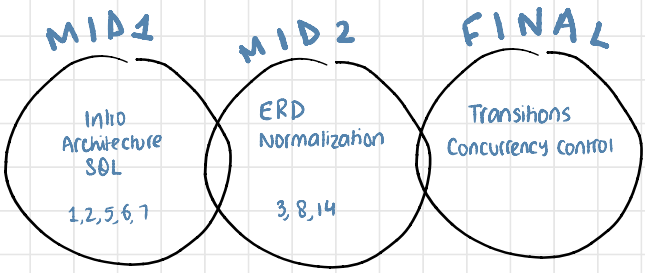
Database Systems

Office Location: Room 101 opposite lab1 first floor CS building.
Email: anam.qureshi@nu.edu.pk

Mid 1: 15marks
Mid 2: 15 marks
Assignments: 10 marks → 5 assignments
↓
best 4 selected
Project: 10 marks
Final Exam: 50 marks



COURCE PROJECT

- all Theory + Lab concepts
- frontend is connected to backend
- frontend design dosent matter
- can Pick templates

CHP 1

Database
↳ a collection of related data

Mini-world
↳ part of real world data stored in a database
student grades at university

Entity
↳ faculty
↳ cources
↳ students
↳ sections

Relations
↳ Student take cources
↳ faculty assigned sections
↳

Data
↳ known facts

Salary
$15 k
$13 k
$16 k

$15 k
$13 k
$16 k
random numbers

Student → Table

S-ID	S-name	S-Ph no.	S-cource

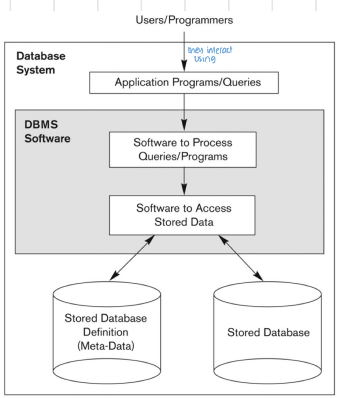
attributes

Database system
↳ DBMS software + data itself + applications (sometimes)

entities/relations/objects/table
SAME THING

DBMS (Database management system)
↳ software for creating/maintaing database

FUNCTIONALITY
1. Data manipulation (Query, transaction, modify) (insert/delete)
2. Data catalogue
3. Cache → loads whole database to secondary storage medium
4. Security and Privacy → Prevent unauthorised access
5. Access control → which data has access
6. Concurrency
7. Presentation and Visualization of data → supports multiple views (each user sees a diff view of db)



Applications interact with database by generating

Query
↳ only read
↳ fetches data

Transaction
↳ read and modify/write
↳ atomic in nature → execute fully OR not at all → COMMIT OR ROLL BACK

* Applications dont allow unauthorized users to access data

Simple database

COURSE

Course_name	Course_number	Credit_hours	Department
Intro to Computer Science	CS1310	4	CS
Data Structures	CS3320	4	CS
Discrete Mathematics	MATH2410	3	MATH
Database	CS3380	3	CS

SECTION

Section_identifier	Course_number	Semester	Year	Instructor
85	MATH2410	Fall	04	King
92	CS1310	Fall	04	Anderson
102	CS3320	Spring	05	Knuth
112	MATH2410	Fall	05	Chang
119	CS1310	Fall	05	Anderson
135	CS3380	Fall	05	Stone

GRADE_REPORT

Student_number	Section_identifier	Grade
17	112	B
17	119	C
8	85	A
8	92	A
8	102	B
8	135	A

PREREQUISITE

Course_number	Prerequisite_number
CS3380	CS3320
CS3380	MATH2410
CS3320	CS1310

Shamkant B. Navathe

Simplified database catalog

RELATIONS

Relation_name	No_of_columns
STUDENT	4
COURSE	4
SECTION	5
GRADE_REPORT	3
PREREQUISITE	2

COLUMNS

Column_name	Data_type	Belongs_to_relation
Name	Character (30)	STUDENT
Student_number	Character (4)	STUDENT
Class	Integer (1)	STUDENT
Major	Major_type	STUDENT
Course_name	Character (10)	COURSE
Course_number	XXXXNNNN	COURSE
....	....	.....
....	....	.....
....	....	.....
Prerequisite_number	XXXXNNNN	PREREQUISITE

Note: Major_type is defined as an enumerated type with all known majors. XXXXNNNN is used to define a type with four alpha characters followed by four digits

Example of a Database (with a Conceptual Data Model)

- **Mini-world for the example:**
  - Part of a UNIVERSITY environment.
- **Some mini-world *entities*:**
  - STUDENTs
  - COURSEs
  - SECTIONs (of COURSEs)
  - (academic) DEPARTMENTs
  - INSTRUCTORs
- **Some mini-world *relationships*:**
  - SECTIONs *are of specific* COURSEs
  - STUDENTs *take* SECTIONs
  - COURSEs *have prerequisite* COURSEs
  - INSTRUCTORs *teach* SECTIONs
  - COURSEs *are offered by* DEPARTMENTs
  - STUDENTs *major in* DEPARTMENTs

MAIN CHARACTERISTICS OF Database APPROACH

Database catalog

↳ description (meta data) of a database
  ↳ type of attribute
  ↳ constant of attribute (e.g. unique, required) (size)
  ↳ data structures
  ↳ constraints

Program data independence

↳ insulation b/w programs and data
↳ allows changing
  ↳ data structures
    ↳ storage organisation
  without changing the DBMS access programs

Data Abstraction

↳ a data model is used
↳ to hide storage details
↳ present users a conceptual view

Multiple view support

↳ each user see's a different view of db
↳ shows data only which is of interest to that user

- **Sharing of data and multi-user transaction processing:**
  - Allowing a set of **concurrent users** to retrieve from and to update the database.
  - *Concurrency control* within the DBMS guarantees that each **transaction** is correctly executed or aborted
  - *Recovery* subsystem ensures each completed transaction has its effect permanently recorded in the database
  - **OLTP** (Online Transaction Processing) is a major part of database applications. This allows hundreds of concurrent transactions to execute per second.

Database Users

Actors on the Scene

those who
- use and control the db content
- design, develop and maintain db applications

1. Database administrators
   - authorise access to db
   - acquire hardware and software resources
   - monitor efficiency of operations

2. Database designers
   - define the content, structure, constraints, functions, transactions
   - communicate with end users → to understand their needs

3. End Users
   - use data for queries, report, update db content
   1. Casual
      - access db when needed
   2. Naive/Parametric → makes up large section of end user
      - canned transactions → previously well defined functions
      - mostly users of
        - mobile apps
        - Bank tellers / reservation clerks
        - Social media users → post and read info on websites

4. Sophisticated
   - many use tools in the form of software packages
   - mostly users are
     - business analysts
     - scientists, engineers
     - people familiar with system capabilities

Workers behind the Scene

those who
- design and develop DBMS software and computer system operators

1. **System Designers and Implementors:** Design and implement DBMS packages in the form of modules and interfaces and test and debug them. The DBMS must interface with applications, language compilers, operating system components, etc.
2. **Tool Developers**: Design and implement software systems called tools for modeling and designing databases, performance monitoring, prototyping, test data generation, user interface creation, simulation etc. that facilitate building of applications and allow using database effectively.
3. **Operators and Maintenance Personnel:** They manage the actual running and maintenance of the database system hardware and software environment.

5. **Stand-alone:**
   - Mostly maintain personal databases using ready-to-use packaged applications.
   - An example is the user of a tax program that creates its own internal database.
   - Another example is a user that maintains a database of personal photos and videos.

6. **System Analysts and Application Developers**

   This category currently accounts for a very large proportion of the IT work force.

7. **System Analysts**: They understand the user requirements of naïve and sophisticated users and design applications including canned transactions to meet those requirements.
8. **Application Programmers:** Implement the specifications developed by analysts and test and debug them before deployment.
9. **Business Analysts**: There is an increasing need for such people who can analyze vast amounts of business data and real-time data ("Big Data") for better decision making related to planning, advertising, marketing etc.

