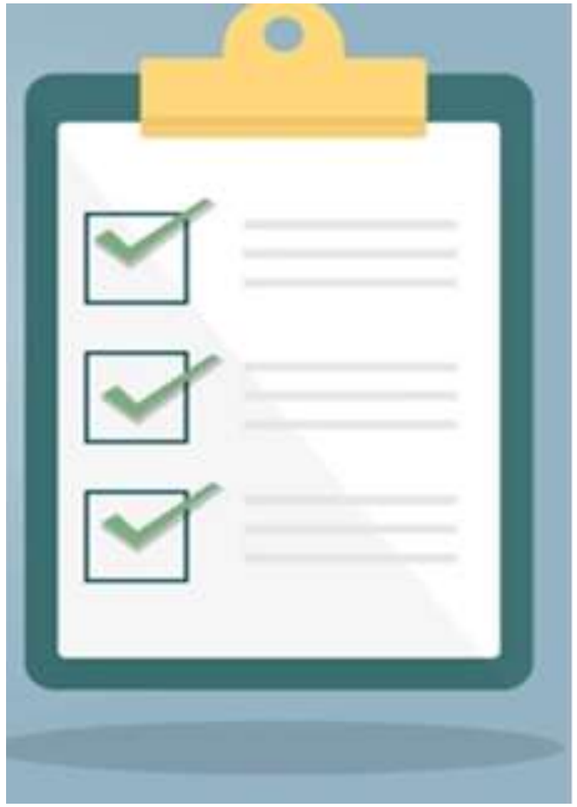
CC5051 - DATABASES

Introduction to Database
&
Database Management System

WEEK 1

Agendas

- Introduction to Module
- Structure of the Course
- Module Aim
- Learning Outcomes
- Assessment



- Data and Information
- Manual File System
- File Processing System
- Database
- Database Management System

Introduction to Module and Indicative Syllabus

- Module Code: CC5051
- Module Title: Databases
- Module Level: Intermediate(05)
- Credit Rating For Module: 15

- Databases and Database Management Systems
- Data Analysis and Modelling
- Database Models
- Introduction to Database languages and SQL
- Relational Database Theory
- Relational Database Languages
- Relational Algebra

Module Aim

- Allows students to understand, and put into practice, techniques available for **Analysing, Designing and Developing Database Systems**.
- To give introduction to **issues governing Design and Implementation of Database Systems.**
- To provide I**ntroduction** to both **Theoretical aspects** of **Designing sound Database Systems** as well as **Practical aspects** of Implementing such Systems.

Structure of the Course

- 1 Lecture (1.5 hours/week)
- 1 Tutorial (1.5 hours/week)
- 1 Computer Lab / Workshop (1.5 hours/week)
- Total study hours: 150
  - Scheduled learning & teaching activities:36 hours
  - Guided independent study: 72 hours
  - Assessment Preparation / Delivery: 42 hours

Assessment

- Assessment will consist of two components
- MCQ + Written Exam (40%)
- Coursework (60 %)
- Followed by a compulsory viva
  - Assess students' understanding and application of theoretical concepts (e.g., design notation and modeling).
  - Evaluate practical proficiency in using popular DBMS environments.
  - Ensure a balanced assessment of both theory and hands-on skills gained through tutorials and coursework.

Learning Outcomes

- Produce an Entity-Relationship model from a realistic Problem Specification
- Use Formal Design Techniques (e.g. Normalisation) to produce Database Schema
- Design and Implement a Database System from a Conceptual Data Model
- Manipulate / Extract data from the Database using Relational Algebra and SQL
- Discuss the relative merits of the Relational environment

Data and Information

Data

- Raw, unorganized fact that doesn’t have meaning
- Can be in the form of a number, figure, character, symbol, audio, video and so on

Information

- **Processed data** that gives **meaning**
- Suitable for **human interpretation**
- Can be used for **decision-making**

Data and Information

Data

Processing

Information

Data are **raw facts**; or **Unprocessed information**

Process involves **acquisition, storage, manipulation, retrieval and distribution.**

Information is data that have been **processed and presented** in a form suitable for human interpretation

Data and Information

Data

Processing

Information

Raw, unorganized facts.

A list of numbers —

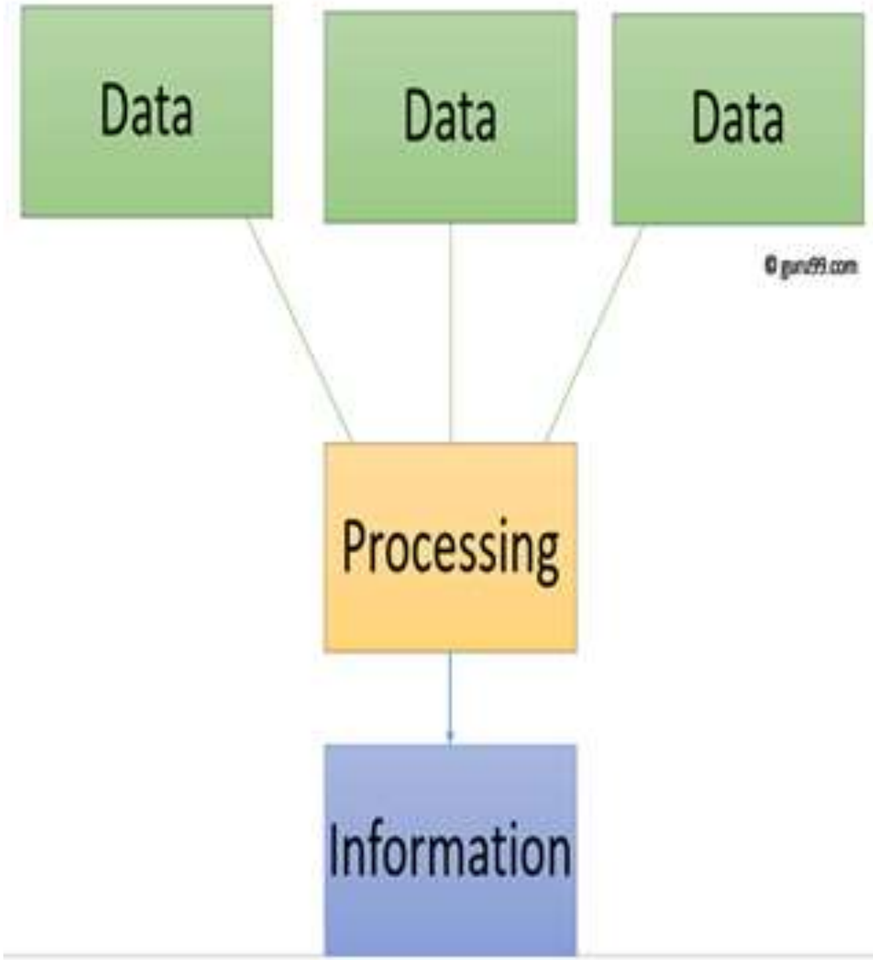
90, 85, 76, 88

(90 + 85 +76 +88)/4= 84.75

These numbers represent the marks obtained by students in a test. The average score is 84.75.

Data versus Information

- Data constitutes building blocks of Information
- Information produced by processing Data
- Information reveals meaning of Data
- Good, timely and relevant Information keys to decision making
- Good decision making keys to organizational survival



Q. Which of the following statements is correct?

1. Data can only be understood by highly skilled people.
2. Data is described as the processed information.
3. Data refers to the raw and unprocessed facts and figures that have no context or purposeful meaning.
4. Data is irrelevant facts and is worthless.

Q. Which of the following statements is correct?

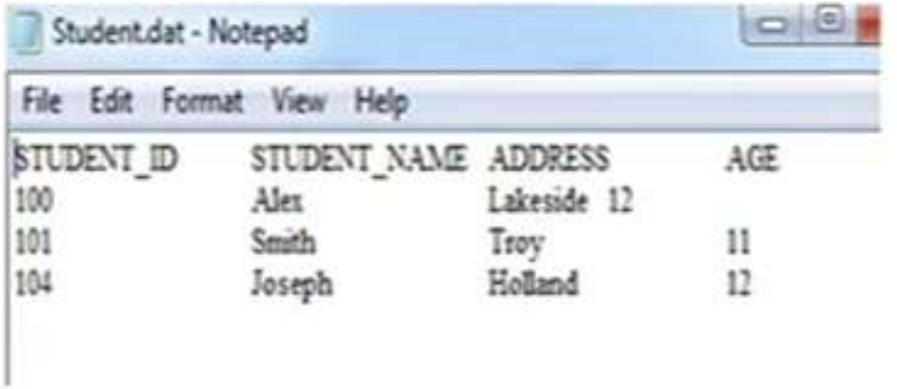
1. Data can only be understood by highly skilled people.
2. Data is described as the processed information.
3. Data refers to the raw and unprocessed facts and figures that have no context or purposeful meaning.
4. Data is irrelevant facts and is worthless.

Manual File Systems

- Traditionally composed of a collection of **File Folders kept in a File Cabinet**
- Organization within Folders was based on **data's expected use** (Ideally, logically related)
- The system was adequate for small amounts of data with few reporting requirements
- Finding and using data in growing collections of File Folders became time-consuming and cumbersome

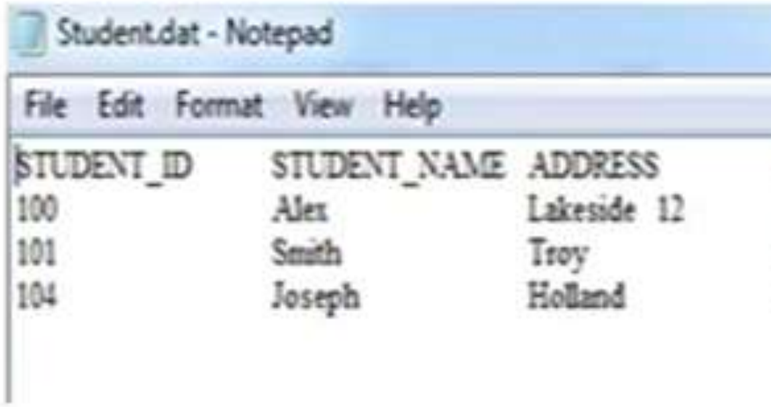
File Processing Approach

- A file processing system is a collection of programs that store and manage files in computer hard-disk.
- Traditional approach to Information System Design.
- In a typical file processing system, each department or area within an organization has its own set of files.



File Processing Approach

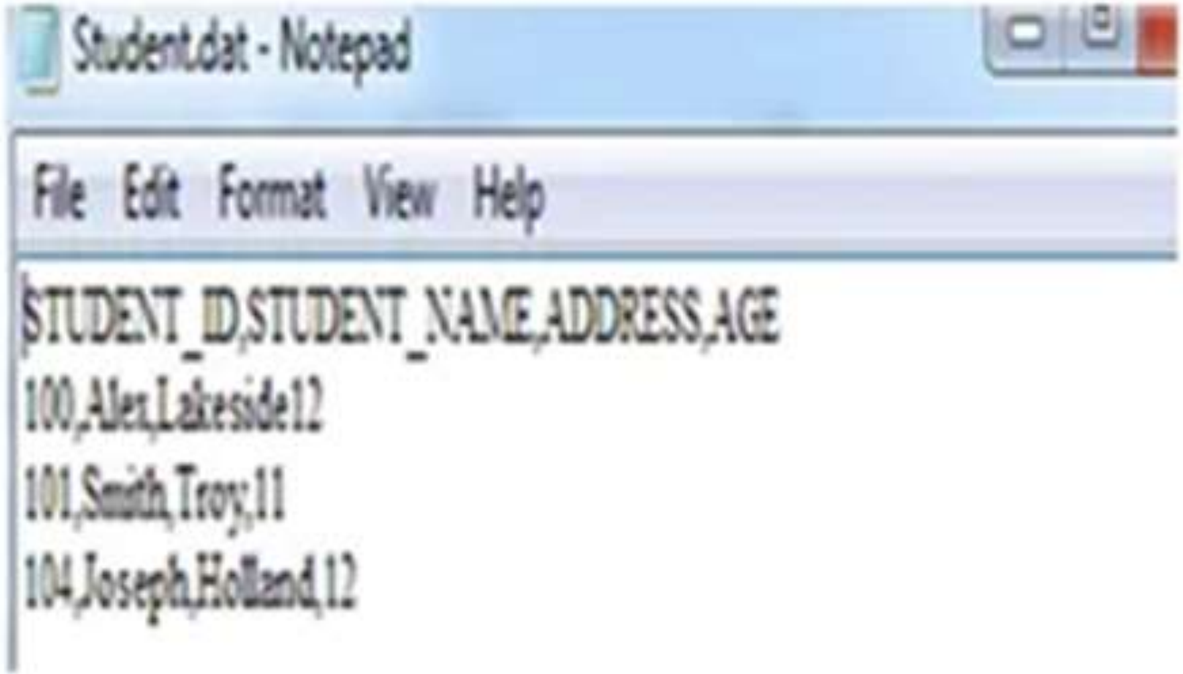
- The records in one file may not relate to the records in any other file.
- Organizations have used file processing systems for many years.
- Many of these systems, however, have two major weaknesses: they have redundant data and they isolate data.
- Data separated and isolated.



File Processing Approach

Five major operations can be performed on file are:

- Creation of a new file.
- Opening an existing file.
- Reading data from a file.
- Writing data in a file.
- Closing a file



File Processing Approach

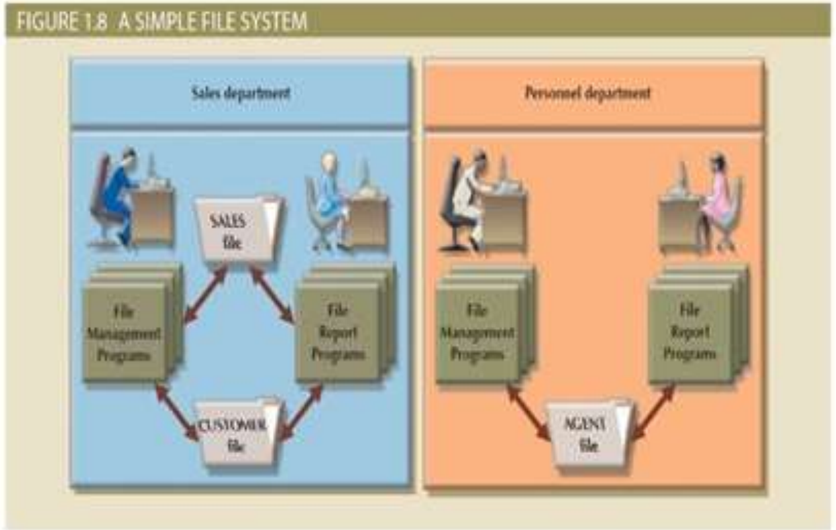
- Focuses on data processing needs of individual departments.
- Data often duplicated
- Application programs dependent on file formats
- Files often incompatible with one another
- Difficult to represent data in a user’s perspective

Evolution of Simple File System

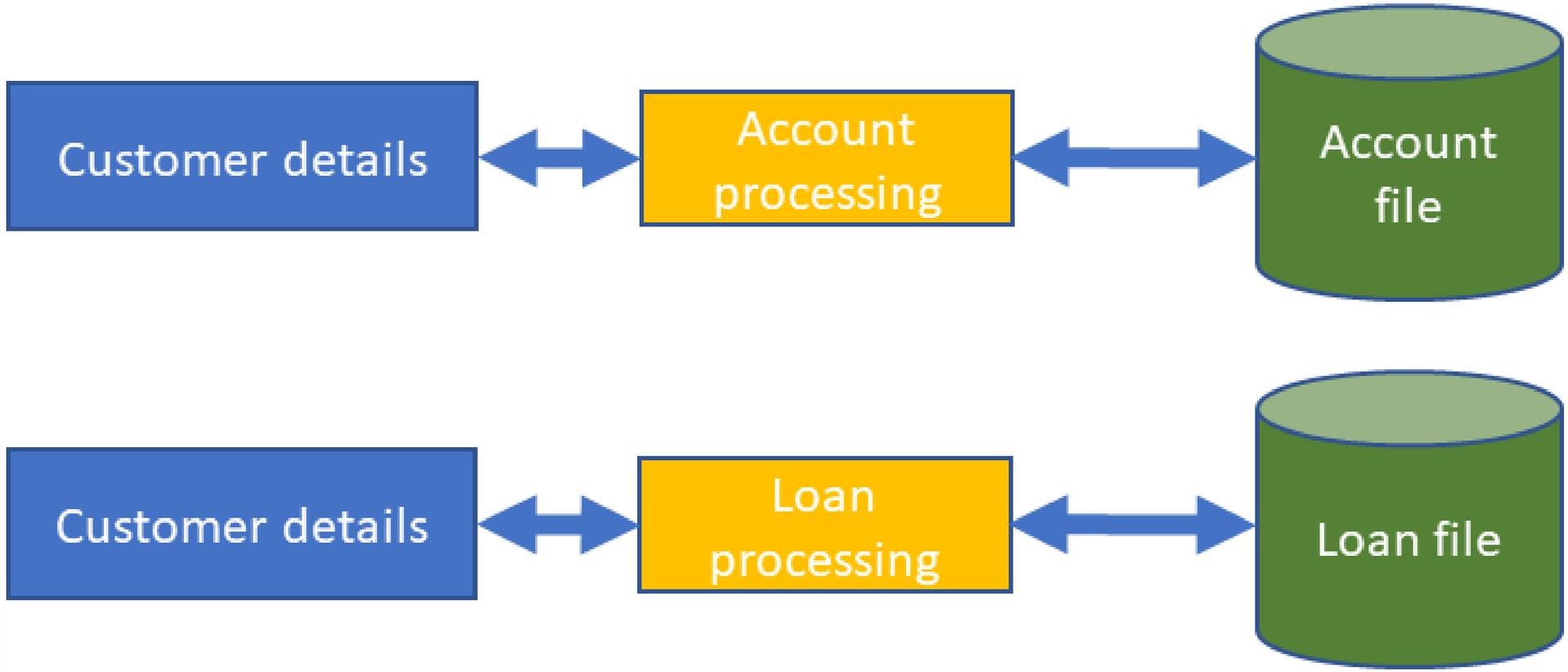
Manual files: Paper based storage in cabinets , slow and error prone.

File processing approach: Each program kept its own data file, caused redundancy and isolation.

Indexed / structured files: Uses indexing for faster access and data retrieval but still inflexible.



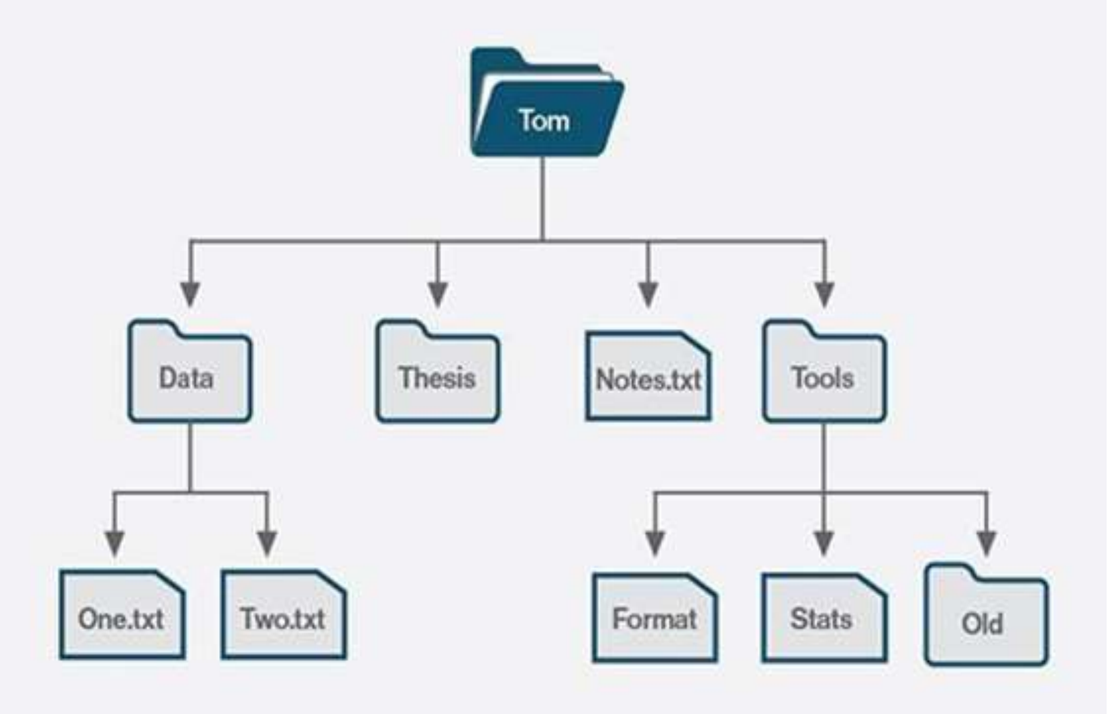
File Processing Approach



File Processing Approach

Advantages

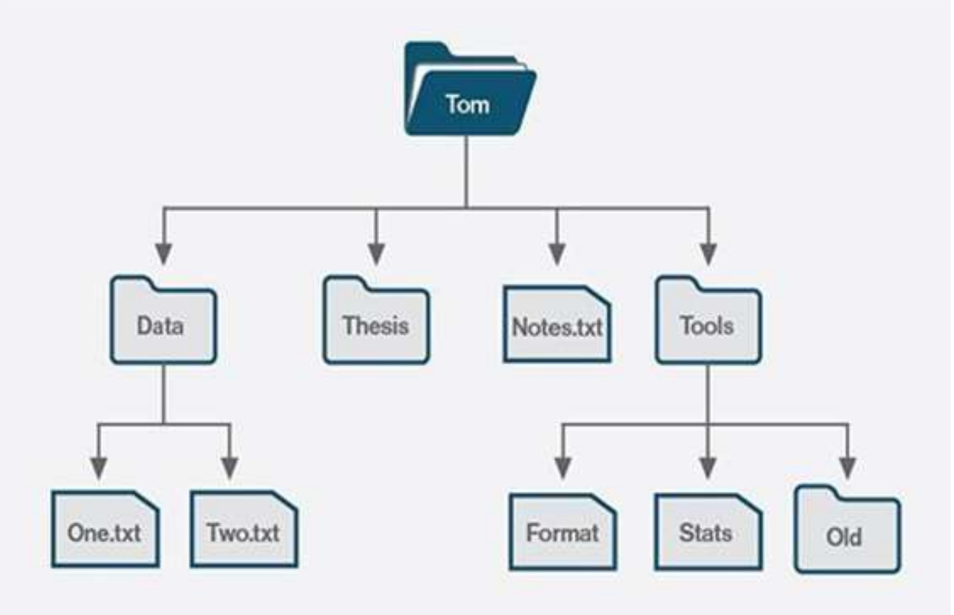
- Simple to design and implement.
- Low setup and maintenance cost.
- Fast access for well-defined, specific tasks.



File Processing Approach

Disadvantages

- Uncontrolled Redundancy
- Inconsistent Data
- Inflexibility
- Limited Data Sharing
- Poor Enforcement of Standards
- Excessive Program Maintenance



Database

- It is **collection of interrelated data** stored together **without unnecessary redundancy** to serve multiple operations.
- Databases can store, manipulate and retrieve data
- Used across wide range of organizations
- Single user to hundreds (or thousands) of users

Database vs. File System

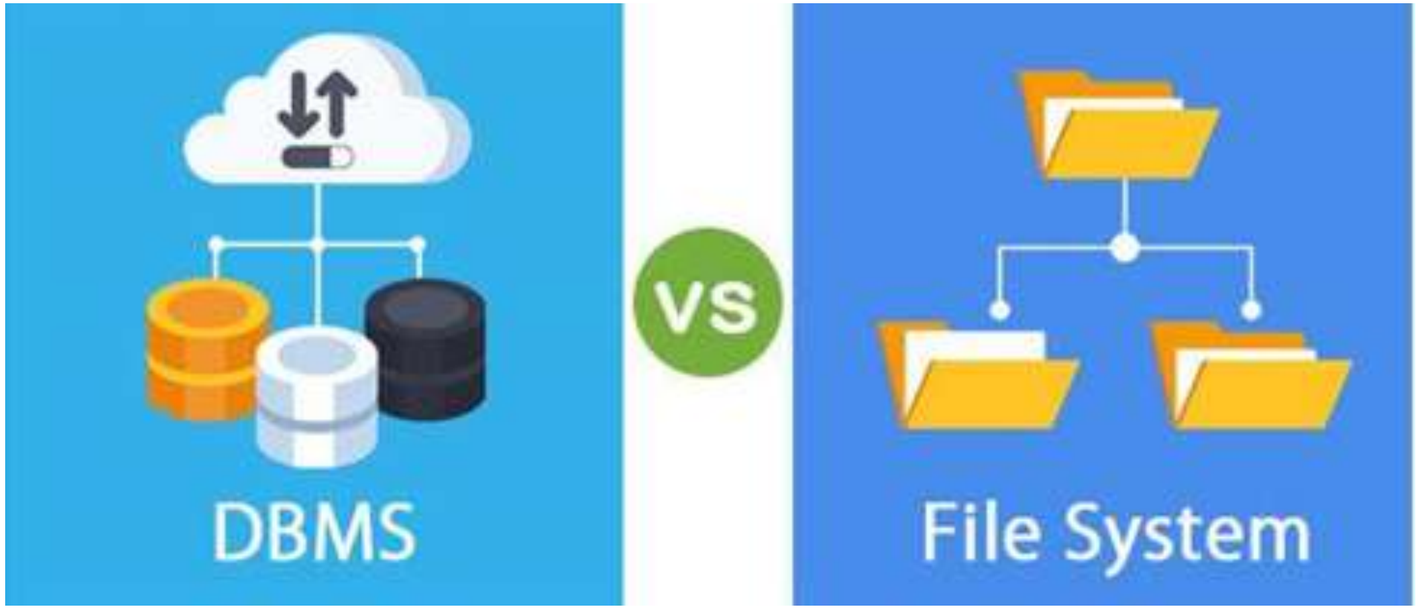
- Problems inherent in File Systems make using a Database System desirable

File system

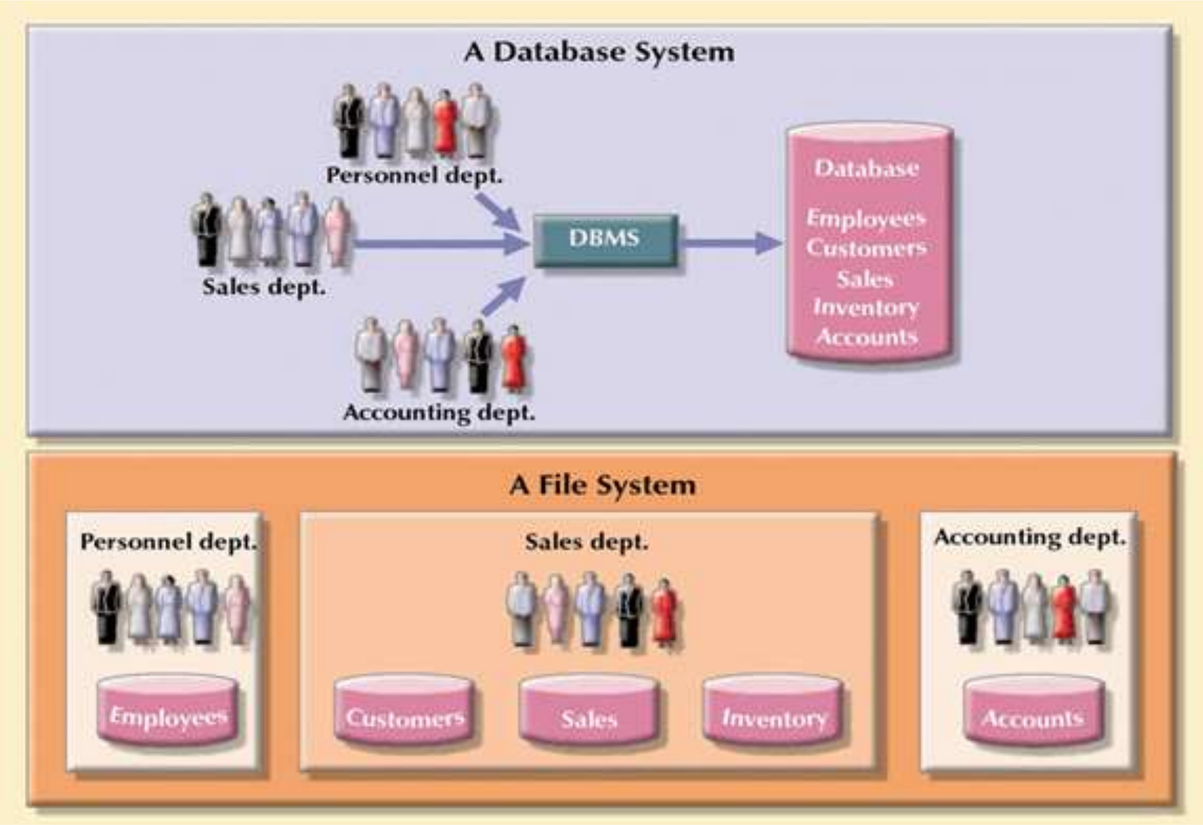
- Many separate and unrelated files

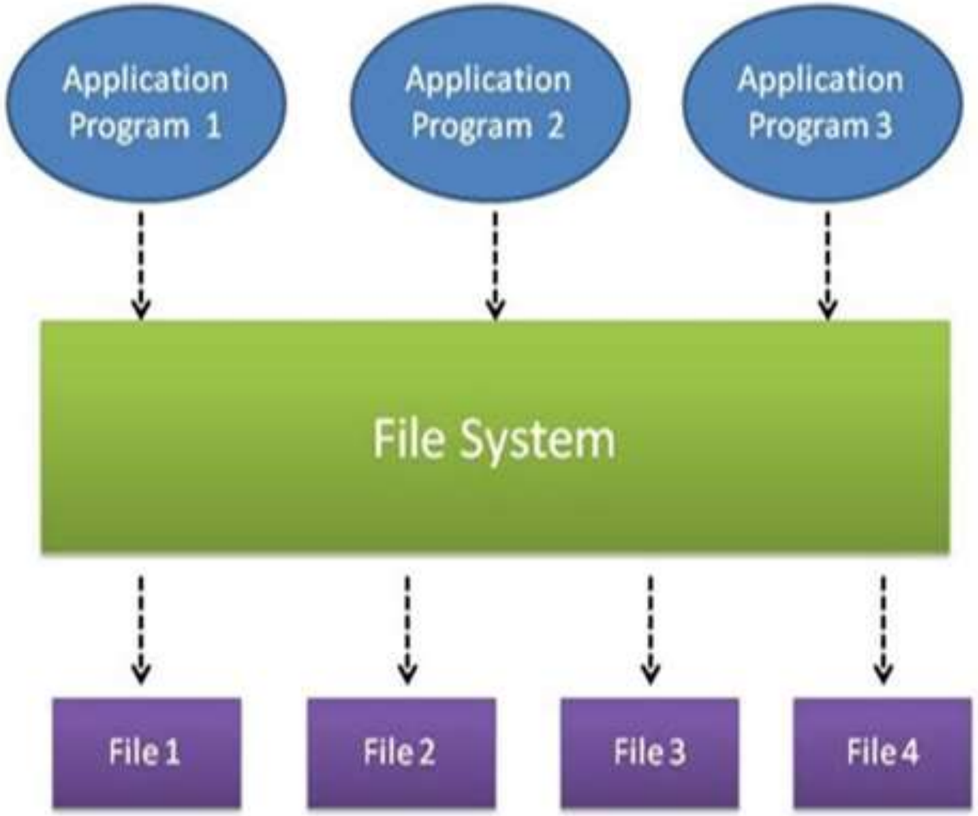
Database

- Logically related data stored in a single Logical Data Source



Contrasting Database and File Systems



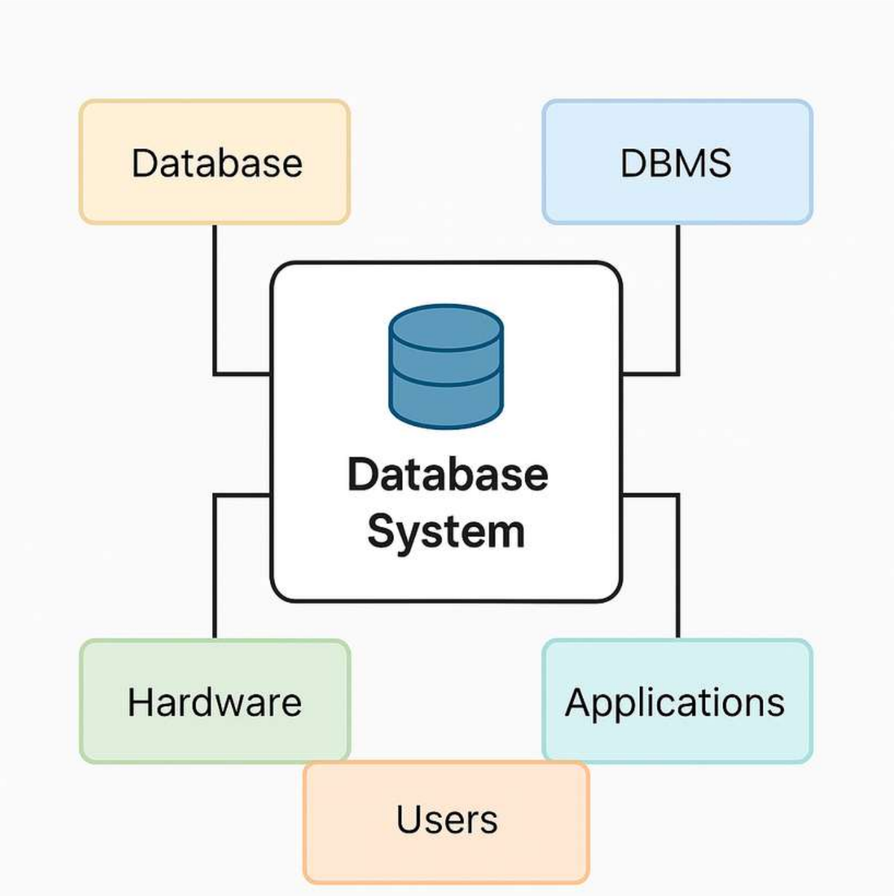


Contrasting Database and File Systems

Feature	File System	Database System
Data Storage	Separate, isolated files	Centralized, related data
Redundancy	High, same data in many files	Low, minimized by design
Data Sharing	Difficult across departments	Easy and centralized
Security	Handled by each app	Enforced by DBMS
Data Integrity	Hard to enforce	Maintained using rules
Scalability	Poor	High
Maintenance	Manual and time-consuming	Easier with tools/support

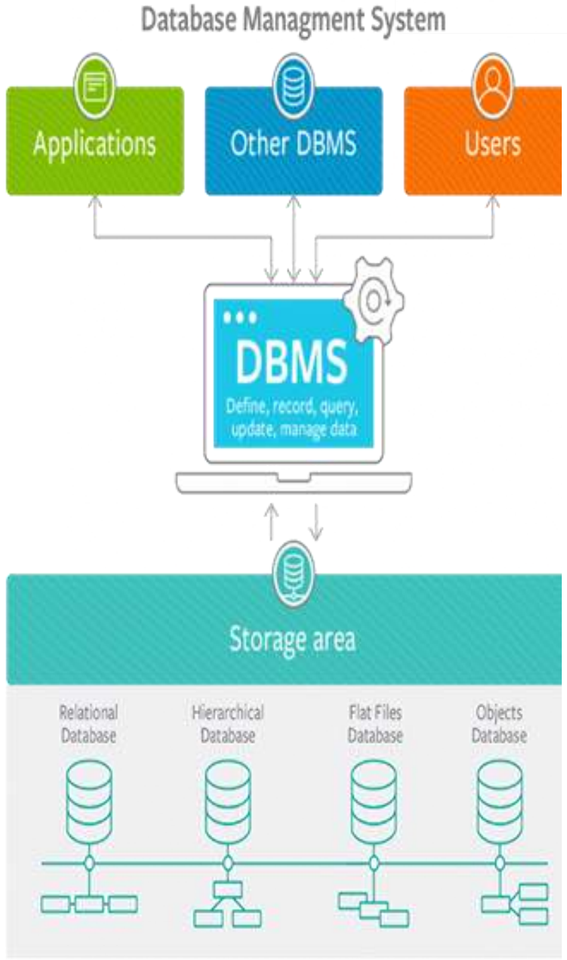
Basic Concept of Database System

A Database System is the complete environment used for managing data efficiently. It includes the **Database Management System (DBMS)** software, **database**, **the users**, and other supporting components such as **hardware**, **applications**, and tools



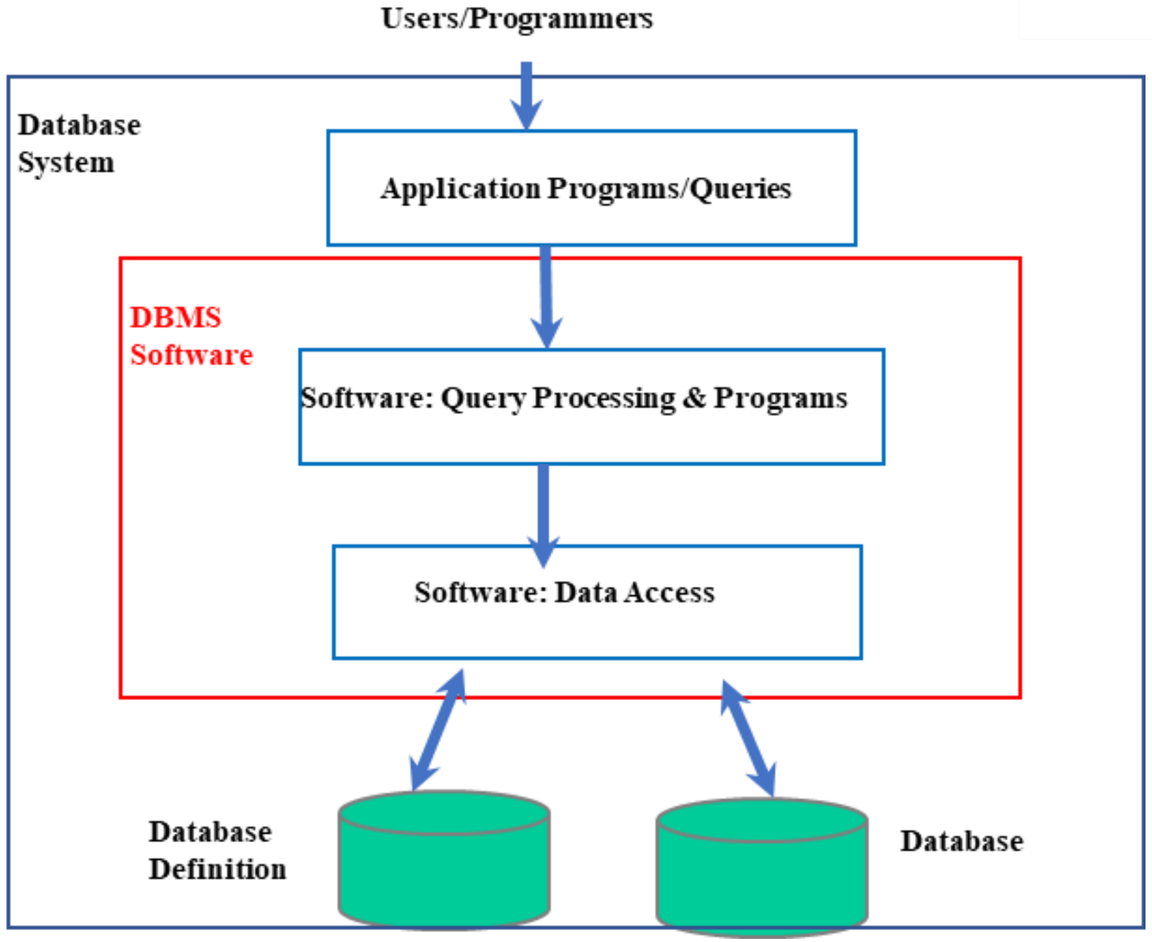
Basic Concept of Database System

- Different programs can be written, each using same Database.
- Database is collection of Logically related Data Files which is integrated and organized so as to provide single **comprehensive File System.**
- Purpose of Database is to provide convenient access to common **data** for wide variety of users and user needs.



