FREEMIND

Date

Page

Textbook: Database System Concepts by Sudarshan et. al. 7th edition

DBMS → allows us to handle large amount of data
[Data is money (valuable)]

→ When data is large, we should take care of consistency (data should remain consistent), duplicacy (data should not be duplicate),

Integrity constraints eg: age of a person can not be less than zero.

Concurrency: When more than one person are trying to modify the data.
↳ update should be atomic
↳ one update at a time

SQL → standard language for accessing Database
↳ (Structure query language)

MySQL ⟹ Software

University Database (we will understand all examples (SQL) using this database

- Students
- Instructors
- Classes

FREEMIND

Date

Page

Data → We can describe data by relation b/w different data points

Semantics → Allows us to understand data

Data constraints : eg. student can not take more than courses in a semester. etc.

Two types of Data Models

- Relational → Tabular
- entity - relationship data model →

★ Relational Model

① Data is stored in the form of tables

	ID	instructor	Dept.	Salary
①	1400			
②	1401			

We need to develop (Abstraction) ~~in order to~~ while storing data. so that appropriate person have appropriate view.

FREEMIND

Date

Page

→ View level → logical structure of Database

Before creating Database → we need to finalize (Schema)

very large amount of data can not be stored at one place → That also need to be taken care of

DDL (Data Definition language) → used to create database data points (schema)

CREATE TABLE (name, col1, ...

DML (Data manipulation language) → used to manipulate data

- procedural languages → you need to specify the whole procedure like C++. (you have to mention everything)
- Declarative languages → when you just declare like join two tables. eg. SQL.

eg. Print the name of Instructor from the table instructor whose dept is maths.

```
SELECT name
FROM Instructor
WHERE Dept = Maths
```

Date

Page

Limitations of SQL

SQL can not be use to develop applications

can " " used to make visulization for user

→ These queries can be embedded in highes ~~too~~ level languages like C++, java, Python...

While creating Database, we need to take care of several things.

→ What is the application of Database

→

Functional units of a database

① store manager

② Query processor

③ Transaction management component.

Relational DBMS → RDBMS

Tables

each instance → Tuple or Row

Attributes: The columns of a relation. eg. name etc in the relation instructor.

Domain of Attributes: The data values allowed in attribute.

Tuple: A row of the relation.

FREEMIND

Date

Page

Database Schema: The logical structure of the database.

Database Instance: A snapshot of database at some particular time.

eg. Schema:

Instructor (columns as building, name, salary, dept_name)

Super Key: The set of attributes of relation which can uniquely identify each tuple in the relation.

Suppose $R = \{$ building, dept_name, salary, name $\}$
Then super key $K \subset R$.

Candidate Keys: Subset of super key ~~which~~ having ~~have~~ least no. of attributes which can uniquely identify ~~the~~ each tuple in the relation.

Primary key:

foreign key:

The attribute building is said to be a foreign key which references to relation 'buildings' in the instructor are also present in ~~this~~ "buildings" and it should be a primary key in "buildings" relation.

Date
Page

Relational algebra

A language which takes one or more relations as input and produces a relation as output.
Not turing equivalent.

① SELECT σ

It allows to select some tuple from the database based on certain attribute values.

instructor

name	dept_name	Salary	building
Ram	maths	1000	S.B.
Shyam	CSE	900	M.B
Aarti	EE	1400	E.B.

Predicate ⟶ will result either in true or false.

→ σ(salary ≥ 1000) (instructor) ⟶ will print Ram Aarti tuple.

(=, ≠, >, <, ≥, ≤)

→ σ(salary ≥ 1000) ∧ (dept_name = maths) (instructor) ⟶ Ram tuple

∧(AND), ∨(OR), ¬(NOT)

To combine predicate.

② Project operation (Π) ⟶ unary operator

Π(name, dept_name) (instructor)
will print name and dept_name.

name	dept_name

FREEMIND

Date

Page

Relational algebra

composition of operations :

eg. $\Pi_{(name)}(\sigma_{(salary \geq 1000)}(instructor))$ → will print Ram.