Adv of using DB Aproach

Sharing of same data among multiple users

- ↳ controlling redundancy in data storage
- ↳ restrict unaumorised acess to data → Privellege commands
- ↳ Provide persistent storage for program objects
- ↳ Provide storage structures (eg index) → for efficient query processing
- ↳ provide optimization of queries → for efficient processing
- ↳ Backup and recovery services
- ↳ multiple interfaces → different view for different class of users
- ↳ Represent complex relationships among data
- ↳ Enforce integrity constraints on db
- ↳ Drawing inferences and actions

When NOT to use DBMS

- Main inhibitors (costs) of using a DBMS:
  - High initial investment and possible need for additional hardware.
  - Overhead for providing generality, security, concurrency control, recovery, and integrity functions.
- When a DBMS may be unnecessary:
  - If the database and applications are simple, well defined, and not expected to change.
  - If access to data by multiple users is not required.
- When a DBMS may be infeasible:
  - In embedded systems where a general purpose DBMS may not fit in available storage
- When no DBMS may suffice:
  - ↳ if stringent real-time requirements → OS, DBMS overheads a problem → Telephone Switching Systems
  - ↳ Complex data → OS, DBMS has modeling limitations → protein db
  - ↳ Special operations → GIS, location based service } not supported by DBMS

Database systems

1. Data integrity and consistency — enforces data integrity using constraints and validation
2. Advance search capabilities — filtering
3. Concurrency → multiple users can acess same data
4. Better support for large and complex datasets
5. Built in security features → robust security features
6. Redundancy → data normalization
7. Efficient revival — SQL queries and indexing
8. Scalibility → Sharding, replication facilities
9. Program data independence / insulation
10. Self describing nature
11. Multiple views → student and teachers both have diffent view

Filebase System

1. Simplicity and ease of setup
2. Greater control → directly manage files and folders
3. Better performance for specific tasks → large files with less processing
4. Lower overhead and resource usage → good for constrains
5. Portability to move/distribute data → when access different systems
6. Simplified backup process → copy files to another location

Data model

- what is structure of data
- what constraints (restrictions on valid data) to be applied on data
- what operations to be applied on data
  - create / read
  - delete / search (↳ condition ↳ filter)

Data model operations

- used to specify db retrivals and updates
- includes
  1. basic model operations → insert / delete / update
  2. user defined operations → compute_gpa, update_inventory

Constructs

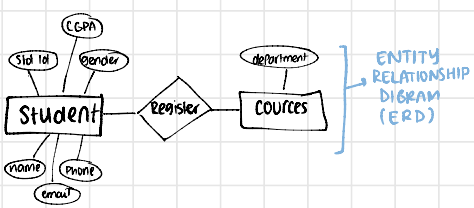
- define db structure
- includes
  1. elements → and their data type
  2. groups of elements → entity / record / table
  3. relationships

Types of Data Model

Conceptual data model

(can make all changes at this point)

- user friendly → easily understood by everyone



CONVERTED TO

Implementation data model

(aka representation)

- conceptual + physical data model

```
create table Student(
    Std_Id int primary key   (is automatically NOT NULL, UNIQUE)
    S_name varchar(40)
    S_gender char(1)
    S_phone varchar(13) NOT NULL
    S_add varchar(40)
    );
```

(constrain – type def)

Physical data model

- all about data structures
- how data is stored
- D+ tree is used to store data in a storage medium

Self describing data model

- at a time key and value both explained
- providing attributes and data at the same time

```
Select * Students from University
where Stdid = "1234" AND CGPA > 2.0
```

VS

Database Schema

aka INTENSION

better to not change once made

- description of database
- includes
  - data types
  - constraints on db
  - description of db structure

Database State/Instance

aka EXTENSION

- content of a db at a particular moment in time
- includes collection of all data in db

Schema Diagram

- illustrative display of db schema

Schema Construct

- a component of schema
  - STUDENT, COURSE

Initial Database State

- when db is initially loaded in system

Valid State

- when constraints and structure of db is satisfied

Distinction

- db schema changes very infrequently
- db state changes everytime db is updated

Example of a Database Schema

STUDENT

Name	Student_number	Class	Major

Schema diag
database in

COURSE

Course_name	Course_number	Credit_hours	Department

PREREQUISITE

Course_number	Prerequisite_number

SECTION

Section_identifier	Course_number	Semester	Year	Instructor

GRADE_REPORT

Student_number	Section_identifier	Grade

Example of a database state

COURSE

Course_name	Course_number	Credit_hours	Department
Intro to Computer Science	CS1310	4	CS
Data Structures	CS3320	4	CS
Discrete Mathematics	MATH2410	3	MATH
Database	CS3380	3	CS

SECTION

Section_identifier	Course_number	Semester	Year	Instructor
85	MATH2410	Fall	04	King
92	CS1310	Fall	04	Anderson
102	CS3320	Spring	05	Knuth
112	MATH2410	Fall	05	Chang
119	CS1310	Fall	05	Anderson
135	CS3380	Fall	05	Stone

GRADE_REPORT

Student_number	Section_identifier	Grade
17	112	B
17	119	C
8	85	A
8	92	A
8	102	B
8	135	A

PREREQUISITE

Course_number	Prerequisite_number
CS3380	CS3320
CS3380	MATH2410
CS3320	CS1310

Figure 1.2
A database that stores student and course information.

3 schema Architecture

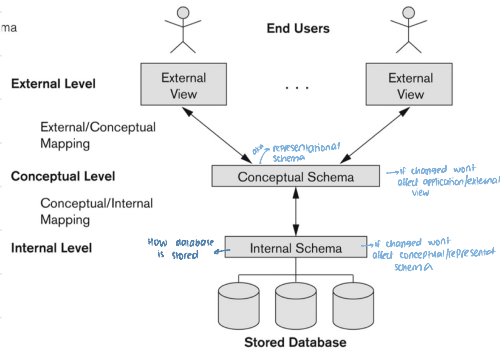
↳ defines DBMS schema at 3 levels

1. **Internal schema** → uses physical data model
   - ↳ at internal level
   - ↳ to describe physical storage structures
   - ↳ to acess paths
2. **conceptual schema** → uses conceptual/implementation data model
   - ↳ at conceptual level
   - ↳ describe structure and constraints
3. **External schema** → uses conceptual data model
   - ↳ at external level
   - ↳ describe various user views

↳ support 2 characteristics

1. **Program data Independence**
   - **logical**
     - ↳ change conceptual schema w/o changing external schema
   - **Physical**
     - ↳ change internal schema w/o changing conceptual schema
     - For example, the internal schema may be changed when certain file structures are reorganized or new indexes are created to improve database performance
2. **Multiple views**
   - ↳ diff views for diff users
   - ↳ view data from virtual table
     - ↳ table wont be created but you can view data from table

The three-schema architecture



if database schema changed it will reflect application/external layer

Three-Schema Architecture

- Mappings among schema levels are needed to transform requests and data.
  - Programs refer to an external schema, and are mapped by the DBMS to the internal schema for execution.
  - Data extracted from the internal DBMS level is reformatted to match the user's external view (e.g. formatting the results of an SQL query for display in a Web page)

Data Independence

- When a schema at a lower level is changed, only the **mappings** between this schema and higher-level schemas need to be changed in a DBMS that fully supports data independence.
- The higher-level schemas themselves are **unchanged**.
  - Hence, the application programs need not be changed since they refer to the external schemas.