Database Processing Systems

- Integrated Data
- Reduced Data Duplication
- Program / Data Independence
- Easier Representation of Users Perspectives
- A database processing system is a combination of machines, people, and processes involved in the database



Objectives of Database System

- Database must be shared by community of users.
- Integrity of Database must be preserved.
- Database should integrate operational files of corporate organization.
- Database should be capable of evolving, both in short term and in longer term.

Database management system (DBMS)

A **group** of **programs** or **software** which allows data organized in a database to be **added** to, **deleted**, **modified** or **retrieved**.

ORACLE

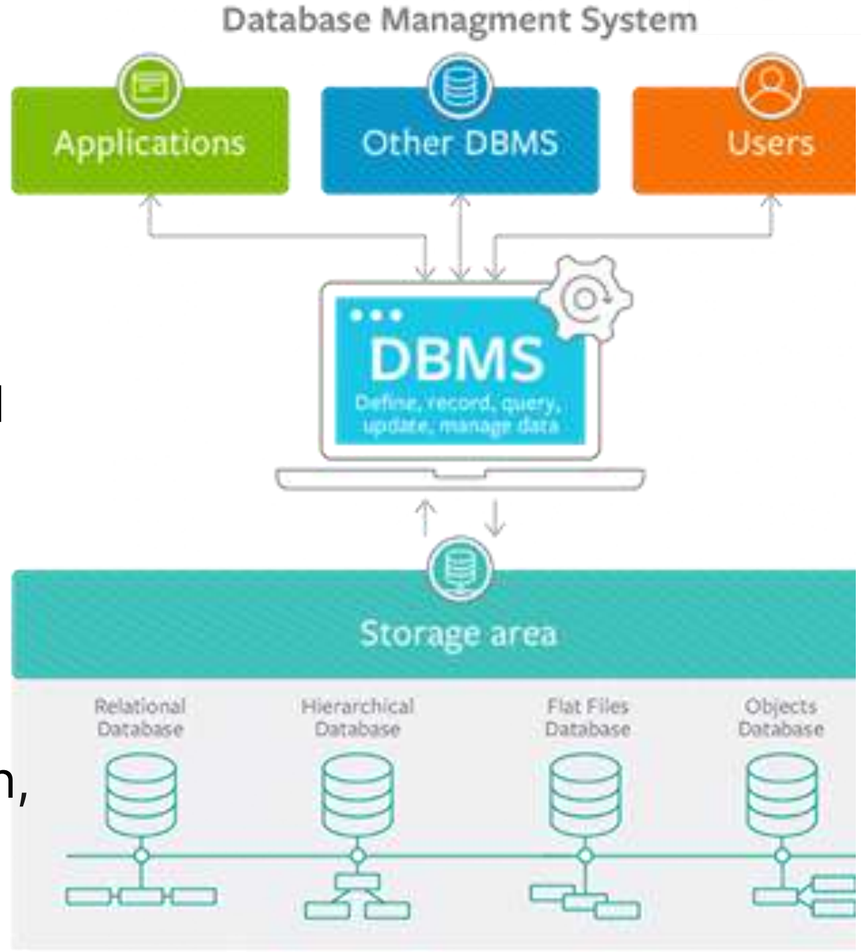
MySQL

PostgreSQL

Microsoft SQL Server

Database Management System (DBMS)

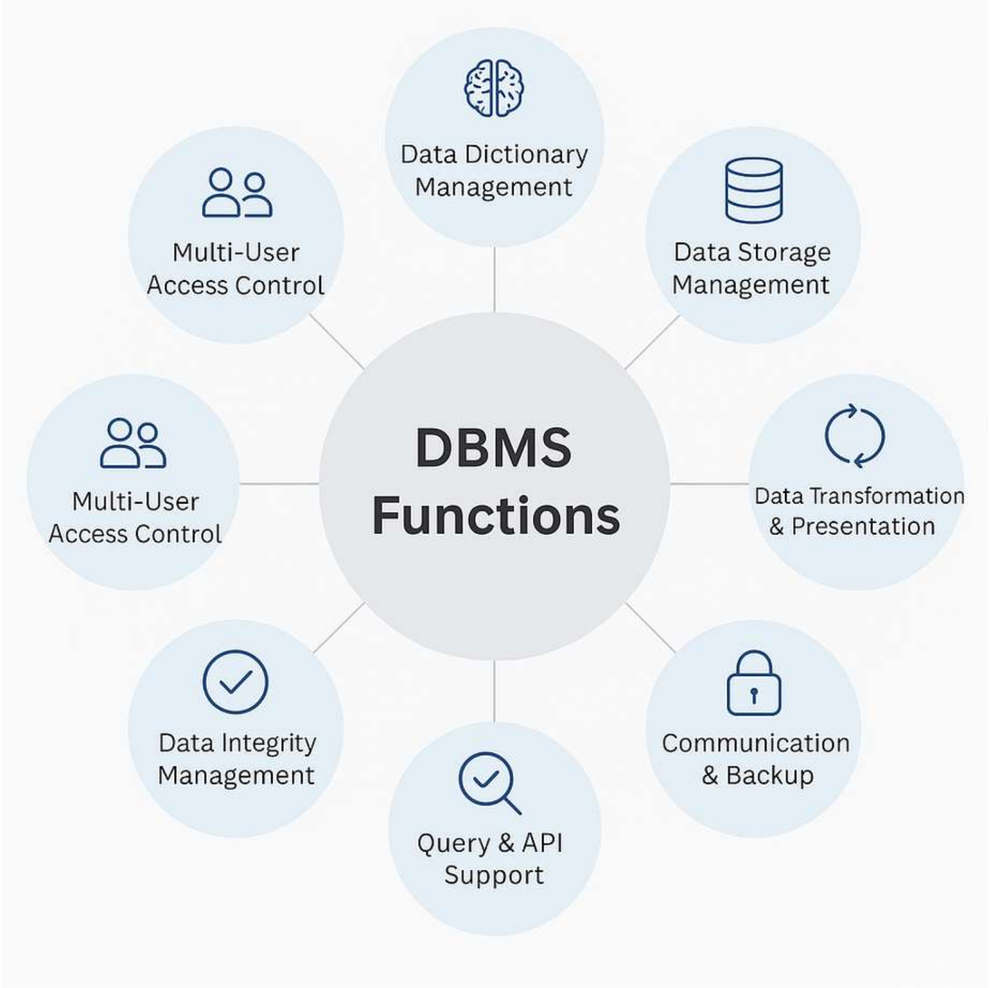
- A database management system (DBMS) is system software for **creating and managing databases.**
- It is suite of programs which allow data housed in Database to be a**dded to, deleted, modified or retrieved**
- DBMS facilitates **concurrent processing.**
- It can enforce **security and integrity.**
- It provides other utilities e.g. bulk loader, interface design, report generation.



KEY MESSAGE

DBMS acts as a Facilitator/Interface for us to interact with the Database

DBMS Functions



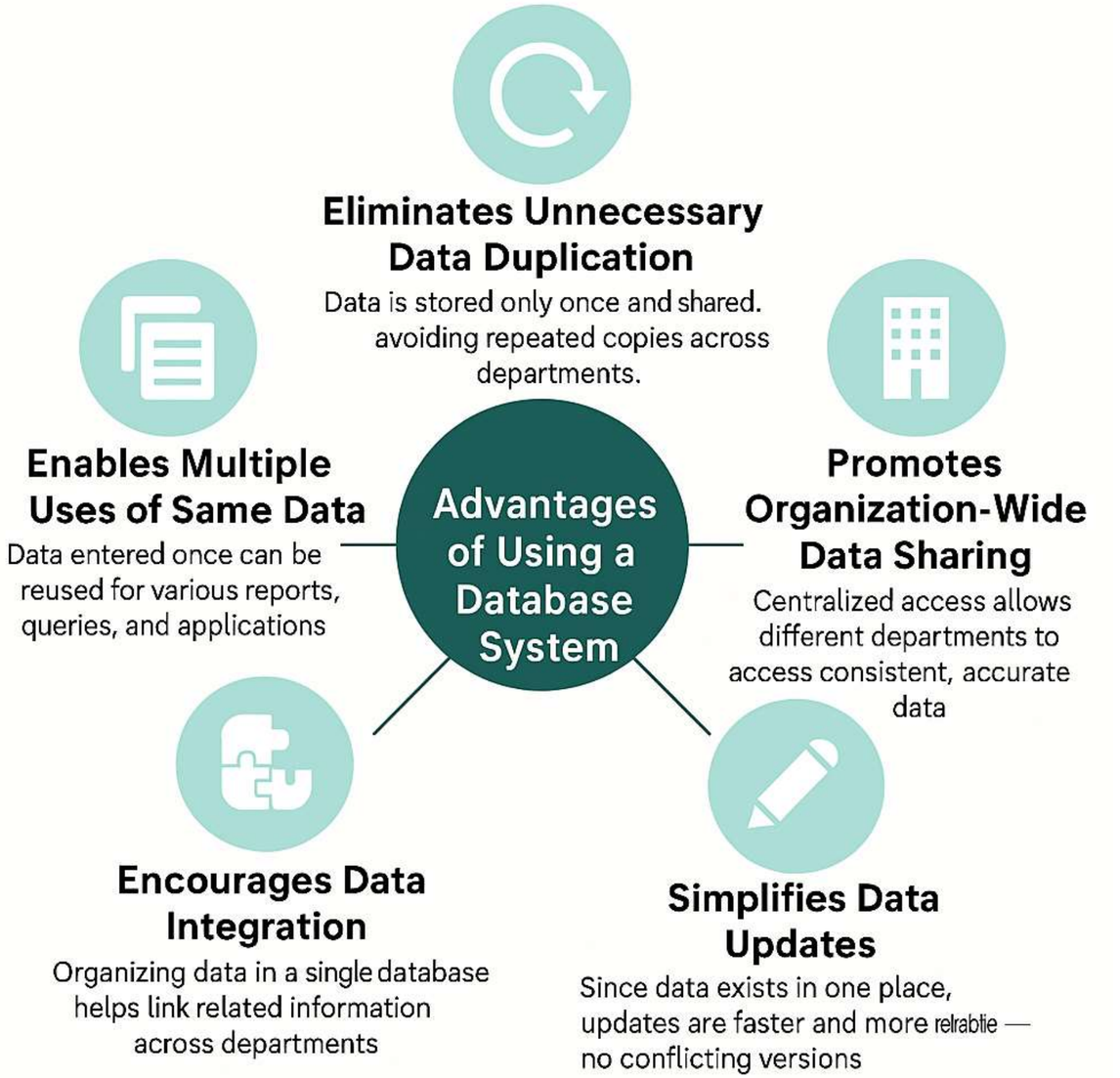
DBMS Functions

- Performs functions that guarantee the integrity and consistency of data
  - **Data Dictionary Management**: Maintains metadata — definitions of tables, fields, data types, and relationships.
  - **Data Storage Management**: Handles efficient storage of data, forms, reports, and access methods.
  - **Data Transformation and Presentation**: Converts user queries into low-level commands to retrieve and format data.
  - **Communication & Backup**: Supports networked access and ensures backup & recovery in case of failures.

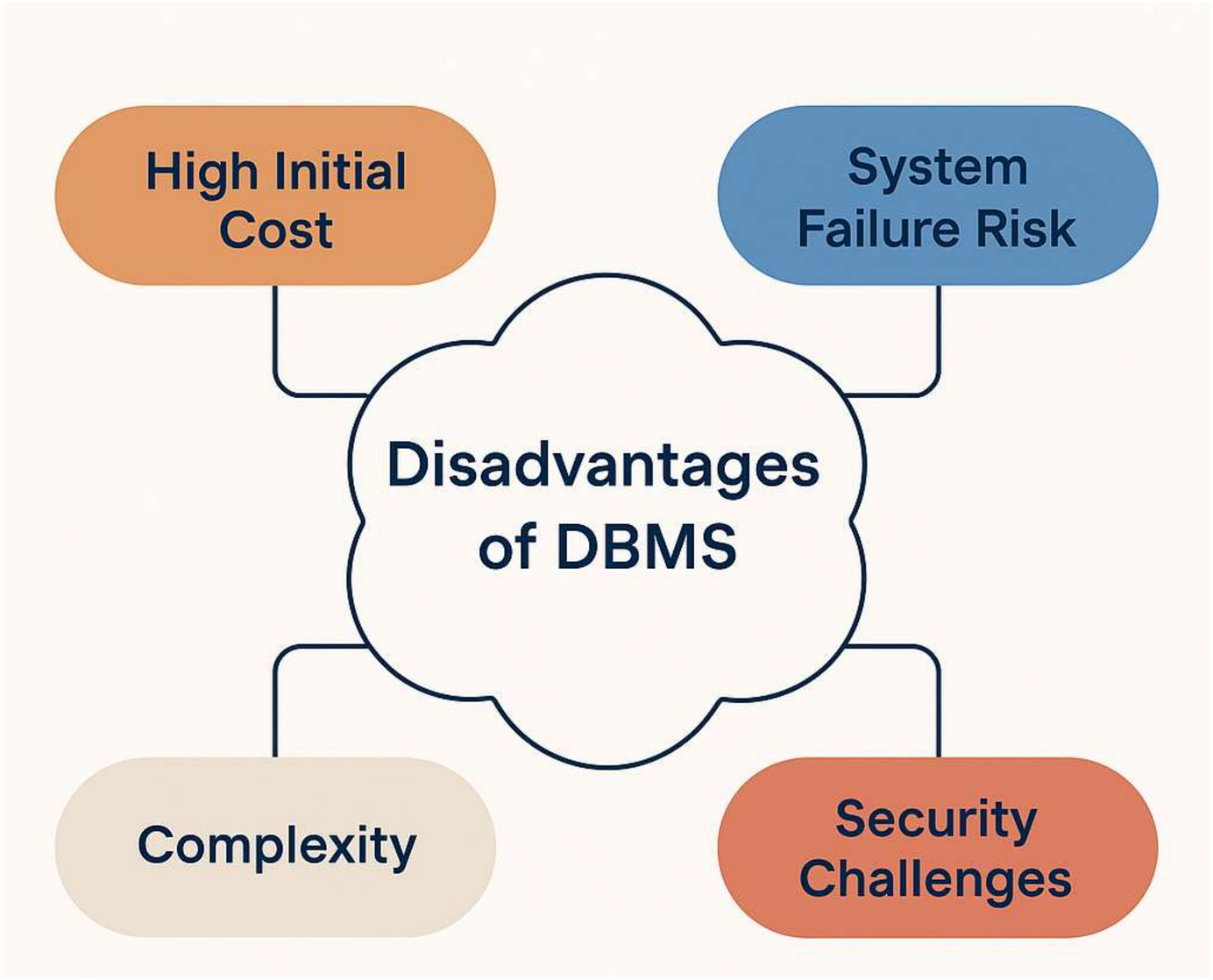
DBMS Functions

- **Security Management**: Controls access based on user roles, ensuring data privacy and protection.
- **Multi-User Access Control**: Allows multiple users to access data concurrently without conflict or corruption.
- **Data Integrity Management**: Enforces rules (e.g., uniqueness, constraints) to ensure accurate and reliable data.
- **Query & API Support**: Provides structured access to data via SQL and APIs for application developers.

Advantages of Database Management System



Disadvantages of Database Management System



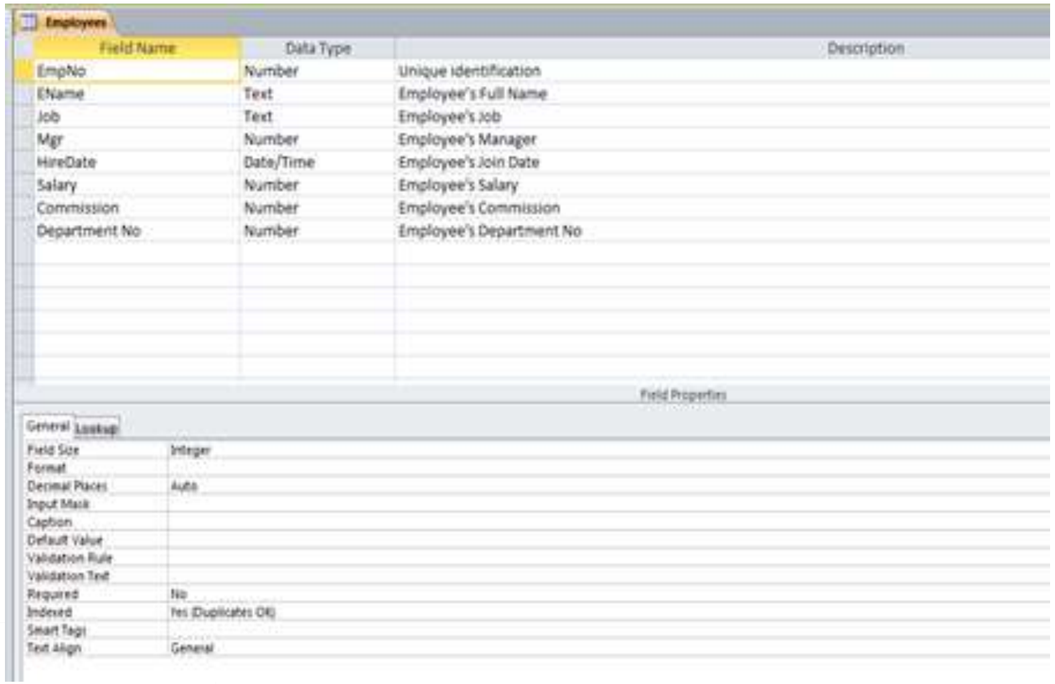
Metadata

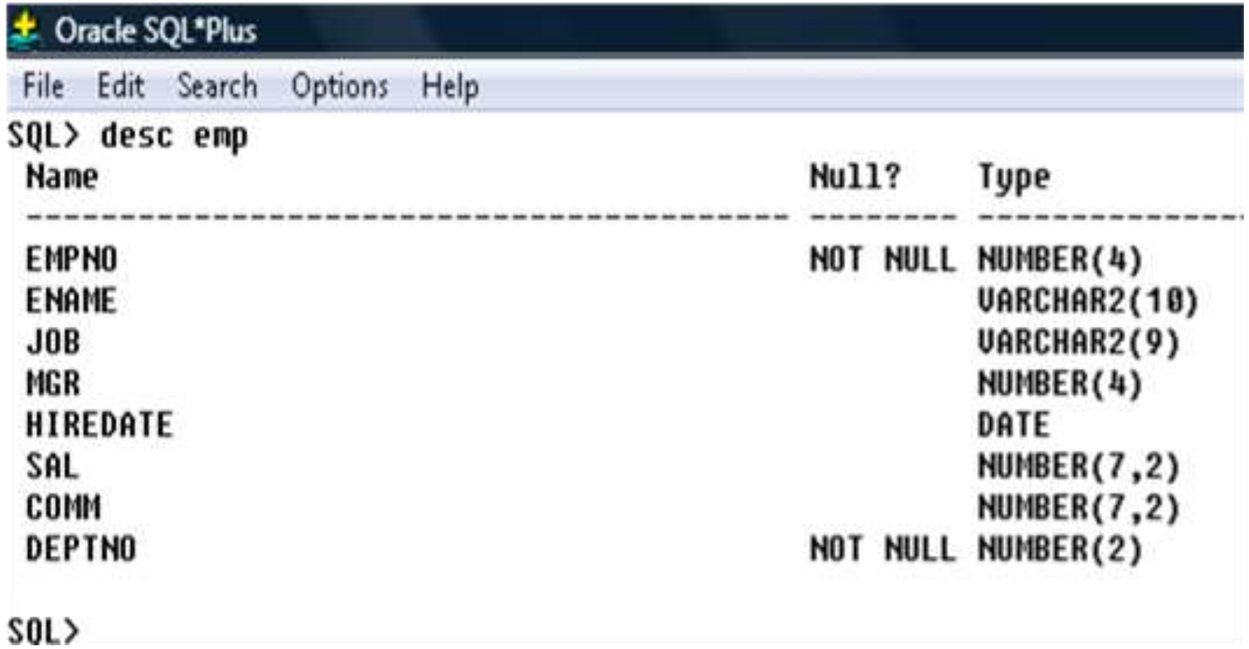
Database is self-describing, which means that it contains description of its structure as part of itself.

This description of structure is called metadata.

Name	Type	Length	Description
Student	Character	50	Student's full name
Student ID	Number	8	Unique identification number for each student
Date of Birth	Date	8	Student's date of birth in the format '01.01.80'

Metadata in Microsoft Access and Oracle





Data in Microsoft Access and Oracle

Employees

EName	EmpNo	Job	Mgr	HireDate	Salary	Commission	Department
Ram Kumar Shrestha	1	Team Leader		9/1/2010	50000	10000	10
Ramesh Joshi	2	Analyst	1	9/1/2010	30000	5000	10
Hari Dhakal	3	Programmer	1	9/1/2010	15000	3000	10
Ganesh Gurung	4	Manager		9/1/2010	40000	7000	20
Asha Thapa	5	Receptionist	4	9/1/2010	15000	2000	20

```
SQL> /

     EMPNO ENAME      JOB              MGR HIREDATE         SAL       COMM     DEPTNO
---------- ---------- --------- ---------- --------- ---------- ---------- ----------
      7369 SMITH      CLERK           7902 17-DEC-80        800                    20
      7499 ALLEN      SALESMAN        7698 20-FEB-81       1600        300         30
      7521 WARD       SALESMAN        7698 22-FEB-81       1250        500         30
      7566 JONES      MANAGER         7839 02-APR-81       2975                    20
      7654 MARTIN     SALESMAN        7698 28-SEP-81       1250       1400         30
      7698 BLAKE      MANAGER         7839 01-MAY-81       2850                    30
      7782 CLARK      MANAGER         7839 09-JUN-81       2450                    10
      7788 SCOTT      ANALYST         7566 19-APR-87       3000                    20
      7839 KING       PRESIDENT            17-NOV-81       5000                    10
      7844 TURNER     SALESMAN        7698 08-SEP-81       1500          0         30
      7876 ADAMS      CLERK           7788 23-MAY-87       1100                    20
      7900 JAMES      CLERK           7698 03-DEC-81        950                    30
      7902 FORD       ANALYST         7566 03-DEC-81       3000                    20
      7934 MILLER     CLERK           7782 23-JAN-82       1300                    10

14 rows selected.

SQL>
```

Field, Record, File

- **Field**: Character or group of characters that has specific meaning.
  - Field is used to define and store data
- **Record**: Logically connected set of one or more fields describing something.
- **File**: Collection of related records is called file

Hierarchy	Example
Database	Student Database Basic info file Tuition fees file Result file
File	Student info files Name, Section, GPA Monir, A, 4:50 Kobir, B, 4:60 Rahat, C, 5:00
Record	Student Record Name, Section, GPA Monir, A, 5:00
Field	Student Name field Name Monir

About Oracle Database XE

Oracle Database Express Edition (XE) is a free, lightweight, and entry-level version of the Oracle Database software designed for small-scale development, testing, and educational purposes.

Key features of Oracle Database XE include:

- Free to Use
- Limited Resources
- User-Friendly Installer
- Data Security

ITAHARI INTERNATIONAL COLLEGE
islington college (इस्लिङ्टन कलेज)
informatics college • pokhara

Thank you

End of Lecture

CC5051NT - DATABASES

Relational Algebra

WEEK 2 - Lecture

Agendas

- Introduction to Relational Algebra
- Operators
  - RESTRICT
  - PROJECT
  - CARTESIAN PRODUCT
  - UNION
  - INTERSECTION
  - DIFFERENCE
  - JOIN
  - DIVIDE

Introduction to Relational Algebra

Relational Algebra is a formal language for querying and manipulating relational databases, comprising of operations such as **selection, projection, union, and joining**. It offers a mathematical framework for searching databases, ensuring efficient data retrieval and processing. Relational algebra provides the mathematical basis for query SQL.

Types of operators in relational algebra

- Basic operators
- Derived operators

Basic operators

Basic operators are fundamental operations that include **selection (σ), projection (π), union (U), set difference (-), Cartesian product (×).**

SELECTION

The Selection Operation is mostly used to filter rows from a given database depending on a specified condition. It basically allows us to receive only the rows that match the condition specified in the SQL query.

The select operator is denoted by σ.

σ<condition>(Relation)

σ is the select operator.

<condition> is a Boolean expression involving attributes of the relation.

Relation is the table (or relation) you are querying.

Restrict or Select Example

Assume we have a relation called **Employee**:

EmpID	Name	Dept	Salary
1	Alice	HR	40000
2	Bob	IT	60000
3	Charlie	IT	50000
4	Diana	HR	70000

selecting employees from IT departments.

$$\sigma_{\text{Dept = 'IT'}}\,\textbf{(Employee)}$$

Output

EmpID	Name	Dept	Salary
2	Bob	IT	60000
3	Charlie	IT	50000

Restrict or Select Example

Syntax: $\sigma_{p}(r)$

$\sigma_{age > 17}(Student)$ → This will fetch the tuples(rows) from table Student, for which age will be greater than 17.

ID	Name	Age
1	Akon	17
2	Bkon	19
3	Ckon	15
4	Dkon	13

Output

Id	Name	Age
2	Bkon	19

Guess What will be the output?

Example Answer

You can also use, and, or operators, to specify two conditions

$\sigma_{\text{age} > 12 \wedge \text{id} >= 3}$ (Student)

Id	Name	Age
3	Ckon	15
4	Dkon	13

ID	Name	Age
1	Akon	17
2	Bkon	19
3	Ckon	15
4	Dkon	13

Note:

- ^ this symbol denote AND
- V this symbol denote OR

Projection

The selection operation operates on rows, and the projection operation of relational algebra works on columns. It basically allows us to select selected columns from a relational table depending on a given condition while disregarding all of the other columns.

The project operator is denoted by π.

Syntax

π<attribute_list>(Relation)

- **π** is the project operator.
- **<attribute_list>** is the list of attributes (columns) to keep.
- **Relation** is the table (relation) you're querying.

Projection Example

Let's use the same **Employee** relation:

EmpID	Name	Dept	Salary
1	Alice	HR	40000
2	Bob	IT	60000
3	Charlie	IT	50000
4	Diana	HR	70000

Output

Name
Alice
Bob
Charlie
Diana

Getting only the names of employees

π Name (Employee)

Projection Example

Syntax: ΠA1, A2...(R)

ΠName, Age(Student) → This will will show us only the Name and Age columns for all the rows of data in Student table.

ID	Name	Age
1	Akon	17
2	Bkon	19
3	Ckon	15
4	Dkon	13

Output

Name	Age
Akon	17
Bkon	19
Ckon	15
Dkon	13

Union

The Union Operator is mostly used to merge the results of two searches into a single result. The sole requirement is that both searches return the same number of columns with identical data types. The union operation in relational algebra is equivalent to the union operation in set theory.

In standard relational algebra, **Union automatically eliminates duplicate tuples**

The union operator is denoted by U

Syntax
Relation1 ∪ **Relation2**
Both relations must be **union-compatible**, meaning:
They have the **same number of attributes.**
The **corresponding attributes** must have the same domain (data type).

Example

Relation : Manager

EmpID	Name
1	Alice
2	Bob

Relation: Developer

EmpID	Name
3	Charlie
2	Bob

List all unique employees who are either managers or developers

Manager ∪ **Developer**

Output

EmpID	Name
1	Alice
2	Bob
3	Charlie

Union Example

Syntax: A U B

Πtime(RegularClass) ∪ Πtime(ExtraClass) →

This will give all unique time slots where either a regular class or an extra class is conducted.

Output

Time
morning
evening
afternoon

RegularClass

class_id	subject	teacher	time
R1	Math	Mr. A	morning
R2	Science	Mrs. B	evening
R3	English	Mr. C	morning

ExtraClass

class_id	activity	instructor	time
E1	Yoga	Ms. D	morning
E2	Robotics	Mr. E	afternoon
E3	Chess	Ms. F	morning

Set difference

Set difference essentially returns the rows that exist in one table but not in another. The set difference operation in relational algebra is the same as the one used in set theory.

The set difference operator is denoted by (-)

Syntax

Relation1 - Relation2

Like union, both relations must be union-compatible (same number and type of attributes).

Example

Relation : Manager

EmpID	Name
1	Alice
2	Bob

Relation: Developer

EmpID	Name
3	Charlie
2	Bob

List employees who are managers but not developers

Manager – Developer

Output

EmpID	Name
1	Alice

Cartesian product

The Cartesian product joins every row of one table with every row of another, yielding all conceivable combinations. It is mostly used as a prelude to more sophisticated operations such as joins. Let's assume A and B; therefore, the cross product of A X B will produce all attributes of the first relation, followed by all attributes of the second relation.

The Cartesian product operator is denoted by (×)

Syntax
Relation1 × Relation2

The result is all possible pairwise combinations of tuples from both relations.
If relation A has m tuples and relation B has n tuples, then A × B will have m × n tuples.

Example

Relation : Employee

EmpID	Name
1	Alice
2	Bob

Relation : Department

DeptID	DeptName
10	HR
20	IT

For example: **Employee × Department**

Output

EmpID	Name	DeptID	DeptName
1	Alice	10	HR
1	Alice	20	IT
2	Bob	10	HR
2	Bob	20	IT

The Cartesian product is usually followed by a SELECT (σ) to form a JOIN condition.

Derived operators

Derived operators are composed of fundamental operators and comprise operations such as join, intersection, and division. These operators assist with more complicated inquiries by combining fundamental operations to satisfy unique data retrieval requirements.

Join operators

JOIN combines related tuples from two relations based on a common attribute (or condition).

Natural join

Auto-join on all common attributes with same name.

Employee ⋈ Department

DeptID	DeptName
10	HR
20	IT

EmID	Name	DeptID
1	Alice	10
2	Bob	20
3	Charlie	10
4	Diana	30
5	Evan	20

Output

EmID	Name	DeptID	DeptName
1	Alice	10	HR
2	Bob	20	IT
3	Charlie	10	HR
5	Evan	20	IT

Equi Join

A special case of theta join where condition is only equality (=).

Employee ⋈ Employee.DeptID = Department.DeptID Department

EmID	Name	DeptID
1	Alice	10
2	Bob	20
3	Charlie	10
4	Diana	30
5	Evan	20

DeptID	DeptName
10	HR
20	IT

Output

EmID	Name	DeptID	DeptName
1	Alice	10	HR
2	Bob	20	IT
3	Charlie	10	HR
5	Evan	20	IT

Left outer join

Includes all rows from the left relation + matched rows from the right.

If there is no match, the right-side columns are filled with NULL.

Employee ⟕ Department

DeptID	DeptName
10	HR
20	IT

EmID	Name	DeptID
1	Alice	10
2	Bob	20
3	Charlie	10
4	Diana	30
5	Evan	20

Output

EmID	Name	DeptID	DeptName
1	Alice	10	HR
2	Bob	20	IT
3	Charlie	10	HR
4	Diana	30	NULL
5	Evan	20	IT

Right outer join

Includes all rows from the right relation + matched rows from the left.

If there is no match in the left table, the left-side columns are NULL.

Employee ⋈ Department

DeptID	DeptName
10	HR
20	IT
30	Finance

EmpID	Name	DeptID
1	Alice	10
2	Bob	20
3	Charlie	10

Output

EmpID	Name	DeptID	DeptName
1	Alice	10	HR
3	Charlie	10	HR
2	Bob	20	IT
NULL	NULL	30	Finance

Full outer join

Includes all rows from both relations, with NULLs where no match exists.

Employee ⟗ Department

- **DeptID 40 (Diana) has no department → DeptName = NULL**
- **DeptID 30 (Finance) has no employee → Employee columns = NULL**

EmpID	Name	DeptID
1	Alice	10
2	Bob	20
3	Charlie	10
4	Diana	40

DeptID	DeptName
10	HR
20	IT
30	Finance

Output

EmpID	Name	DeptID	DeptName
1	Alice	10	HR
3	Charlie	10	HR
2	Bob	20	IT
4	Diana	40	NULL
NULL	NULL	30	Finance

Set intersection

Set Intersection simply allows you to obtain only the rows of data that are shared by two sets of relational tables. Set Intersection in relational algebra is equivalent to set intersection in set theory.

Syntax

Relation1 ∩ Relation2

Both relations must be **union-compatible:**

Same number of attributes

Matching attribute domains

Example

Relation: Manager

EmpID	Name
1	Alice
2	Bob

Relation: Developer

EmpID	Name
3	Charlie
2	Bob

Find employees who are both Managers and Developers

- Only employees who are both managers and developers are included.

Manager ∩ Developer

Output

EmpID	Name
2	Bob

Bob is both a manager and a developer.

Division

The Division operator is used when we want to find tuples in one relation that are associated with all tuples in another relation.

It is especially useful for queries that involve phrases like “**for all**” or “**every**.”

Purpose: Returns all values of attributes in R that are related to every tuple in S.

Syntax

R ÷ S

Where:

R has attributes: (A, B)

S has attributes: (B)

Example

Relation: Enrolled

Student	Course
Alice	DBMS
Alice	OS
Bob	DBMS

Relation: AvailableCourses

Course
DBMS
OS
DBMS

Find students who are enrolled in ALL Available Courses

Enrolled ÷ AvailableCourses

Output

Student
Alice

- **Division (÷) selects students who have every course in RequiredCourses.**
- **Bob is missing OS, so he is excluded.**

ITAHARI INTERNATIONAL COLLEGE

islington college (इस्लिङ्टन कलेज)

informatics college • pokhara

Thank you

End of Lecture

CC5051 - DATABASES

INTRODUCTION TO SQL

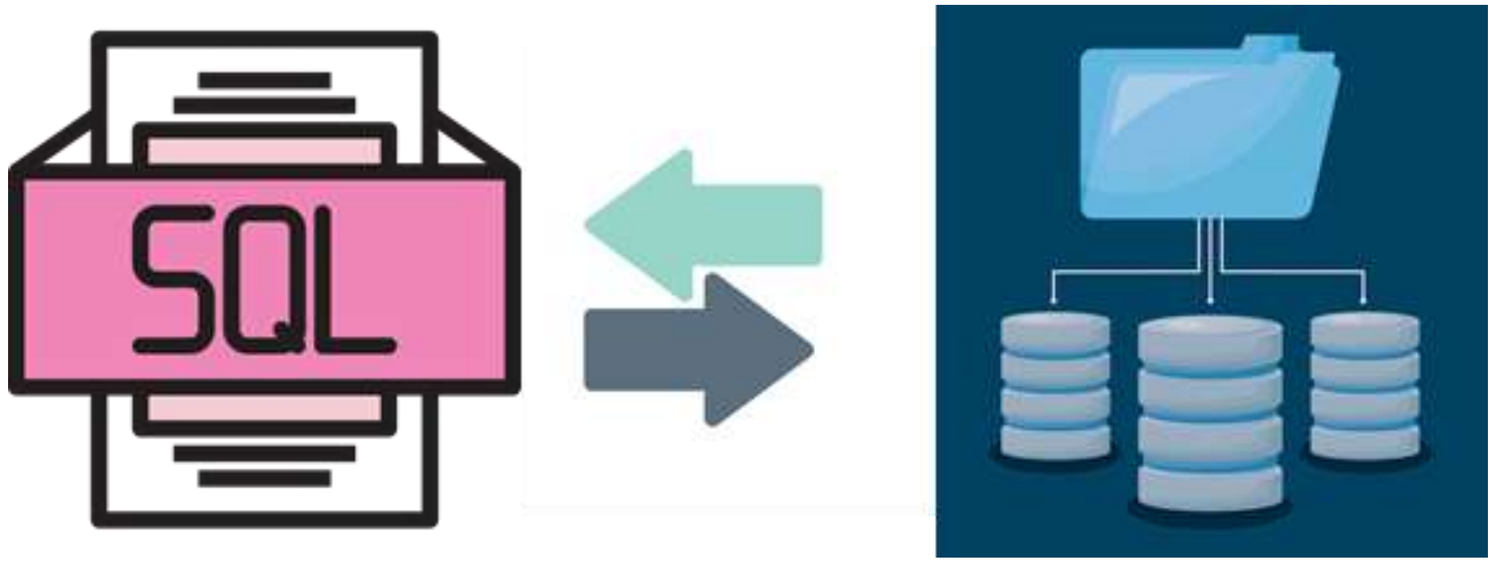
WEEK 3- Lecture

Topics

- Database Languages
- What is SQL?
- Why use SQL?
- Why is SQL important?
- SQL statement construction

What is SQL?

- **Structured Query Language (SQL)** is a standard database language which is used to store, manipulate and retrieve data in database.



What is SQL Used For?

- SQL is used to **create new database and database tables**
- SQL can be used to **retrieve data from a database**
- SQL can be used to **insert, update and delete data in a database**

What is SQL Used For?

- SQL can be used to set access permissions for database tables
- SQL can be used to create stored procedures for database
- SQL is used to create views in database

From Relational Algebra to SQL

In 1970, Edgar F. Codd introduced the Relational Model at IBM.

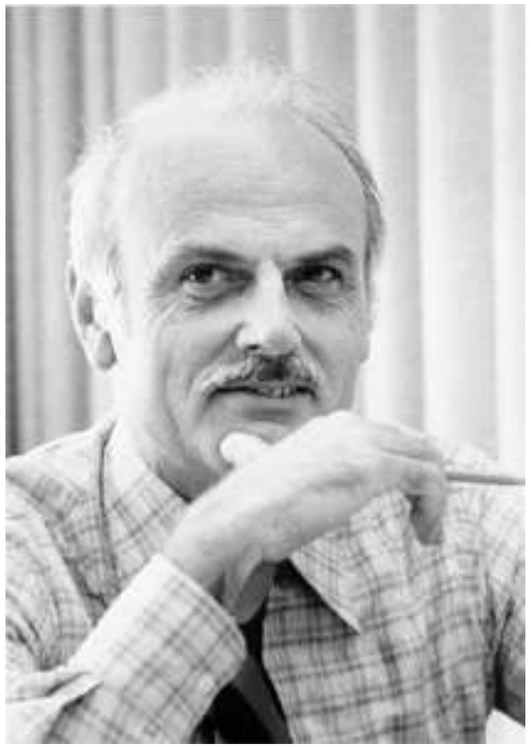
Proposed Relational Algebra & Relational Calculus as formal query languages.

Relational Algebra:

A mathematical system for manipulating relations (tables).

Operators: Selection (σ), Projection (π), Join (⋈), Union (∪), Difference (-), Division (÷).

Provided the **theoretical foundation** for querying databases.



From Relational Algebra to SQL

In Mid-1970s: IBM developed **SEQUEL** (Structured English Query Language) which **later renamed SQL.**

SQL was designed to be declarative: users specify what data they want, not how to get it.

SQL concepts directly map to relational algebra operations:

σ → WHERE

π → SELECT column

⋈ → JOIN

∪ → UNION

Example:

Relational Algebra: σclass="Maths"(STUDENT)

SQL: SELECT * FROM STUDENT WHERE class='Maths';

Evolution of SQL

1980s Standardization

- SQL officially became the ANSI (1986) and ISO (1987) standard.
- Ensured compatibility across different database systems.

Extensions Beyond Relational Algebra

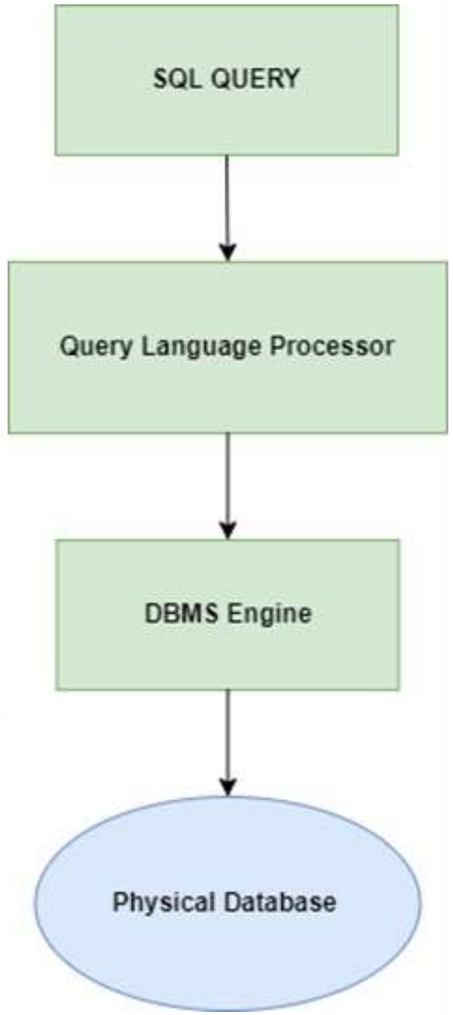
- **Aggregates (SUM, COUNT, AVG):** Enabled summarizing and analyzing data.
- **NULL values:** Introduced the concept of missing or unknown data.
- **Constraints:** Enforced rules (PRIMARY KEY, FOREIGN KEY, UNIQUE, CHECK).
- **Views & Triggers**: Allowed virtual tables and automatic actions on data changes.
- **Transactions (ACID):** Guaranteed reliability with Atomicity, Consistency, Isolation, Durability.