The result of the operations is a relation which can then be used as input for another operator. The operators can be composed into relational algebra expression

→ Binary operation

Cartesian Product operation (X)

Instructor × Teaches

→ will have 9 rows and 7 columns

Teaches

name	course	no. of students
Ram	maths1	10
Ram	maths2	12
Aarti	EE1	7

	instructor.name	dept_name	salary	building	teaches.name	coure	no.of.students
1st row ✓	Ram				Ram	maths1	
2nd row ✓	"				Ram	maths2	
x	"				Aarti	EE1	
x	Shyam				Ram	maths1	
x	"				Ram	maths2	
x	"				Aarti	EE1	
x	Aarti				Ram	maths1	
x	"				Ram	maths2	
3rd row ✓	"				Aarti	EE1	

$\sigma_{(instructor.name = teaches.name)}(instructor \times teaches)$

→ Table with 3 rows 7 col.

FREEMIND

Date

Page

① JOIN OPERATION ($\bowtie$) → Binary

It is denoted as $r \bowtie_\theta s$ where r and s are relations, and θ is a predicate with values True / False for ~~all~~ each tuple.

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

UNION OPERATION ($\cup$) → Binary ($r \cup s$)

① The number of columns in 'r' and 's' must be the same

② Each column must have the same data type

① Find instructor's name ~~which~~ whose salary $\geq$ 1000 or they teach maths?

② $\Pi_{name}(\sigma(salary \geq 1000)(instructor)) \cup \Pi_{name}(\sigma_{course = math}(teaches))$

name
Ram
Aarti

INTERSECTION OPERATION ($\cap$) → Binary → $r \cap s$

$\Pi_{name}(\sigma(salary \neq 1000)(instructor)) \cap \Pi_{name}(\sigma_{course = math}(teaches))$

name
~~Ram~~ Shyam
Aarti

name
Ram

Intersection is ϕ → empty

FREEMIND

Date

Page

DIFFERENCE OPERATION (−) (r − s)

→ tuples present in s will removed from r.

→ It allows us to find the set of tuples in one table (r) but not in (s).

ASSIGNMENT OPERATION (←)

~~r2 ← $\Pi_0(\sigma(r, s))$~~

$r2 \leftarrow \Pi_0(\sigma_{\theta_1}(r)) \cup \sigma_{\theta_2}(t)$

RENAME OPERATOR

This operator will name the resultant relation of RA expression as x

↑ Relational algebra

$(\rho_x(\text{RA expression}))$

Database:

- branch (branch_name, city, assets)
- Customer (ID, name, street, city)
- loan (loan_no, branch_name, amount)
- borrower (ID, loan_no)
- account (ac_no, branch_name, balance)
- depositor (ID, ac_no)

Q → find out each loan no. with loan amount greater than 10K?

$\Pi_{loan\_no}(\sigma_{amount > 10,000}(loan))$

OR $\Pi_{ID}\left(\sigma_{\substack{Balance \\ >5,000}}\left(\sigma_{\substack{depositor.ac\_no \\ = account.ac\_no}}(account \times depositor)\right)\right)$

→ (cartesian product)

FREEMIND

Date

Page

(b) Find the ID of each depositor who has an account balance greater than 5k.

$$\Pi_{ID}\left(\sigma_{\substack{balance > \\ 5,000}}(account \bowtie depositor)\right)$$

→ natural join

(c) Find the ID of each depositor who has an account with balance at least 5k at Dhanbad branch.

$$\Pi_{ID}\left(\sigma_{(balance \geq 5,000) \wedge (branch\_name = 'Dhanbad')}(account \bowtie depositor)\right)$$

OR

$$Q1 \leftarrow (depositor \bowtie_{dep.ac\_no = ac.ac\_no} account)$$

$$\Pi_{ID}\left(\sigma_{(balance \geq 5,000) \wedge (branch\_name = 'Dhanbad')}(Q1)\right)$$

SQL

Domain types in SQL

→ [Data types]

