INDEX
- CREATE USER

* ALTER TABLE:

→ adding a new attribute.

ALTER TABLE ␣ table name ␣ ADD ␣ (new row) ␣ type;

(VARCHAR2 (10));

#Modifying datatype of an attribute —

ALTER TABLE Person MODIFY mobile_number Number;

- Person → Table name
- mobile_number → Attribute name
- Number (data type) → new change data type

Drop column

```
ALTER TABLE Person DROP COLUMN mobile_number;
```

Person → Table name

mobile_number → the name of the column which want to drop

Rename attribute –

```
ALTER TABLE Person RENAME COLUMN name to P_name;
```

Person → Current table name

name → Current attribute name

P_name → After rename the attribute name

Rename table name:

```
ALTER TABLE Person RENAME TO PersonData;
```

Person → Current table name

PersonData → Updated table name

Adding a new constraint:

```
ALTER TABLE PersonData ADD Constraint PersonData_PK
PRIMARY KEY (id);
```

PersonData → Table name

PersonData_PK → constraint name as u like

Adding check to already existing attribute –

```
ALTER TABLE PersonData ADD Constraint PD_CHK_Income
CHECK (Income>0);
```

PersonData → Table name

Condition.

Dropping a constraint -

```
ALTER TABLE PersonData DROP CONSTRAINT PD_CHK_Income
```

SUM, AVG, MAX, MIN

```
SELECT SUM(salary) as T_salary, AVG(salary) as avg_salary,
COUNT(ID) as Number_of_INST
FROM instructor;
```

#DML (Aggregate Queries):

Sum, MIN, MAX, COUNT, AVG, MEDIAN

→ Find the sum of salary of all instructors.

```
SELECT sum(salary)
FROM instructors.
```

→ Find number of instructor,

```
SELECT COUNT (id) as NO_OF_INST
FROM instructor;
```

LAB-3(4)

22.10.23

#HAVING Clause —

→ It is placed after GROUP BY.

→ It applies a condition after grouping.

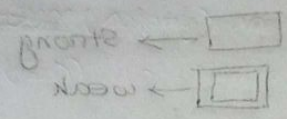
* Calculate the average credit completed by students for each department. where avg credit complited is more than 90.

⇒

```
Select dept_name, avg (tot_cred)
from student
GROUP By dept_name
HAVING avg (tot_cred) >90;
```

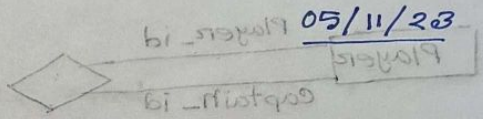
Nested Query:

```
SELECT Name, salary
FROM instructor
WHERE salary = (SELECT max(salary) FROM instructor);
```



LAB-4 (5)

05/11/23

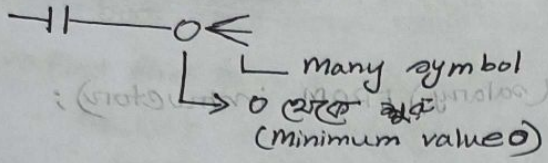


Nested subquery within From clause –

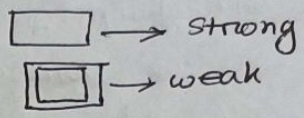
-- Find the average instructor's salaries of those departments where the average salary is greater than $42,000

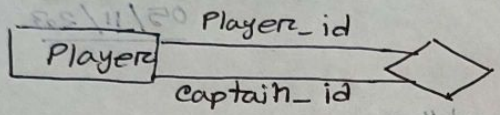
① Select dept_name, avg (salary)
FROM instructor
GROUP BY dept_name
HAVING avg (salary) > 42000;

LAB - 6



Strong entiti set → Player
weak entiti set → Injury





entity 1co ws.

LAB - 8

Buses
Destination
Tot_ Route
C_ Details
T_ Passenger
C_ Reservation
P_ Reservation
R_ Ticket

Identification

Type → Select list
List of values (SQL Query)

```
Select dept_name as display,
    dept_name as target
from department
```

```
Select dept_name ||'-'|| building
as display, dept_name as target
from department.
```

Home এ Add এর করো -
Body → create reason

Html (image)

<img src = " ">

Authentication, **

trigger

75
+125
100
100

363-75

Authorization → User Admin alada করবো,
Right thik করবো,

Access control page
↓
create anew page. create a new navigation.

BLOB
Binary Large Object

→ ER model
→ ER model → relational model
→ Generate DDL statement
} SQL developer

pex
→ Create the schema
→ Application create.
  * Create interactive Grid for each table
  * Calculation of some values. (view)
→ create reports based on queries.
→ Create some charts.

master detail

:PIS_student_name

name match झालेले

where name Like '%:PIS name%';

LIKE '%' || :PIS_Name || '%';

Electricity Billing system for Desco

```
Create view V1 as
( select dept_name, count (*)
  FROM student
  Group By dept_name) ;
```

Consumption

Date	Reading
Oct 2023	110
Nov 2023	510
Dec 2023	1010
~~November 2023~~	

view

Novem 2023	400
Dec 2023	500