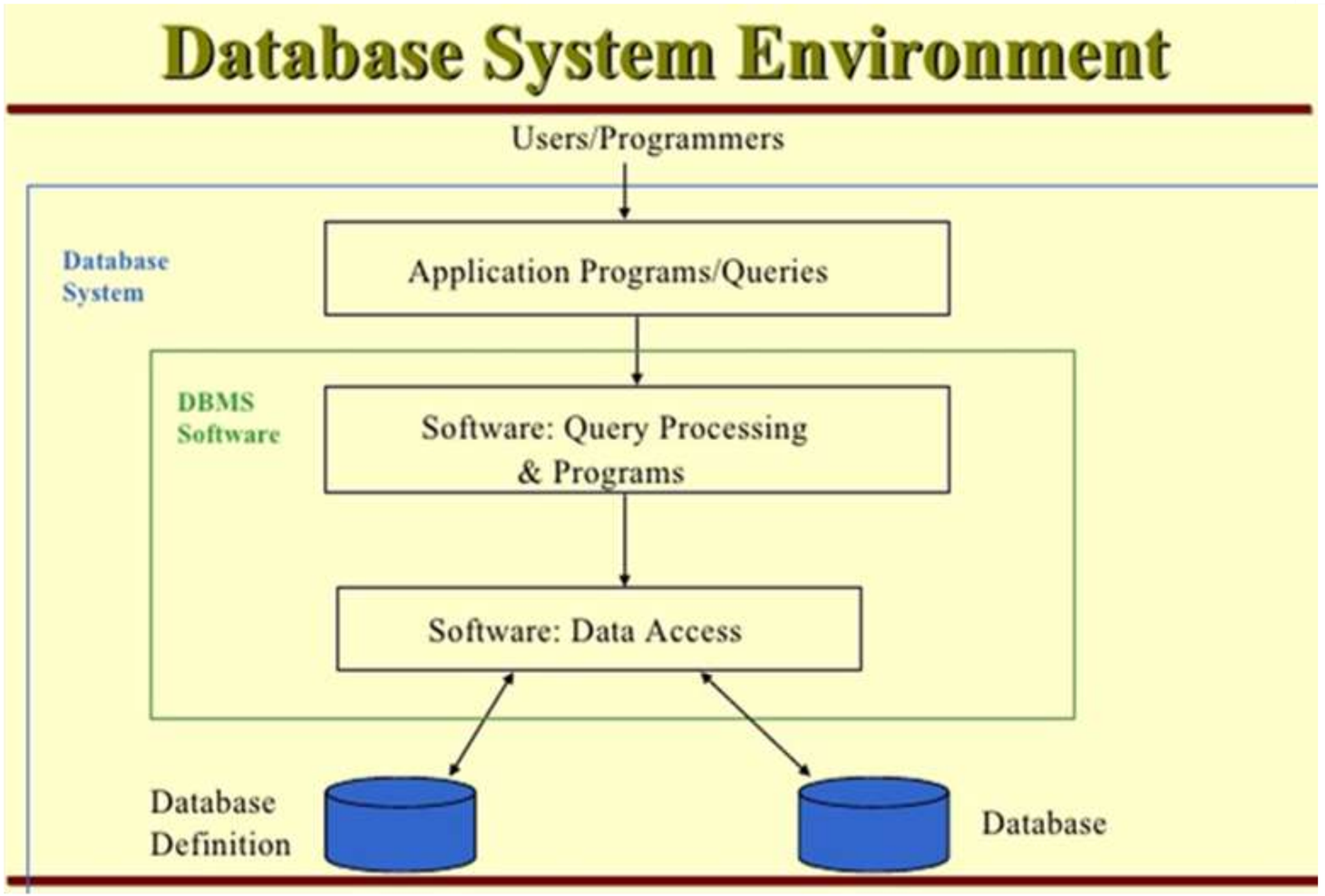
Modern SQL

- Combines the mathematical rigor of relational algebra with practical database features.
- Forms the core query language for all major relational DBMS:

  Oracle, MySQL, PostgreSQL, SQL Server etc.

SQL Process


SQL QUERY
Query Language Processor
DBMS Engine
Physical Database


Database System Environment
Users/Programmers
Database System
Application Programs/Queries
DBMS Software
Software: Query Processing & Programs
Software: Data Access
Database Definition
Database

Database Languages

Every DBMS includes a set of database languages that are used to interact with the database system. These languages allow users to **define**, **manipulate**, **control**, and **retrieve data** as needed.

- The common components include:
- Data Definition Language (DDL)
- Data Manipulation Language (DML)
- Data Control Language (DCL)
- Transaction Control Language(TCL)
- Query Language
- Host Language

Host Language

- A host language is a general-purpose programming language that is used in combination with SQL to build database-driven applications.
- Allows SQL statements to be embedded into applications with programming languages like Java, C, etc.
- Host languages - C, Java, COBOL, Visual Basic, etc.

Host Language

We use SQL embedded within a host language to:

- Accept input from users
- Display results
- Perform logic and processing

```java
import java.sql.*;

public class Main {
    public static void main(String[] args) throws Exception {
        Connection con = DriverManager.getConnection(
            "jdbc:mysql://localhost:3306/school", "root", "password");

        Statement stmt = con.createStatement();
        ResultSet rs = stmt.executeQuery("SELECT name FROM students");

        while (rs.next()) {
            System.out.println(rs.getString("name"));
        }

        con.close();
    }
}
```

Query Language

A query language is used to retrieve and display data from a database.

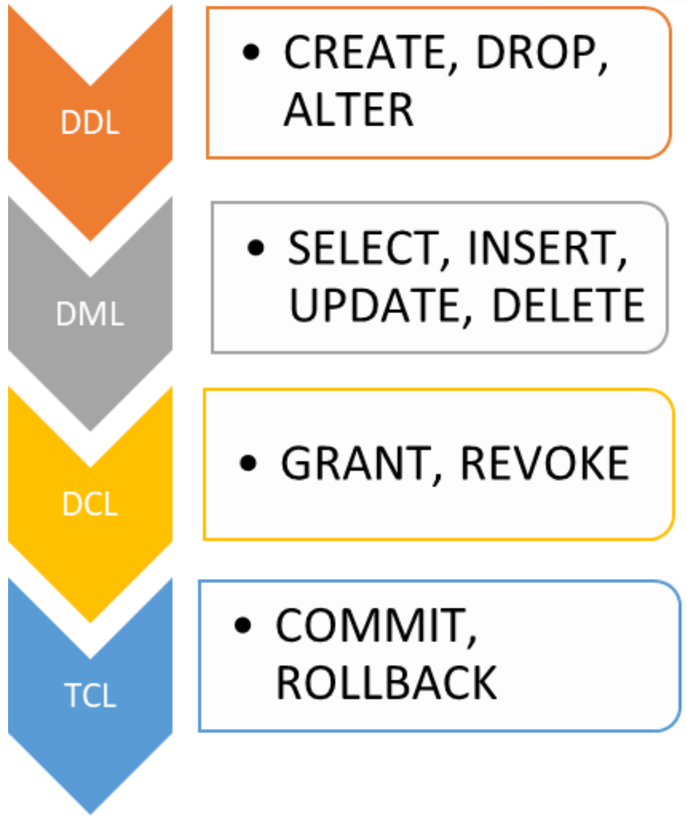
- **The most common query language is SQL (Structured Query Language).**
- Used to fetch specific information from large datasets efficiently.

Example: SELECT name, age FROM students WHERE grade = 'A';

SQL category

Classified by type of operation

- **Data Definition Language (DDL):** Facilitates Creation and Description of Database.
- **Data Manipulation Language (DML):** Deals with Manipulation and Processing of Data.
- **Data Control Language (DCL):** Deals with Data Access.
- **Transaction Control Language (TCL):** Manages transactions in the database and the changes made to the data in a table by DML statements.



Why use SQL

- Allow users to:
  - **Create Database** and Relational Structures
  - **Insert, Modify and Delete** Database Data
  - Perform simple and complex queries.
- Performs all these tasks with minimal user effort, and command structure / syntax (relatively) simple to learn.
- Portable and Standardized and have DDL and DML component parts to it.
- Declarative: State question to answer not how to answer it

SQL – Statement Construction

- SQL statements consists of reserved words and user-defined words.
  - Reserved words are a fixed part of SQL and must be spelt exactly as required and cannot be split across lines.
  - User-defined words are made up by user and represent names of various database objects such as relations, columns, views.
- Most components of an SQL statement are case insensitive, except for literal character data.

Rules Followed by SQL

- SQL queries consists of keywords that perform certain functions

```
SELECT * FROM students;
```

- SQL keywords are not case sensitive but are represented in upper case as a convention.
- A single SQL query can be written in one or more than one line and ends through the use of semicolon (;).

```
SELECT first_name, last_name
FROM employees
WHERE department_id = 10;
```

SQL – Statement Construction

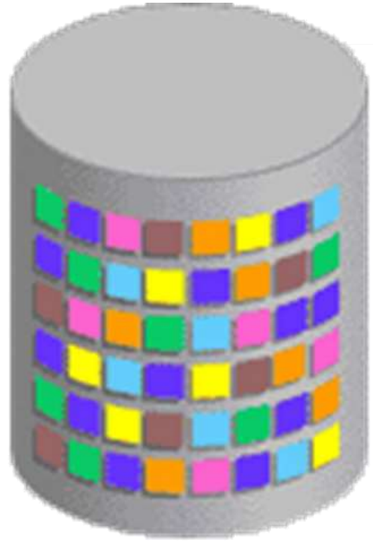
- More readable with indentation and lineation:
  - Each clause should begin on new line.
  - Start of clause should line up with start of other clauses.
  - Where clause has several parts, each should appear on separate line and be indented under start of clause.

DDL Statements

- Data definition language (DDL) statements
- Syntax for creating and modifying database objects such as tables, users etc.
- Let you to perform these tasks: **Create, alter, drop, rename and truncate**

Create Table Statement

- You must have:
  - **CREATE TABLE** privilege
  - A storage area
  - Have immediate effect on database
  - Record information in the data dictionary.
- You specify:
  - Table name, Column name, column data type, and column size



```
CREATE TABLE [schema.]table
            (column datatype [DEFAULT expr][, ...]);
```

Naming Rules

- Table names and column names:
  - Must begin with a letter
  - Must be 1–30 characters long
  - Must contain only A–Z, a–z, 0–9, _, $, and # (legal characters, but their use is discouraged).
  - Must not duplicate the name of another object owned by the same user
  - Must not be an Oracle server reserved word

Creating Tables

- **Create the table.**

```
SQL> CREATE TABLE test1 (deptno  NUMBER(2),
  2            dname  VARCHAR2(14),
  3            email  VARCHAR2(14));

Table created.
```

- **Confirm Table Creation**

```
SQL> describe test1
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 DEPTNO                                             NUMBER(2)
 DNAME                                              VARCHAR2(14)
 EMAIL                                              VARCHAR2(14)
```

Data Types

Data Type	Description	Data size
VARCHAR2(*size*)	Variable-length character data	4000 bytes
CHAR(*size*)	Fixed-length character data	2000 bytes
NUMBER(*p*,*s*)	Variable-length numeric data	Precision ranges from 1 to 38 while the scale range from -84 to 127
DATE	Date and time values	Valid date range from January 1, 4712 BC to December 31, 9999 AD.
LONG	Variable-length character data (up to 2 GB)	2 gigabytes, or 2^{31} -1 bytes.
CLOB	Character data (up to 4 GB)	Maximum size: (4 GB - 1) * DB_BLOCK_SIZE initialization parameter (8 TB to 128 TB)
RAW and LONG RAW	Raw binary data	Maximum size: 2000 bytes
BLOB	Binary data (up to 4 GB)	Maximum size: (4 GB - 1) * DB_BLOCK_SIZE initialization parameter (8 TB to 128 TB)
BFILE	Binary data stored in an external file (up to 4 GB)	Maximum size: 4 GB
ROWID	A base-64 number system representing the unique address of a row in its table	Base 64 string representing the unique address of a row in its table

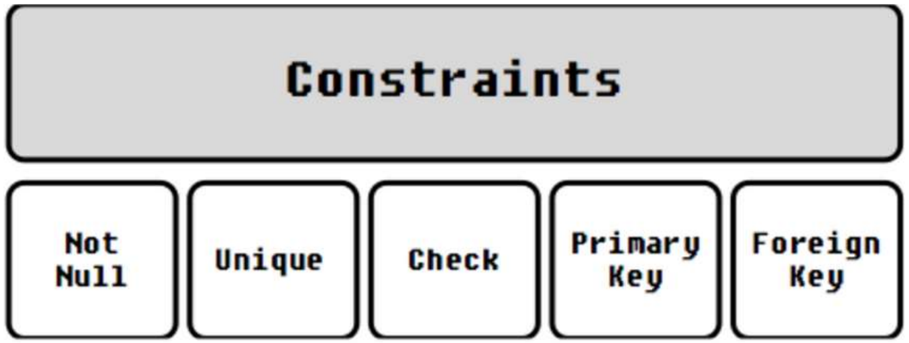
Data Types

By default, Oracle uses BYTE or Characters in if you don’t explicitly specify BYTE or CHARACTER,Default value is used.

Data Type	LENGTH
VARCHAR2(*size*)	4000
CHAR(*size*)	2000
NUMBER(*p*,*s*)	38-digit precision
DATE	Range from January 1, 4712 BC to December 31, 9999 AD.

Constraints

- Rules to preserve the data integrity in the application.
- Imposed on a column of a database table, to check data.
- The key milestones achieved by the usage of constraints are:
- Validate NULL property of the data
- Validate uniqueness of the data
- Validate referential integrity of the data



Defining Constraints

- **Syntax:**

```
CREATE TABLE [schema.]table
      (column datatype [DEFAULT expr]
      [column_constraint],
      ...
      [table_constraint][,...]);
```

- **Column-level constraint:**

```
column [CONSTRAINT constraint_name]
constraint_type,
```

- **Table-level constraint:**

```
column,...
  [CONSTRAINT constraint_name] constraint_type
  (column, ...),
```

Defining Constraints

- **Column-level constraint:**

```
CREATE TABLE employees(
  employee_id  NUMBER(6)
    CONSTRAINT emp_emp_id_pk PRIMARY KEY,
  first_name   VARCHAR2(20),
  ...);
```

- **Table-level constraint**

```
CREATE TABLE employees(
  employee_id  NUMBER(6),
  first_name   VARCHAR2(20),
  ...
  job_id       VARCHAR2(10) NOT NULL,
  CONSTRAINT emp_emp_id_pk
    PRIMARY KEY (EMPLOYEE_ID));
```

Not Null Constraint

- Ensures that the column contains no null values.
- Columns without the **NOT NULL** constraint can contain null values by default.
- **NOT NULL** constraints must be defined at the column level.
- Syntax

  [COLUMN NAME] [DATA TYPE] [CONSTRAINT NAME)]

Not Null

- **Column-level constraint:**

```
CREATE TABLE table_name
( ... column_name data_type NOT NULL
…..);
```

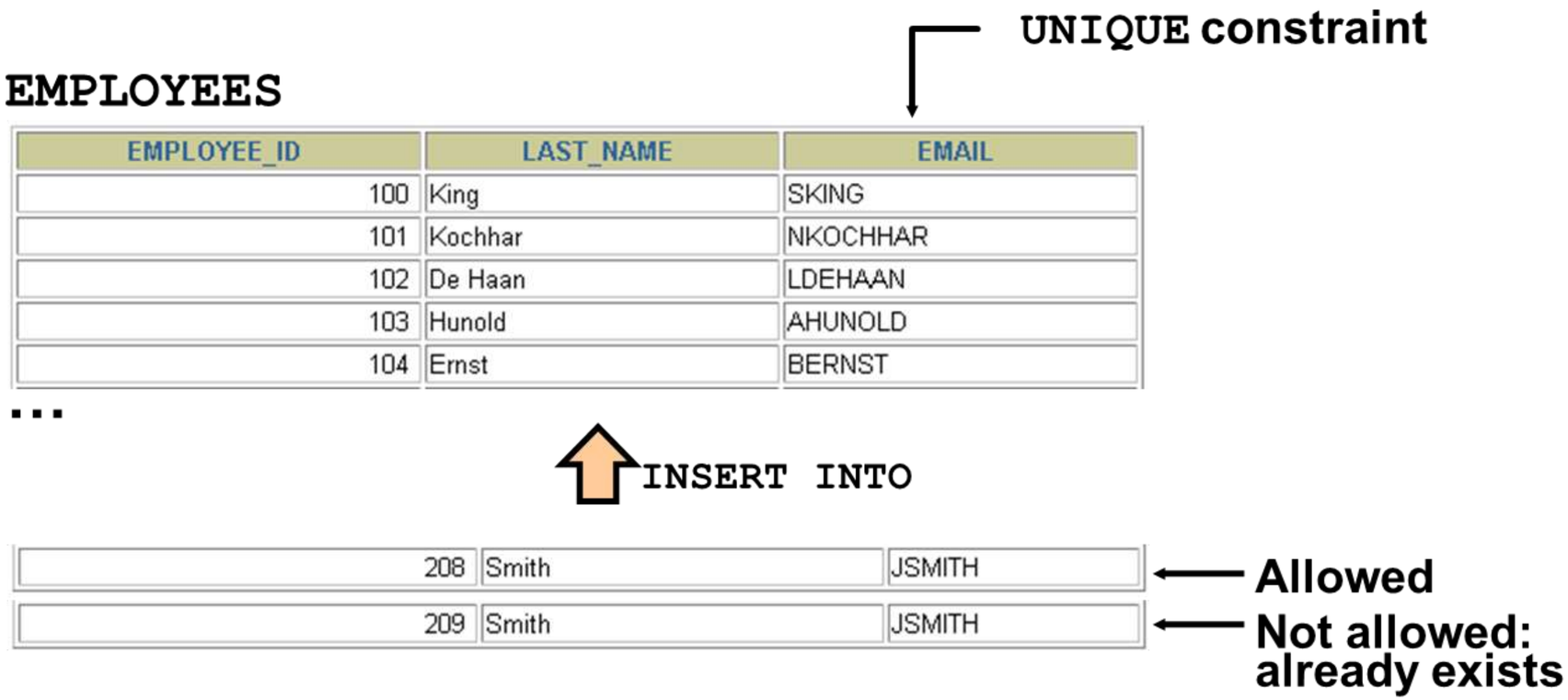
1

```
SQL> CREATE TABLE surcharges (
  2  surcharge_id NUMBER,
  3  surcharge_name VARCHAR2(255) NOT NULL,
  4  amount NUMBER(9,2));

Table created.
```

Unique Constraint

- Ensures the data stored in a column, or a group of columns, is unique among the rows in a table.



Unique

- **Column-level constraint:**

```
CREATE TABLE table_name
 ... column_name data_type UNIQUE ... );
```

```
SQL> CREATE TABLE charges (
  2  charge_name VARCHAR2(255) UNIQUE,
  3  amount NUMBER(9,2));

Table created.
```

```
SQL> DESCRIBE charges
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 CHARGE_NAME                                        VARCHAR2(255)
 AMOUNT                                             NUMBER(9,2)
```

Unique Constraint

Defined at either the table level or the column level:

```
CREATE TABLE table_name ( ... column_name data_type
UNIQUE, ... );
```

```
CREATE TABLE table_name ( ... column_name data_type,
..., CONSTRAINT unique_constraint_name
UNIQUE(column_name) );
```

Column-level constraint:

```
CREATE TABLE table_name ( ... column_name1 data_type,
column_name2 data_type, ..., CONSTRAINT unique_constraint_name
UNIQUE(column_name1, column_name2) );
```

Unique Constraint

Defined at either the table level or the column level:

```
CREATE TABLE clients (
client_id NUMBER, first_name VARCHAR2(50) NOT NULL,
last_name VARCHAR2(50) NOT NULL, company_name
VARCHAR2(255) NOT NULL,
email VARCHAR2(255) NOT NULL, phone VARCHAR(25),
CONSTRAINT UNQ_C_EMAIL UNIQUE (email)
);
```

Unique Constraint

```
SQL> CREATE TABLE clients (
  2  client_id NUMBER, first_name VARCHAR2(50) NOT NULL, last_name VARCHAR2(50) NOT NULL, company_name VARCHAR2(255) N
OT NULL,
  3  email VARCHAR2(255) NOT NULL, phone VARCHAR(25),
  4  CONSTRAINT UNQ_C_EMAIL UNIQUE (email)
  5  );
```

Table level constraint

```
SQL> DESCRIBE Clients
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 CLIENT_ID                                          NUMBER
 FIRST_NAME                                NOT NULL VARCHAR2(50)
 LAST_NAME                                 NOT NULL VARCHAR2(50)
 COMPANY_NAME                              NOT NULL VARCHAR2(255)
 EMAIL                                     NOT NULL VARCHAR2(255)
 PHONE                                              VARCHAR2(25)
```

Check

- Allows you to enforce domain integrity by limiting the values accepted by one or more columns.
- To create a check constraint, you define a logical expression that returns true or false.
- You can apply a check constraint to a column or a group of columns. A column may have one or more check constraints.
- **Syntax:**

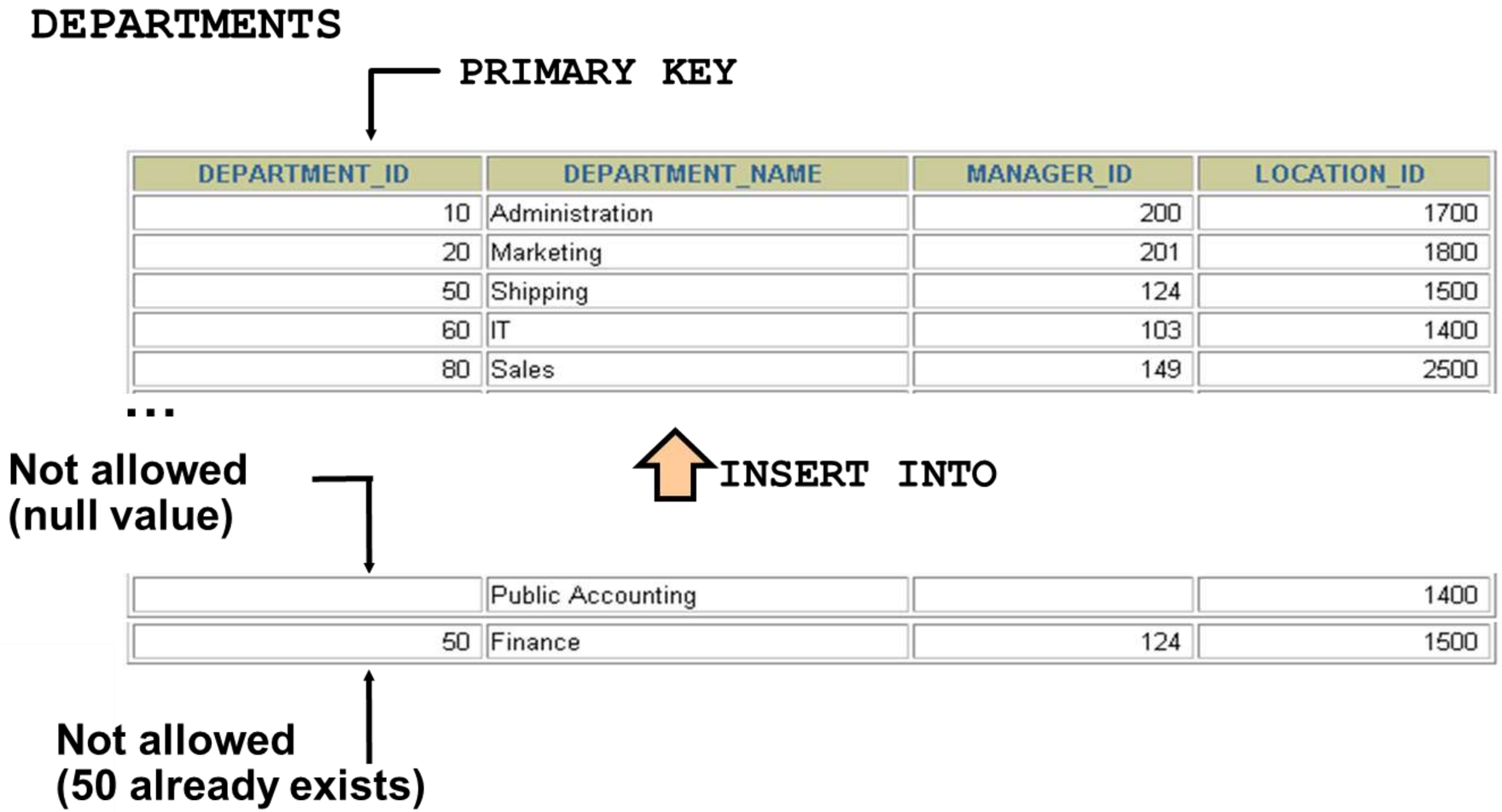
```
CREATE TABLE table_name
( ... column_name data_type CHECK
(expression),
);
```

```
CREATE TABLE parts ( part_id NUMBER,
buy_price NUMBER(9,2) CHECK(buy_price > 0),);
```

Primary Key Constraint

- A **primary key is a column or combination of columns in a table that uniquely identifies a row in the table**.
- The following are rules that make a column a primary key:
  - A primary key column cannot contain a NULL value or an empty string.
  - A primary key value must be unique within the entire table.
  - A primary key value should not be changed over time.

Primary Key Constraint



Primary Key Constraint

```
CREATE TABLE table_name
        ( column1 datatype null/not null,
        column2 datatype,
... CONSTRAINT constraint_name PRIMARY KEY
(column1, column2, ... column_n) );
```

At column level, CONSTRAINT keyword and CONSTRAINT_NAME are optional.

```
CREATE TABLE supplier (
 supplier_id numeric(10) not null,
 supplier_name varchar2(50) not null,
 contact_name varchar2(50), CONSTRAINT supplier_pk PRIMARY KEY
(supplier_id) );
```

Primary Key Constraint

```
CREATE TABLE supplier (
 supplier_id numeric(10) not null,
 supplier_name varchar2(50) not null,
 contact_name varchar2(50), CONSTRAINT supplier_pk PRIMARY KEY
(supplier_id) );
```

```
SQL> CREATE TABLE supplier (
  2   supplier_id numeric(10) not null,
  3   supplier_name varchar2(50) not null,
  4   contact_name varchar2(50), CONSTRAINT supplier_pk PRIMARY KEY (supplier_id) );

Table created.
```

```
SQL> DESCRIBE supplier
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 SUPPLIER_ID                               NOT NULL NUMBER(10)
 SUPPLIER_NAME                             NOT NULL VARCHAR2(50)
 CONTACT_NAME                                       VARCHAR2(50)
```

Primary Key Constraint

PRIMARY KEY Constraint

```
CREATE TABLE Persons (
    ID int NOT NULL PRIMARY KEY,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int
);
```

```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    CONSTRAINT PK_Person PRIMARY KEY (ID,LastName)
);
```

Primary Key Constraint

```
SQL> CREATE TABLE Persons (
  2     ID int NOT NULL,
  3     LastName VARCHAR(25) NOT NULL,
  4     FirstName VARCHAR(25),
  5     Age int,
  6     CONSTRAINT PK_Person PRIMARY KEY (ID)
  7  );

Table created.
```


Table level constraint

```
SQL> DESCRIBE Persons
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 ID                                        NOT NULL NUMBER(38)
 LASTNAME                                  NOT NULL VARCHAR2(25)
 FIRSTNAME                                          VARCHAR2(25)
 AGE                                                NUMBER(38)
```

Foreign Key Constraint

- A **FOREIGN KEY is a key used to link two tables together**.
- A FOREIGN KEY is a field (or collection of fields) in one table that refers to the PRIMARY KEY in another table.
- The table containing the foreign key is called the child table, and the table containing the candidate key is called the referenced or parent table.

Foreign Key Constraint

- **FOREIGN KEY:** Defines the column in the child table at the table-constraint level
- **REFERENCES**: Identifies the table and column in the parent table

Column Level:
COLUMN [data type] [CONSTRAINT] [constraint name] [REFERENCES] [table name (column name)]

[**Table level:**
CONSTRAINT [constraint name] [FOREIGN KEY (foreign key column name) REFERENCES]
[referenced table name (referenced column name)]

Foreign Key Constraint

```sql
CREATE TABLE Orders (
    OrderID int NOT NULL,
    OrderNumber int NOT NULL,
    PersonID int,
    PRIMARY KEY (OrderID),
    FOREIGN KEY (PersonID) REFERENCES Persons(PersonID)
);
```

```sql
CREATE TABLE Orders (
    OrderID int NOT NULL,
    OrderNumber int NOT NULL,
    PersonID int,
    PRIMARY KEY (OrderID),
    CONSTRAINT FK_PersonOrder FOREIGN KEY (PersonID)
    REFERENCES Persons(PersonID)
);
```

Foreign Key Constraint

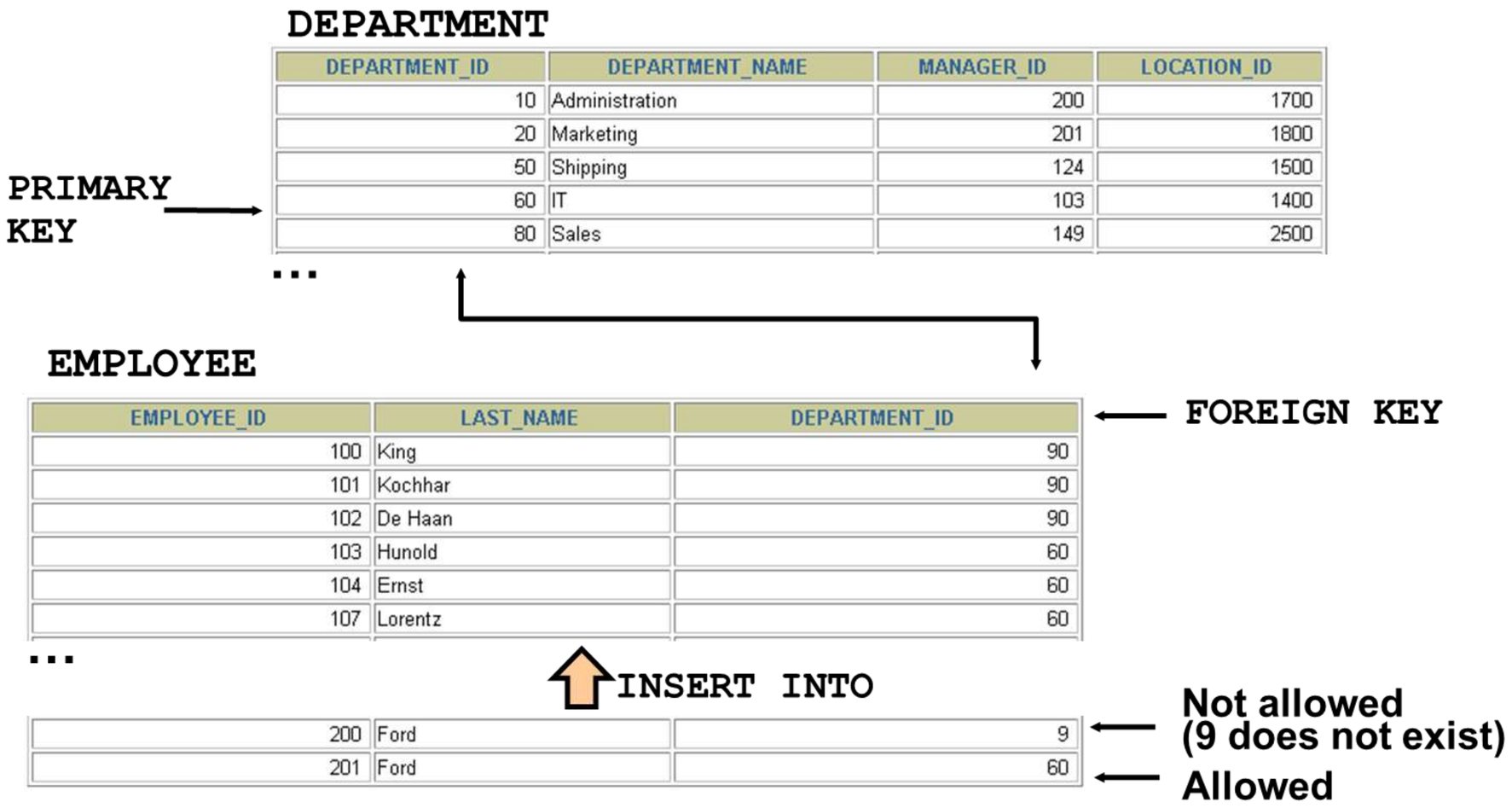
```
SQL> CREATE TABLE Orders (
  2     OrderID int NOT NULL,
  3     OrderNumber int NOT NULL,
  4     ID int,
  5     PRIMARY KEY (OrderID),
  6     CONSTRAINT FK_PersonOrder FOREIGN KEY (ID)
  7     REFERENCES Persons(ID)
  8  );

Table created.
```

Table level constraint

```
SQL> DESCRIBE Orders
 Name                                      Null?    Type
 ----------------------------------------- -------- ------------
 ORDERID                                   NOT NULL NUMBER(38)
 ORDERNUMBER                               NOT NULL NUMBER(38)
 ID                                                 NUMBER(38)
```

Foreign Key Constraint



Alter Table Statement

Use the ALTERTABLE statement to:

- Add a new column
- Modify an existing column
- Define a default value for the new column
- Drop a column

Alter Table Statement

Syntax:

ALTER TABLE table_name **action**;

1. **Add one or more columns**
2. **Modify column definition**
3. **Drop one or more columns**
4. **Rename columns**
5. **Rename table**

```
ALTER TABLE persons MODIFY(
    phone VARCHAR2(20) NOT NULL,
    email VARCHAR2(255) NOT NULL
);
```

```
ALTER TABLE persons
ADD (
    phone VARCHAR(20),
    email VARCHAR(100)
);
```

```
ALTER TABLE persons
ADD birthdate DATE NOT NULL;
```

```
ALTER TABLE persons RENAME TO people;
```

```
ALTER TABLE persons
DROP
  COLUMN birthdate;
```

Dropping a Table

- All data and structure in the table are deleted.
- Any pending transactions are committed.
- All indexes are dropped.
- All constraints are dropped.
- You cannot roll back the DROP TABLE statement.

```
DROP TABLE dept80;
Table dropped.
```

Truncate Statement

- **Removes all rows from a table,** leaving the table empty and the table structure intact
- **Is a data definition language (DDL) statement** rather than a DML statement; cannot easily be undone
- Syntax:

```
TRUNCATE TABLE table_name;
```

- Example:

```
TRUNCATE TABLE copy_emp;
```

DML Statements

- DML is Data Manipulation Language and is used to manipulate data.
- Examples of DML are insert, update and delete statements.
  - **SELECT** - retrieve data from a database
  - **INSERT** - insert data into a table
  - **UPDATE** - updates existing data within a table
  - **DELETE** - Delete all records from a database table

INSERT Statement Syntax

- Add new rows to a table by using the INSERT statement:
- With this syntax, only one row is inserted at a time.

```
INSERT INTO   table [(column [, column...])]
VALUES        (value [, value...]);
```

Inserting New Rows

- Insert a new row containing values for each column.
- List values in the default order of the columns in the table.
- Optionally, list the columns in the INSERT clause.

```
INSERT INTO departments(department_id,
       department_name, manager_id, location_id)
VALUES (70, 'Public Relations', 100, 1700);
1 row created.
```

- Enclose character and date values in single quotation marks.

Inserting Rows with Null Values

- **Implicit method**: Omit the column from the column list.

```
INSERT INTO    departments (department_id,
                            department_name)
VALUES          (280, 'ECA'       );
1 row created.
```

- **Explicit method**: Specify the NULL keyword in the VALUES clause.

```
INSERT INTO    departments
VALUES          (290,'Operation', NULL, NULL);
1 row created.
```

Method	Description
Implicit	Omit the column from the column list.
Explicit	Specify the NULL keyword in the VALUES list; specify the empty string ('') in the VALUES list for character strings and dates.

Inserting Rows with Null Values: Implicit Method

```
SQL> INSERT INTO departments
  2  (department_id, department_name)
  3  VALUES (280, 'ECA');

1 row created.
```

```
DEPARTMENT_ID DEPARTMENT_NAME                MANAGER_ID LOCATION_ID
------------- ------------------------------ ---------- -----------
          230 IT Helpdesk                                      1700
          240 Government Sales                                 1700
          250 Retail Sales                                     1700
          260 Recruiting                                       1700
          270 Payroll                                          1700
          280 ECA
```

Inserting Rows with Null Values: Explicit Method

```
SQL> INSERT INTO departments
  2  VALUES (290, 'Operation', NULL, NULL);
```

```
DEPARTMENT_ID DEPARTMENT_NAME                MANAGER_ID LOCATION_ID
------------- ------------------------------ ---------- -----------
          230 IT Helpdesk                                      1700
          240 Government Sales                                 1700
          250 Retail Sales                                     1700
          260 Recruiting                                       1700
          270 Payroll                                          1700
          280 ECA
          290 Operation
```

Changing Data in a Table

EMPLOYEES

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	HIRE_DATE	JOB_ID	SALARY	DEPARTMENT_ID	COMMISSION_P
100	Steven	King	SKING	17-JUN-87	AD_PRES	24000	90	
101	Neena	Kochhar	NKOCHHAR	21-SEP-89	AD_VP	17000	90	
102	Lex	De Haan	LDEHAAN	13-JAN-93	AD_VP	17000	90	
103	Alexander	Hunold	AHUNOLD	03-JAN-90	IT_PROG	9000	60	
104	Bruce	Ernst	BERNST	21-MAY-91	IT_PROG	6000	60	
107	Diana	Lorentz	DLORENTZ	07-FEB-99	IT_PROG	4200	60	
124	Kevin	Mourgos	KMOURGOS	16-NOV-99	ST_MAN	5800	50	

Update rows in the EMPLOYEES table:

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	HIRE_DATE	JOB_ID	SALARY	DEPARTMENT_ID	COMMISSIO
100	Steven	King	SKING	17-JUN-87	AD_PRES	24000	90	
101	Neena	Kochhar	NKOCHHAR	21-SEP-89	AD_VP	17000	90	
102	Lex	De Haan	LDEHAAN	13-JAN-93	AD_VP	17000	90	
103	Alexander	Hunold	AHUNOLD	03-JAN-90	IT_PROG	9000	30	
104	Bruce	Ernst	BERNST	21-MAY-91	IT_PROG	6000	30	
107	Diana	Lorentz	DLORENTZ	07-FEB-99	IT_PROG	4200	30	
124	Kevin	Mourgos	KMOURGOS	16-NOV-99	ST_MAN	5800	50	

UPDATE Statement Syntax

- Modify existing rows with the UPDATE statement:

```
UPDATE      table
SET         column = value [, column = value, ...]
[WHERE      condition];
```

- Update more than one row at a time (if required).