1. Char(n) → string of fixed length 'n'
2. Varchar(n) → string of maximum length 'n'
3. numeric (p,d) → 'p' is total no. of digits and 'd' is no. of digits after decimal
4. float (n)
5. double precision
6. int

FREEMIND

Date

Page

SQL

How to create a table in the Database ?

```
CREATE TABLE CUSTOMER (
    name (varchar(15)),
    branch_name (varchar(10)),
    ac-no (int),
    ID (int), income (int),
    PRIMARY KEY (ID),
    FOREIGN KEY (branch_name) references branch)
```

COMMANDS:

INSERT → INSERT INTO CUSTOMER VALUES( )

DELETE → DELETE FROM

DROP →

ALTER → ALTER TABLE CUSTOMER ADD city varchar(10)
→ will add attribute city to table Customer

" " " DROP branch_name
→ will delete attribute branch_name from table Customer.

Date

Page

SQL Query

→ SELECT name → name of attributes which we want to display
FROM CUSTOMER if we put * (asterisk) instead, then all attributes will be displayed.
WHERE branch_name = 'Dhanbad'

will display names of customer whose branch_name is Dhanbad from table customer.

→ ALTER CUSTOMER ADD yearly_income int

→ SELECT name, yearly_income/12
FROM CUSTOMER
WHERE branch_name = 'Dhanbad'

name	yearly_income/12

→ SELECT *
FROM customer, teaches

} This will return all the attributes and tuples of cartesian product of table customer and teaches.

→ SELECT *
FROM customer, branch
WHERE customer.branch_name = branch.branch_name

→ SELECT name, yearly_income/12 AS monthly_income → will rename the attribute while displaying
FROM CUSTOMER
WHERE branch_name = 'Dhanbad'

name	monthly_income

FREEMIND

Date

Page

→ SELECT name.
FROM CUSTOMER AS T, CUSTORER AS S
WHERE T.income > S.income AND
S.branch_name = 'Dhanbad'

Instructor (name, dept_name, salary, ID)

Ques. Find instructor whose salary is greater than at least some instructors in CSE?

```
SELECT name
FROM instructor
WHERE salary > SOME
    ( SELECT salary
      FROM instructor
      WHERE dept_name = 'CSE')
```

} my soln (nested query)

OR

```
SELECT T.name
FROM instructor AS T, instructor AS S
WHERE T.salary > S.salary
  AND S.dept_name = 'CSE'
```

} using cartesian product

STRING Operations

① % matches with any substring (set of characters)
② _ matches with any character

Check whether given string "Aarti" has t in it.

%t%

Date
Page

Q. Find Select instructors who has t in their name.

```
SELECT Name
From Instructor
WHERE name LIKE %t%
```

① Intro% → matches with any string beginning with intro as introduction, introduce etc

② In_t__ → matches with any string of exact 6 length and 1st, 2nd 4th character should be I, n, t respectively

③ ___% → matches with any string of length at least 3

④ ___ → matches with any string of length exactly 3

Sorting In tables
Order:

* SELECT name
  FROM Instructor
  ORDER BY name
  } By default Ascending

* SELECT name
  FROM Instructor
  ORDER BY name DESC/ASC (DESC → descending, ASC → ascending)

* SELECT name, dept_name
  FROM Instructor
  ORDER BY name, dept_name
  } will order by name then by dept_name.

FREEMIND

Date

Page

Find the instructor name whose salary is between 1000 and 1500

```
SELECT name, dept_name
FROM instructor
WHERE salary BETWEEN 1000 AND 1500
```

OR

```
SELECT name
FROM instructor
WHERE (salary > 1000) AND (salary < 1500)
```

OR

```
(SELECT name
 FROM instructor
 WHERE salary > 1000)
INTERSECT
(SELECT name
 FROM instructor
 WHERE salary < 1500)
```

① UNION
② INTERSECT
③ EXCEPT → SET Difference

By default remove duplicate tuples.

★ If WE Want duplicates as well then we write
① UNION ALL
② INTERSECT ALL
③ EXCEPT ALL

need to check ?