DBMS LANGUAGES

Data Definition language (DDL)

↳ used by db designers → to specify conceptual schema

↳ coding which starts with create
- table
- view
- tablespace
- database

Data manipulation language (DML)

↳ used to specify db retrivals and updates

↳ coding operations
- create / read
- delete
- search
  - ↳ condition
  - ↳ filter

TYPES OF DML

NON procedural language

↳ specifies what data to retrive

↳ aka high level or declarative

↳ no functions
↳ no loops
↳ no conditions

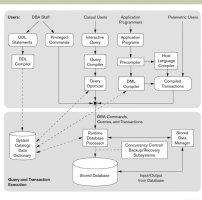
Procedural language

↳ retrieves data one at a time

↳ aka low level

↳ uses 3rd party languages

Typical DBMS Component Modules



→ better version 2 pages later

Copyright © 2016 Ramez Elmasri and Shamkant B. Navathe

Slide 2- 2:

Structural → SQL

↳ relates to DB schema and organisation

↳ create, modify, delete objects → tables, indexes, views, constraints

↳ defines structure, relationships and constrains

↳ changes can impact entire DB system

no of students
course name

↳ google

non structural → no SQL

↳ focuses on actual content/records in DB

↳ querying, inserting, updating, deleting data in tables
- querying: retrieve specific info from storage as structure

↳ manipulates info

↳ more common

↳ less likely to affect overall DB design

↳ mostly used in Big data

if data is variety velocity volume then non structual

Big data apps → insta, fb, sc

Centrallied DBMS

→ aka one tier architecture

↳ combines everything into a single system

↳ front end, backend, Db

↳ e.g. prototype of mobile app

clients

↳ provide appropriate interfaces

↳ connected to servers via network

DBMS server

↳ provide db query and transaction services to clients

A Physical Centralized Architecture

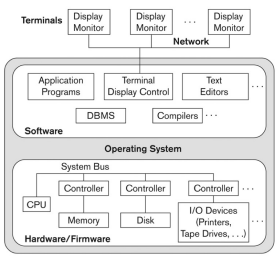


Figure 2.4
A physical centralized architecture.

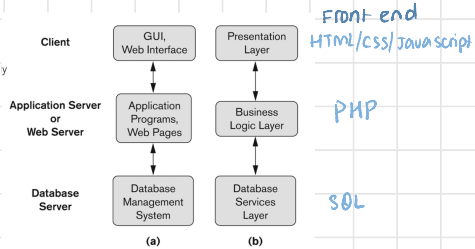
Slide

Three-tier Client-server

↳ e.g. websites

↳ no direct acess to server

↳ enhanced security



front end
HTML/CSS/Javascript

PHP

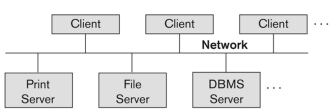
SQL

Two-tier Client-server

↳ for organisational

↳ seperate client and server

↳ client direct acess to server → disaster

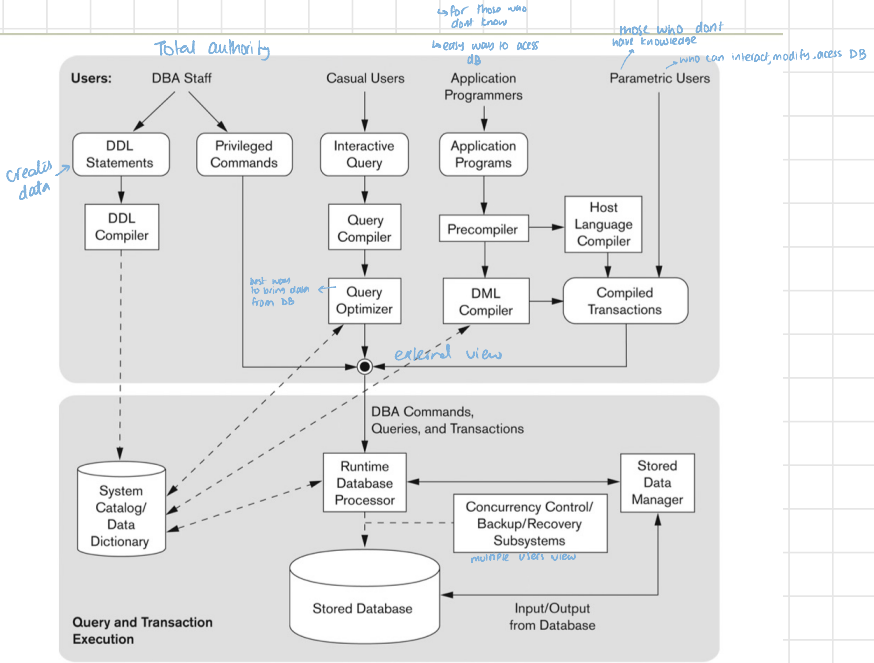


Three Tier Client-Server Architecture

- Common for Web applications
- Intermediate Layer called Application Server or Web Server:
  - Stores the web connectivity software and the business logic part of the application used to access the corresponding data from the database server
  - Acts like a conduit for sending partially processed data between the database server and the client.
- Three-tier Architecture Can Enhance Security:
  - Database server only accessible via middle tier
  - Clients cannot directly access database server
  - Clients contain user interfaces and Web browsers
  - The client is typically a PC or a mobile device connected to the Web

Two Tier Client-Server Architecture

- Client and server must install appropriate client module and server module software for ODBC or JDBC
- A client program may connect to several DBMSs, sometimes called the data sources.
- In general, data sources can be files or other non-DBMS software that manages data.
- See Chapter 10 for details on Database Programming


Total authority
for those who dont know
Users:
DBA Staff
Casual Users
Application Programmers
Parametric Users
DDL Statements
Privileged Commands
Interactive Query
Application Programs
DDL Compiler
Query Compiler
Precompiler
Host Language Compiler
Query Optimizer
DML Compiler
Compiled Transactions
external view
DBA Commands, Queries, and Transactions
System Catalog/ Data Dictionary
Runtime Database Processor
Stored Data Manager
Concurrency Control/ Backup/Recovery Subsystems
Stored Database
Input/Output from Database
Query and Transaction Execution

CHP 5

RELATIONAL MODEL CONCEPTS

↓based on

Relation

↳ mathematical concept based on sets

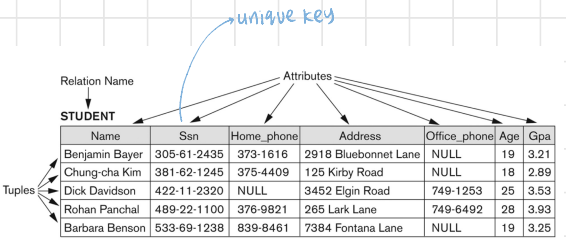
↳ contains a set of rows → tuples

Schema of Relation

CUSTOMER (Cust-id, Cust-name, Address, Phone#)

- CUSTOMER is the relation name
- Defined over the four attributes: Cust-id, Cust-name, Address, Phone#

- Each attribute has a **domain** or a set of valid values.
  - For example, the domain of Cust-id is 6 digit numbers.



Name	Ssn	Home_phone	Address	Office_phone	Age	Gpa
Benjamin Bayer	305-61-2435	373-1616	2918 Bluebonnet Lane	NULL	19	3.21
Chung-cha Kim	381-62-1245	375-4409	125 Kirby Road	NULL	18	2.89
Dick Davidson	422-11-2320	NULL	3452 Elgin Road	749-1253	25	3.53
Rohan Panchal	489-22-1100	376-9821	265 Lark Lane	749-6492	28	3.93
Barbara Benson	533-69-1238	839-8461	7384 Fontana Lane	NULL	19	3.25

Figure 5.1
The attributes and tuples of a relation STUDENT.

Formal Definitions - Tuple

- A **tuple** is an ordered set of values (enclosed in angled brackets '< … >')
- Each value is derived from an appropriate *domain*.
- A row in the CUSTOMER relation is a 4-tuple and would consist of four values, for example:
  - <632895, "John Smith", "101 Main St. Atlanta, GA 30332", "(404) 894-2000">
  - This is called a 4-tuple as it has 4 values
  - A tuple (row) in the CUSTOMER relation.
- A relation is a **set** of such tuples (rows)

- A **domain** has a logical definition:
  - Example: "USA_phone_numbers" are the set of 10 digit phone numbers valid in the U.S.
- A domain also has a data-type or a format defined for it.
  - The USA_phone_numbers may have a format: (ddd)ddd-dddd where each d is a decimal digit.
  - Dates have various formats such as year, month, date formatted as yyyy-mm-dd, or as dd mm,yyyy etc.
- The attribute name designates the role played by a domain in a relation:
  - Used to interpret the meaning of the data elements corresponding to that attribute
  - Example: The domain Date may be used to define two attributes named "Invoice-date" and "Payment-date" with different meanings

Definition Summary

Informal Terms		Formal Terms
Table		Relation
Column Header		Attribute
All possible Column Values		Domain
Row		Tuple
Table Definition		Schema of a Relation
Populated Table		State of the Relation

- All values are considered atomic (indivisible).
- Each value in a tuple must be from the domain of the attribute for that column

CONSTRAINTS

Constraints determine which values are permissible and which are not in the database.