Updating Rows in a Table

- Specific row or rows are modified if you specify the WHERE clause:

```
UPDATE employees
SET    department_id = 70
WHERE  employee_id = 113;
1 row updated.
```

- All rows in the table are modified if you omit the WHERE clause:

```
UPDATE          copy_emp
SET             department_id = 110;
22 rows updated.
```

Removing a Row from a Table

- DEPARTMENTS

DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
10	Administration	200	1700
20	Marketing	201	1800
30	Purchasing		
100	Finance		
50	Shipping	124	1500
60	IT	103	1400

- Delete a row from the DEPARTMENTS table:

DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
10	Administration	200	1700
20	Marketing	201	1800
30	Purchasing		
50	Shipping	124	1500
60	IT	103	1400

DELETE Statement

- You can remove existing rows from a table by using the DELETE statement:

```
DELETE [FROM]   table
[WHERE   condition];
```

Deleting Rows from a Table

- Specific rows are deleted if you specify the WHERE clause:

```
 DELETE FROM departments
 WHERE  department_name = 'Finance';
1 row deleted.
```

- All rows in the table are deleted if you omit the WHERE clause

```
DELETE FROM  copy_emp;
22 rows deleted.
```

ITAHARI INTERNATIONAL COLLEGE
Islington college (इस्लिङ्टन कलेज)
informatics college • pokhara

Thank you

End of Lecture

CC5051 - DATABASES

INTRODUCTION TO SQL

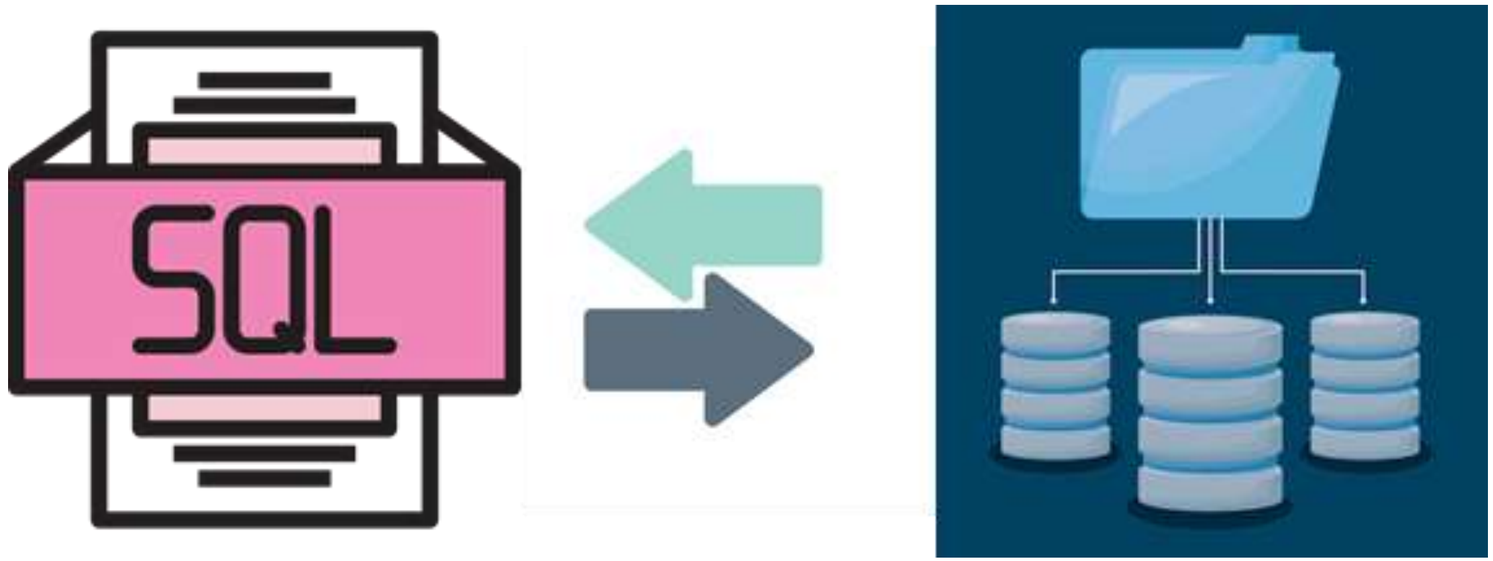
WEEK 3- Lecture

Topics

- Database Languages
- What is SQL?
- Why use SQL?
- Why is SQL important?
- SQL statement construction

What is SQL?

- **Structured Query Language (SQL)** is a standard database language which is used to store, manipulate and retrieve data in database.



What is SQL Used For?

- SQL is used to **create new database and database tables**
- SQL can be used to **retrieve data from a database**
- SQL can be used to **insert, update and delete data in a database**

What is SQL Used For?

- SQL can be used to set access permissions for database tables
- SQL can be used to create stored procedures for database
- SQL is used to create views in database

From Relational Algebra to SQL

In 1970, Edgar F. Codd introduced the Relational Model at IBM.

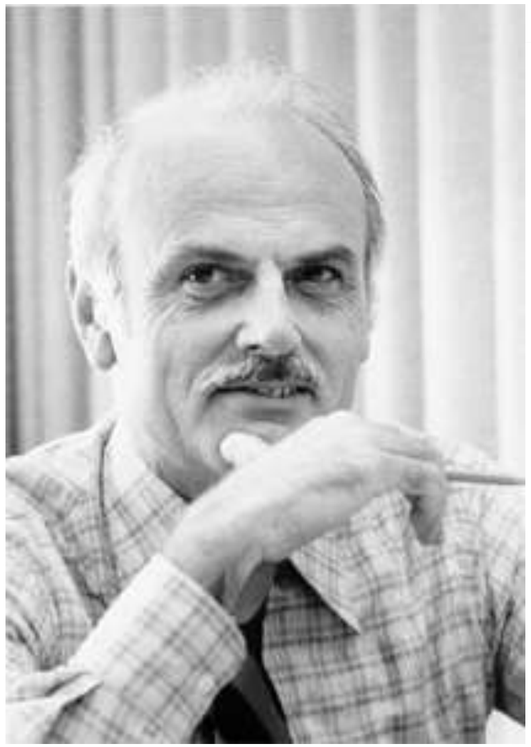
Proposed Relational Algebra & Relational Calculus as formal query languages.

Relational Algebra:

A mathematical system for manipulating relations (tables).

Operators: Selection (σ), Projection (π), Join (⋈), Union (∪), Difference (-), Division (÷).

Provided the **theoretical foundation** for querying databases.



From Relational Algebra to SQL

In Mid-1970s: IBM developed **SEQUEL** (Structured English Query Language) which **later renamed SQL.**

SQL was designed to be declarative: users specify what data they want, not how to get it.

SQL concepts directly map to relational algebra operations:

σ → WHERE

π → SELECT column

⋈ → JOIN

∪ → UNION

Example:

Relational Algebra: σclass="Maths"(STUDENT)

SQL: SELECT * FROM STUDENT WHERE class='Maths';

Evolution of SQL

1980s Standardization

- SQL officially became the ANSI (1986) and ISO (1987) standard.
- Ensured compatibility across different database systems.

Extensions Beyond Relational Algebra

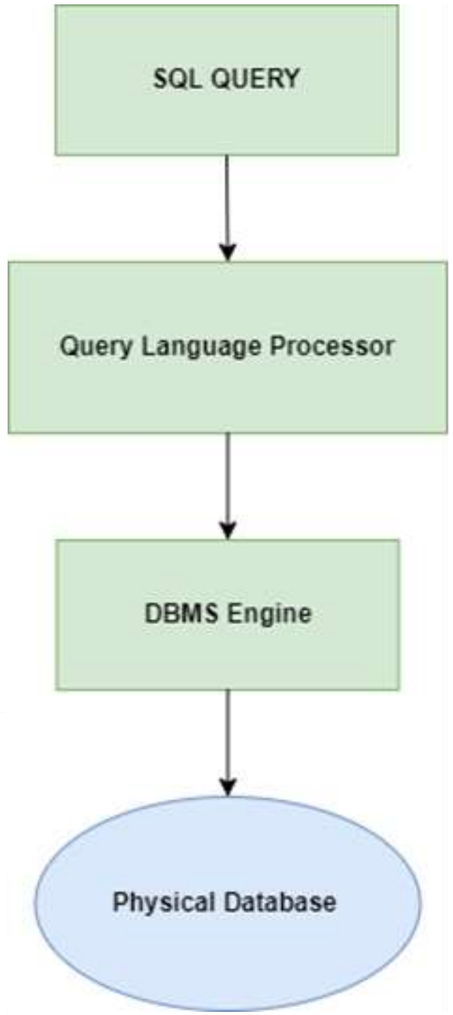
- **Aggregates (SUM, COUNT, AVG):** Enabled summarizing and analyzing data.
- **NULL values:** Introduced the concept of missing or unknown data.
- **Constraints:** Enforced rules (PRIMARY KEY, FOREIGN KEY, UNIQUE, CHECK).
- **Views & Triggers**: Allowed virtual tables and automatic actions on data changes.
- **Transactions (ACID):** Guaranteed reliability with Atomicity, Consistency, Isolation, Durability.

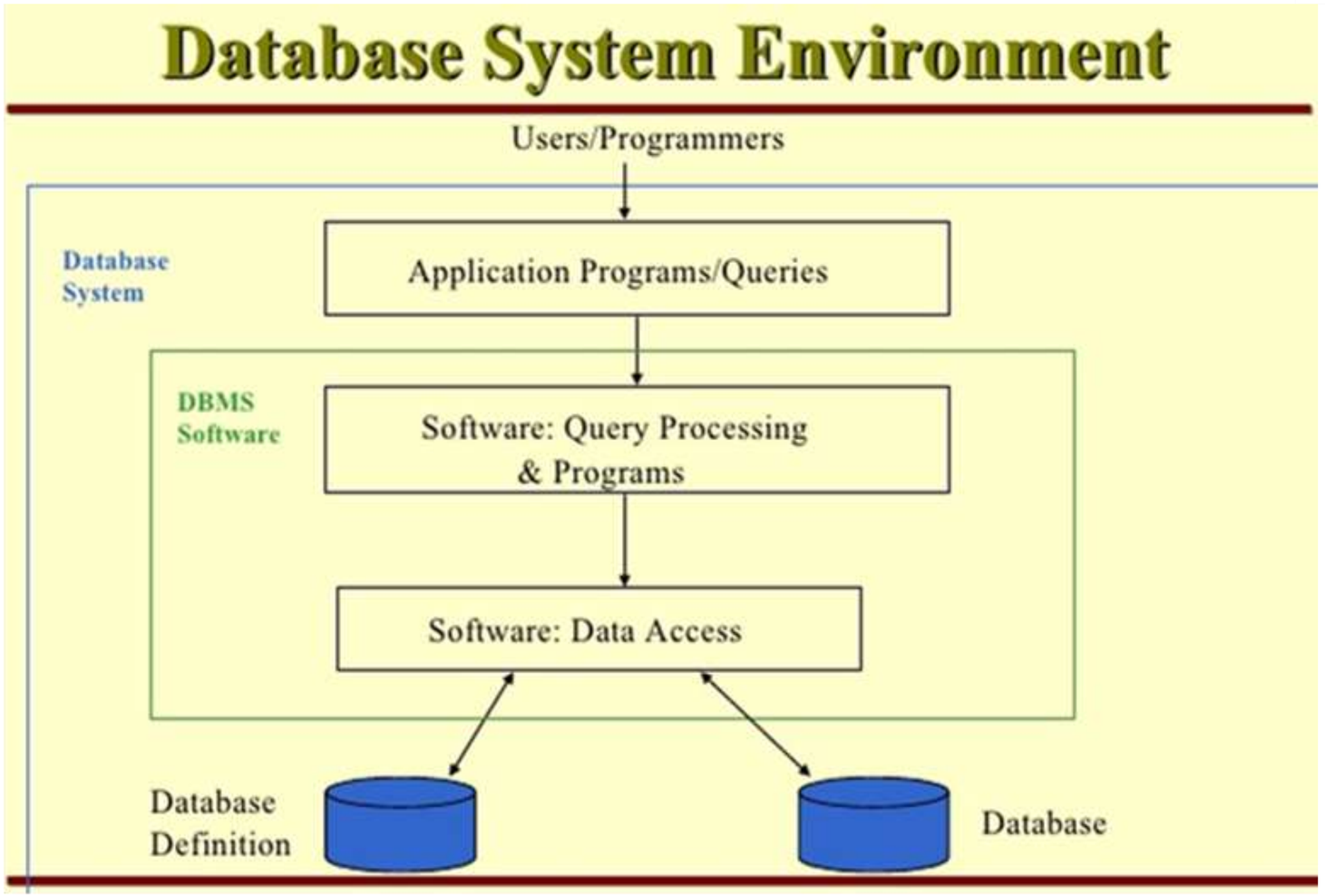
Modern SQL

- Combines the mathematical rigor of relational algebra with practical database features.
- Forms the core query language for all major relational DBMS:

  Oracle, MySQL, PostgreSQL, SQL Server etc.

SQL Process


SQL QUERY
Query Language Processor
DBMS Engine
Physical Database


Database System Environment
Users/Programmers
Database System
Application Programs/Queries
DBMS Software
Software: Query Processing & Programs
Software: Data Access
Database Definition
Database

Database Languages

Every DBMS includes a set of database languages that are used to interact with the database system. These languages allow users to **define**, **manipulate**, **control**, and **retrieve data** as needed.

- The common components include:
- Data Definition Language (DDL)
- Data Manipulation Language (DML)
- Data Control Language (DCL)
- Transaction Control Language(TCL)
- Query Language
- Host Language

Host Language

- A host language is a general-purpose programming language that is used in combination with SQL to build database-driven applications.
- Allows SQL statements to be embedded into applications with programming languages like Java, C, etc.
- Host languages - C, Java, COBOL, Visual Basic, etc.

Host Language

We use SQL embedded within a host language to:

- Accept input from users
- Display results
- Perform logic and processing

```java
import java.sql.*;

public class Main {
    public static void main(String[] args) throws Exception {
        Connection con = DriverManager.getConnection(
            "jdbc:mysql://localhost:3306/school", "root", "password");

        Statement stmt = con.createStatement();
        ResultSet rs = stmt.executeQuery("SELECT name FROM students");

        while (rs.next()) {
            System.out.println(rs.getString("name"));
        }

        con.close();
    }
}
```

Query Language

A query language is used to retrieve and display data from a database.

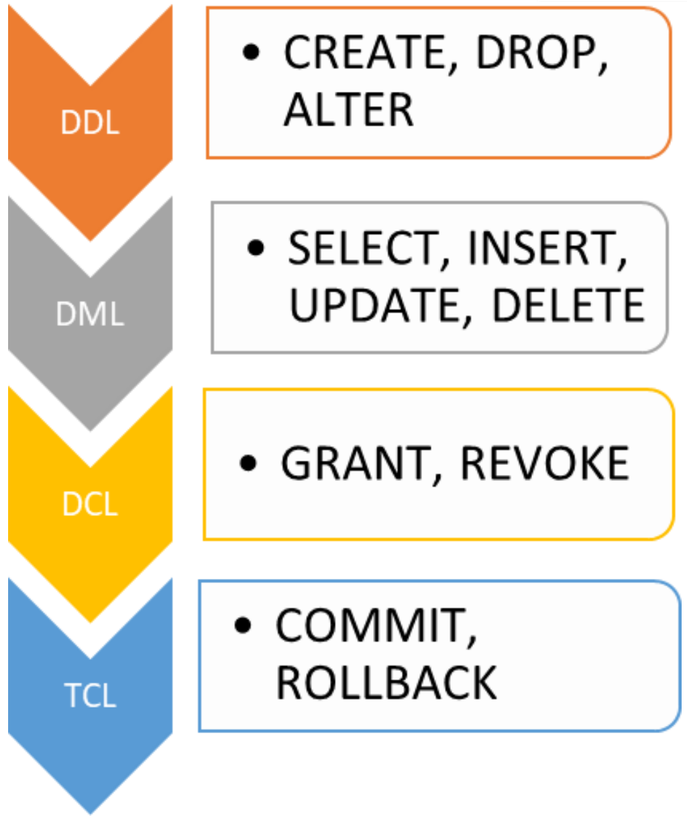
- **The most common query language is SQL (Structured Query Language).**
- Used to fetch specific information from large datasets efficiently.

Example: SELECT name, age FROM students WHERE grade = 'A';

SQL category

Classified by type of operation

- **Data Definition Language (DDL):** Facilitates Creation and Description of Database.
- **Data Manipulation Language (DML):** Deals with Manipulation and Processing of Data.
- **Data Control Language (DCL):** Deals with Data Access.
- **Transaction Control Language (TCL):** Manages transactions in the database and the changes made to the data in a table by DML statements.



Why use SQL

- Allow users to:
  - **Create Database** and Relational Structures
  - **Insert, Modify and Delete** Database Data
  - Perform simple and complex queries.
- Performs all these tasks with minimal user effort, and command structure / syntax (relatively) simple to learn.
- Portable and Standardized and have DDL and DML component parts to it.
- Declarative: State question to answer not how to answer it

SQL – Statement Construction

- SQL statements consists of reserved words and user-defined words.
  - Reserved words are a fixed part of SQL and must be spelt exactly as required and cannot be split across lines.
  - User-defined words are made up by user and represent names of various database objects such as relations, columns, views.
- Most components of an SQL statement are case insensitive, except for literal character data.

Rules Followed by SQL

- SQL queries consists of keywords that perform certain functions

```
SELECT * FROM students;
```

- SQL keywords are not case sensitive but are represented in upper case as a convention.
- A single SQL query can be written in one or more than one line and ends through the use of semicolon (;).

```
SELECT first_name, last_name
FROM employees
WHERE department_id = 10;
```

SQL – Statement Construction

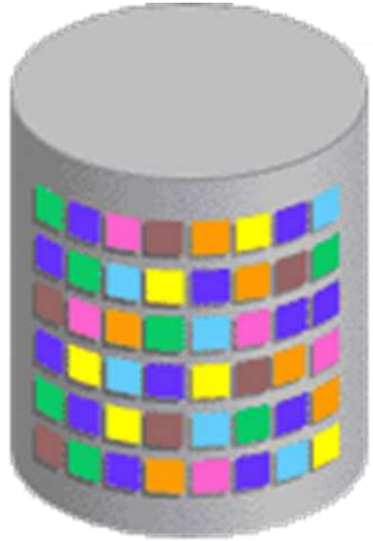
- More readable with indentation and lineation:
  - Each clause should begin on new line.
  - Start of clause should line up with start of other clauses.
  - Where clause has several parts, each should appear on separate line and be indented under start of clause.

DDL Statements

- Data definition language (DDL) statements
- Syntax for creating and modifying database objects such as tables, users etc.
- Let you to perform these tasks: **Create, alter, drop, rename and truncate**

Create Table Statement

- You must have:
  - **CREATE TABLE** privilege
  - A storage area
  - Have immediate effect on database
  - Record information in the data dictionary.
- You specify:
  - Table name, Column name, column data type, and column size



```
CREATE TABLE [schema.]table
            (column datatype [DEFAULT expr][, ...]);
```

Naming Rules

- Table names and column names:
  - Must begin with a letter
  - Must be 1–30 characters long
  - Must contain only A–Z, a–z, 0–9, _, $, and # (legal characters, but their use is discouraged).
  - Must not duplicate the name of another object owned by the same user
  - Must not be an Oracle server reserved word

Creating Tables

- **Create the table.**

```
SQL> CREATE TABLE test1 (deptno  NUMBER(2),
  2            dname  VARCHAR2(14),
  3            email  VARCHAR2(14));

Table created.
```

- **Confirm Table Creation**

```
SQL> describe test1
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 DEPTNO                                             NUMBER(2)
 DNAME                                              VARCHAR2(14)
 EMAIL                                              VARCHAR2(14)
```

Data Types

Data Type	Description	Data size
VARCHAR2(*size*)	Variable-length character data	4000 bytes
CHAR(*size*)	Fixed-length character data	2000 bytes
NUMBER(*p*,*s*)	Variable-length numeric data	Precision ranges from 1 to 38 while the scale range from -84 to 127
DATE	Date and time values	Valid date range from January 1, 4712 BC to December 31, 9999 AD.
LONG	Variable-length character data (up to 2 GB)	2 gigabytes, or 2^{31} -1 bytes.
CLOB	Character data (up to 4 GB)	Maximum size: (4 GB - 1) * DB_BLOCK_SIZE initialization parameter (8 TB to 128 TB)
RAW and LONG RAW	Raw binary data	Maximum size: 2000 bytes
BLOB	Binary data (up to 4 GB)	Maximum size: (4 GB - 1) * DB_BLOCK_SIZE initialization parameter (8 TB to 128 TB)
BFILE	Binary data stored in an external file (up to 4 GB)	Maximum size: 4 GB
ROWID	A base-64 number system representing the unique address of a row in its table	Base 64 string representing the unique address of a row in its table

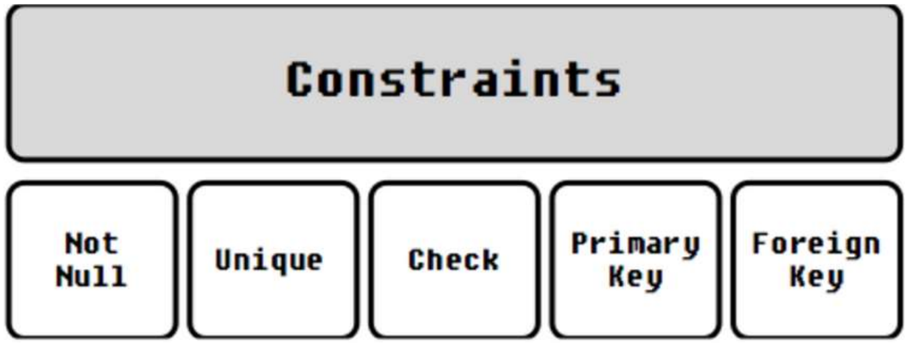
Data Types

By default, Oracle uses BYTE or Characters in if you don’t explicitly specify BYTE or CHARACTER,Default value is used.

Data Type	LENGTH
VARCHAR2(*size*)	4000
CHAR(*size*)	2000
NUMBER(*p*,*s*)	38-digit precision
DATE	Range from January 1, 4712 BC to December 31, 9999 AD.

Constraints

- Rules to preserve the data integrity in the application.
- Imposed on a column of a database table, to check data.
- The key milestones achieved by the usage of constraints are:
- Validate NULL property of the data
- Validate uniqueness of the data
- Validate referential integrity of the data



Defining Constraints

- **Syntax:**

```
CREATE TABLE [schema.]table
      (column datatype [DEFAULT expr]
      [column_constraint],
      ...
      [table_constraint][,...]);
```

- **Column-level constraint:**

```
column [CONSTRAINT constraint_name]
constraint_type,
```

- **Table-level constraint:**

```
column,...
  [CONSTRAINT constraint_name] constraint_type
  (column, ...),
```

Defining Constraints

- **Column-level constraint:**

```
CREATE TABLE employees(
  employee_id  NUMBER(6)
    CONSTRAINT emp_emp_id_pk PRIMARY KEY,
  first_name   VARCHAR2(20),
  ...);
```

- **Table-level constraint**

```
CREATE TABLE employees(
  employee_id  NUMBER(6),
  first_name   VARCHAR2(20),
  ...
  job_id       VARCHAR2(10) NOT NULL,
  CONSTRAINT emp_emp_id_pk
    PRIMARY KEY (EMPLOYEE_ID));
```

Not Null Constraint

- Ensures that the column contains no null values.
- Columns without the **NOT NULL** constraint can contain null values by default.
- **NOT NULL** constraints must be defined at the column level.
- Syntax

  [COLUMN NAME] [DATA TYPE] [CONSTRAINT NAME)]

Not Null

- **Column-level constraint:**

```
CREATE TABLE table_name
( ... column_name data_type NOT NULL
…..);
```

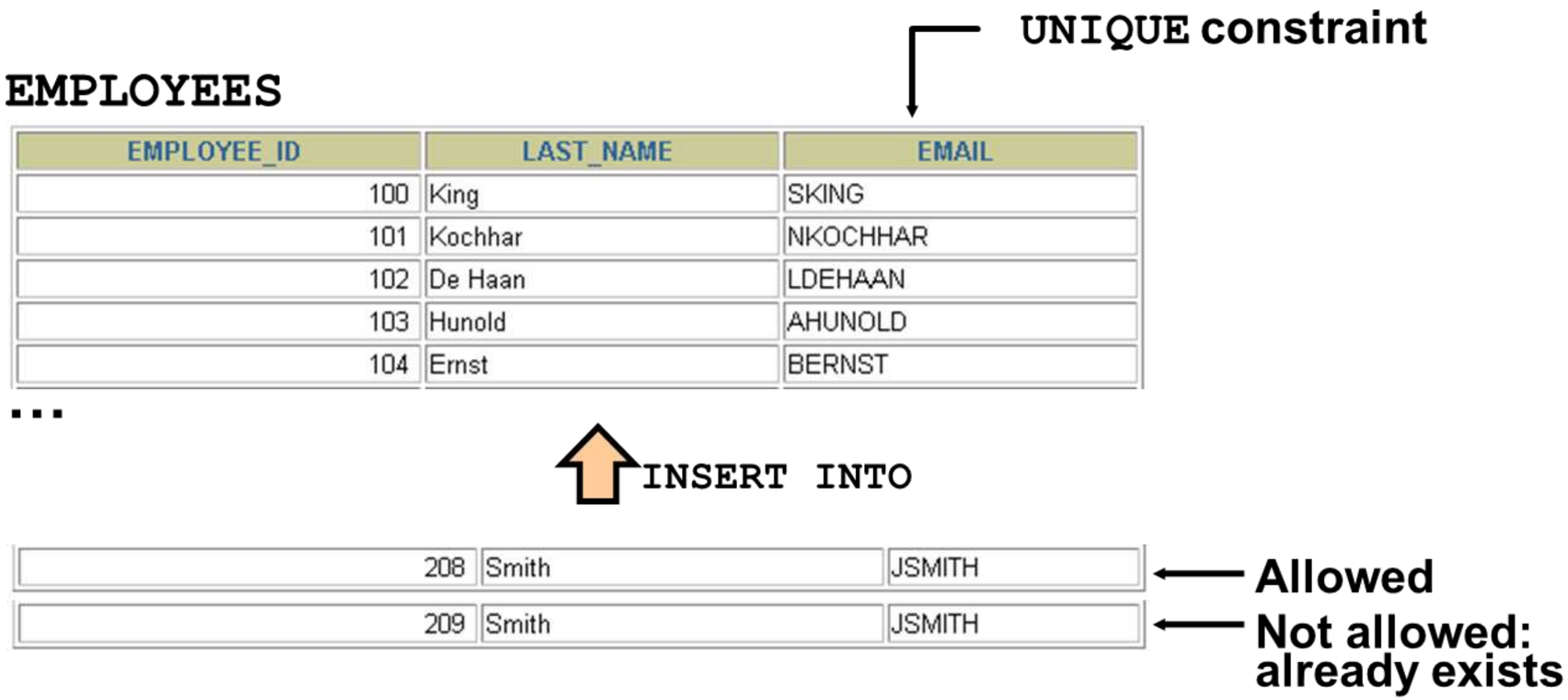
1

```
SQL> CREATE TABLE surcharges (
  2  surcharge_id NUMBER,
  3  surcharge_name VARCHAR2(255) NOT NULL,
  4  amount NUMBER(9,2));

Table created.
```

Unique Constraint

- Ensures the data stored in a column, or a group of columns, is unique among the rows in a table.



Unique

- **Column-level constraint:**

```
CREATE TABLE table_name
 ... column_name data_type UNIQUE ... );
```

```
SQL> CREATE TABLE charges (
  2  charge_name VARCHAR2(255) UNIQUE,
  3  amount NUMBER(9,2));

Table created.
```

```
SQL> DESCRIBE charges
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 CHARGE_NAME                                        VARCHAR2(255)
 AMOUNT                                             NUMBER(9,2)
```

Unique Constraint

Defined at either the table level or the column level:

```
CREATE TABLE table_name ( ... column_name data_type
UNIQUE, ... );
```

```
CREATE TABLE table_name ( ... column_name data_type,
..., CONSTRAINT unique_constraint_name
UNIQUE(column_name) );
```

Column-level constraint:

```
CREATE TABLE table_name ( ... column_name1 data_type,
column_name2 data_type, ..., CONSTRAINT unique_constraint_name
UNIQUE(column_name1, column_name2) );
```

Unique Constraint

Defined at either the table level or the column level:

```
CREATE TABLE clients (
client_id NUMBER, first_name VARCHAR2(50) NOT NULL,
last_name VARCHAR2(50) NOT NULL, company_name
VARCHAR2(255) NOT NULL,
email VARCHAR2(255) NOT NULL, phone VARCHAR(25),
CONSTRAINT UNQ_C_EMAIL UNIQUE (email)
);
```

Unique Constraint

```
SQL> CREATE TABLE clients (
  2  client_id NUMBER, first_name VARCHAR2(50) NOT NULL, last_name VARCHAR2(50) NOT NULL, company_name VARCHAR2(255) N
OT NULL,
  3  email VARCHAR2(255) NOT NULL, phone VARCHAR(25),
  4  CONSTRAINT UNQ_C_EMAIL UNIQUE (email)
  5  );
```

Table level constraint

```
SQL> DESCRIBE Clients
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 CLIENT_ID                                          NUMBER
 FIRST_NAME                                NOT NULL VARCHAR2(50)
 LAST_NAME                                 NOT NULL VARCHAR2(50)
 COMPANY_NAME                              NOT NULL VARCHAR2(255)
 EMAIL                                     NOT NULL VARCHAR2(255)
 PHONE                                              VARCHAR2(25)
```

Check

- Allows you to enforce domain integrity by limiting the values accepted by one or more columns.
- To create a check constraint, you define a logical expression that returns true or false.
- You can apply a check constraint to a column or a group of columns. A column may have one or more check constraints.
- **Syntax:**

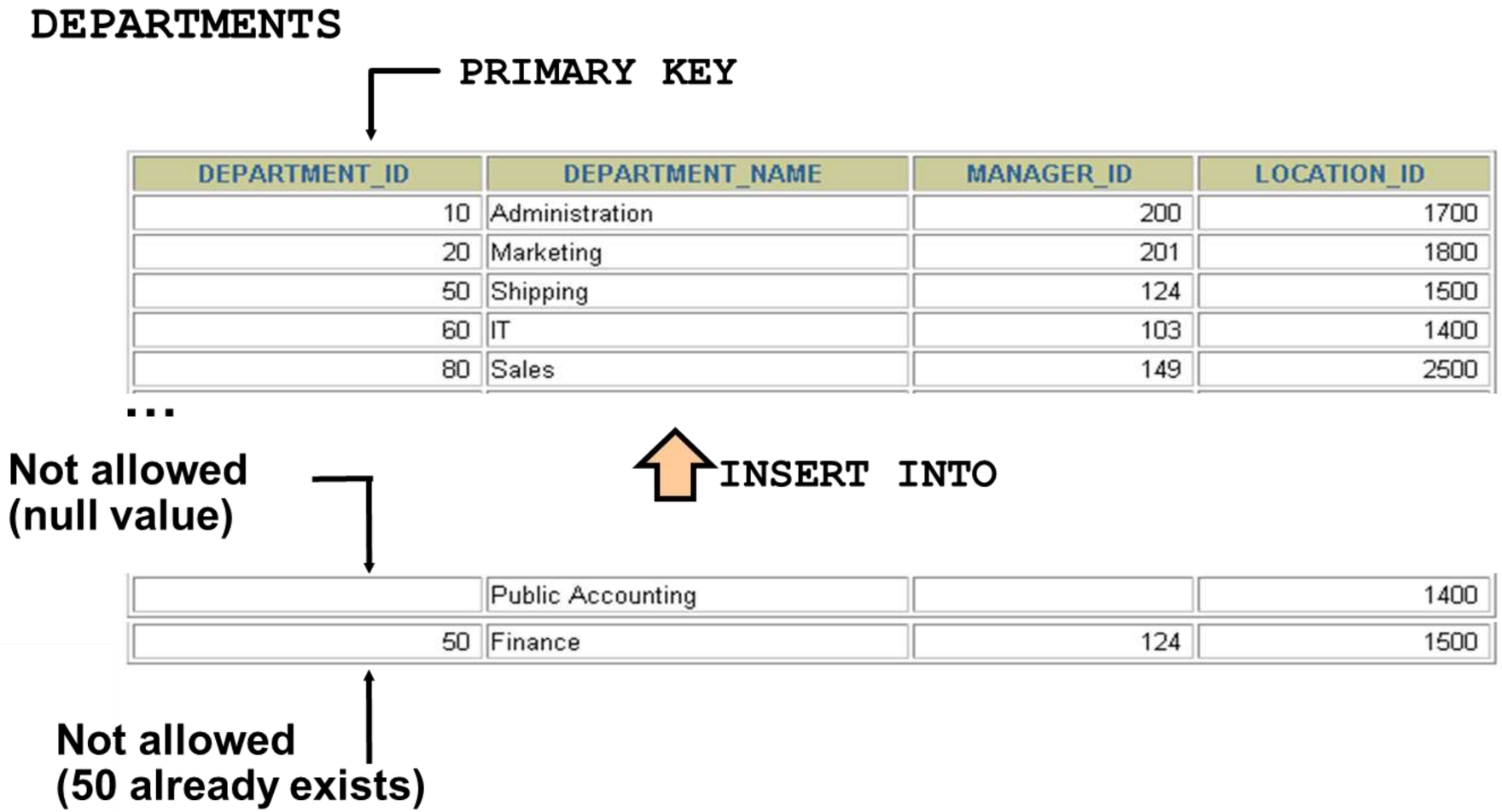
```
CREATE TABLE table_name
( ... column_name data_type CHECK
(expression),
);
```

```
CREATE TABLE parts ( part_id NUMBER,
buy_price NUMBER(9,2) CHECK(buy_price > 0),);
```

Primary Key Constraint

- A **primary key is a column or combination of columns in a table that uniquely identifies a row in the table**.
- The following are rules that make a column a primary key:
  - A primary key column cannot contain a NULL value or an empty string.
  - A primary key value must be unique within the entire table.
  - A primary key value should not be changed over time.

Primary Key Constraint



Primary Key Constraint

```
CREATE TABLE table_name
        ( column1 datatype null/not null,
        column2 datatype,
... CONSTRAINT constraint_name PRIMARY KEY
(column1, column2, ... column_n) );
```

At column level, CONSTRAINT keyword and CONSTRAINT_NAME are optional.

```
CREATE TABLE supplier (
 supplier_id numeric(10) not null,
 supplier_name varchar2(50) not null,
 contact_name varchar2(50), CONSTRAINT supplier_pk PRIMARY KEY
(supplier_id) );
```

Primary Key Constraint

```
CREATE TABLE supplier (
 supplier_id numeric(10) not null,
 supplier_name varchar2(50) not null,
 contact_name varchar2(50), CONSTRAINT supplier_pk PRIMARY KEY
(supplier_id) );
```

```
SQL> CREATE TABLE supplier (
  2   supplier_id numeric(10) not null,
  3   supplier_name varchar2(50) not null,
  4   contact_name varchar2(50), CONSTRAINT supplier_pk PRIMARY KEY (supplier_id) );

Table created.
```

```
SQL> DESCRIBE supplier
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 SUPPLIER_ID                               NOT NULL NUMBER(10)
 SUPPLIER_NAME                             NOT NULL VARCHAR2(50)
 CONTACT_NAME                                       VARCHAR2(50)
```

Primary Key Constraint

PRIMARY KEY Constraint

```
CREATE TABLE Persons (
    ID int NOT NULL PRIMARY KEY,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int
);
```

```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    CONSTRAINT PK_Person PRIMARY KEY (ID,LastName)
);
```

Primary Key Constraint

```
SQL> CREATE TABLE Persons (
  2     ID int NOT NULL,
  3     LastName VARCHAR(25) NOT NULL,
  4     FirstName VARCHAR(25),
  5     Age int,
  6     CONSTRAINT PK_Person PRIMARY KEY (ID)
  7  );

Table created.
```


Table level constraint

```
SQL> DESCRIBE Persons
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 ID                                        NOT NULL NUMBER(38)
 LASTNAME                                  NOT NULL VARCHAR2(25)
 FIRSTNAME                                          VARCHAR2(25)
 AGE                                                NUMBER(38)
```

Foreign Key Constraint

- A **FOREIGN KEY is a key used to link two tables together**.
- A FOREIGN KEY is a field (or collection of fields) in one table that refers to the PRIMARY KEY in another table.
- The table containing the foreign key is called the child table, and the table containing the candidate key is called the referenced or parent table.

Foreign Key Constraint

- **FOREIGN KEY:** Defines the column in the child table at the table-constraint level
- **REFERENCES**: Identifies the table and column in the parent table

Column Level:
COLUMN [data type] [CONSTRAINT] [constraint name] [REFERENCES] [table name (column name)]

[**Table level:**
CONSTRAINT [constraint name] [FOREIGN KEY (foreign key column name) REFERENCES]
[referenced table name (referenced column name)]

Foreign Key Constraint

```sql
CREATE TABLE Orders (
    OrderID int NOT NULL,
    OrderNumber int NOT NULL,
    PersonID int,
    PRIMARY KEY (OrderID),
    FOREIGN KEY (PersonID) REFERENCES Persons(PersonID)
);
```

```sql
CREATE TABLE Orders (
    OrderID int NOT NULL,
    OrderNumber int NOT NULL,
    PersonID int,
    PRIMARY KEY (OrderID),
    CONSTRAINT FK_PersonOrder FOREIGN KEY (PersonID)
    REFERENCES Persons(PersonID)
);
```

Foreign Key Constraint

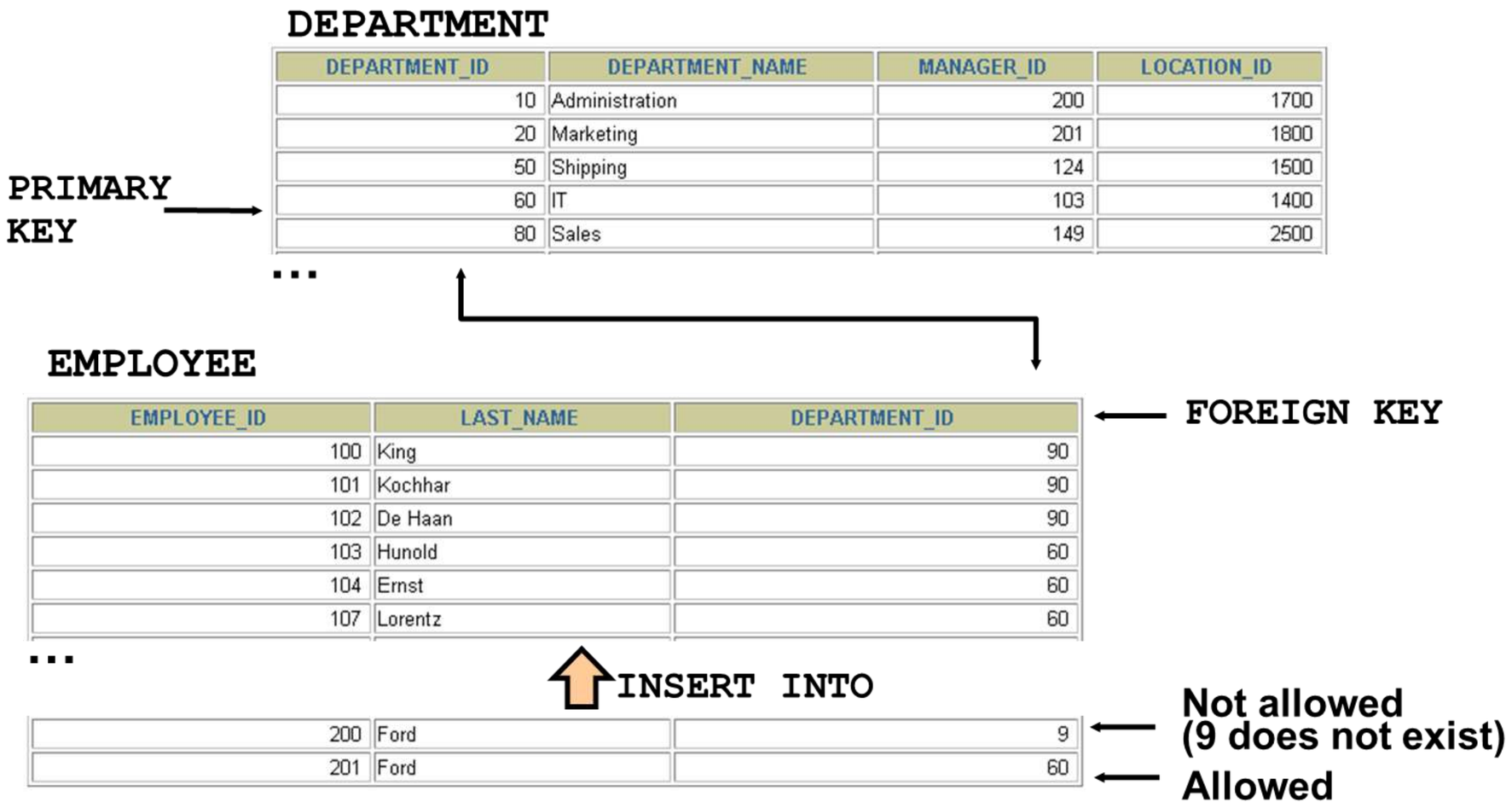
```
SQL> CREATE TABLE Orders (
  2     OrderID int NOT NULL,
  3     OrderNumber int NOT NULL,
  4     ID int,
  5     PRIMARY KEY (OrderID),
  6     CONSTRAINT FK_PersonOrder FOREIGN KEY (ID)
  7     REFERENCES Persons(ID)
  8  );

Table created.
```

Table level constraint

```
SQL> DESCRIBE Orders
 Name                                      Null?    Type
 ----------------------------------------- -------- ------------
 ORDERID                                   NOT NULL NUMBER(38)
 ORDERNUMBER                               NOT NULL NUMBER(38)
 ID                                                 NUMBER(38)
```

Foreign Key Constraint



Alter Table Statement

Use the ALTERTABLE statement to:

- Add a new column
- Modify an existing column
- Define a default value for the new column
- Drop a column

Alter Table Statement

Syntax:

ALTER TABLE table_name **action**;

1. **Add one or more columns**
2. **Modify column definition**
3. **Drop one or more columns**
4. **Rename columns**
5. **Rename table**

```
ALTER TABLE persons MODIFY(
    phone VARCHAR2(20) NOT NULL,
    email VARCHAR2(255) NOT NULL
);
```

```
ALTER TABLE persons
ADD (
    phone VARCHAR(20),
    email VARCHAR(100)
);
```

```
ALTER TABLE persons
ADD birthdate DATE NOT NULL;
```

```
ALTER TABLE persons RENAME TO people;
```

```
ALTER TABLE persons
DROP
  COLUMN birthdate;
```

Dropping a Table

- All data and structure in the table are deleted.
- Any pending transactions are committed.
- All indexes are dropped.
- All constraints are dropped.
- You cannot roll back the DROP TABLE statement.

```
DROP TABLE dept80;
Table dropped.
```

Truncate Statement

- **Removes all rows from a table,** leaving the table empty and the table structure intact
- **Is a data definition language (DDL) statement** rather than a DML statement; cannot easily be undone
- Syntax:

```
TRUNCATE TABLE table_name;
```

- Example:

```
TRUNCATE TABLE copy_emp;
```

DML Statements

- DML is Data Manipulation Language and is used to manipulate data.
- Examples of DML are insert, update and delete statements.
  - **SELECT** - retrieve data from a database
  - **INSERT** - insert data into a table
  - **UPDATE** - updates existing data within a table
  - **DELETE** - Delete all records from a database table

INSERT Statement Syntax

- Add new rows to a table by using the INSERT statement:
- With this syntax, only one row is inserted at a time.

```
INSERT INTO   table [(column [, column...])]
VALUES        (value [, value...]);
```

Inserting New Rows

- Insert a new row containing values for each column.
- List values in the default order of the columns in the table.
- Optionally, list the columns in the INSERT clause.

```
INSERT INTO departments(department_id,
       department_name, manager_id, location_id)
VALUES (70, 'Public Relations', 100, 1700);
1 row created.
```

- Enclose character and date values in single quotation marks.