NULL values and NULL operations

NULL + 5 ⟶ NULL

NULL = NULL ⟶ unknown

FREEMIND

Date

Page

NULL values

① They are used to denote missing values in database

② SQL treats as unknown the following

$5 > null \longrightarrow$ unknown

$null > null \longrightarrow$ unknown

$null = null \longrightarrow$ unknown

$5 > 2 \longrightarrow$ True

$2 > 5 \longrightarrow$ false

some examples:

① $(5 > null) \vee (1 > 2) \longrightarrow$ unknown

($5 > null$): unknown (may be true or false); $(1 > 2)$: false

② $(5 > null) \vee (5 > 2) \longrightarrow$ True.

($5 > null$): unknown (may be true or false); $(5 > 2)$: True

whatever be the value of unknown, since here is an OR operator with true hence the result will be True.

instructor

ID	name	dept_name	Salary
1	Ram	CSE	1000
2	Shyam	Maths	900
3	Aarvi	Maths	1500
4	Sarabh	CSE	700

FREEMIND

Date

Page

AGGREGATE FUNCTIONS

AVG

COUNT

MIN

MAX

Sum

Q → find average salary in 'CSE' dept.

```
SELECT AVG(salary)
FROM instructor
WHERE dept_name = 'CSE'
```

Q → Count the no. of instructor whose salary is greater than 1000 from CSE dept

```
SELECT COUNT(salary)
FROM instructor
WHERE dept_name = 'CSE' AND salary > 1000
```

GROUP BY

①

```
SELECT dept_name , AVG(salary)
FROM instructor
GROUP BY dept_name
```

output

dept_name	salary
CSE	850
Maths	1200

Date
Page

② SELECT dept_name, avg(salary)
FROM instructor
GROUP BY dept_name
HAVING AVG(salary) > 1000

section

Course_id	Section	Semester	Year
MTH101	A	Monsoon	2020
CSE101	B	Winter	2021
MTH102	A	Winter	2022
MTH103	B	Monsoon	2021
MTH101	B	Winter	2021

NESTED QUERIES

Qu. Find the courses that were offered in Monsoon 2020 and (not in) Winter 2021.

SELECT course_id
FROM section
WHERE semester = 'Monsoon'
AND year = 2020
AND course_id IN (NOT IN) (SELECT course_id
FROM section
WHERE semester = 'winter'
AND Year = 2021)

If we will try to do this query in a single query, we will have to use cartesian product of table section because we need to process two tuples at the same time

FREEMIND

Date

Page

EXIST ( ) ⟶ True if $r \neq \phi$, false otherwise

NOT EXIST ( ) ⟶ True if $r = \phi$, false otherwise

SOME

ALL

Q Find name of instructors whose salary is greater than some instructor in CSE?

```
SELECT name
FROM instructors
WHERE salary > SOME ( SELECT salary
                      FROM instructor
                      WHERE dept_name = 'CSE')
```

* $5 > \text{SOME} \begin{pmatrix} 6 \\ 7 \\ 8 \end{pmatrix}$	false	$5 > \text{ALL} \begin{pmatrix} 6 \\ 7 \\ 8 \end{pmatrix}$	~~False~~ True
$4 < \text{SOME} \begin{pmatrix} 0 \\ 1 \\ -5 \end{pmatrix}$	false	$4 < \text{ALL} \begin{pmatrix} 0 \\ 1 \\ -5 \end{pmatrix}$	False
$5 = \text{SOME} \begin{pmatrix} 0 \\ 1 \\ 5 \end{pmatrix}$	True	$5 = \text{ALL} \begin{pmatrix} 0 \\ 1 \\ 5 \end{pmatrix}$	False
$3 \neq \text{SOME} \begin{pmatrix} 3 \\ 0 \end{pmatrix}$	True	$3 \neq \text{ALL} \begin{pmatrix} 3 \\ 0 \end{pmatrix}$	False