They are of three main types:

1. **Inherent or Implicit Constraints**: These are based on the data model itself. (E.g., relational model does not allow a list as a value for any attribute)
2. **Schema-based or Explicit Constraints**: They are expressed in the schema by using the facilities provided by the model. (E.g., max. cardinality ratio constraint in the ER model)
3. **Application based or semantic constraints**: These are beyond the expressive power of the model and must be specified and enforced by the application programs.

- There are three *main types* of (explicit schema-based) constraints that can be expressed in the relational model:
  - **Key** constraints
  - **Entity integrity** constraints
  - **Referential integrity** constraints
- Another schema-based constraint is the **domain** constraint
  - Every value in a tuple must be from the *domain of its attribute* (or it could be **null**, if allowed for that attribute)

Relational Database State

- A **relational database state** DB of S is a set of relation states DB = $\{r_1, r_2, \ldots, r_m\}$ such that each r_i is a state of R_i and such that the r_i relation states satisfy the integrity constraints specified in IC.
- A relational database *state* is sometimes called a relational database *snapshot* or *instance*.
- We will not use the term *instance* since it also applies to single tuples.
- A database state that does not meet the constraints is an invalid state

Populated database state

- Each *relation* will have many tuples in its current relation state
- The *relational database state* is a union of all the individual relation states
- Whenever the database is changed, a new state arises
- Basic operations for changing the database:
  - INSERT a new tuple in a relation
  - DELETE an existing tuple from a relation
  - MODIFY an attribute of an existing tuple
- Next slide (Fig. 5.6) shows an example state for the COMPANY database schema shown in Fig. 5.5.

is in the employee

Entity Integrity

↳ more than 1 table

↳ Primary key → unique, can't be null

↳ forgien key → primary key of another table

- can be duplicated, be null
- name can be different
- domain will be same as primary key
- can't be a Primary key

EMPLOYEE

E.no	E.id	F.name	CNIC	Gender	Ph no	Add	D no

E.no: Primary key

D no: forgien key

dont create in Employee table add as forgien key

Department

Dno.	D.name	D cnic (Manager)	D strt date	D.location

Dno.: Primary key

NULL

123 → resigning

so will be deleted and

Will be null in other duplicate tables

Delete

it deleted all other tables will

↳ set null → be null

↳ set default → be default value set

↳ cacade → like nested loops, have make all tables null then delete

↳ reject

↳ restrict

LOCATION → this table has no Primary key → so we combine both and make the whole Primary key

D.no	D.location

D.no: forgien key

forgien key part of primary key so cannot null but can be duplicated

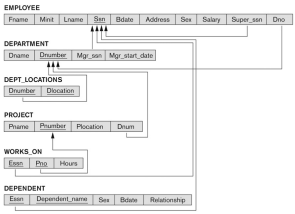
Refrential Integrity

↳ more than 1 table

↳ makes use of forgien key to connect tables

Referential Integrity Constraints for COMPANY database

Figure 5.7
Referential integrity constraints displayed on the COMPANY relational database schema.



Stored procedures

↳ functions that are stored already and called when needed

concatinate columns

column1 || column2 As joined

violated by each operation, if any, and the different ways of enforcing these constraints.

a. Insert <'Robert', 'F', 'Scott', '943775543', '1972-06-21', '2365 Newcastle Rd, Bellaire, TX', M, 58000, '888665555', 1> into EMPLOYEE.
b. Insert <'ProductA', 4, 'Bellaire', 2> into PROJECT.
c. Insert <'Production', 4, '943775543', '2007-10-01'> into DEPARTMENT.
d. Insert <'677678989', NULL, '40.0'> into WORKS_ON.
e. Insert <'453453453', 'John', 'M', '1990-12-12', 'spouse'> into DEPENDENT.
f. Delete the WORKS_ON tuples with Essn = '333445555'.
g. Delete the EMPLOYEE tuple with Ssn = '987654321'.
h. Delete the PROJECT tuple with Pname = 'ProductX'.
i. Modify the Mgr_ssn and Mgr_start_date of the DEPARTMENT tuple with Dnumber = 5 to '123456789' and '2007-10-01', respectively.
j. Modify the Super_ssn attribute of the EMPLOYEE tuple with Ssn = '999887777' to '943775543'.
k. Modify the Hours attribute of the WORKS_ON tuple with Essn = '999887777' and Pno = 10 to '5.0'.

5.12. Consider the AIRLINE relational database schema shown in Figure 5.8, which describes a database for airline flight information. Each FLIGHT is identified by a Flight_number, and consists of one or more FLIGHT_LEGs with Leg_numbers 1, 2, 3, and so on. Each FLIGHT_LEG has scheduled arrival and departure times, airports, and one or more LEG_INSTANCEs—one for each Date on which the flight travels. FAREs are kept for each FLIGHT. For each FLIGHT_LEG instance, SEAT_RESERVATIONs are kept, as are the AIRPLANE used on the leg and the actual arrival and departure times and airports. An AIRPLANE is identified by an Airplane_id and is of a particular AIRPLANE_TYPE. CAN_LAND relates AIRPLANE_TYPEs to the AIRPORTs at which they can land. An AIRPORT is identified by an Airport_code. Consider an update for the AIRLINE database to enter a reservation on a particular flight or flight leg on a given date.

a. Give the operations for this update.
b. What types of constraints would you expect to check?
c. Which of these constraints are key, entity integrity, and referential integrity constraints, and which are not?
d. Specify all the referential integrity constraints that hold on the schema shown in Figure 5.8.

5.13. Consider the relation CLASS(Course#, Univ_Section#, Instructor_name, Semester, Building_code, Room#, Time_period, Weekdays, Credit_hours). This represents classes taught in a university, with unique Univ_section#s. Identify what you think should be various candidate keys, and write in your own words the conditions or assumptions under which each candidate key would be valid.

Figure 5.6
One possible database state for the COMPANY relational database schema.

EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
John	B	Smith	123456789	1965-01-09	731 Fondren, Houston, TX	M	30000	333445555	5
Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5
Alicia	J	Zelaya	999887777	1968-01-19	3321 Castle, Spring, TX	F	25000	987654321	4
Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	43000	888665555	4
Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble, TX	M	38000	333445555	5
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5
Ahmad	V	Jabbar	987987987	1969-03-29	980 Dallas, Houston, TX	M	25000	987654321	4
James	E	Borg	888665555	1937-11-10	450 Stone, Houston, TX	M	55000	NULL	1

DEPARTMENT

Dname	Dnumber	Mgr_ssn	Mgr_start_date
Research	5	333445555	1988-05-22
Administration	4	987654321	1995-01-01
Headquarters	1	888665555	1981-06-19

DEPT_LOCATIONS

Dnumber	Dlocation
1	Houston
4	Stafford
5	Bellaire
5	Sugarland
5	Houston

WORKS_ON

Essn	Pno	Hours
123456789	1	32.5
123456789	2	7.5
666884444	3	40.0
453453453	1	20.0
453453453	2	20.0
333445555	2	10.0
333445555	3	10.0
333445555	10	10.0
333445555	20	10.0
999887777	30	30.0
999887777	10	10.0
987987987	10	35.0
987987987	30	5.0
987654321	30	20.0
987654321	20	15.0
888665555	20	NULL

PROJECT

Pname	Pnumber	Plocation	Dnum
ProductX	1	Bellaire	5
ProductY	2	Sugarland	5
ProductZ	3	Houston	5
Computerization	10	Stafford	4
Reorganization	20	Houston	1
Newbenefits	30	Stafford	4

DEPENDENT

Essn	Dependent_name	Sex	Bdate	Relationship
333445555	Alice	F	1986-04-05	Daughter
333445555	Theodore	M	1983-10-25	Son
333445555	Joy	F	1958-05-03	Spouse
987654321	Abner	M	1942-02-28	Spouse
123456789	Michael	M	1988-01-04	Son
123456789	Alice	F	1988-12-30	Daughter
123456789	Elizabeth	F	1967-05-05	Spouse

INSERT may violate any of the constraints:

- Domain constraint:
  - if one of the attribute values provided for the new tuple is not of the specified attribute domain
- Key constraint:
  - if the value of a key attribute in the new tuple already exists in another tuple in the relation
- Referential integrity:
  - if a foreign key value in the new tuple references a primary key value that does not exist in the referenced relation
- Entity integrity:
  - if the primary key value is null in the new tuple