Inserting Rows with Null Values

- **Implicit method**: Omit the column from the column list.

```
INSERT INTO    departments (department_id,
                            department_name)
VALUES          (280, 'ECA'       );
1 row created.
```

- **Explicit method**: Specify the NULL keyword in the VALUES clause.

```
INSERT INTO    departments
VALUES          (290,'Operation', NULL, NULL);
1 row created.
```

Method	Description
Implicit	Omit the column from the column list.
Explicit	Specify the NULL keyword in the VALUES list; specify the empty string ('') in the VALUES list for character strings and dates.

Inserting Rows with Null Values: Implicit Method

```
SQL> INSERT INTO departments
  2  (department_id, department_name)
  3  VALUES (280, 'ECA');

1 row created.
```

```
DEPARTMENT_ID DEPARTMENT_NAME                MANAGER_ID LOCATION_ID
------------- ------------------------------ ---------- -----------
          230 IT Helpdesk                                      1700
          240 Government Sales                                 1700
          250 Retail Sales                                     1700
          260 Recruiting                                       1700
          270 Payroll                                          1700
          280 ECA
```

Inserting Rows with Null Values: Explicit Method

```
SQL> INSERT INTO departments
  2  VALUES (290, 'Operation', NULL, NULL);
```

```
DEPARTMENT_ID DEPARTMENT_NAME                MANAGER_ID LOCATION_ID
------------- ------------------------------ ---------- -----------
          230 IT Helpdesk                                      1700
          240 Government Sales                                 1700
          250 Retail Sales                                     1700
          260 Recruiting                                       1700
          270 Payroll                                          1700
          280 ECA
          290 Operation
```

Changing Data in a Table

EMPLOYEES

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	HIRE_DATE	JOB_ID	SALARY	DEPARTMENT_ID	COMMISSION_P
100	Steven	King	SKING	17-JUN-87	AD_PRES	24000	90	
101	Neena	Kochhar	NKOCHHAR	21-SEP-89	AD_VP	17000	90	
102	Lex	De Haan	LDEHAAN	13-JAN-93	AD_VP	17000	90	
103	Alexander	Hunold	AHUNOLD	03-JAN-90	IT_PROG	9000	60	
104	Bruce	Ernst	BERNST	21-MAY-91	IT_PROG	6000	60	
107	Diana	Lorentz	DLORENTZ	07-FEB-99	IT_PROG	4200	60	
124	Kevin	Mourgos	KMOURGOS	16-NOV-99	ST_MAN	5800	50	

Update rows in the EMPLOYEES table:

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	HIRE_DATE	JOB_ID	SALARY	DEPARTMENT_ID	COMMISSIO
100	Steven	King	SKING	17-JUN-87	AD_PRES	24000	90	
101	Neena	Kochhar	NKOCHHAR	21-SEP-89	AD_VP	17000	90	
102	Lex	De Haan	LDEHAAN	13-JAN-93	AD_VP	17000	90	
103	Alexander	Hunold	AHUNOLD	03-JAN-90	IT_PROG	9000	30	
104	Bruce	Ernst	BERNST	21-MAY-91	IT_PROG	6000	30	
107	Diana	Lorentz	DLORENTZ	07-FEB-99	IT_PROG	4200	30	
124	Kevin	Mourgos	KMOURGOS	16-NOV-99	ST_MAN	5800	50	

UPDATE Statement Syntax

- Modify existing rows with the UPDATE statement:

```
UPDATE      table
SET         column = value [, column = value, ...]
[WHERE      condition];
```

- Update more than one row at a time (if required).

Updating Rows in a Table

- Specific row or rows are modified if you specify the WHERE clause:

```
UPDATE employees
SET    department_id = 70
WHERE  employee_id = 113;
1 row updated.
```

- All rows in the table are modified if you omit the WHERE clause:

```
UPDATE          copy_emp
SET             department_id = 110;
22 rows updated.
```

Removing a Row from a Table

- DEPARTMENTS

DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
10	Administration	200	1700
20	Marketing	201	1800
30	Purchasing		
100	Finance		
50	Shipping	124	1500
60	IT	103	1400

- Delete a row from the DEPARTMENTS table:

DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
10	Administration	200	1700
20	Marketing	201	1800
30	Purchasing		
50	Shipping	124	1500
60	IT	103	1400

DELETE Statement

- You can remove existing rows from a table by using the DELETE statement:

```
DELETE [FROM]   table
[WHERE   condition];
```

Deleting Rows from a Table

- Specific rows are deleted if you specify the WHERE clause:

```
 DELETE FROM departments
 WHERE  department_name = 'Finance';
1 row deleted.
```

- All rows in the table are deleted if you omit the WHERE clause

```
DELETE FROM  copy_emp;
22 rows deleted.
```

ITAHARI INTERNATIONAL COLLEGE
Islington college (इस्लिङ्टन कलेज)
informatics college • pokhara

Thank you

End of Lecture

CC5051 - DATABASES

Database Models

&

Conceptual Data Modeling

Week 4

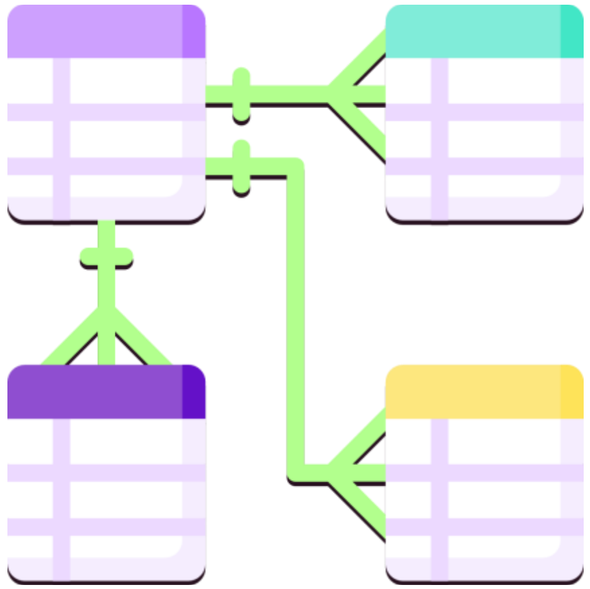
Agendas

- Data Modeling
- Types of Data Model
- Database Design Process
- Level of Abstraction
- Entity Relationship Diagram
- ERD components

Data Model and Data Modeling

A data model is a **conceptual framework** that describes **how data is organized, stored, and related to one another** within a database.

It provides a structured way to **represent entities**, their **attributes**, and the **relationships** between them — usually using symbols, rules, and diagrams.



Data Modeling

- In databases, data modelling is the **process of creating models to store the data** in the database.
- Data modeling is the **process of creating a visual representation** of data structures, requirements, and relationships — typically at the conceptual, logical, or physical level.
- It helps database designers, developers, and stakeholders communicate, plan, and build efficient database systems that meet real-world needs.

Real world scenario of Data Modeling

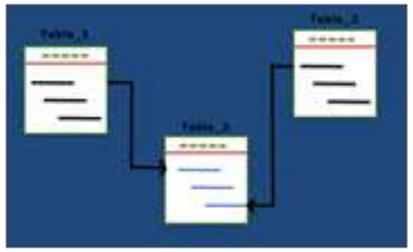
- Imagine Setting Up a **Library**:
  - **Would you just throw books onto shelves randomly?**
    - Of course not!
  - How would you organize them? By making categories like:
    - Genre (e.g., Fiction, Science)
    - **Author**
    - **Alphabetical order**
  - You'd also track:
    - Who borrows books (**Members**)
    - Which book was borrowed (**Loans**)
    - When it's due (**Return Date**)
- Data modeling is the process of planning how to structure, store, and connect data in a database — just like planning a library layout.

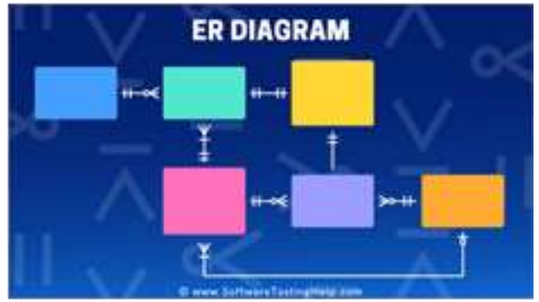
Why is data modelling Important?

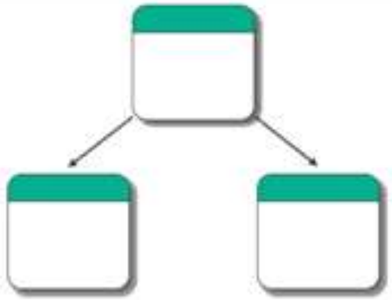
- **To obtain quality data**: Goal of Data Model is to make sure that all Data Objects required by Database are completely and accurately represented.
- **To easily retrieve/analyse data and To reduce cost**
- **To set a clear scope**: Data Model is also detailed enough to be used by Database Developers to use as "**blueprint**" for building Physical Database.
- It will give us a clear insight into what things we need to focus on more and **how the system will develop in the days to come.**

Types of Database Model

- Hierarchical model
- Network model
- Relational model
- Entity-relationship model
- Object-oriented model
- Graph model
- Document model



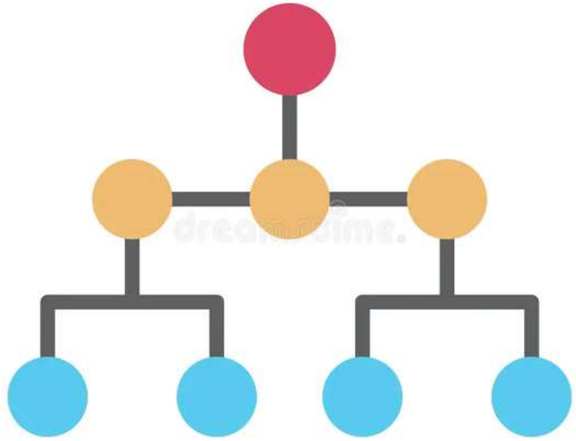




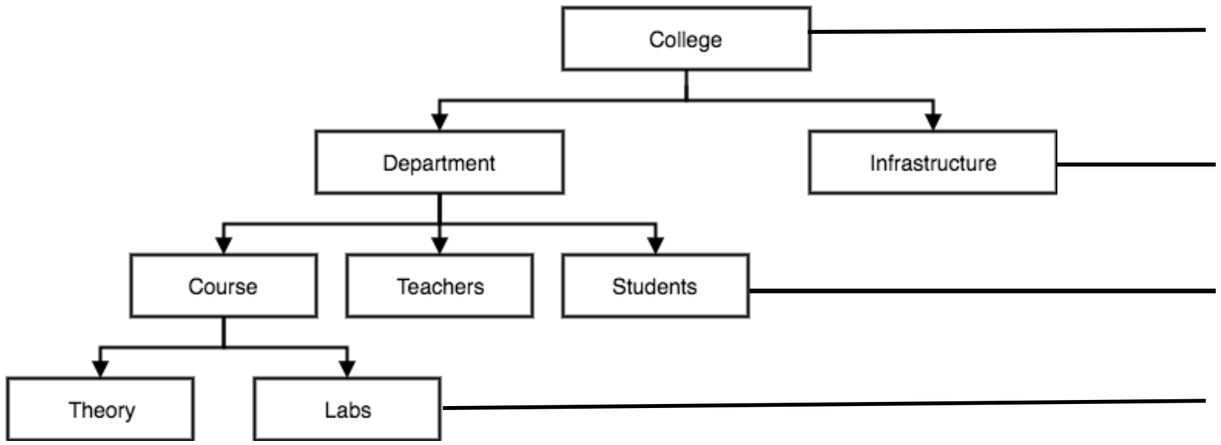
Hierarchical Model

The Hierarchical Model is a data model that **organizes data in a tree-like structure**, where each record has a single parent and possibly many children — forming a one-to-many relationship.

- Data is **organized in a hierarchy of parent–child relationships.**
- Top-level node is called the "**root**", and lower levels are "**children**."
- **A child can have only one parent, but a parent can have multiple children**



Hierarchical Model



- Root segment: College
- Level 1 Segment (Child of Root): Department, Infrastructure
- Level 2 Segment (Child of Level 1): Course, Teachers, Students
- Level 3 Segment (Child of Level 2): Theory, Labs

- **Advantages**:
  - Fast access for well-structured, repetitive data.
  - Easy to navigate if the hierarchy is fixed.

- **Disadvantages**:
  - Difficult to reorganize if the hierarchy changes.
  - Rigid structure":- No support for many-to-many relationships.

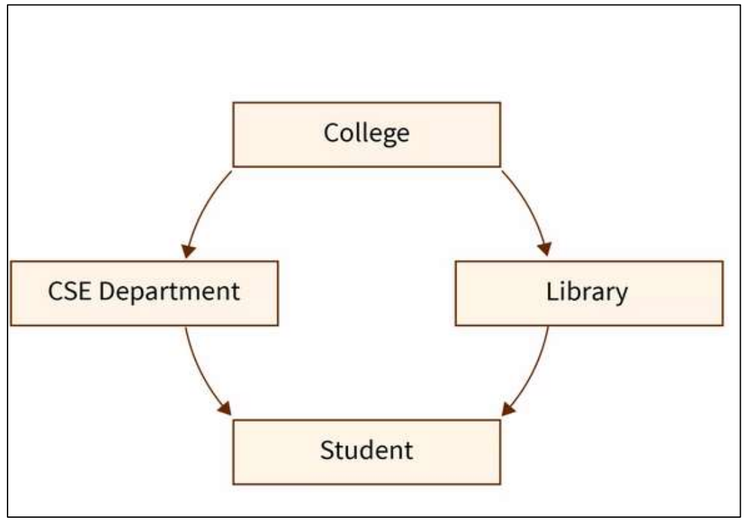
To find a specific lab, you must first identify which course, department, and college it is part of — reflecting the top-down nature of hierarchical databases.

Network Model

A data model where **records are organized using graph structures**, allowing each record (or node) to have multiple parent and child records, enabling many-to-many relationships.

- **Allows a child to have multiple parents (many-to-many relationships)**
- Data is accessed using **pointers or links**
- A user can perceive the network database as a collection of records in one-to-many relationships.

Network Model



- The node 'student' has two parents, the CSE department and the library
- Each student is part of the CSE department as well as the library
- This model can manage one-to-one relationships as well as many-to-many relationships.

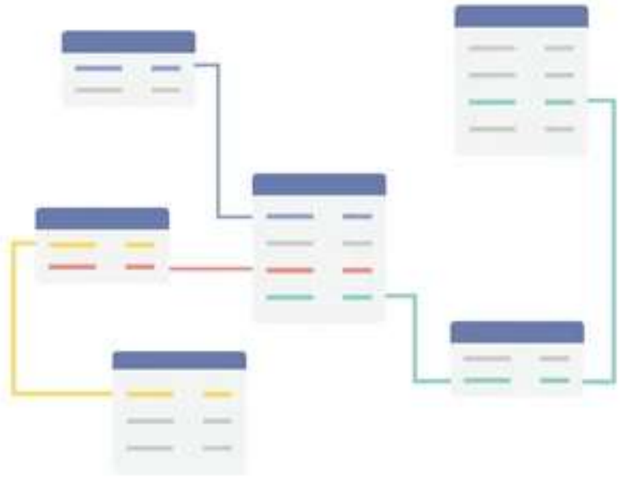
- **Advantages**:
  - Supports **complex relationships (many-to-many, shared children**)
  - More **flexible** than a hierarchical model

- **Disadvantages**:
  - **Complex to design and maintain**
  - Requires users to know access paths (not easy for ad-hoc queries)

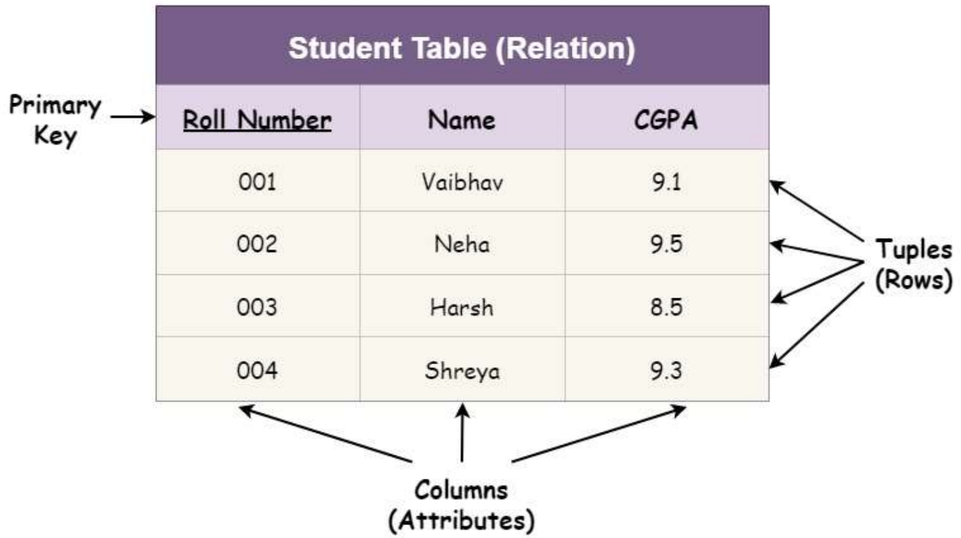
Relational Model

The Relational Model **organizes data into tables (called relations)** consisting of rows and columns. It is the most widely used data model in modern database systems.

- Data is stored in **tables (relations)**
- Each row is called a **tuple (represents a record)**
- Each column is an **attribute (represents a field)**
- Each table has a **primary key and a foreign key (where required)**
- Uses **Structured Query Language (SQL)** for data manipulation
- Based on mathematical principles of set theory and predicate logic



Relational Model



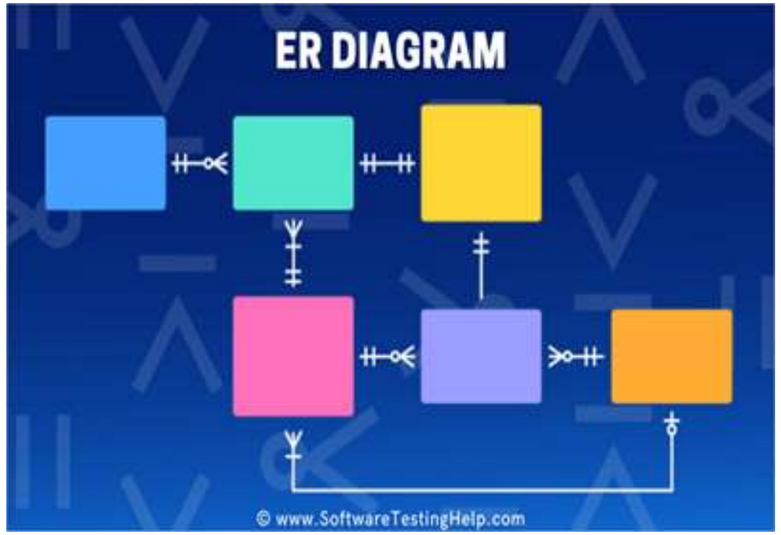
- **Advantages**:
  - Simple and easy to understand
  - **Data independence** — logical and physical separation
  - With SQL, users can access and manipulate data easily
  - Supports **data integrity through constraints and normalization**

- **Disadvantages**:
  - May be **slower for complex, recursive relationships**
  - Join operations can become expensive on very large datasets
  - Not ideal for hierarchical or graph-based data structures

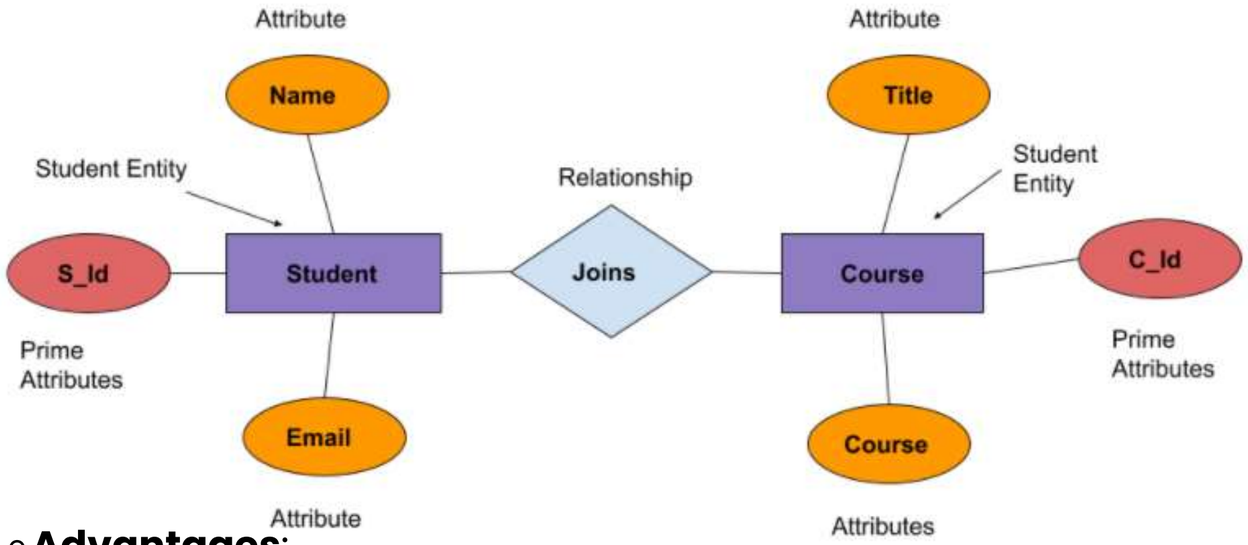
Entity-Relationship Model

The Entity–Relationship (ER) Model is a high-level conceptual data model used to visually describe data and their relationships in a system. It's typically used in database design before building the actual database.

- It is a graphical representation of **entities** and their **relationships** in a database structure.
- Represents data using entities, attributes, and relationships
- **Entities** are real-world objects (e.g., Student, Course)
- **Attributes** describe properties of entities (e.g., Name, ID, Age)
- **Relationships** show how entities are connected (e.g., Enrolls, Teaches)



Entity-Relationship Model



Relational data model and ERM are combined to provide foundation for structured database design.

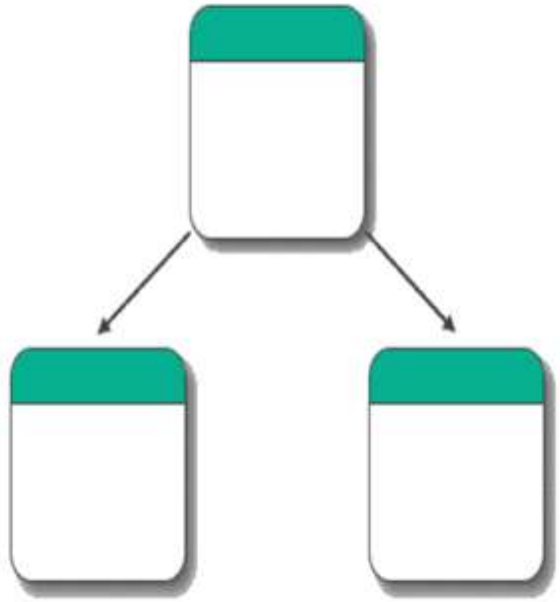
- **Advantages**:
  - Helps in **planning and communication** before actual database creation
  - Provides a clear visual structure of data and relationships
  - **Ensures proper normalization and logical design**
  - Easily convertible to relational schema

- **Disadvantages**:
  - Not used for **actual data storage — only for design**
  - **Doesn't represent physical storage** or performance details
  - May become complex with large systems

Object-Oriented Model

The Object-Oriented Model organizes data as objects, similar to object-oriented programming. Each object contains data (attributes) and methods (operations) that can act on the data.

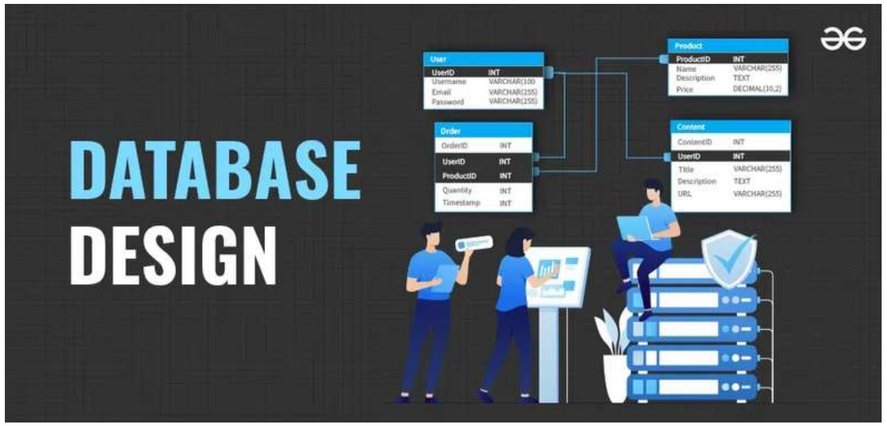
- Uses classes to define object types and create object instances
- Suitable for complex data and applications requiring both data and behavior
- Integrates well with object-oriented programming languages like Java, C++, and Python



Object-Oriented Model

- **Advantages**:
  - Models complex data naturally
  - Easier to represent real-world entities and relationships
- **Disadvantages**:
  - More complex to design and implement than relational models
  - Not widely supported by all database management systems
  - Querying can be less straightforward than SQL

Database Design

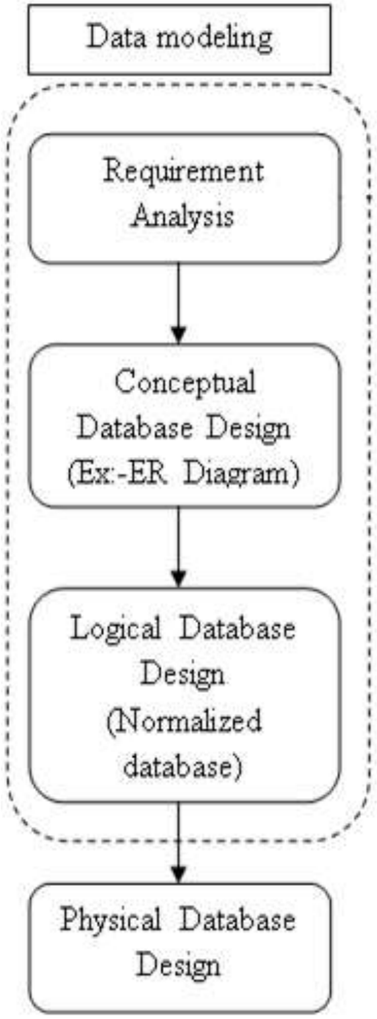


Database Design is the process of creating a detailed data model that defines how data will be stored, organized, and managed in a database system.

- Properly designed databases are easy to maintain, improve data consistency, and are cost-effective in terms of disk storage space.
- It enhances data integrity and security

Database Design Process

The Database Design Process involves a series of steps to create a well-structured database that efficiently stores and manages data, ensuring data integrity and supporting user requirements.



1.Requirement Collection and Analysis

- Understand user needs and gather detailed data requirements via Interviews, Observation, looking at current data
- Identify what data must be stored and how it will be used

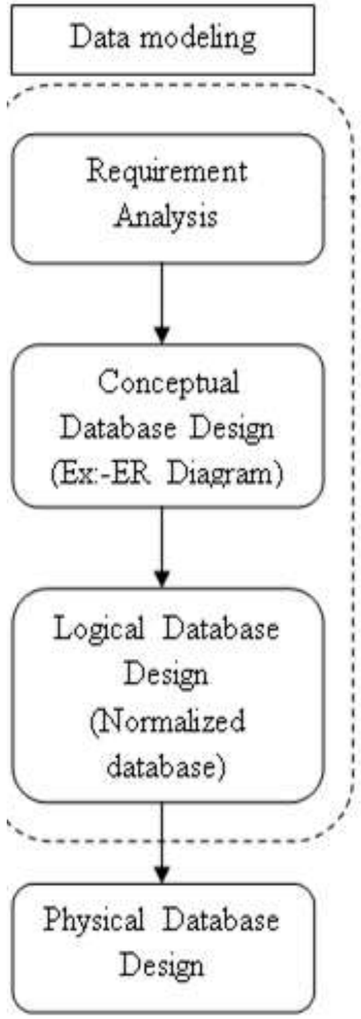
Database Design Process

2. Conceptual database design

- Create a high-level data model, often using the Entity-Relationship (ER) Model
- Define entities, attributes, and relationships without worrying about physical storage
- Defines **WHAT** the system contains through data analysis and requirements gathering

3. Logical database design(Data model mapping)

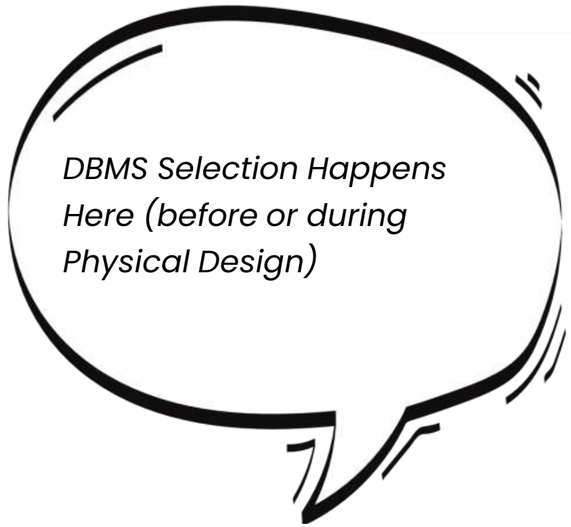
- Transform the conceptual design into a logical schema suitable for a specific database model (usually the Relational Model)
- Define tables, keys, constraints, and normalization to reduce redundancy
- Defines **HOW** the system should be implemented regardless of the DBMS.



Database Design Process

4. Physical Design

- Decide how the logical schema will be physically implemented on storage devices
- Optimize storage, indexing, and access paths for performance
- Describes **Where** the system will be implemented using a specific DBMS system



In summary

- **Requirement Analysis**:- Understand **what users need.**
- **Conceptual Design**:- Design using high-level models like **ER diagrams**.
- **Logical Design**:- Convert to **relational schema, define tables, keys**, etc.
- **Physical Design**:- Decide on i**ndexes, storage, and performance** based on the chosen DBMS.

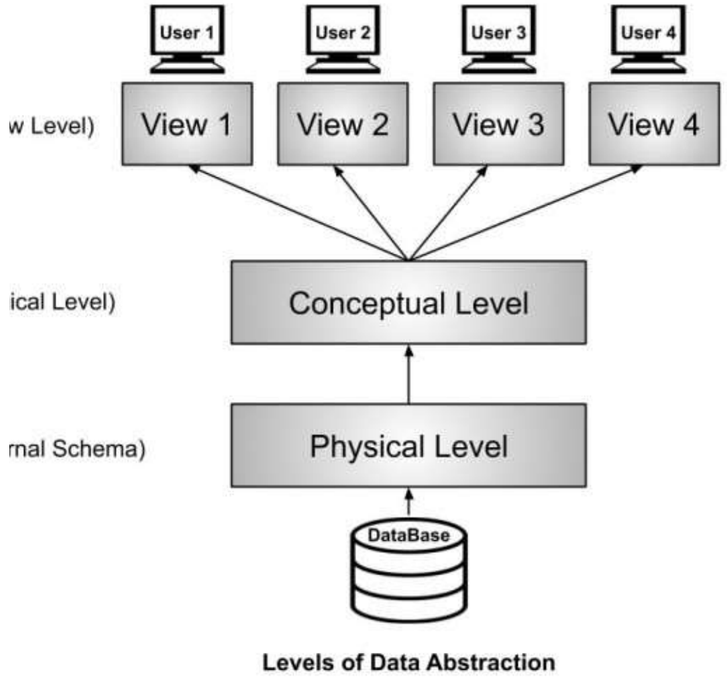
Database Design Process

In summary

- **Requirement Analysis**:- Understand **what users need.**
- **Conceptual Design**:- Design using high-level models like **ER diagrams**.
- **Logical Design**:- Convert to **relational schema, define tables, keys**, etc.
- **Physical Design**:- Decide on i**ndexes, storage, and performance** based on the chosen DBMS.

Levels of Abstraction

- The levels of abstraction in database systems define how data is viewed and interacted with at different layers — from how it's physically stored to how users see it. This helps in hiding complexity and separating concerns during database design and usage.
- **Abstraction** hide irrelevant details from the users and approach simplifies **database design**.