Date

Page

Que. Find the name of instructors whose salary is greater than all the instructors in CSE.

```
SELECT name
FROM instructors
WHERE salary > ALL (SELECT salary
                    FROM instructor
                    WHERE dept_name = 'CSE')
```

Que. Find the average salary of instructors in the department where average salary is greater than 900 ? (do it without using HAVING CLAUSE, use nested subquery)

```
SELECT dept_name, AVG(salary)
FROM instructor
WHERE
```

```
SELECT dept_name, avg_salary
FROM ( SELECT dept_name, AVG(salary) AS avg_salary
       FROM instructor
       GROUP BY dept_name)
WHERE avg_salary > 900
```

department

dept_name	building	floor
CSE	Science Bui	3
maths	Khosla Bui	2
history	Huma Bi	2

Instructor

name	dept_name	salary	ID
Ram	CSE	900	1
Shyam	maths	300	2
Aarti	CSE	1200	3
Sarvabh	history	1300	4

Views

Views are subset of relations that are not materialised but rather stored as an expression.

→ Create a view without salary column in instructor

```
CREATE VIEW view_name AS
        SELECT ID, name, dept_name
        FROM Instructor
```

→ We can create another view from views.
If v_1 and v_2 be two views.
eg. CREATE VIEW v_3 AS (v_1, v_2)

→ We may sometimes update through views ↑ but sometimes a conflict may arise.

→ Create a view ~~which~~ with columns: ID, instructor.name, dept_name.building.

```
CREATE VIEW view2 AS
    ( SELECT ID, name, building
      FROM instructor, building
```

Date

Page

WHERE instructor.dept_name = department.dept_name)

* now suppose someone try to insert a tuple into this view. This will lead to a conflict.

* If a view is created using more than one relation, a insertion into that view may lead to a conflict. But if a view is created from only one relation, most of the languages allows updation in that case.

If a view contains detail of instructors in CSE dept. Then updating it with (sanjay, biology, 3) is not possible.

SIMPLE VIEWS → when SELECT has only
should not have any aggregate functions

TRANSACTIONS → represent a unit of work
↳ group certain no. of queries to perform a task.

→ Transaction is a unit of work that consists of a sequence of queries.
It ends with:
① Commit work:

② Rollback work:

FREEMIND

Date

Page

Isolation

Concurrent transactions

Basic purpose of these transactions is to fulfill integrity ~~integrity~~ constrains.

Check (command)

↳ will make sure given constraints are satisfied.

```
CREATE TABLE instructor
    (ID varchar
     name
     salary Rupees
     PRIMARY KEY (ID)
     CHECK (salary >100)
     FOREIGN KEY dept_name REFERENCES department
     ON UPDATE CASCADE ON DELETE CASCADE)
```

data type defined on next page.

→ Cascading actions ~~on delete~~

If an update violates foriegn key constraint then either rejects it or do cascade operations

→ ASSERTIONS

CREATE ASSERTION → are also and allow us to maintain consistency of data.

CREATE ASSERTION ____ CHECK (____)

FREEMIND

Date

Page

DATA TYPES

→ There are by default certain data types as date, But we can also create our own data types.

example:

① Date : with format 'YYYY-MM-DD'

② time :

③ timestamp:

→ CREATE TYPE Rupees AS
numeric(10,2)

~~Data types~~

→ CREATE DOMAIN dept-name AS varchar(10)
CONSTRAIN dept_domain-constrain
CHECK (VALUE IN ('maths', 'CSE', 'biology'))

Index → help to retrieve a tuple quickly

↳ It allows to search the database efficiently without scanning the whole database again for retrieving a particular tuple. SQL will not create index by default. you will have to give a command for that.

~~CREATE INDEX index-name~~

FREEMIND

Date

Page

CREATE INDEX index_name ON instructor (ID);

SELECT *
FROM student/instructor
WHERE ID = 170

→ After creating an index, this query will run ~~re~~ much faster.

AUTHORIZATION:

Read If you give authorization Read, then user can only read the Data

Insert

Modify

Delete

These accesses can be given on indices, ~~relations~~ resources (or relations), Alteration, Drop.