Possible violations for each operation

- INSERT may violate any of the constraints:
  - Domain constraint:
    - if one of the attribute values provided for the new tuple is not of the specified attribute domain
  - Key constraint:
    - if the value of a key attribute in the new tuple already exists in another tuple in the relation
  - Referential integrity:
    - if a foreign key value in the new tuple references a primary key value that does not exist in the referenced relation
  - Entity integrity:
    - if the primary key value is null in the new tuple

- DELETE may violate only referential integrity:
  - If the primary key value of the tuple being deleted is referenced from other tuples in the database
    - Can be remedied by several actions: RESTRICT, CASCADE, SET NULL (see Chapter 6 for more details)
      - RESTRICT option: reject the deletion
      - CASCADE option: propagate the new primary key value into the foreign keys of the referencing tuples
      - SET NULL option: set the foreign keys of the referencing tuples to NULL
  - One of the above options must be specified during database design for each foreign key constraint

- UPDATE may violate domain constraint and NOT NULL constraint on an attribute being modified
- Any of the other constraints may also be violated, depending on the attribute being updated:
  - Updating the primary key (PK):
    - Similar to a DELETE followed by an INSERT
    - Need to specify similar options to DELETE
  - Updating a foreign key (FK):
    - May violate referential integrity
  - Updating an ordinary attribute (neither PK nor FK):
    - Can only violate domain constraints

Specifying Constraints in SQL

Basic constraints:

- Relational Model has 3 basic constraint types that are supported in SQL:
  - **Key** constraint: A primary key value cannot be duplicated
  - **Entity Integrity** Constraint: A primary key value cannot be null
  - **Referential integrity** constraints : The “foreign key “ must have a value that is already present as a primary key, or may be null.

CHP 6

SQL CREATE TABLE data definition statements for defining the COMPANY schema from Figure 5.7 (Fig. 6.1)

```
CREATE TABLE EMPLOYEE
        ( Fname                 VARCHAR(15)             NOT NULL,
          Minit                 CHAR,
          Lname                 VARCHAR(15)             NOT NULL,
          Ssn                   CHAR(9)                 NOT NULL,
          Bdate                 DATE,
          Address               VARCHAR(30),
          Sex                   CHAR,
          Salary                DECIMAL(10,2),
          Super_ssn             CHAR(9),
          Dno                   INT                     NOT NULL,
        PRIMARY KEY (Ssn),
CREATE TABLE DEPARTMENT
        ( Dname                 VARCHAR(15)             NOT NULL,
          Dnumber               INT                     NOT NULL,
          Mgr_ssn               CHAR(9)                 NOT NULL,
          Mgr_start_date        DATE,
        PRIMARY KEY (Dnumber),
        UNIQUE (Dname),
        FOREIGN KEY (Mgr_ssn) REFERENCES EMPLOYEE(Ssn) );
CREATE TABLE DEPT_LOCATIONS
        ( Dnumber               INT                     NOT NULL,
          Dlocation             VARCHAR(15)             NOT NULL,
        PRIMARY KEY (Dnumber, Dlocation),
        FOREIGN KEY (Dnumber) REFERENCES DEPARTMENT(Dnumber) );
```

continued on next slide

Attribute Data Types and Domains in SQL

- Basic **data types**
  - **Numeric** data types
    - Integer numbers: `INTEGER`, `INT`, and `SMALLINT`
    - Floating-point (real) numbers: `FLOAT` or `REAL`, and `DOUBLE PRECISION`
  - **Character-string** data types
    - Fixed length: `CHAR(n)`, `CHARACTER(n)`
    - Varying length: `VARCHAR(n)`, `CHAR VARYING(n)`, `CHARACTER VARYING(n)`
  - **Bit-string** data types
    - Fixed length: `BIT(n)`
    - Varying length: `BIT VARYING(n)`
  - **Boolean** data type
    - Values of `TRUE` or `FALSE` or `NULL`
  - **DATE** data type
    - Ten positions
    - Components are `YEAR`, `MONTH`, and `DAY` in the form YYYY-MM-DD
    - Multiple mapping functions available in RDBMSs to change date formats
- Additional data types
  - **Timestamp** data type

  Includes the `DATE` and `TIME` fields
    - Plus a minimum of six positions for decimal fractions of seconds
    - Optional `WITH TIME ZONE` qualifier
  - **INTERVAL** data type
    - Specifies a relative value that can be used to increment or decrement an absolute value of a date, time, or timestamp
  - **DATE, TIME, Timestamp, INTERVAL** data types can be **cast** or converted to string formats for comparison.

Integrity Constraints

- **`PRIMARY KEY`** clause
  - Specifies one or more attributes that make up the primary key of a relation
  - `Dnumber INT PRIMARY KEY;`
- **`UNIQUE`** clause
  - Specifies alternate (secondary) keys (called CANDIDATE keys in the relational model).
  - `Dname VARCHAR(15) UNIQUE;`

Specifying Constraints on Tuples Using CHECK

- Additional Constraints on individual tuples within a relation are also possible using CHECK
- `CHECK` clauses at the end of a `CREATE TABLE` statement
- Apply to each tuple individually
- `CHECK (Dept_create_date <= Mgr_start_date);`

Arithmetic Operators

- **`LIKE`** comparison operator
  - Used for string **pattern matching**
  - % replaces an arbitrary number of zero or more characters
  - underscore (_) replaces a single character
  - Examples: **WHERE** Address **LIKE** '%Houston,TX%';
  - **WHERE** Ssn **LIKE** '_ _ 1_ _ 8901';
- **`BETWEEN`** comparison operator

E.g., in Q14 :

WHERE(Salary **BETWEEN** 30000 **AND** 40000)
AND Dno = 5;

Aliases or tuple variables

- Declare alternative relation names E and S to refer to the EMPLOYEE relation twice in a query:

Query 8. For each employee, retrieve the employee's first and last name and the first and last name of his or her immediate supervisor.

```
SELECT  E.Fname, E.Lname, S.Fname, S.Lname
FROM    EMPLOYEE AS E, EMPLOYEE AS S
WHERE E.Super_ssn=S.Ssn;
```

Putting 's' will make join

- Recommended practice to abbreviate names and to prefix same or similar attribute from multiple tables.

```
U1:   INSERT INTO   EMPLOYEE
      VALUES        ( 'Richard', 'K', 'Marini', '653298653', '1962-12-30', '98
                      Oak Forest, Katy, TX', 'M', 37000, '653298653', 4 );
```

The variation below inserts multiple tuples v

BULK LOADING OF TABLES

- Another variation of **INSERT** is used for bulk-loading of several tuples into tables
- A new table TNEW can be created with the same attributes as T and using LIKE and DATA in the syntax, it can be loaded with entire data.
- EXAMPLE:

```
CREATE TABLE D5EMPS  LIKE  EMPLOYEE
   (SELECT   E.*
    FROM          EMPLOYEE AS E
   WHERE     E.Dno=5)
 WITH DATA;
```

DELETE

- Removes tuples from a relation
  - Includes a WHERE-clause to select the tuples to be deleted
  - Referential integrity should be enforced
  - Tuples are deleted from only *one table* at a time (unless CASCADE is specified on a referential integrity constraint)
  - A missing WHERE-clause specifies that *all tuples* in the relation are to be deleted; the table then becomes an empty table
  - The number of tuples deleted depends on the number of tuples in the relation that satisfy the WHERE-clause

```
U4A:   DELETE FROM    EMPLOYEE
       WHERE          Lname='Brown';
```

- Example: Change the location and controlling department number of project number 10 to 'Bellaire' and 5, respectively

```
U5:    UPDATE    PROJECT
       SET       PLOCATION = 'Bellaire',
                       DNUM = 5
       WHERE     PNUMBER=10
```

foreign key can't be null

Primary key can't be null

IN → =, include

* **Select** → columns
* **From** → tables
* **where** → rows, tuples, records → conditional
* **Order by** → DESC, ASC
* **Group by** → repetition → count, min, max, avg, sum
  - ↳ always with select clause
  - ↳ count never counts 'NULL'
  - ↳ group by without functions give distinct clause

```
SELECT
    model_year,
    (count (product_id) AS product_count
FROM products
GROUP BY model_year   → as repeated
```