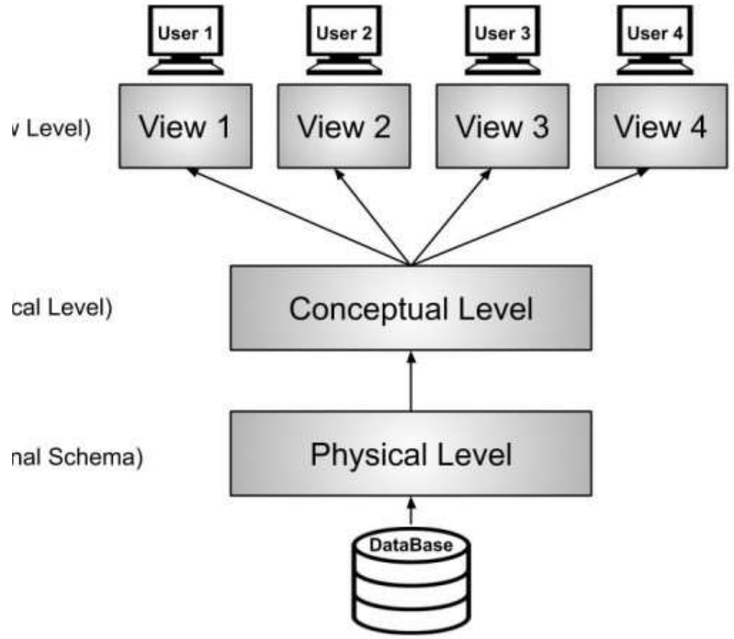
Levels of Data Abstraction

Think of a Car

Level	Description
View Level	Driver uses steering wheel, pedals (simple controls)
Logical Level	Car system understands engine, brakes, sensors
Physical Level	Actual mechanical parts, wires, gears

Three levels of Abstraction

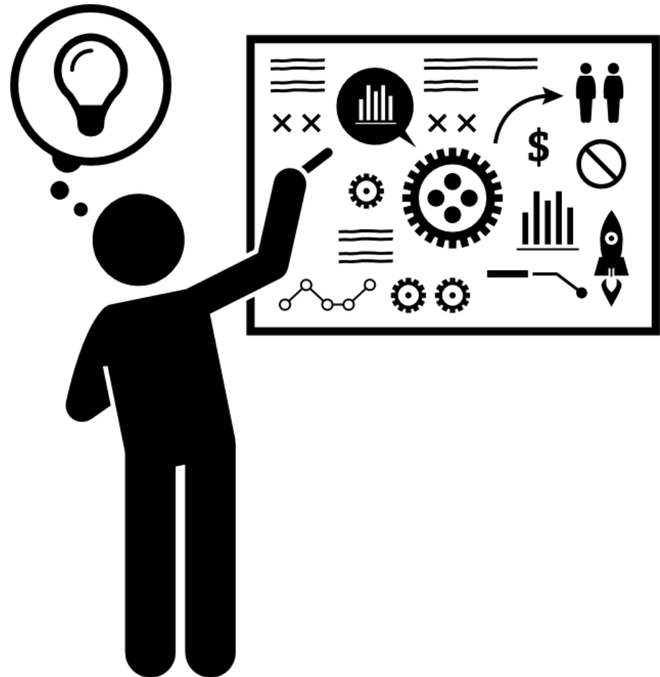
- **View Level (Highest Level)**
  - Describes **how users interact with the database**
  - Shows only relevant parts of the data to each user (via views)
  - Enhances security and simplicity
  - **Hides unnecessary complexity** from end users
- **Logical Level (Middle Level)**
  - Describes **what data is stored** and the relationships among the data
  - Defines tables, fields, data types, relationships, and constraints
  - **Independent of physical** details
- **Physical Level (Lowest Level)**
  - Describes **how data is actually stored** on hardware (e.g., files, indexes)
  - Deals with internal storage structures
  - Concerned with efficiency, performance, and space



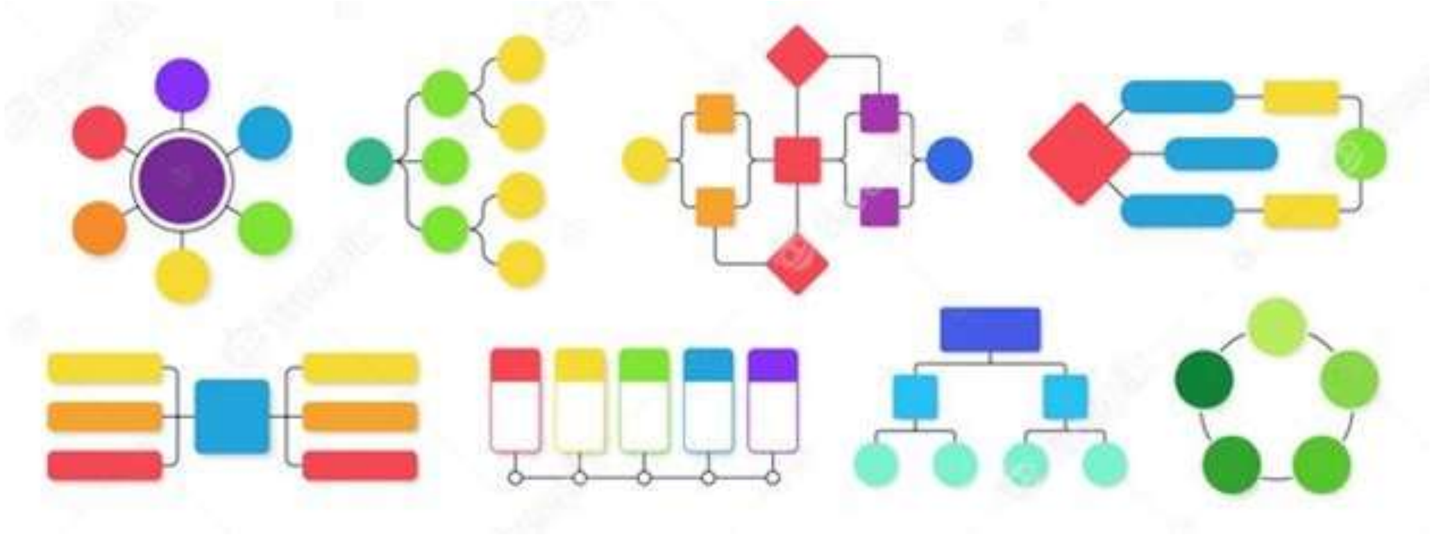
Levels of Data Abstraction

Why Conceptual Design

- Helps visualize the structure of data without worrying about technical details
- Allows clear communication between database designers and stakeholders
- Identifies entities, relationships, and constraints early in the design phase
- Reduces the risk of design errors or missing requirements later
- Serves as a blueprint for logical and physical database design



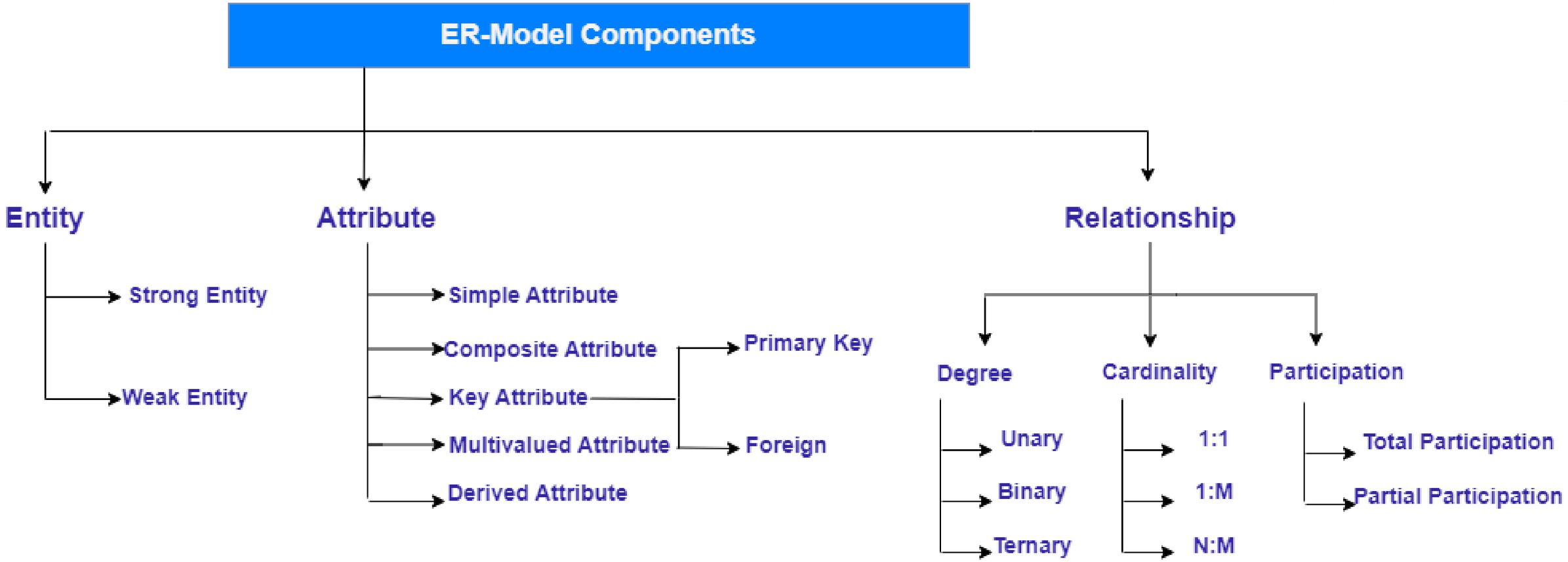
Entity-Relationship Diagram



An Entity–Relationship Diagram (ERD) is a visual representation of **entities**, their **attributes**, and the **relationships** among them within a particular system. It is used during the **conceptual design** phase to model real-world data logically.

Why entity-relationship diagram?

- It is the **visual** starting point for **database design.**
- it is influential in determining **the flow of data and how the complete system works**.
- ER Model can easily be **transformed into Relational Tables.**
- It is simple and easy to understand. The model can be used by Database Designer to communicate design to end user.
- Model can be used as design plan by Database Developer to implement a Data Model in specific DBMS.


ER-Model Components
Entity
Strong Entity
Weak Entity
Attribute
Simple Attribute
Composite Attribute
Key Attribute
Primary Key
Foreign
Multivalued Attribute
Derived Attribute
Relationship
Degree
Unary
Binary
Ternary
Cardinality
1:1
1:M
N:M
Participation
Total Participation
Partial Participation

Entity

An entity is a **real-world object or concept** that can be uniquely identified and about which data is stored in a database. For example:-Customer, Customer Order, Product, Employee, Project, Department, Order

Customer

Laptop

- Name of each entity is in singular form
  - Noun
  - Adjective + Noun
  - Noun + Noun => (noun string)
  - Adjective + Noun + Noun

Employee

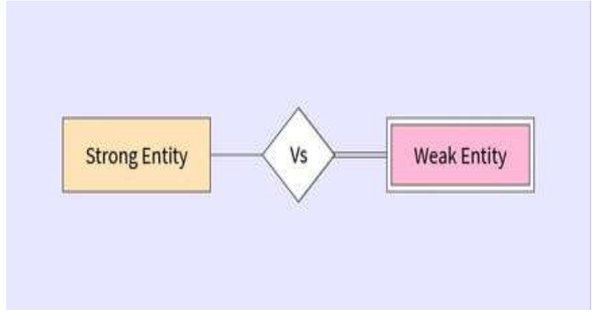
Order

Book

Product

Entity

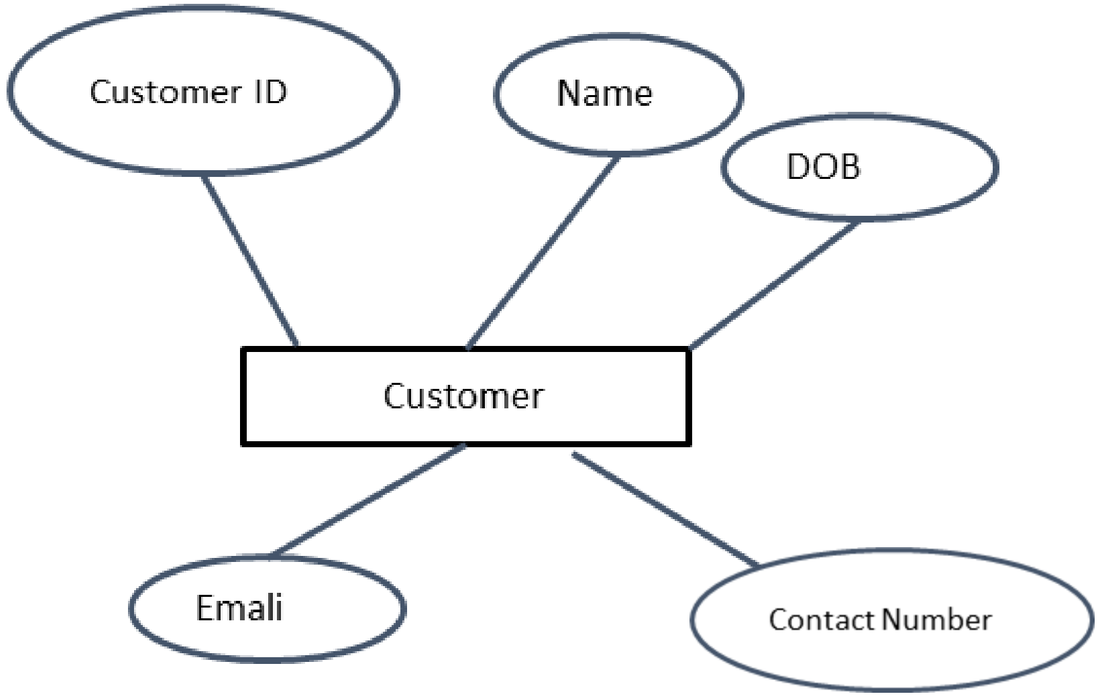
- It is represented by a **rectangular box** in an ER diagram
- Entities have **attributes** (properties or characteristics)
- Each entity must have a **key attribute** (uniquely identifies each instance)
- **Types of Entities:**
  - ➤ **By Existence**
    - **Strong Entity**: Exists independently with a unique key
    - **Weak Entity**: Depends on another entity and has no unique key of its own
  - ➤ **By Nature**
    - **Tangible Entity**: Something physical that you can touch. Example: Student, Book, Car
    - **Intangible Entity**: Something conceptual or abstract. Example: Course, Reservation, Contract



Attribute

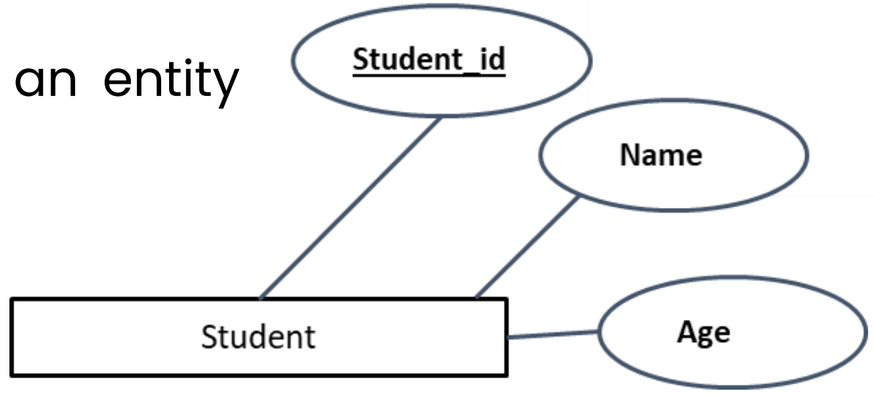
Attributes are the **properties** or **characteristics** of an entity. It is represented by llipses (ovals) connected to their entity or relationship

- Customer
  - Customer ID
  - Name
  - DOB
  - Contact Number
  - Email

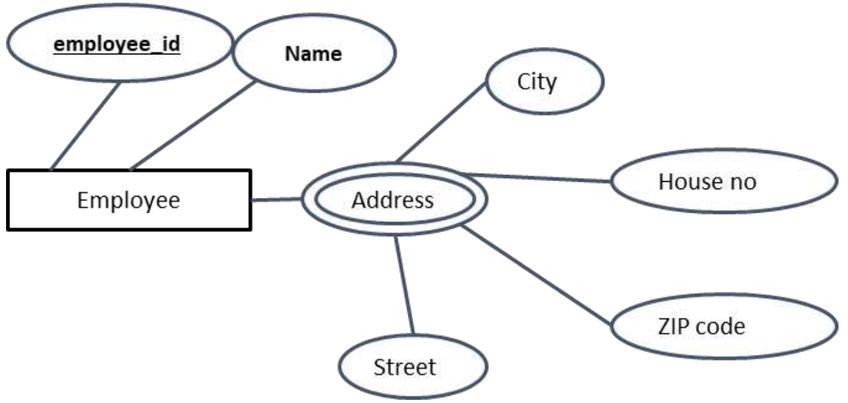


Types of Attribute

- Key attributes:-Can uniquely identify an entity from an entity set

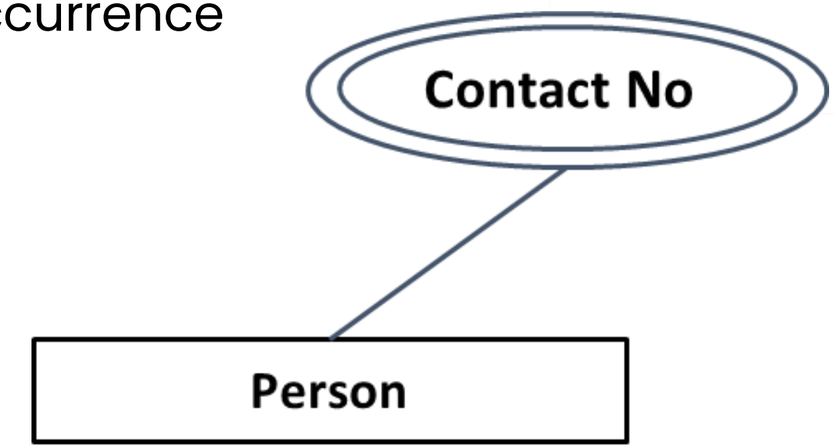


- Composite attributes:-Composed of multiple components, each with an independent existence

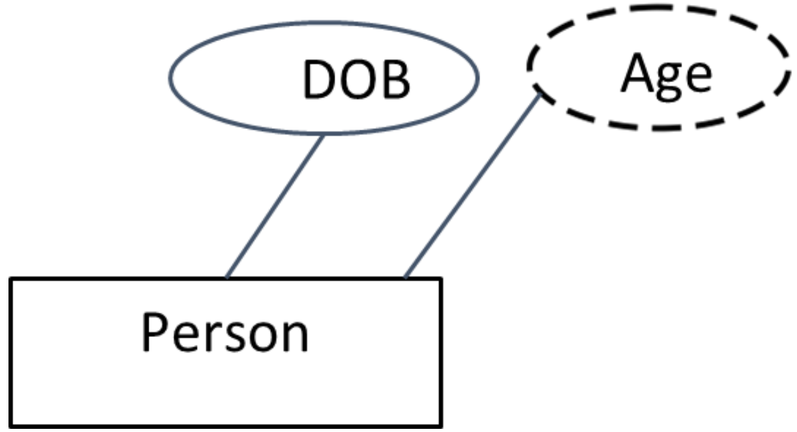


Types of Attribute

- Multi-value attributes:-Hold multiple values for each occurrence in an entity type.



- Derived attributes:-Value can be calculated or derived from other attributes



Relationship

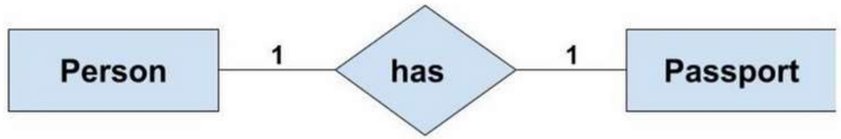
A relationship in ERD shows how two or more entities are connected or associated with each other in a database system.

- Represents interaction or association between entities
- Shown using a diamond (♦) shape in ER diagrams
- Connected to entities via lines

Customer places an **Order.**

Types of Relationship

- **One-to-One (1:1)**
  - **one instance of Entity A** is associated with **one instance of Entity B** and vice versa.
  - Each person has one passport



- **One-to-many(1:M)**
  - **one instance of Entity A**, there are **zero, one, or many instances of Entity B**, but for **one instance of entity B**, there is **only one instance of entity A.**
  - A customer can place many orders but an order has been placed by only one customer.

Types of Relationship

- **Many-to-Many(M:N)**
  - Many-to-Many(M:N) relationship, sometimes called non-specific, is when for **one instance of Entity A**, there are **zero, one, or many instances of Entity B** and for **one instance of Entity B,** there are **zero, one, or many instances of Entity A**.
  - Many-to-Many relationships cannot be directly translated to relational tables but instead must be transformed into two or more one-to-many relationships using associative entities.

- **Employees** can be **assigned to more than two projects** at the same time**; Projects** can have **multiple employees working on them.**

ITAHARI INTERNATIONAL COLLEGE

Islington college (इस्लिङ्टन कालेज)

informatics college • pokhara

Thank you

End of Lecture

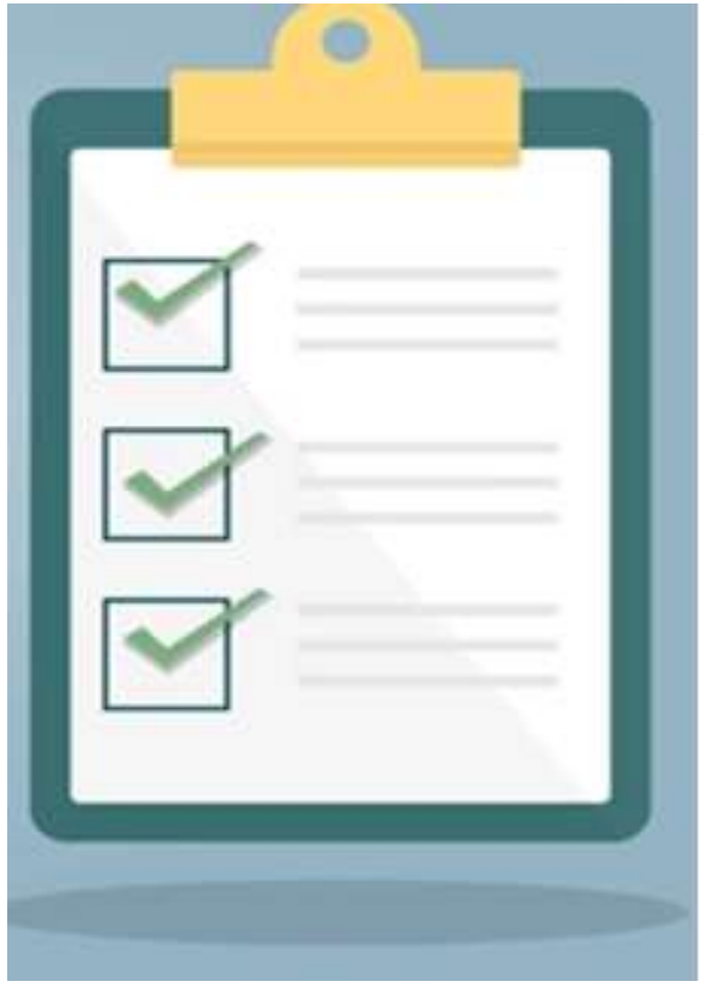
CC5051 - DATABASES

Conceptual Modelling Concepts & Their Representation

WEEK 5

Agendas

- Degree of Relationship
- Connectivity and Cardinality
- Relationship Existence
- Problems with ER Models
- Generalization Hierarchy
- ER Model - Example



Relationship

- Relationship is a set of associations among Entities.
- It indicates that there is a business relationship between these Entity.
- Examples:
  - Customer places an Order
  - Order is placed by Customer
  - Employee works on Project
  - Project has project member Employee

Relationship

- In entity-relationship diagrams, relationships are used to document the connection between entities.

Relationship

- The relationships describe how these entities will interact with each other, if at all.
- Drawing a line in between them shows relationship

- Customer places Order.

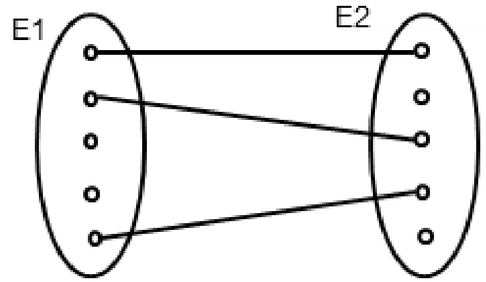
- Order is placed by Customer.

Connectivity and Cardinality

- Connectivity of a relationship describes mapping of associated Entity instances in Relationship.
- Values of connectivity are One or Many
- Cardinality of a Relationship is actual number of related occurrences for each of Two Entities.
- Basic types of connectivity for relations are: One-to-One, One-to-Many and Many-to-Many.

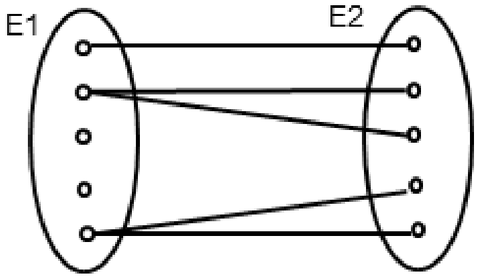
One-to-One

1:1



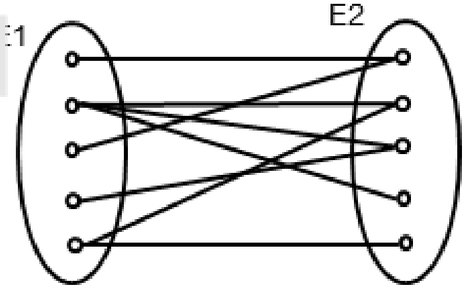
One-to-Many

1:M



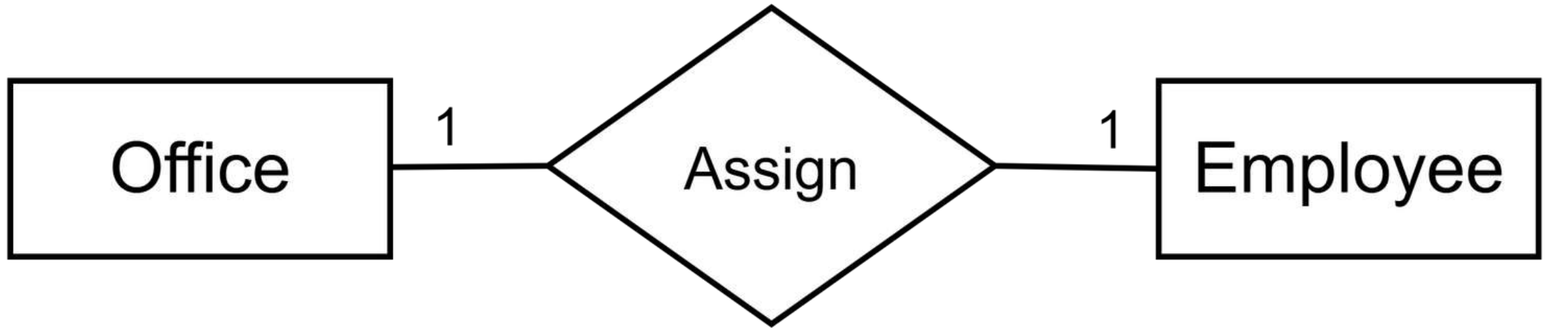
Many-to-Many

M:N



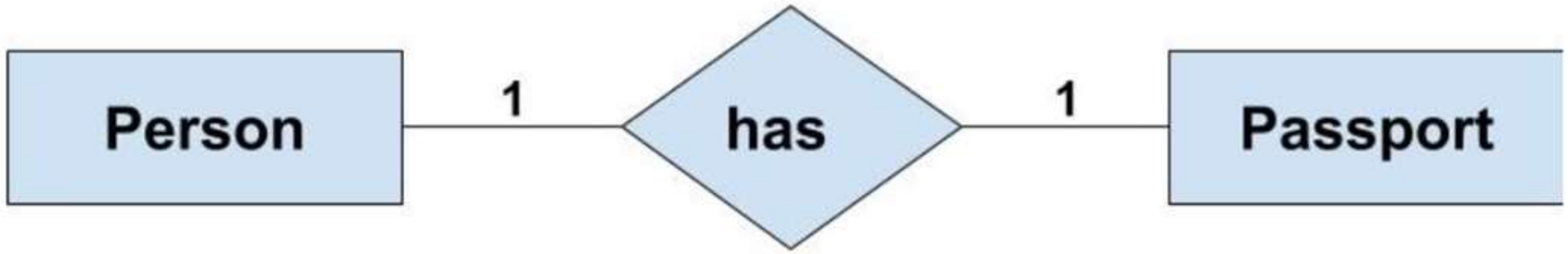
One-to-One

- **One-to-One(1:1) relationship** is when at most one instance of a **Entity A is associated with one instance of Entity B**.
- Example:
  - Employees in the company are each assigned their own office. For each employee there exists a unique office and for each office there exists a unique employee.



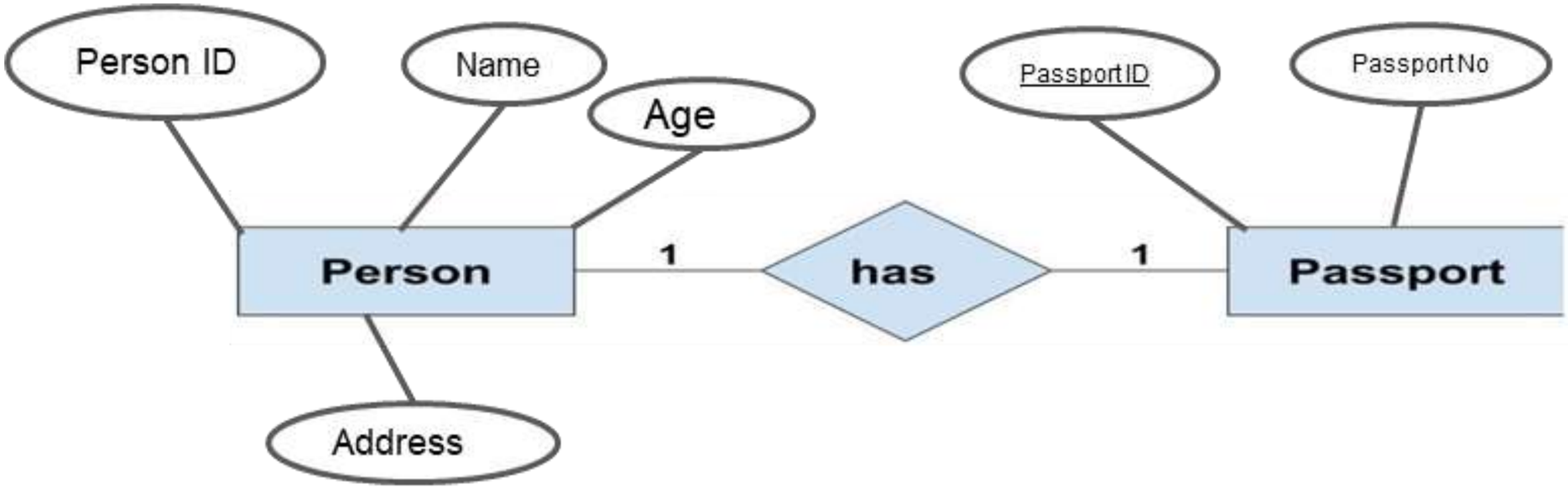
One-To-One Relationship

- How many passports can a person have?
  - Answer: One
- How many persons can a passport belong to?
  - Answer: One



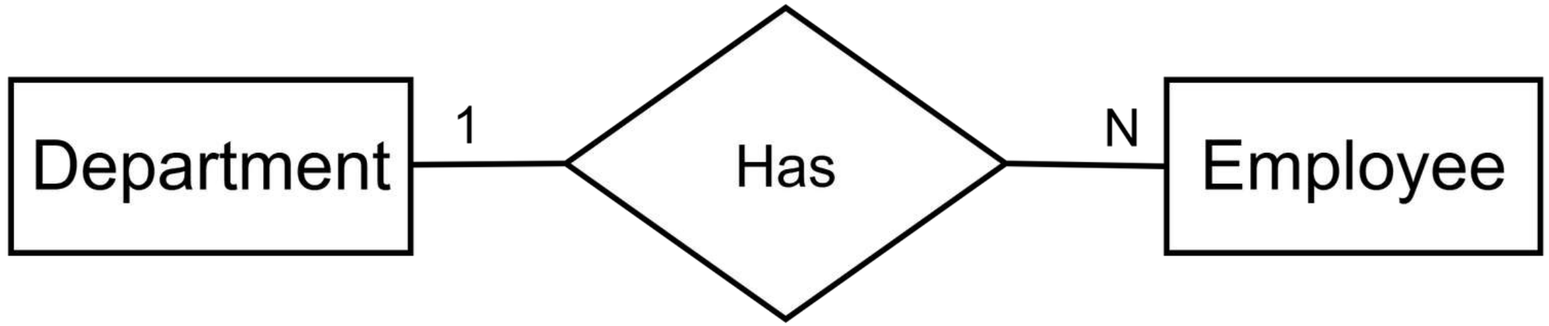
One-To-One Relationship

- One-to-one relationship exists when one instance of entity A is associated with one instance of entity B and vice versa.



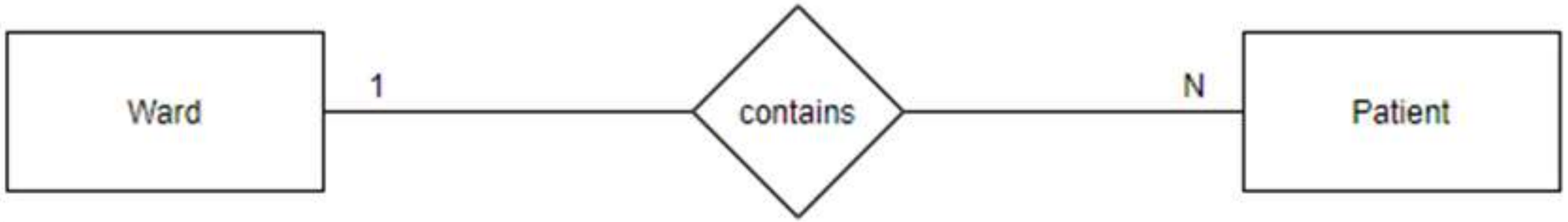
One-To-Many

- **One-to-Many(1:M) relationships** is when for one instance of Entity A, there are zero, one, or many instances of Entity B, but for one instance of entity B, there is only one instance of entity A.
- Example
  - A department has many employees
  - Each employee is assigned to one department

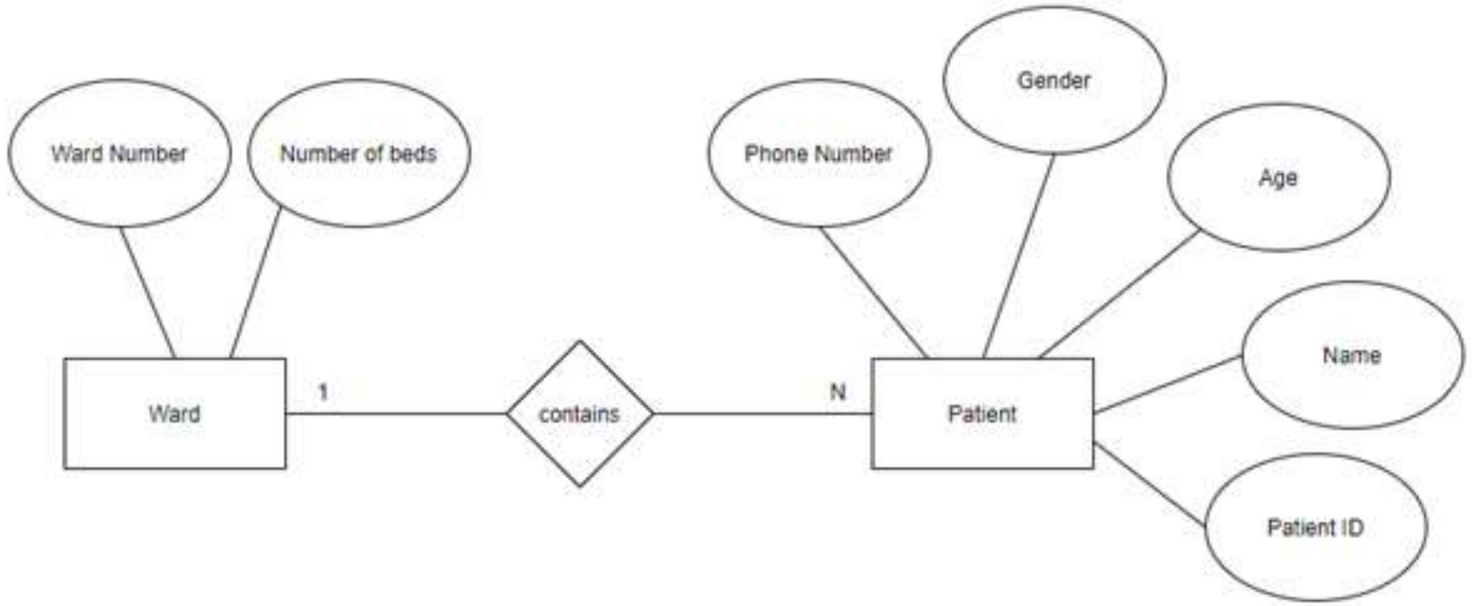


One-To-Many Relationship

- How many patients can be there in a ward?
  - Answer: Many
- How many wards are assigned to a patient?
  - Answer: One



One-To-Many Relationship



- One-to-many relationships exist when one instance of entity A is associated with many instances of entity B, but for one instance of entity B, there is only one instance of entity A.

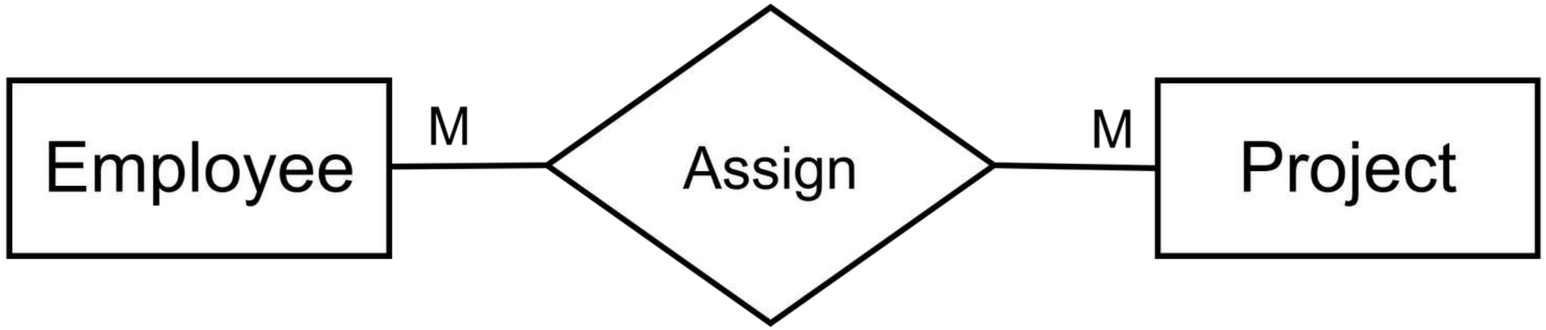
Many-To-Many

- Many-to-Many(M:N) relationship, sometimes called non-specific, is when for **one instance of Entity A, there are zero, one, or many instances of Entity B and for one instance of Entity B there are zero, one, or many instances of Entity A**.
- Many-to-Many relationships cannot be directly translated to relational tables but instead must be transformed into two or more one-to-many relationships using associative entities.

Many-To-Many

Example:

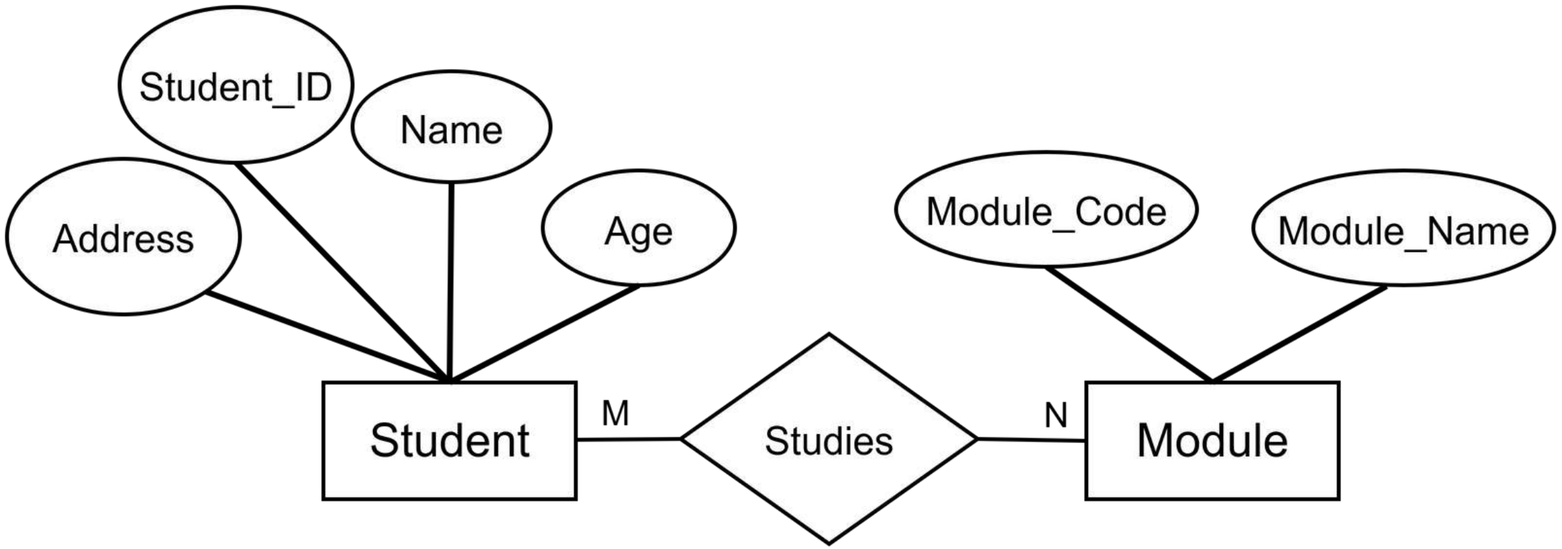
- Employees can be assigned to more than two projects at the same time;
- Projects must have assigned at least three employees
- A single employee can be assigned to many projects; conversely, a single project can have assigned to it many employees. Here the cardinality for the relationship between employees and projects is two and the cardinality between project and employee is three.



Many-To-Many Relationship

- How many module does a student study?
  - Answer: Many
- How many students may study a module?
  - Answer: Many

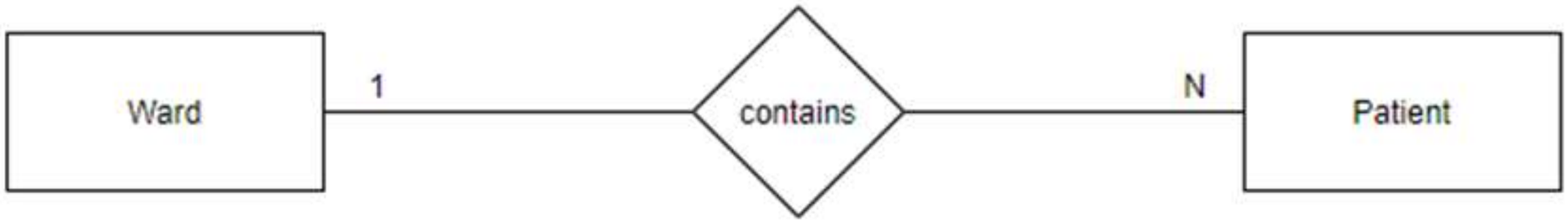
Many-To-Many Relationship



- Many-to-many relationships exist when one instance of entity A is associated with many instances of entity B and for one instance of entity B, there are many instances of entity A.