~~Index~~, ~~Resources~~ → change in DB.

Select

grant select on Instructor to user_id
grant insert " " " "
delete
update
all priviliges

Revoke: to take back priviliges

REVOKE insert ON instructor FROM user_id

→ If PUBLIC → Then all users get that access

Role :

CREATE A ROLE faculty
CREATE A ROLE Dean
GRANT faculty TO Aarti
GRANT Dean to Ram
GRANT MODIFY ON instructor TO Dean

GRANT faculty TO DEAN

Next chapter

CREATING FUNCTIONS WITHIN SQL
↳ (user defined)

dept_count → A function that returns the count of the instructor in a dept from instructor table.

```
SELECT dept_name
FROM department
WHERE { dept_count(dept_name) ≥ 2
```

→ We can embed SQL queries in Python code.

Que Given name of department find the count of instructors in that department.

Instructor(ID, name, dept_name, salary)
department(dept_name, building, floor)

```
CREATE FUNCTION dept_count (dept_name varchar(20))
    RETURNS INTEGER
    BEGIN
```

```
DECLARE  d_count  INTEGER
         SELECT  COUNT(*)  INTO  d_count
         FROM  instructor
         WHERE  instructor.dept_name = dept_name
         RETURN  d_count
END
```

find count of instructors in science block.

```
SELECT  dept_name , dept_count( dept_name )
FROM  department
WHERE  department. building = "science block"
```

#
```
SELECT  dept_name
FROM  department
WHERE  dept count ( dept_name ) > 1
```

Table functions in SQL

Que. Given name of department find the details of all instructors in that department.

```
CREATE  FUNCTION  instructor_of  (dept_name varchar(20))
    RETURNS  TABLE  ( ID  char ,
                      salary  int ,
                      name  char ,
                      dept_name  char )
```

FREEMIND

Date

Page

RETURN TABLE (SELECT ID, salary, name, dept_name)
FROM instructor
WHERE instructor.dept_name = dept_name)

→ SELECT *
FROM table instructor_of (Maths))

FOR

FOR LOOP
DECLARE s_count INTEGER DEFAULT 0;
FOR it AS
SELECT salary
FROM instructor
DO
SET s_count = s_count + it.salary
END FOR

TRIGGERS
↳ similar syntax as that of Functions. [need to write CREATE TRIGGER insted of CREATE FUNCTION]
↳ you can disable triggers in certain situations (when needed)

Course

course_id	prereq_id
MTH104	MTH103
CSE102	CSE101
MTH103	MTH102
MTH102	MTH101

FREEMIND

Date

Page

RECURSION

Qu: Find all the pre-requisites for a given course?

rec_course

course_id	Prereq_id
MTH104	MTH103
CSE102	CSE101
MTH103	MTH102
MTH102	MTH101
MTH104	MTH102
MTH103	MTH101
~~MTH101~~	~~MTH101~~
MTH104	MTH101

Iteration 1

MTH104 MTH102

MTH103 MTH101

Iteration 2:

MTH104 MTH102

MTH103 MTH101

MTH104 MTH101

Query:

```
WITH RECURSIVE rec_course (course_id, prereq_id)
    AS ( SELECT *
         FROM course
         UNION SELECT rec_course.course_id, course.prereq_id
               FROM rec_course , course
               WHERE rec_course.preq_id = course.course_id)

SELECT *
FROM rec_course
```

windowing

Next chapter (E-R Model)

Instructor

name	ID	salary	dept_name

Entity Relationship Model

↳ relation b/w two objects

object

student

name	dept_name

Entity: An object or thing in database distinguishable from other objects. It is described by set of attributes.

Relationship: An association between different entities.

Entity set: A set of entities with same attribute

Instructor
name
ID
salary
dept_name

Relationship set: A mathematical relation between entity sets.

$\{(e_1, e_2, \ldots, e_n) \mid e_1 \in E_1, e_2 \in E_2, e_3 \in E_3, \ldots, e_n \in E_n\}$

each E_i is entity set

e_i is tuple

Date

Page