↳ can't write orderby here, as aggregate function

```
Select
    count (DISTINCT(column_name/* ))   → unique values
    AS count_no
from Table
```

```
Select Dep
From Departments
Group by Dep
```

↓ only one colum can't add more UNLESS others are aggregate

```
SELECT
    COUNT(CASE when price >=500 THEN product_id END)
    AS expensive
FROM products
```

* **Having** → group by → conditional
  - ↳ doesn't work on single clause
  - ↳ on complex group

* **Aggregate function**
  - ↳ always use group by with it
  - ↳ can't use distinct with aggregate function
  - ↳ count, min, max, avg, sum

Employee

Primary key

foreign key

E_id	E_name	D no
123	Ali	1
321	Bilal	3
419	Anmed	3
512	Hassan	1

Department

foreign key

Primary key

D_no	Dname	Dmang_id
1	CS	512
2	EE	NULL
3	AI	321

Select * from Employee E (alias), Department D (alias) where E.E_id = D.Dmang_id

→ **cross product**

* if theres a common column it should be a primary key

E_id	E_name	Dno	Dno	Dname	Dmang_id
123	Ali	1	1	CS	512
123	Ali	1	2	EE	NULL
123	Ali	1	3	AI	321
321	Bilal	3	1	CS	512
321	Bilal	3	2	EE	NULL
321	Bilal	3	3	AI	321
419	Anmed	3	1	CS	512
419	Anmed	3	2	EE	NULL
419	Anmed	3	3	AI	321
512	Hassan	1	1	CS	512
512	Hassan	1	2	EE	NULL
512	Hassan	1	3	AI	321

↘ might not work on oracle compiler as it cant support

Select * from Employee E, Department D where E.E_id = D.Dmang_id

E_id	E_name	Dno	Dno	Dname	Dmang_id
321	Bilal	3	3	AI	321
512	Hassan	1	1	CS	512

Select * from Employee E, Department D where E.E_id = D.Dmang_id AND D.Dname = 'CS'

E_id	E_name	Dno	Dno	Dname	Dmang_id
512	Hassan	1	1	CS	512

JOIN

- by default inner join
- b/w two tables
- necessary b/w common columns
- better query optimization compared to SQL
  - explain/analyse select * from Employees
  - create index ON Employee USING Btree (Department)

Cross Join
- diff tables
- cartesian product

↳ doesnt have where clause

```
Select * from Employee, Department
```
↳ join clause

Self Join
- same tables

```
Select E.fname, E.lname, S.Fname, s.lname
from Employee E JOIN Employee S
ON E.SSN = S.SuperSNN
```
↳ self and inner join

INNER JOIN
↳ returns matching rows

Natural join
↳ if Table 1 and Table 2 have same column name

```
Select * from Department D
JOIN dept_locations
USING D.number
```

EQUI JOIN
↳ if '=' written

```
Select E.fname, E.lname, S.Fname, s.lname
from Employee E, Employee S
where E.SSN = S.SuperSNN
```
- where: if not written cross join
- =: equal join

Outer Join
- for not ordered entities

↳ returns matching rows

↳ to check where NULL comes

Left JOIN
↳ full info of left table and some info of right

LEFT OUTER JOIN

Full JOIN
↳ full info of left and right table

FULL JOIN

Right JOIN
↳ full info of right table and some info of left

↳ if no matching rows then displays NULL

RIGHT JOIN

How to Join more than 2 Tables

not in Paper

```
Select column_name
From T1 Join T2
ON column_name
JOIN T3
ON T2._ = T3._
```

Equal JOIN
- very expensive
- too many computations
- slows system down

which is better

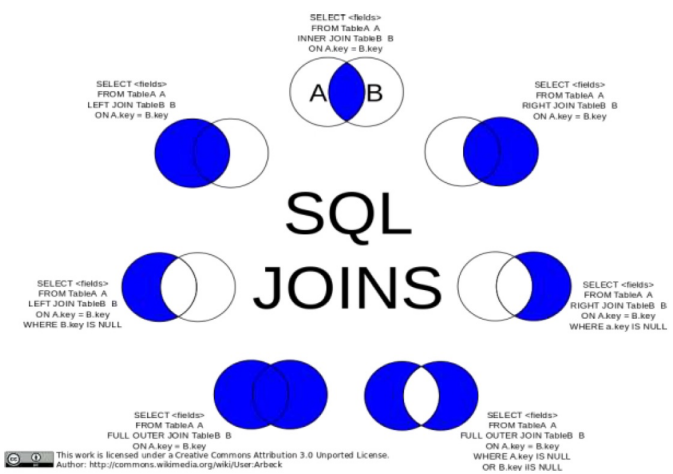
Sub Query OR	JOIN
↳ if no common column	↳ on common columns
↳ if no of tables less	↳ if no of tables greater
↳ easir to read	↳ retrieve data from multiple tables simultaneously
↳ filter or aggregate data	

```
START_DATE = TO_DATE('2001-01-13','YYYY-MM-DD'));
```

TO_CHAR (Hire_date, 'Month')
Extract (year From Hire_date)
TO_DATE (hire_date, 'DD-MM-YYYY)

name like '%a_'
substr (name, -3)
Round (num, 1)

Conclusion for Joins:



ishma hafeez notes

INTRODUCTION TO SET OPERATOR

Query 18. Retrieve the names of all employees who do not have supervisors.

```
Q18:  SELECT   Fname, Lname
      FROM     EMPLOYEE
      WHERE    Super_ssn IS NULL;
```

Nested Queries (cont'd.)

- Use tuples of values in comparisons
  - Place them within parentheses

```
SELECT    DISTINCT Essn
FROM      WORKS_ON
WHERE     (Pno, Hours) IN ( SELECT    Pno, Hours
                            FROM      WORKS_ON
                            WHERE     Essn='123456789' );
```

- Use other comparison operators to compare a single value *v*
  - `= ANY` (or `= SOME`) operator
    - Returns `TRUE` if the value *v* is equal to some value in the set *V* and is hence equivalent to `IN`
  - Other operators that can be combined with `ANY` (or `SOME`): >, >=, <, <=, and <>
  - `ALL`: value must exceed all values from nested query

```
SELECT    Lname, Fname
FROM      EMPLOYEE
WHERE     Salary > ALL  ( SELECT    Salary
                          FROM      EMPLOYEE
                          WHERE     Dno=5 );
```


Correlated Nested Queries

- **Queries that are nested using the = or IN comparison operator** can be collapsed into one single block: E.g., Q16 can be written as:

```
Q16A:   SELECT    E.Fname, E.Lname
        FROM      EMPLOYEE AS E, DEPENDENT AS D
        WHERE     E.Ssn=D.Essn AND E.Sex=D.Sex
                                AND
                  E.Fname=D.Dependent_name;
```

- **Correlated** nested query
  - Evaluated once for each tuple in the outer query

USE of EXISTS

Q7:

```
SELECT Fname, Lname
FROM Employee
WHERE EXISTS (SELECT *
              FROM DEPENDENT
              WHERE Ssn= Essn)

      AND EXISTS (SELECT   *
                  FROM Department
                  WHERE Ssn= Mgr_Ssn)
```

we use double negation this way in SQL.

Query: List first and last name of employees who work on ALL projects controlled by Dno=5.

```
SELECT Fname, Lname
FROM Employee
WHERE NOT EXISTS ( (SELECT  Pnumber
                    FROM PROJECT
                    WHERE Dno=5)

                  EXCEPT (SELECT   Pno
                          FROM WORKS_ON
                          WHERE Ssn= ESsn)
```

The above is equivalent to double negation: List names of those employees for whom there does NOT exist a project managed by department no. 5 that they do NOT work on.