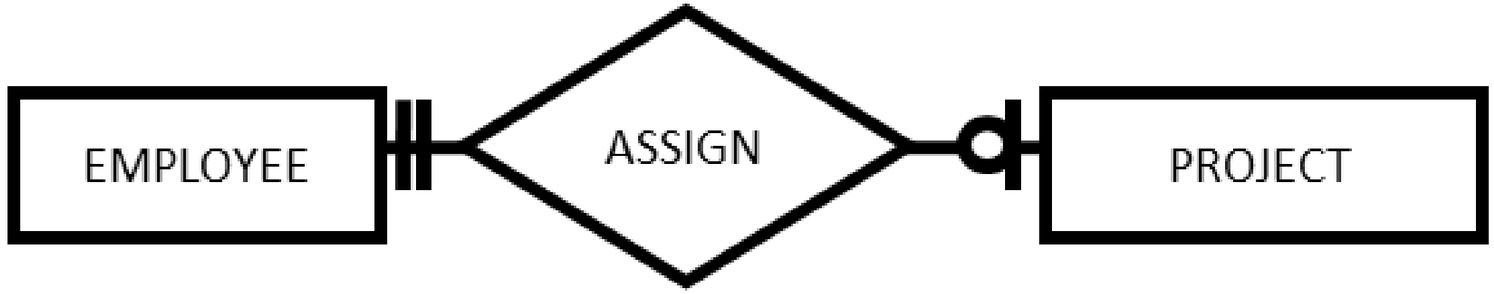
Relationship Existence

- Relationship existence describes whether an entity in a relationship is optional or mandatory.
- If an instance of an entity must always occur for an entity to be included in a relationship, then it is mandatory.

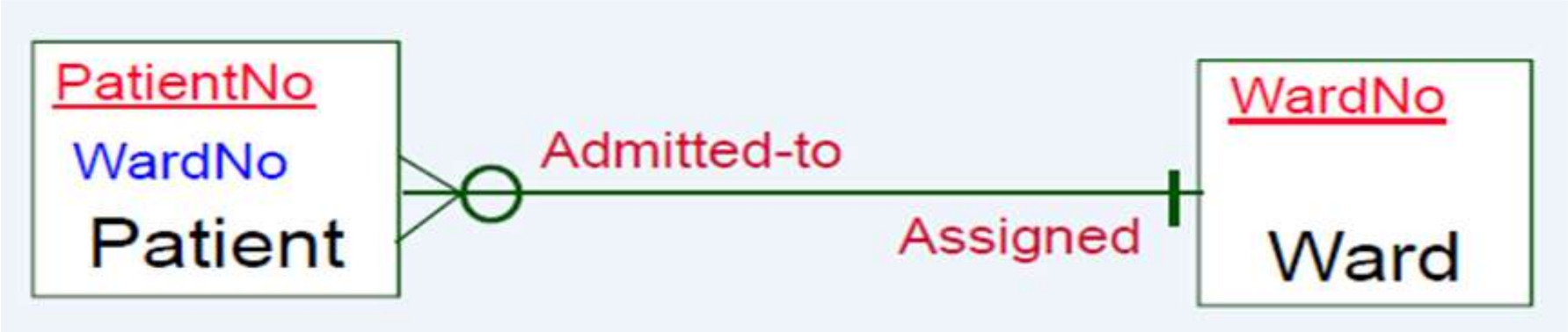


Relationship Existence

- If the instance of the entity is not required, it is Optional.
- An example of Optional existence is the statement, "**employees may be assigned to work on project**".



Identity Relationships



- **Integrity Rules – derived from ER Diagram:**
  - Each patient must be admitted to a single ward
  - Each ward may be assigned one or more patients

Create a Relational Schema

- Ward (WardNo, WName, WType)
- Patient (PatientNo, PName, PType, DOB, Balance, WardNo)

Ward	
WardNo	Varchar2(2)
Wname	Varchar2(12)
WType	Varchar2(15)

Patient	
PatientNo	Number(4)
Pname	Varchar2(10)
Ptype	Varchar2(5)
DOB	Date
Balance	Number(5)
WardNo	Varchar2(2)

Sample Relations

Patient

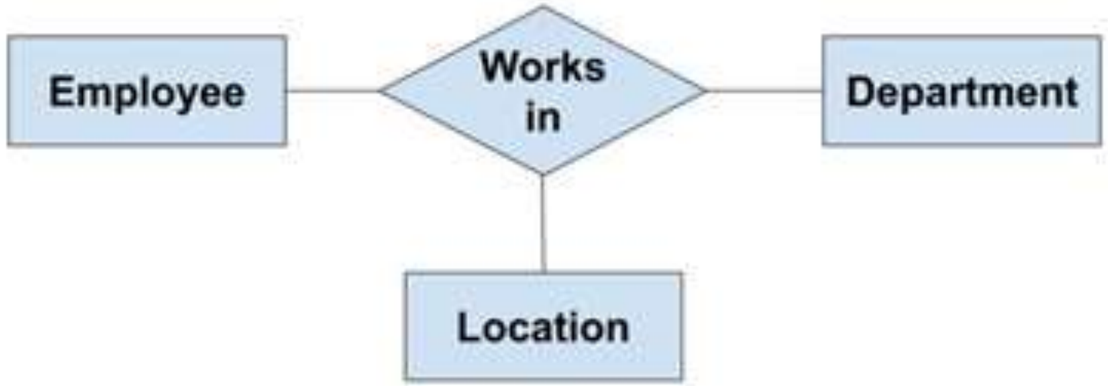
PatientNo	PName	PType	DOB	Balance	WardNo
100	Rajendra	Out	12-MAY-86	1000	W1
101	Rohan	Out	30-APR-92	1200	W1
102	Govinda	In	07-JUL-80	200	W3
104	Aman	In	14-JUN-95	450	W4
105	Birendra	Out	25-SEP-92	1200	W2

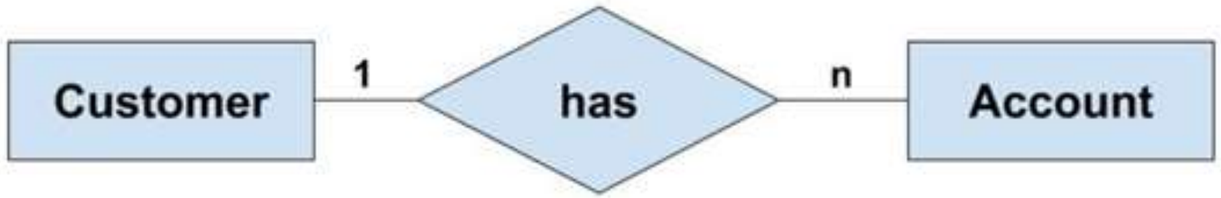
Ward

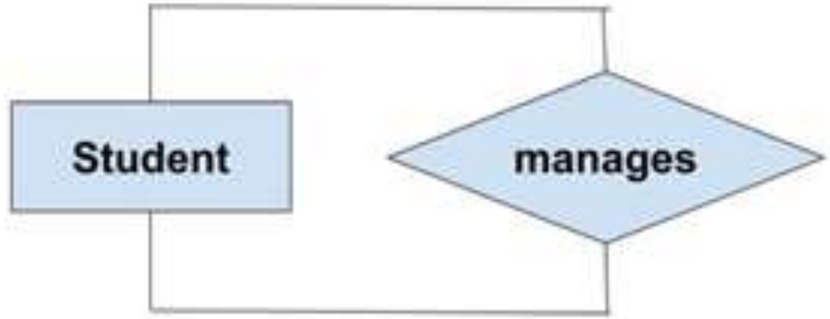
WardNo	WName	Wtype
W1	Tej Narayan	Outpatient
W2	Til Ganga	Optics
W3	ICU	Intensive Care
W4	Makalu	Renal

Degree of relationship

- **Degree of relationship is the number of entities linked with one relationship.**
- Most common relationship degrees are
  - Unary relationship
  - Binary relationship
  - Ternary relationship

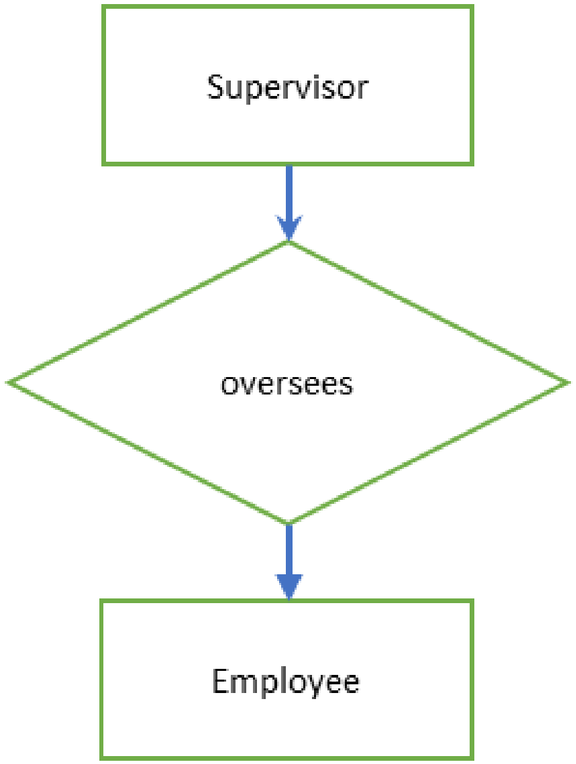




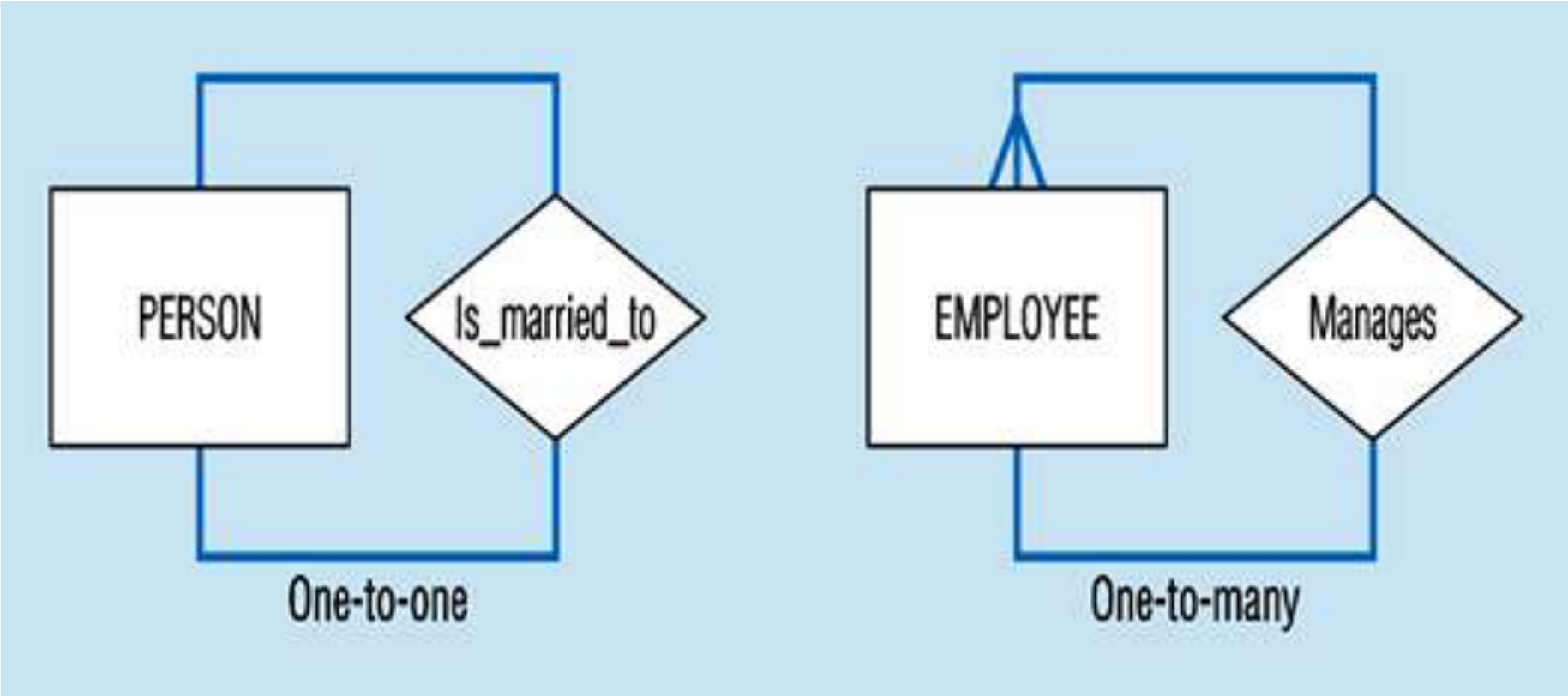


Unary Relationship

- Unary Relationship **occurs when an Entity is related to itself**. It is degree of one relationship
- Relationship between the instances of a single Entity.
- They are also called “recursive relationships”
- Examples:
  - Employee manages employees
  - Some employees are married to other employees.

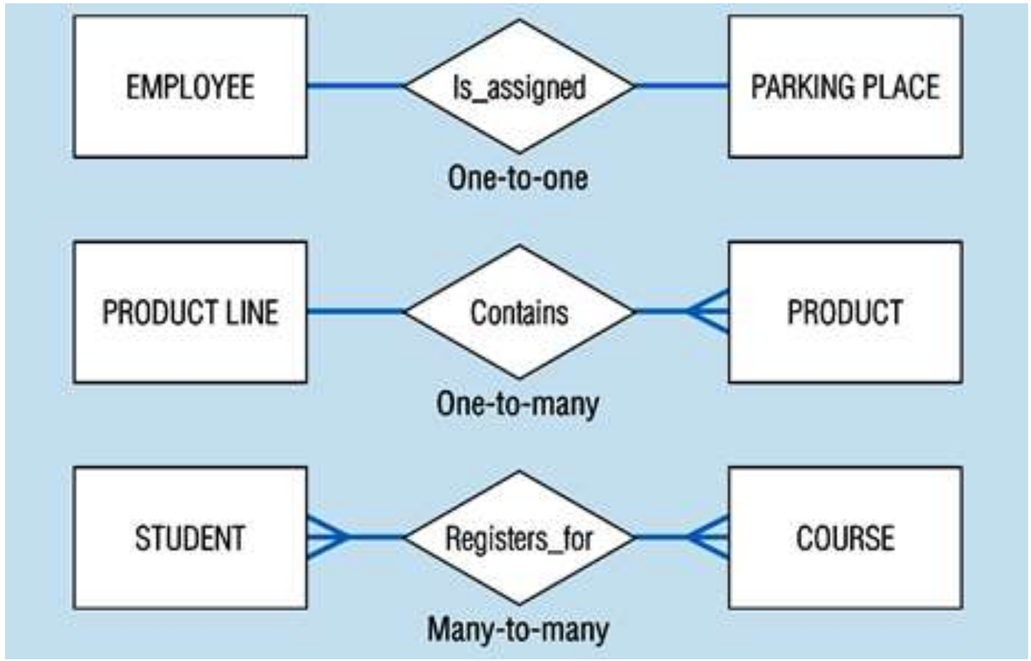


Unary Relationship



Binary Relationship

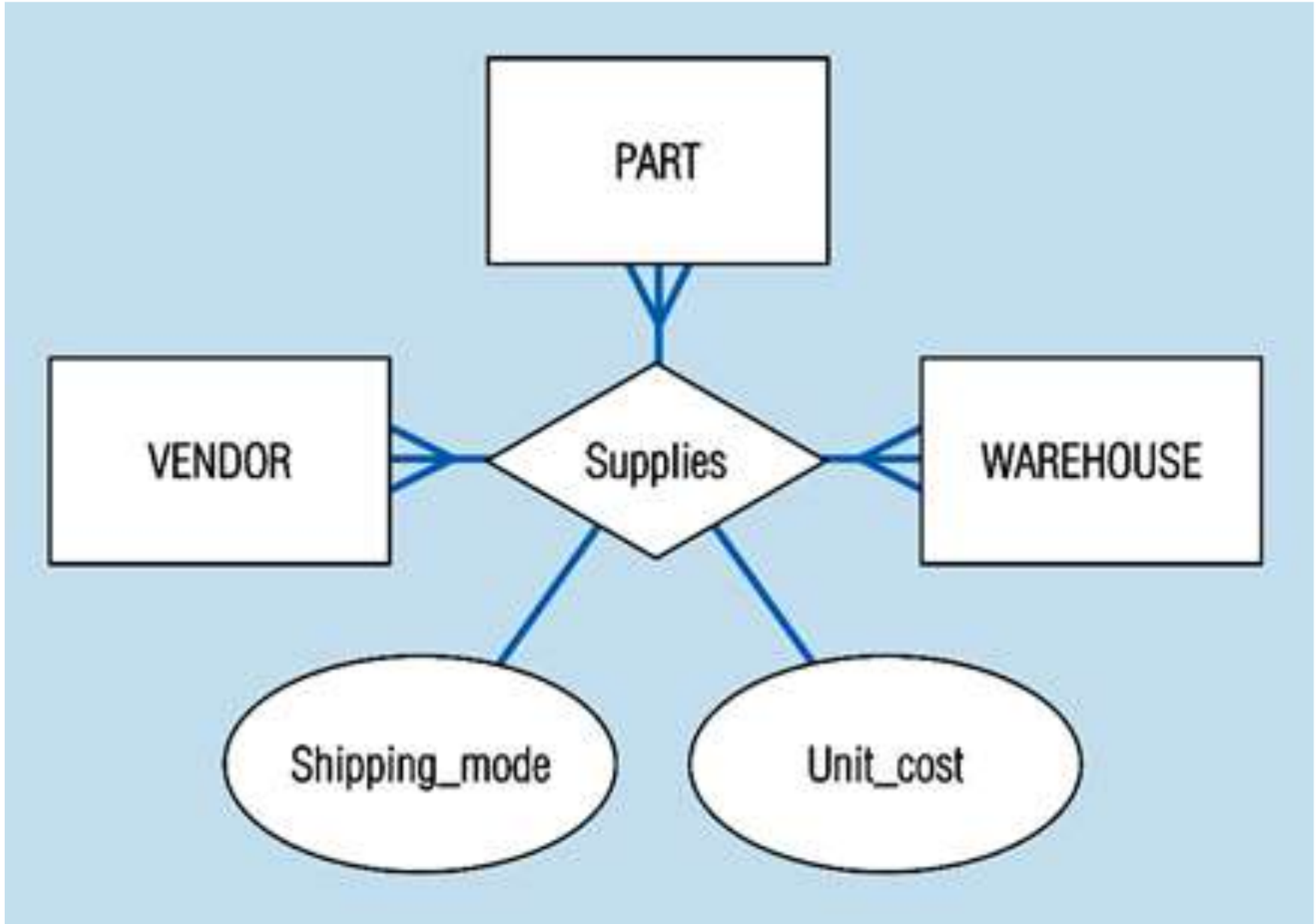
- Relationship between instances of two Entities
- Most commonly encountered
- Examples:
  - **One to one** - Warden takes care of one Hostel
  - **One to Many** - Hostel contains many Rooms
  - **Many to Many** - Students graduating in many Degrees



Ternary Relationship

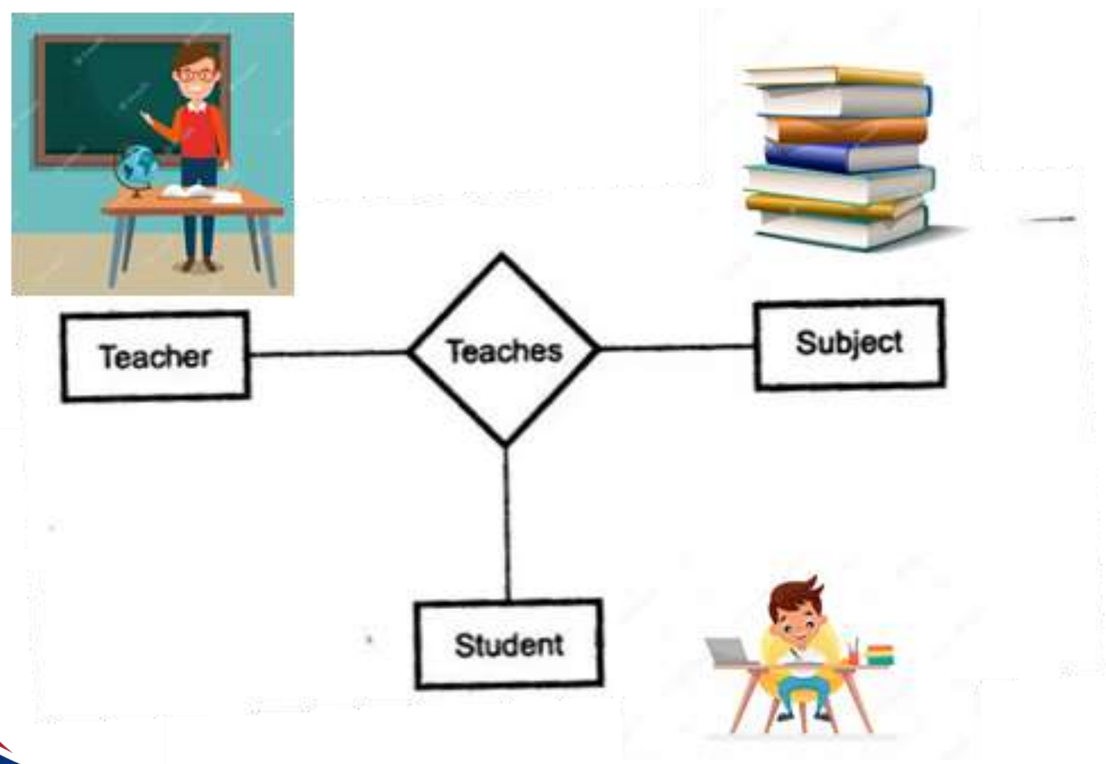
- Ternary Relationship **involves Three Entities and is used when a binary relationship is inadequate**.
- A simultaneous relationship among the instances of three Entities
- Ternary Relationships is not same as three Binary Relationships
- Examples:
  - Course or Section, Instructor, Classroom---> Timetable
- All ternary relationships are represented as associative entities
- Ternary or N-nary Relationships are decomposed into two or more binary relationships.

Ternary Relationship

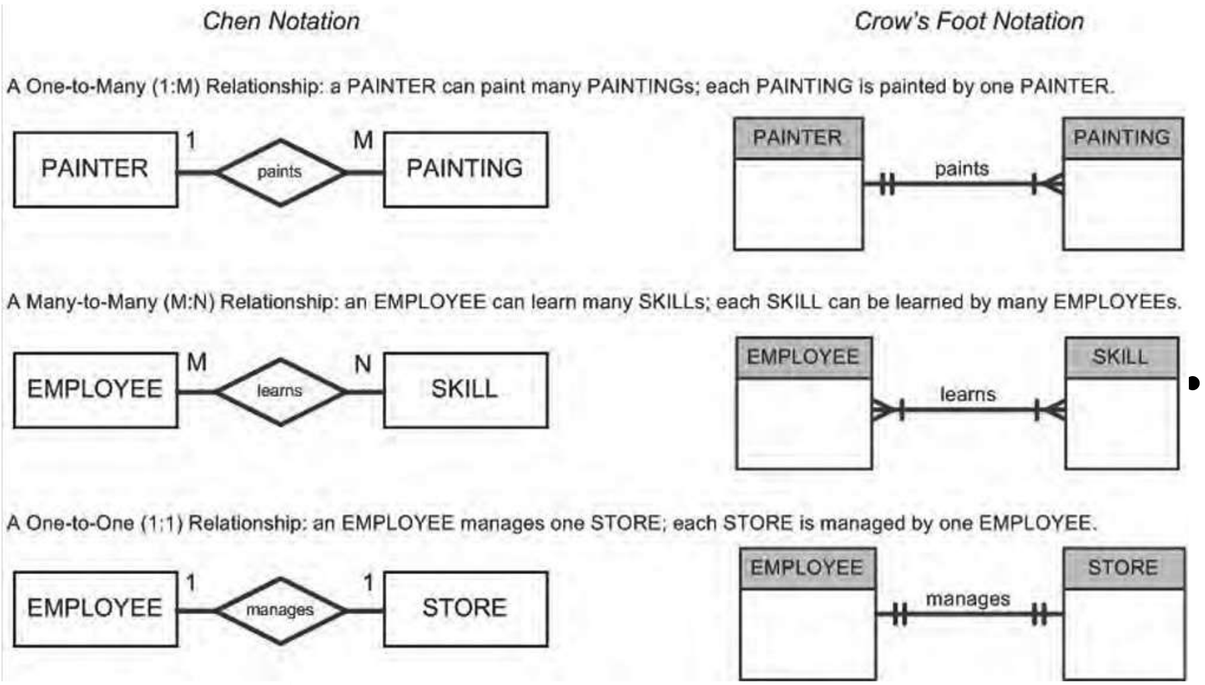


Ternary Relationship - Example

- Relationship between teacher, subject and student.
- Relationship between employee, department and location.

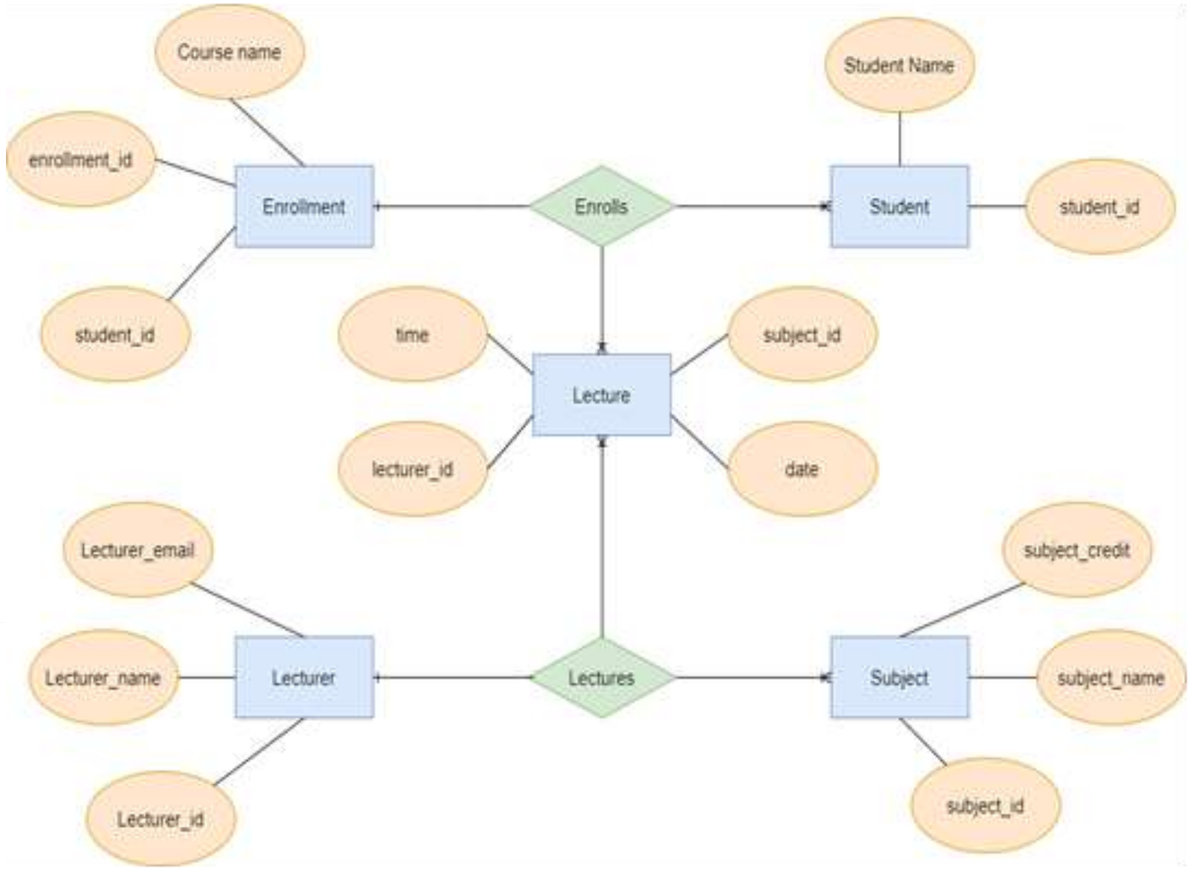


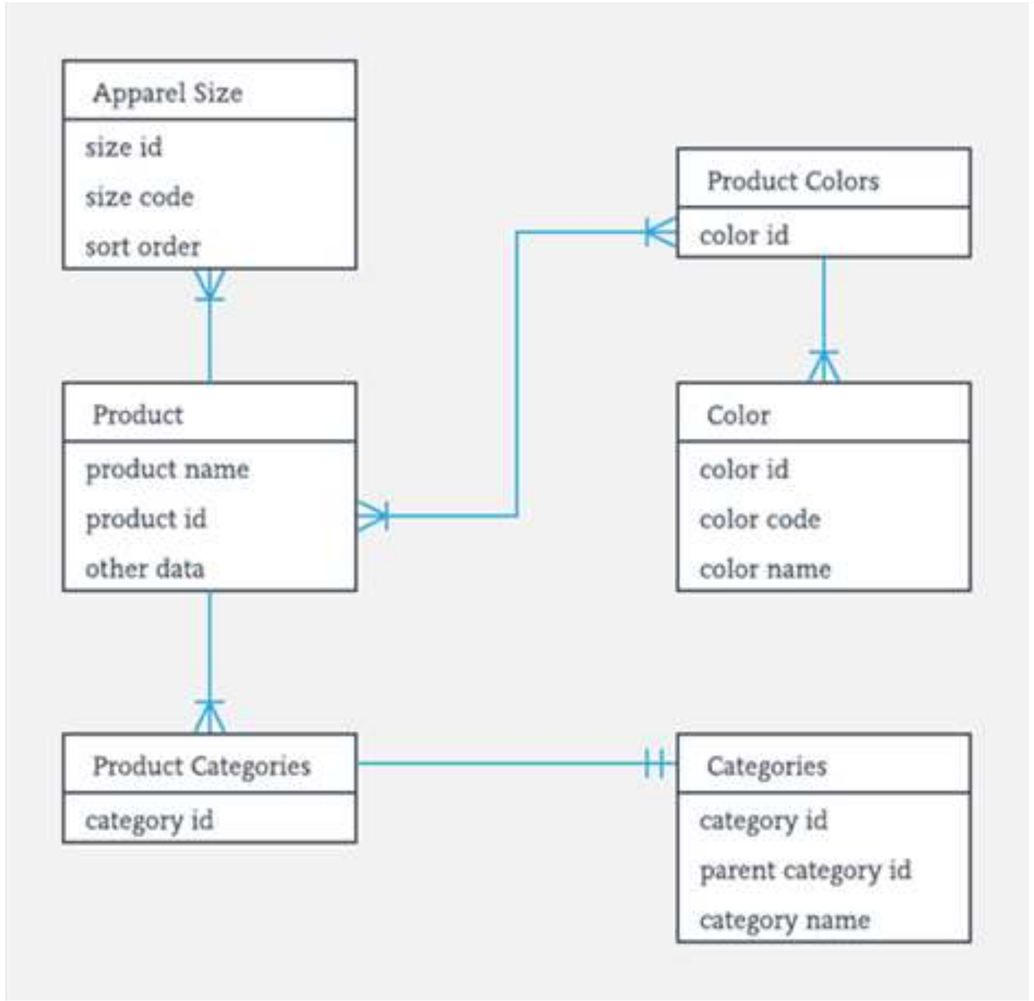
Entity Relationship Model or ER Model



- Peter Chen first introduced ER Data Model in 1976 and is Graphical representation of entities and their relationships in database structure.
- Crow's foot notation is used in Barker's Notation, and represent entities as boxes, and relationships as lines between the boxes.
- Crow's foot notation is used in table diagrams (physical model), Chen's notation is used in ER diagrams (conceptual model).

ER Model Example



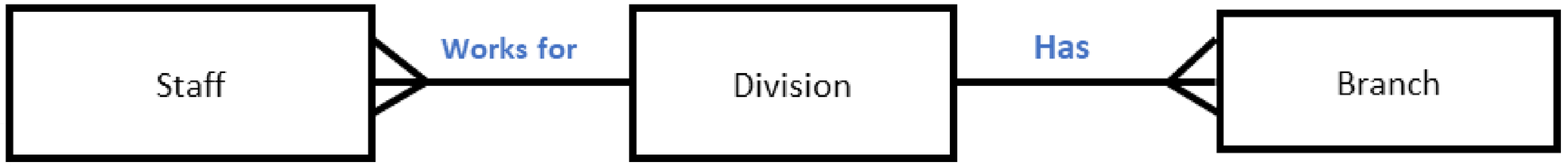


Problems With ER Models

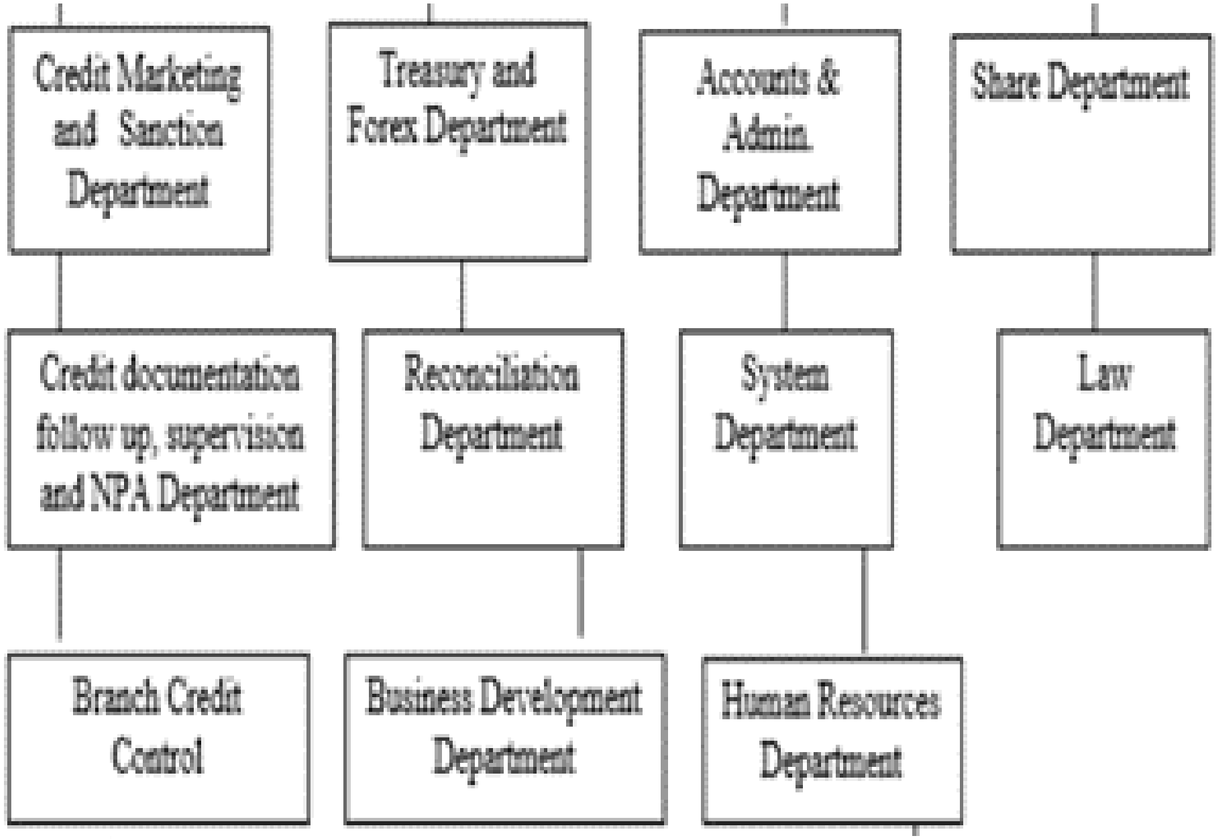
- Problems may arise when creating an ER model.
- Problems are referred to as Connection Traps
- Normally occur due to a misinterpretation of the meaning of certain relationships (Howe, 1989).
  - Fan Traps
  - Chasm Traps

Fan Traps

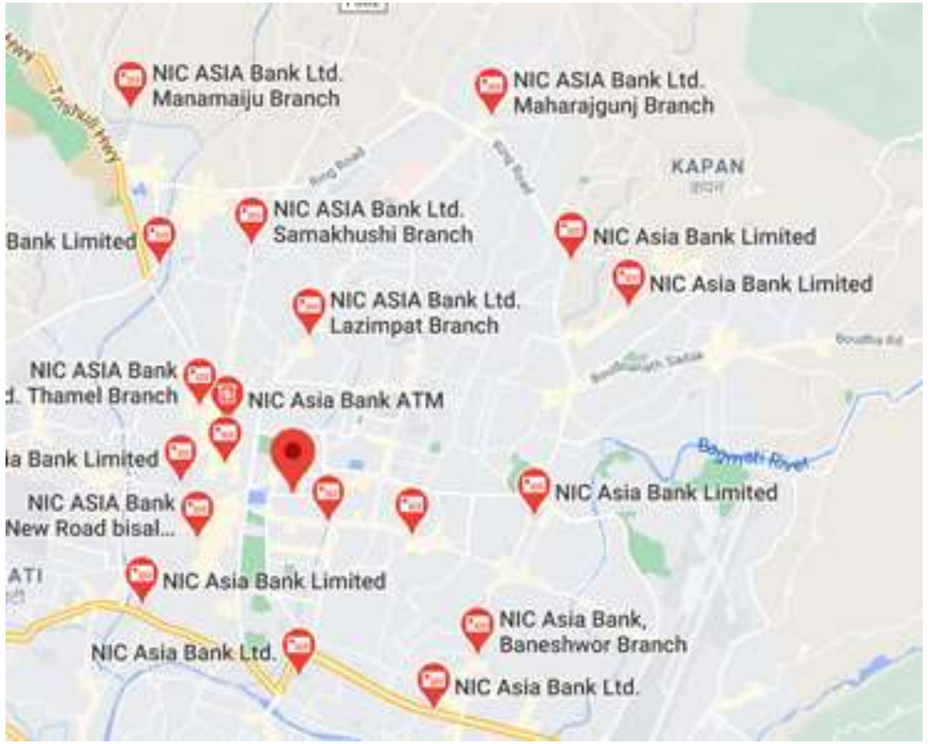
- Model represents a relationship between entity types, but pathway between certain entity occurrences is ambiguous.
- Fan trap may exist where two or more 1:M relationships that fan out from the same entity.



Fan Traps



A branch office is a location, other than the main office, where a business is conducted.



Fan Traps

Staff id	Staff name
01	Ram
02	Raj
03	Sita
04	Rita

Division id	department name
De1	Accounts And Admin
De2	Marketing
De3	Business Development
De4	HR

Branch id	Location
ktm1	Kathmandu
pok2	Pokhara

Staff id	Staff name	Division id
01	Ram	De1
02	Raj	De2
03	Sita	De2
04	Rita	De1

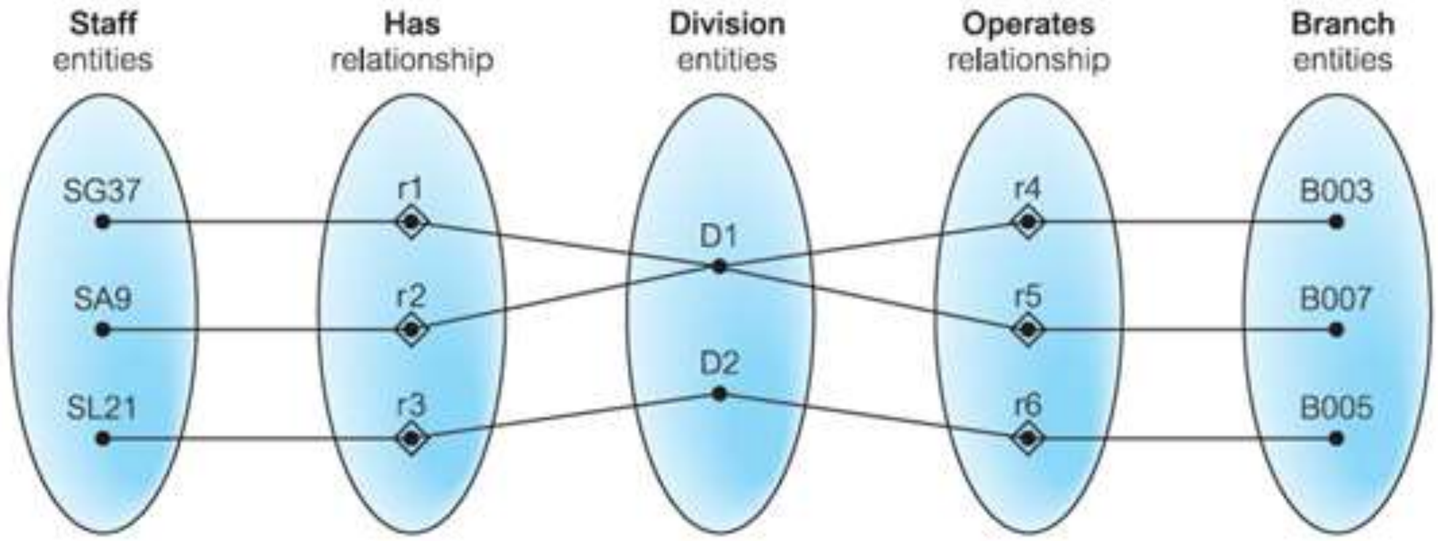
Branch id	Location	Division id
Ktm1	Kathmandu	De1
Ktm1	Kathmandu	De2
ktm1	Kathmandu	De3
pok2	Pokhara	De1

A single division operates one or more branches and has one or more staff.