- Can use explicit set of values in WHERE clause

```
Q17:    SELECT    DISTINCT Essn
        FROM      WORKS_ON
        WHERE     Pno IN (1, 2, 3);
```

Example: LEFT OUTER JOIN

```
SELECT E.Lname AS Employee_Name
       S.Lname AS Supervisor_Name

FROM Employee AS E LEFT OUTER JOIN EMPLOYEE AS S
                   ON E.Super_ssn = S.Ssn)
```

ALTERNATE SYNTAX:

```
SELECT E.Lname , S.Lname
FROM  EMPLOYEE E, EMPLOYEE S
WHERE E.Super_ssn + = S.Ssn
```

Renaming Results of Aggregation

- Following query returns a single row of computed values from EMPLOYEE table:

```
Q19:    SELECT    SUM (Salary), MAX (Salary), MIN (Salary), AVG
                  (Salary)
        FROM      EMPLOYEE;
```

  - Specifies grouping attributes
- COUNT (*) counts the number of rows in the group

Grouping: The GROUP BY and HAVING Clauses (cont'd.)

- **HAVING** clause
  - Provides a condition to select or reject an entire group:
- **Query 26.** For each project *on which more than two employees work*, retrieve the project number, the project name, and the number of employees who work on the project.

```
Q26:    SELECT      Pnumber, Pname, COUNT (*)
        FROM        PROJECT, WORKS_ON
        WHERE       Pnumber=Pno
        GROUP BY    Pnumber, Pname
        HAVING      COUNT (*) > 2;
```

Combining the WHERE and the HAVING Clause

- Consider the query: we want to count the *total* number of employees whose salaries exceed $40,000 in each department, but only for departments where more than five employees work.

- INCORRECT QUERY:

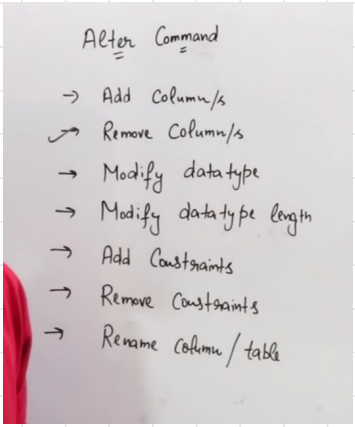
```
SELECT      Dno, COUNT (*)
FROM        EMPLOYEE
WHERE       Salary>40000
GROUP BY    Dno
HAVING      COUNT (*) > 5;
```

Correct Specification of the Query:

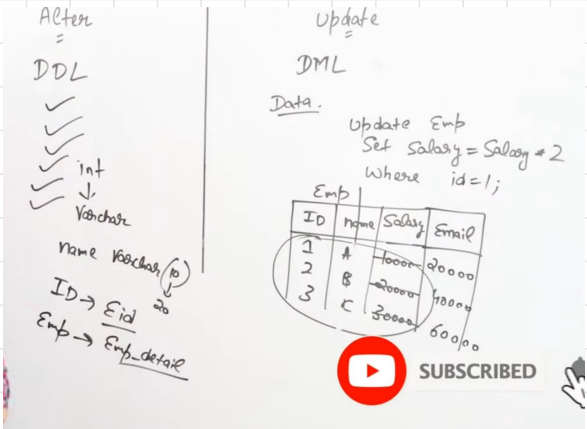
- Note: the WHERE clause applies tuple by tuple whereas HAVING applies to entire group of tuples

Query 28. For each department that has more than five employees, retrieve the department number and the number of its employees who are making more than $40,000.

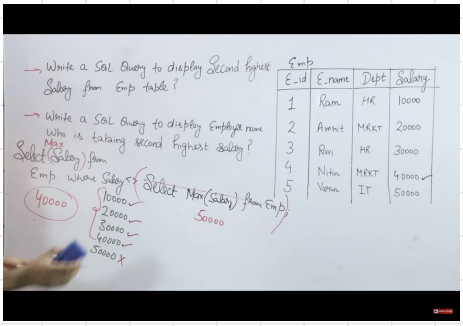
```
Q28:    SELECT      Dnumber, COUNT (*)
        FROM        DEPARTMENT, EMPLOYEE
        WHERE       Dnumber=Dno AND Salary>40000 AND
                    ( SELECT      Dno
                      FROM        EMPLOYEE
                      GROUP BY Dno
                      HAVING      COUNT (*) > 5)
```

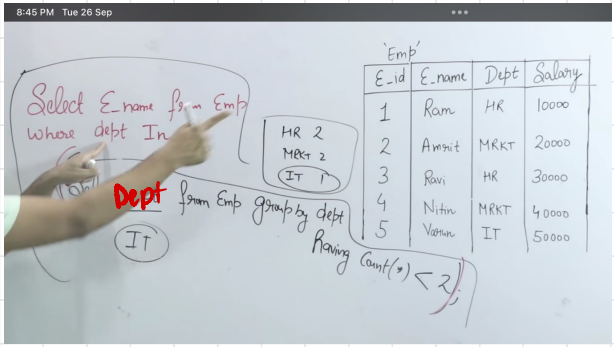


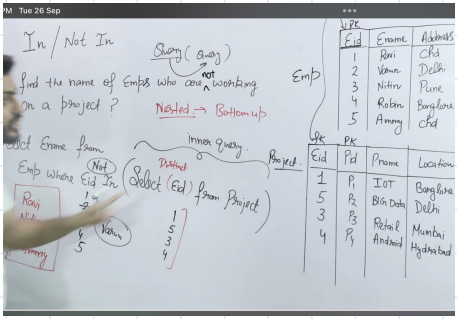

```
alter table employee add address varchar(10);
desc emp11
alter table employee drop column address;
alter table employee modify id varchar(30);
alter table employee rename column id to roll_no;
alter table employee rename to emp11;
```

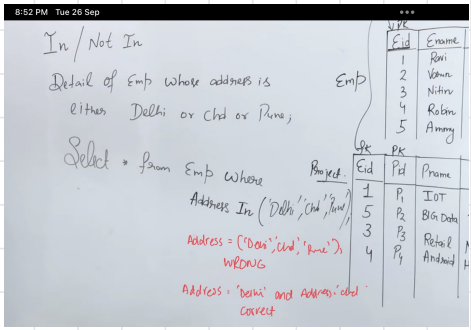


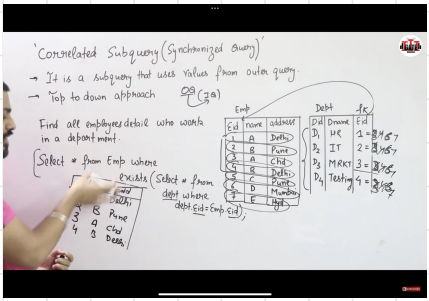
1. capslock matters in 'strings'
2. use IN for listing IN ('101','102') instead of =

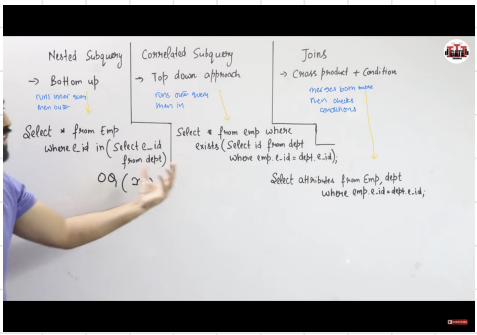

Write a SQL Query to display Second highest Salary from Emp table?
Write a SQL Query to display Employee name who is taking second highest salary?
Select Max(Salary) from Emp where Salary < (Select Max(Salary) from Emp)
40000
10000
20000
30000
40000
30000 X
50000
Emp
E_id | E_name | Dept | Salary
1 | Ram | HR | 10000
2 | Amrit | MRKT | 20000
3 | Ravi | HR | 30000
4 | Nitin | MRKT | 40000
5 | Varun | IT | 50000

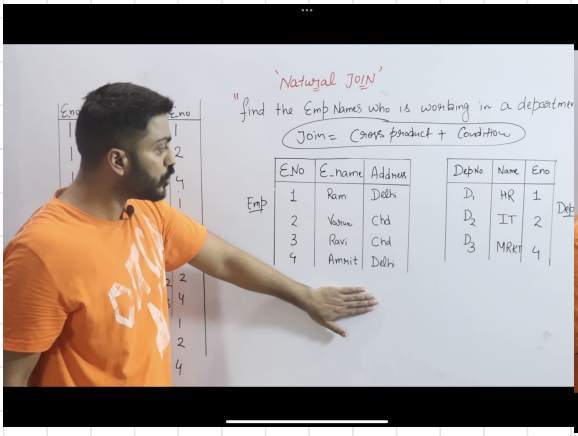

8:45 PM Tue 26 Sep
Select E_name from Emp where dept In
Dept from Emp group by dept
Having Count(*) < 2;
HR 2
MRKT 2
IT 1
IT
'Emp'
E_id | E_name | Dept | Salary
1 | Ram | HR | 10000
2 | Amrit | MRKT | 20000
3 | Ravi | HR | 30000
4 | Nitin | MRKT | 40000
5 | Varun | IT | 50000

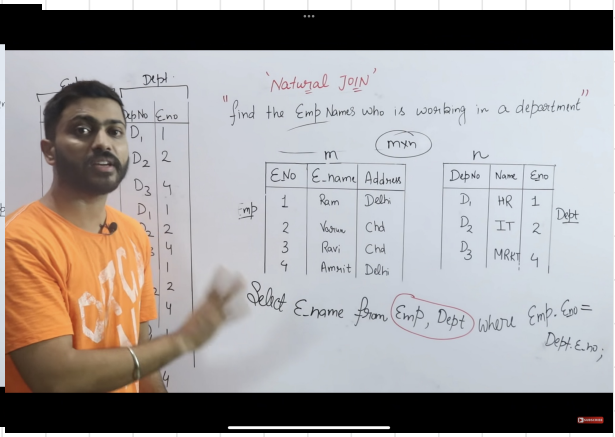

PM Tue 26 Sep
In / Not In
Query (query)
not
find the name of Emps who are working in a project?
Nested → Bottom up
Select Ename from Emp Where Eid In (Select (Eid) from Project)
Inner query
Distinct
Not
Ravi
Nitin
Varun
Emp
PK
Eid | Ename | Address
1 | Ravi | Chd
2 | Varun | Delhi
3 | Nitin | Pune
4 | Robin | Bangalore
5 | Ammy | Chd
Project
FK | PK
Eid | Pid | Pname | Location
1 | P1 | IOT | Bangalore
5 | P2 | BIG Data | Delhi
3 | P3 | Retail | Mumbai
4 | P4 | Android | Hydrabad

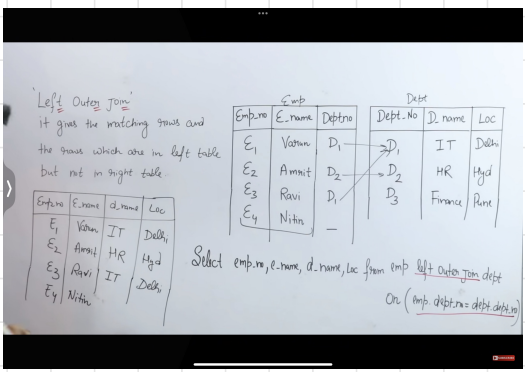

8:52 PM Tue 26 Sep
In / Not In
Detail of Emp whose address is either Delhi or Chd or Pune;
Select * from Emp Where Address In ('Delhi','Chd','Pune');
Address = ('Delhi','Chd','Pune'); WRONG
Address = 'Delhi' and Address = 'Chd' correct
Emp
PK
Eid | Ename
1 | Ravi
2 | Varun
3 | Nitin
4 | Robin
5 | Ammy
Project
FK | PK
Eid | Pid | Pname
1 | P1 | IOT
5 | P2 | BIG Data
3 | P3 | Retail
4 | P4 | Android

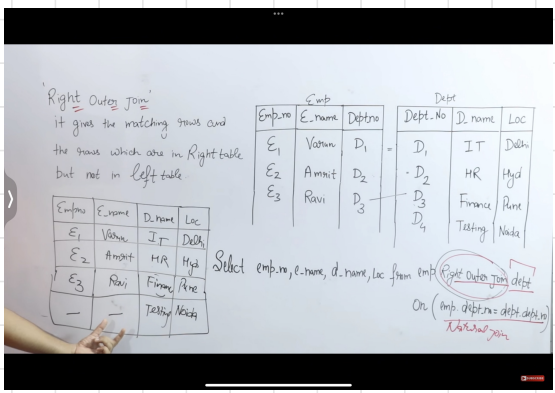

'Correlated Subquery (Synchronized Query)'
→ It is a subquery that uses values from outer query
→ Top to down approach
Find all employees detail who works in a department.
Select * from Emp where exists (Select * from dept where dept.Eid = Emp.Eid);
Emp
Eid | name | address
1 | A | Delhi
2 | B | Pune
3 | A | Chd
4 | B | Delhi
5 | C | Pune
6 | D | Mumbai
7 | E | Hyd
Dept
Dept | Dname | Eid
D1 | HR | 1 = 3 ✓
D2 | IT | 2 = 3 ✗
D3 | MRKT | 3 = 3 ✓
D4 | Testing | 4 = 3 ✗


Nested Subquery
→ Bottom up
runs inner query then outer
Select * from Emp where E_id in (Select E_id from dept)
Correlated Subquery
→ Top down approach
runs outer query then in
Select * from Emp where exists (Select id from dept where emp.E_id = dept.E_id);
Joins
→ Cross product + Condition
matches both tables then checks conditions
Select attributes from Emp, dept where emp.E_id = dept.E_id;









SELECT * FROM jobs where job_title in ('President','Accountant');

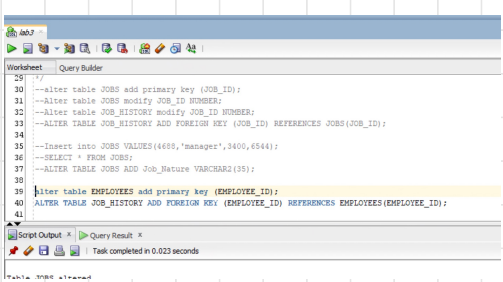
(d) Pattern Match Search Condition: The search condition involves searching for a particular character or string within a column value. Like Operator with the help of pattern matching symbols (_, %) are used find patterns in the column's value. **'_' represents a single character while '%' represents a sequence of characters.**

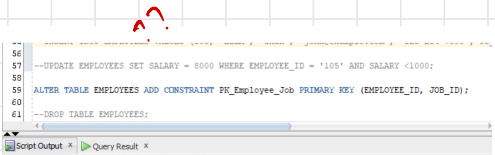
Example: List all the employees whose names contains an 'a' in their first names.

SELECT * FROM employees WHERE first_name LIKE '%a%';

OR **List all employees having L as second letter in their first names.**

SELECT * FROM employees WHERE first_name LIKE '_a %';





- **COMMENT** - add comments to the data dictionary
- **RENAME** - rename an object

1. Create

There are two CREATE statements available in SQL:

- CREATE DATABASE

Example Syntax:
- CREATE DATABASE database_name;
  database_name: name of the database.

- CREATE TABLE

Example Syntax:
- CREATE TABLE table_name (column1 data_type(size), column2 data_type(size), column3 data_type(size),...);

table_name: name of the table.
column1 name of the first column.
data_type: Type of data we want to store in the particular column.
size: Size of the data we can store in a particular column.

2. Alter

ALTER TABLE is used to add, delete/drop or modify columns in the existing table. It is also used to add and drop various constraints on the existing table.

Example Syntax:

ADD
- ALTER TABLE table_name ADD (Columnname_1 datatype, Columnname_2 datatype, ... Columnname_n datatype);

Drop
- ALTER TABLE table_name DROP COLUMN column_name;

Modify
- ALTER TABLE table_name MODIFY column_name column_type;

3. Drop

DROP is used to delete a whole database or just a table. The DROP statement destroys the objects like an existing database, table, index, or view. A DROP statement in SQL removes a component from a relational database management system (RDBMS).

Example Syntax:
- DROP TABLE table_name;
  table_name: Name of the table to be deleted.

- DROP DATABASE database_name;
  database_name: Name of the database to be deleted.

QUESTION 4

1. Define a table constraint on Emp that will ensure that every employee makes at least $10,000.

```
ALTER TABLE Emp
ADD CONSTRAINT MinSalaryCheck CHECK (salary >= 10000);
```

2. Define a table constraint on Dept that will ensure that all managers have age > 30.

```
ALTER TABLE Dept
ADD CONSTRAINT ManagerAgeCheck CHECK (managerid IS NULL OR age > 30);
```