Which members of staff work at a particular branch?

Fan Traps

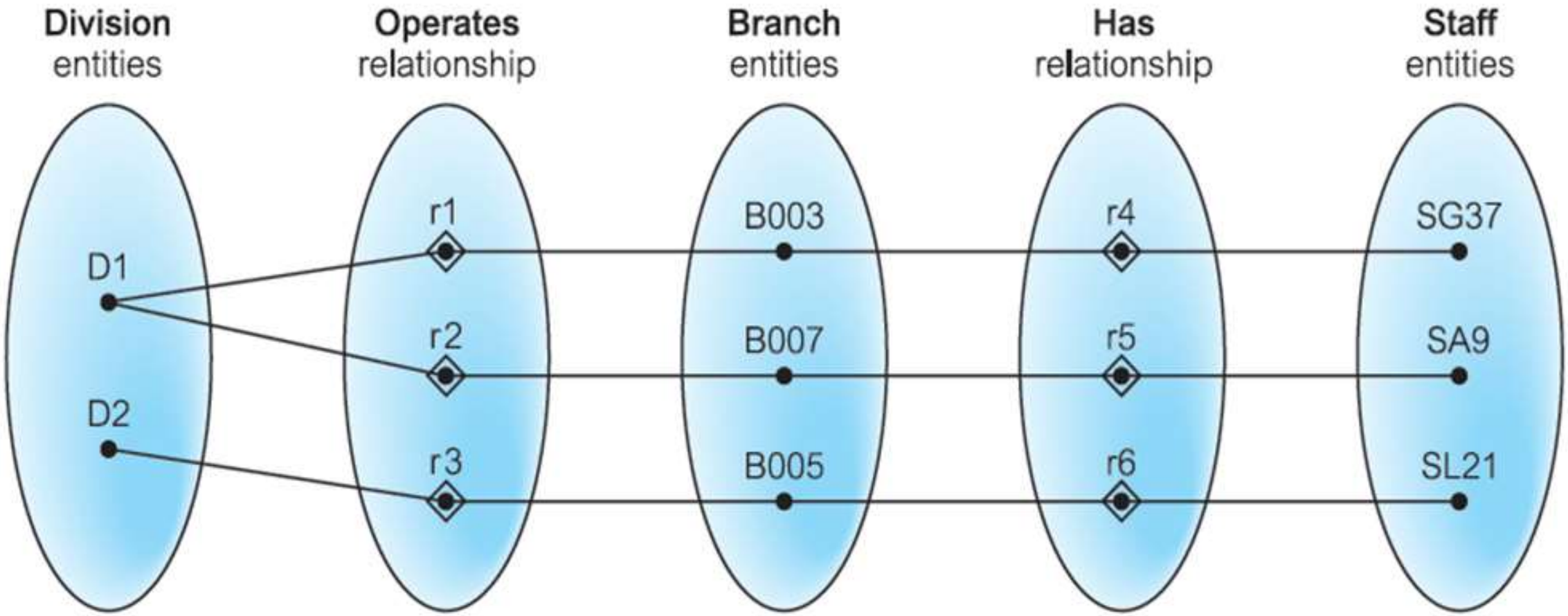
- This model represents the facts that a Single Division operates one or more Branches and Single Division has one or more Staffs.
- Problem arises when we want to know which Staff works at a particular Branch.



A single division operates one or more branches and has one or more staff.

Which members of staff work at a particular branch?

Fan Traps – Better Structure

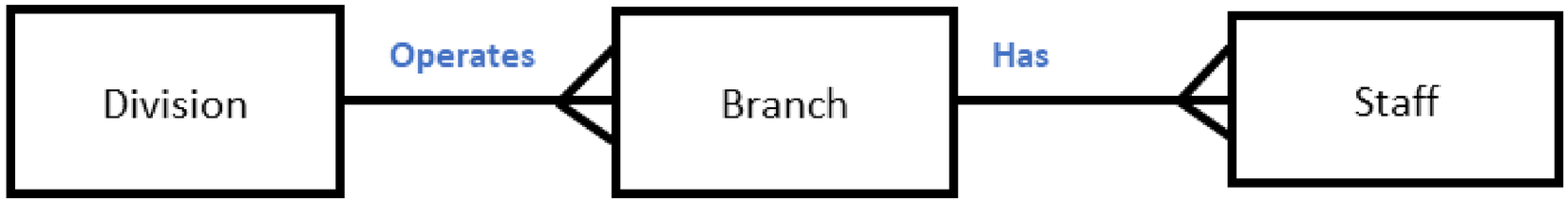


Fan Traps – Better Structure

- This model represents the facts that a Single Division operates one or more Branches and Single Branch has one or more Staffs.

Division id	Branch id
De1	Ktm1
De2	Ktm1
De3	ktm1
De1	pok2

Staff ID	Staff name	Branch id
01	Ram	Ktm1
02	Raj	Ktm1
03	Sita	ktm1
04	Rita	pok2



Chasm Traps

- Chasm trap do not deal with cardinality but with participation and with partial participation between certain entity occurrences.
- Chasm trap may occur where there are one or more relationships with a minimum multiplicity of zero (that is optional participation) forming part of the pathway between related entities.
- This join returns too many rows.
- Due to additional where condition there are too many restrictions in the record

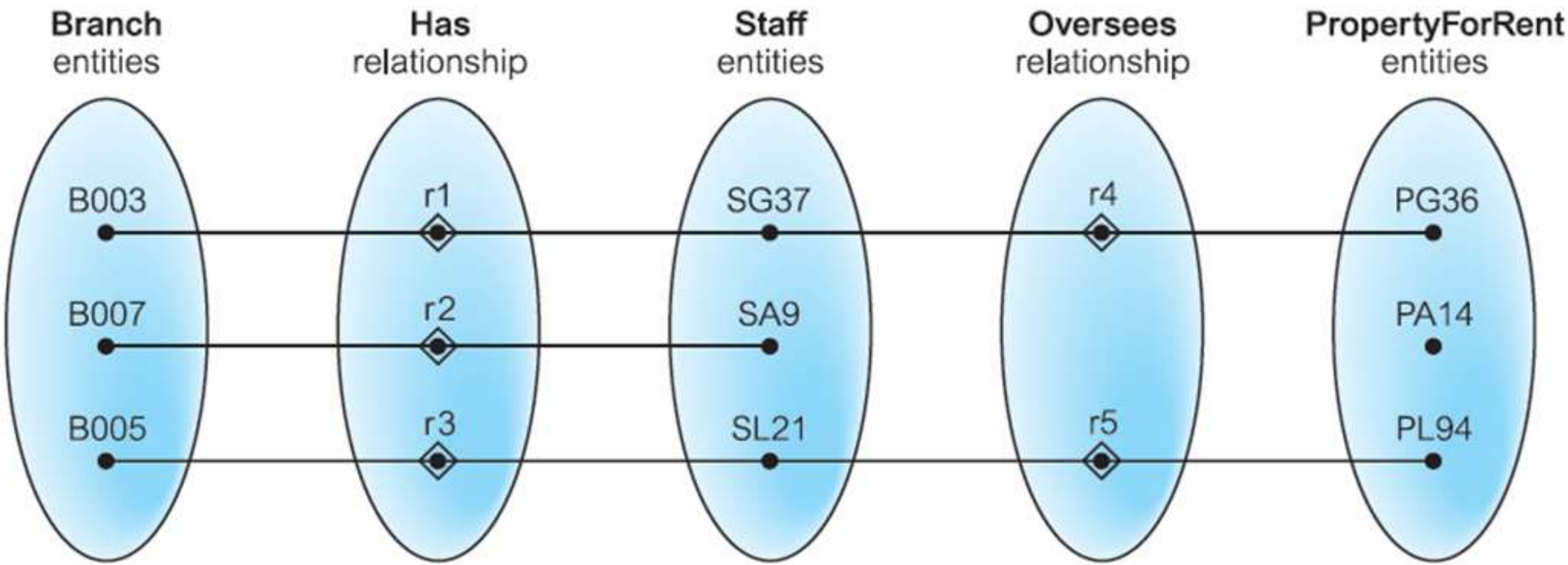
Chasm Traps

- This model represents the facts that a single branch has one or more staff who oversee zero or more properties for rent.

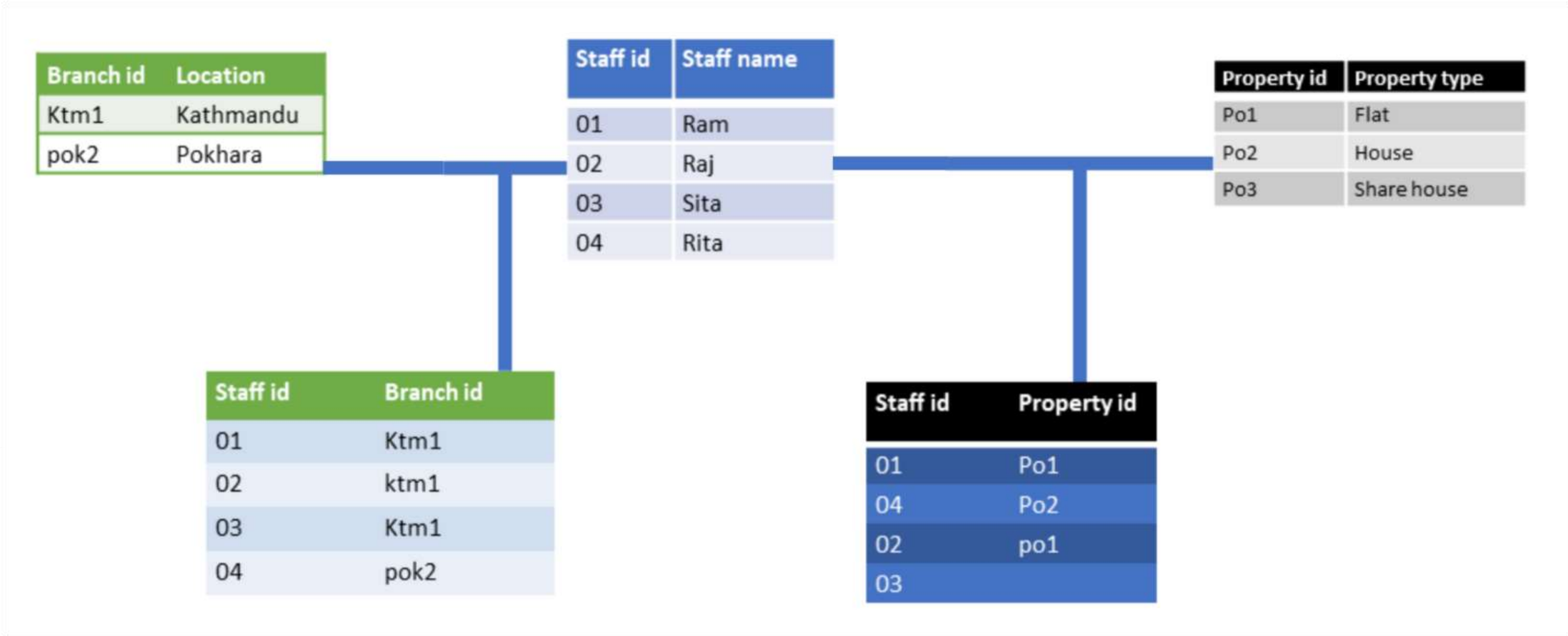
- Not all staff oversee property, and not all properties are overseen by a member of staff.
- Problem arises when we want to know which properties are available at each branch.
- Example: At which branch is property number PA14 available?

Chasm Traps

Chasm Traps


Branch
entities
Has
relationship
Staff
entities
Oversees
relationship
PropertyForRent
entities
B003
B007
B005
r1
r2
r3
SG37
SA9
SL21
r4
r5
PG36
PA14
PL94

Chasm Trap



Branch id	Location
Ktm1	Kathmandu
pok2	Pokhara

Staff id	Staff name
01	Ram
02	Raj
03	Sita
04	Rita

Property id	Property type
Po1	Flat
Po2	House
Po3	Share house

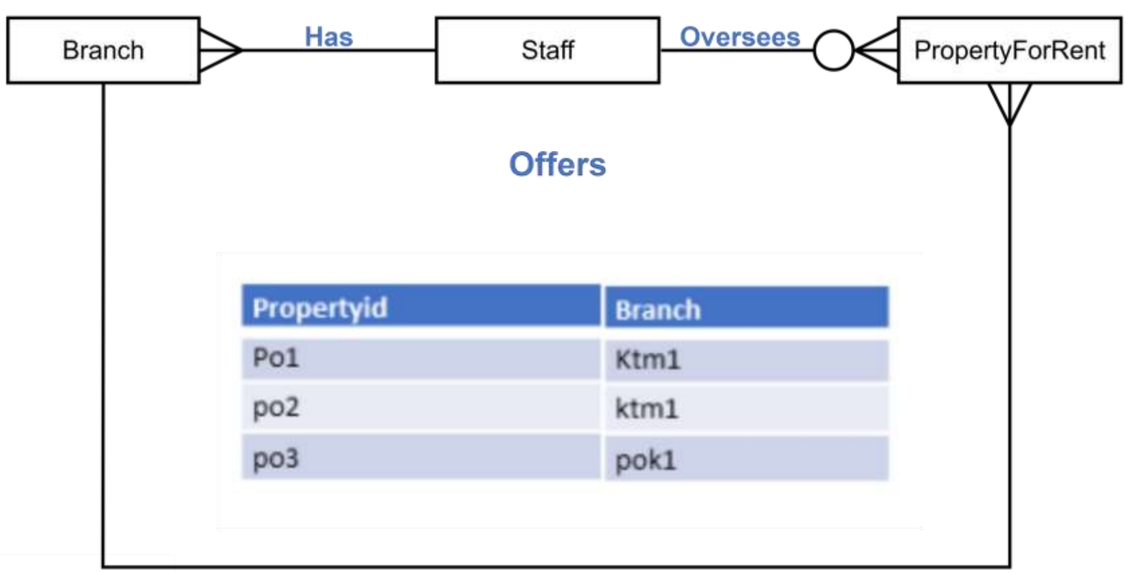
Staff id	Branch id
01	Ktm1
02	ktm1
03	Ktm1
04	pok2

Staff id	Property id
01	Po1
04	Po2
02	po1
03	

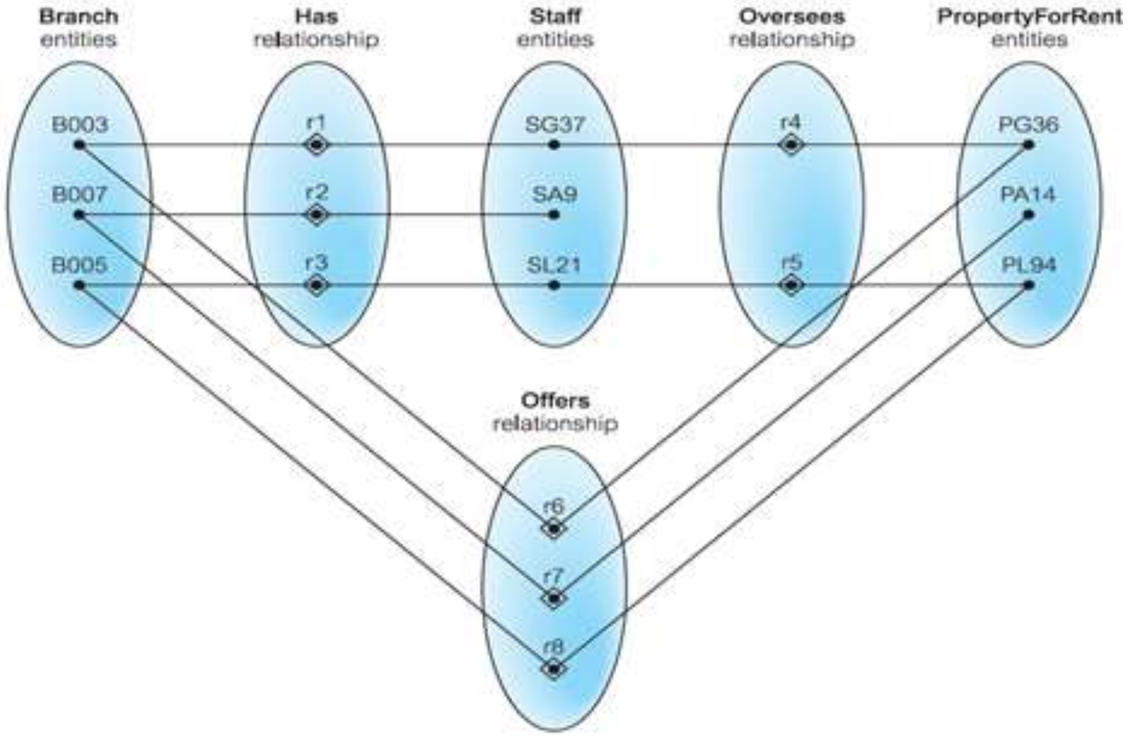
Chasm Traps – Better Structure

- Multiplicity of both Staff and PropertyForRent entities in Oversees relationship has a minimum value of zero
- Which means that some properties cannot be associated with a branch through a member of staff.
- Therefore to solve this problem, we need to identify the missing relationship, which in this case is Offers relationship between Branch and PropertyForRent entities.

Chasm Traps – Better Structure



Propertyid	Branch
Po1	Ktm1
po2	ktm1
po3	pok1



Quiz

Q. Fan traps may exist when there are two or more _____ from the same entity.

A.1:M relationships

B.1:1 relationships

C.M:1 relationships

D.None of the above

Quiz

Q. Fan traps may exist when there are two or more _____ from the same entity.

A.1:M relationships

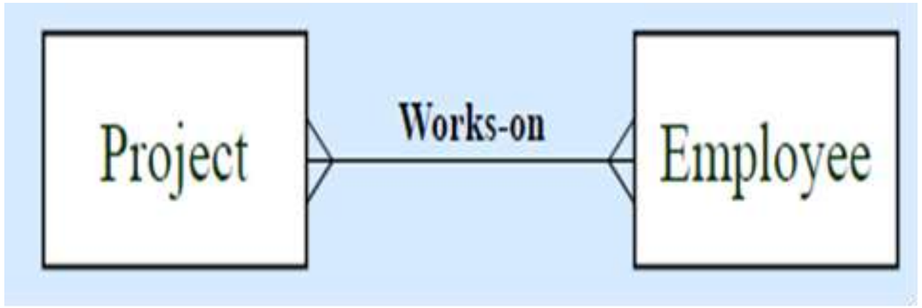
B.1:1 relationships

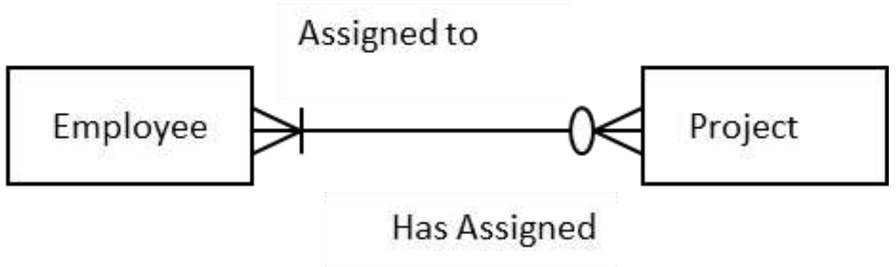
C.M:1 relationships

D.None of the above

Decomposition of Many to Many Relationship

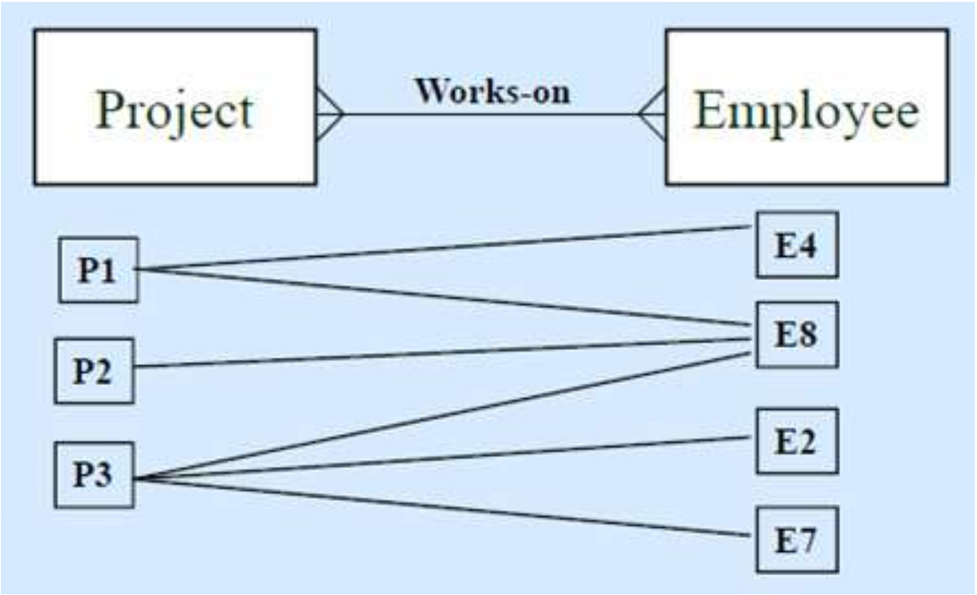
- Many-to-many relationships cannot be used in the Data Model because they cannot be represented by the relational model.
- Therefore, many-to-many relationships must be resolved early in the modeling process.

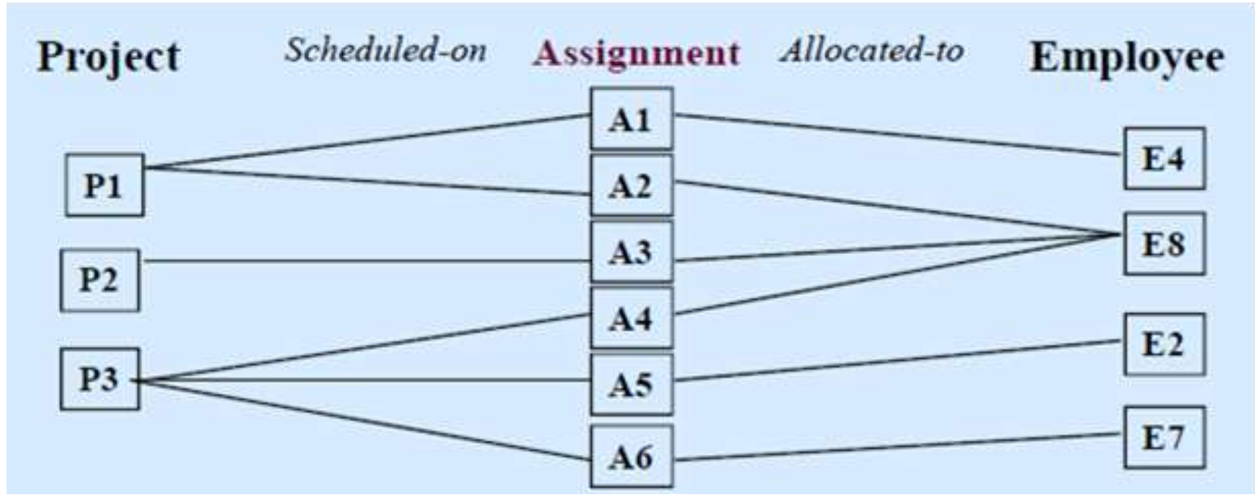




- Strategy for resolving many-to-many relationship is to replace relationship with an association entity and then relate two original entities to the association entity.

Decomposition of Many to Many Relationship

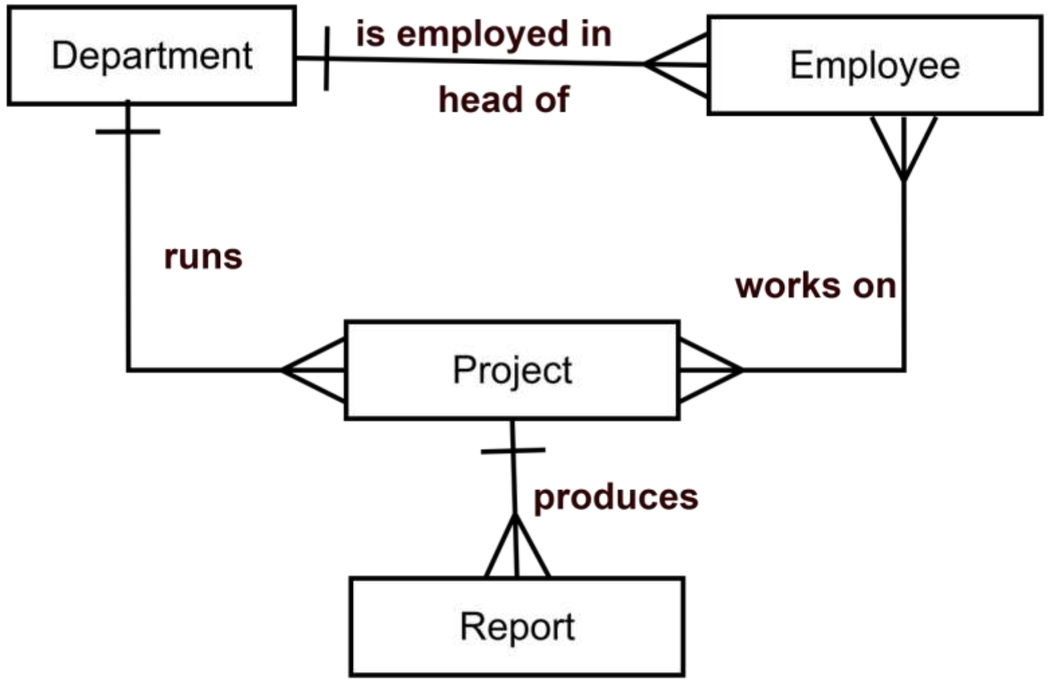




ER Model Example

An ER model for a small company system comprising the entity types and attributes

- **DEPARTMENT** (dname, location.....)
- **PROJECT** (project-code, title, budget, start-date, end-date....)
- **EMPLOYEE** (emp-id, ename, address, salary.....)
- **REPORT** (report-no, title, date.....)



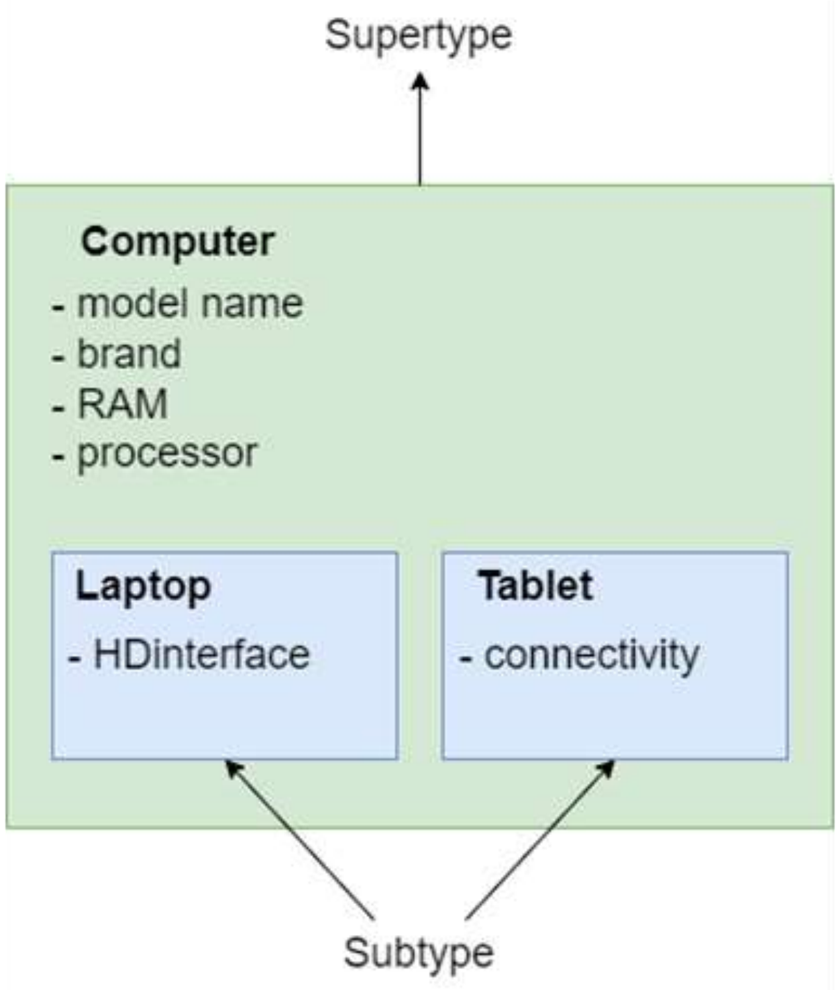
Generalisation Hierarchies

- Structured grouping of entities that share common attributes
- Entities which share multiple common attributes: grouped together as supertype and subtypes

Creating Generalization Hierarchy

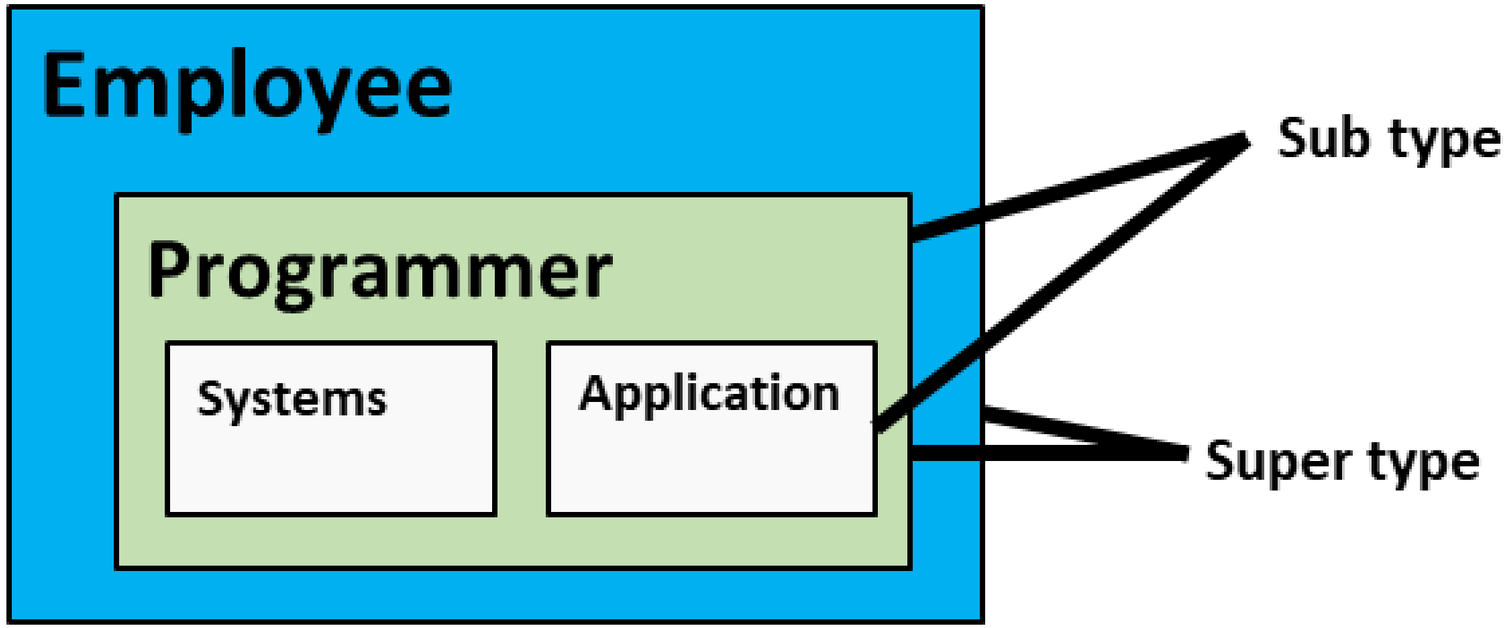
- All common attributes are assigned to **Supertype**.
- **Supertype** is also assigned an attribute, called a discriminator, whose values identify the categories of **Subtypes**
- Attributes unique to a category, are assigned to the appropriate **Subtype**.
- Each **Subtype** also inherits primary key of the **Supertype**.
- **Subtypes** that have only a primary key should be eliminated.
- **Subtypes** are related to the **Supertypes** through a one-to-one relationship.

Generalisation Hierarchies



Generalization Hierarchies

- Systems and Application are Subtypes of Programmer
- Programmer is a Subtype of Employee

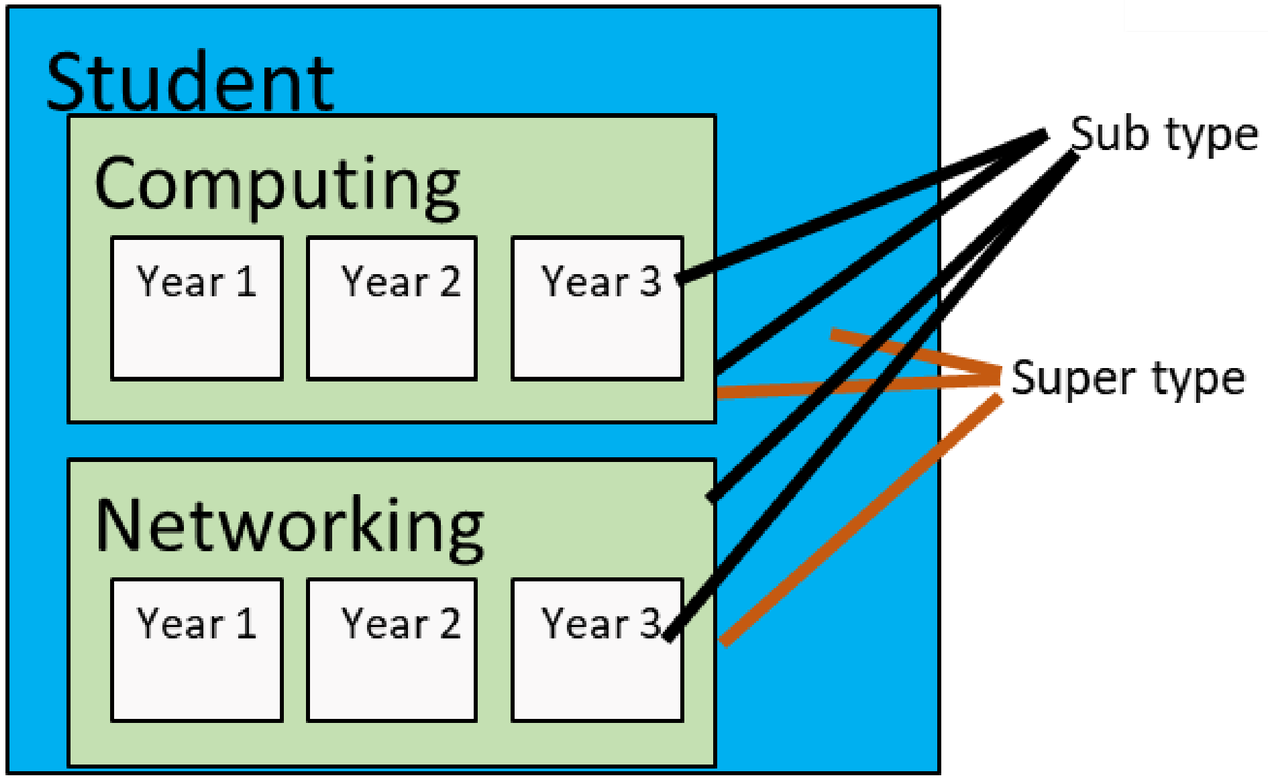


When to Use Generalisation Hierarchies?

- Generalization hierarchies is used when
  - large number of entities appear to be of same type
  - attributes are repeated for multiple entities
  - model is continually evolving.
- Generalization hierarchies improve stability of model by allowing **changes to be made only to those entities relevant to change** and simplify model by reducing number of entities in model.

Generalization Hierarchies

- For example:
  - Year 1 and year 2 are Subtypes of computing
- Computing is a Subtype of Student



Quiz

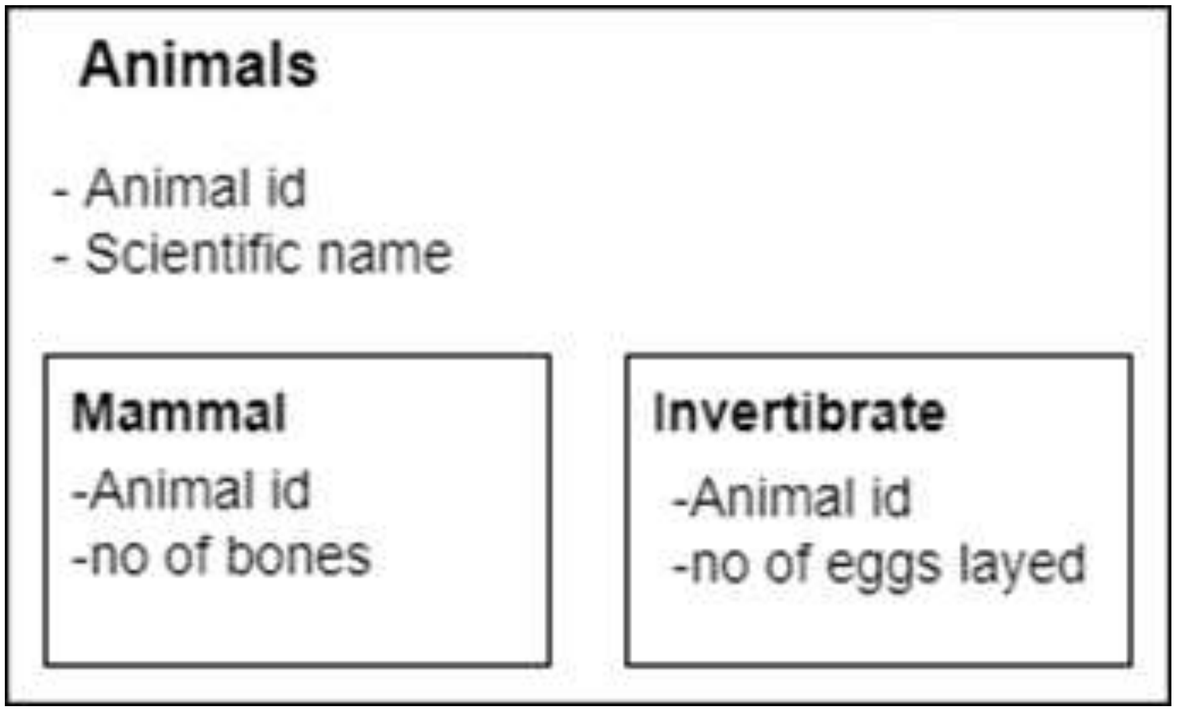
Q. In the given figure, which entity is the supertype?

a) Animals

b) Mammal

c) Invertebrate

d) None of the above

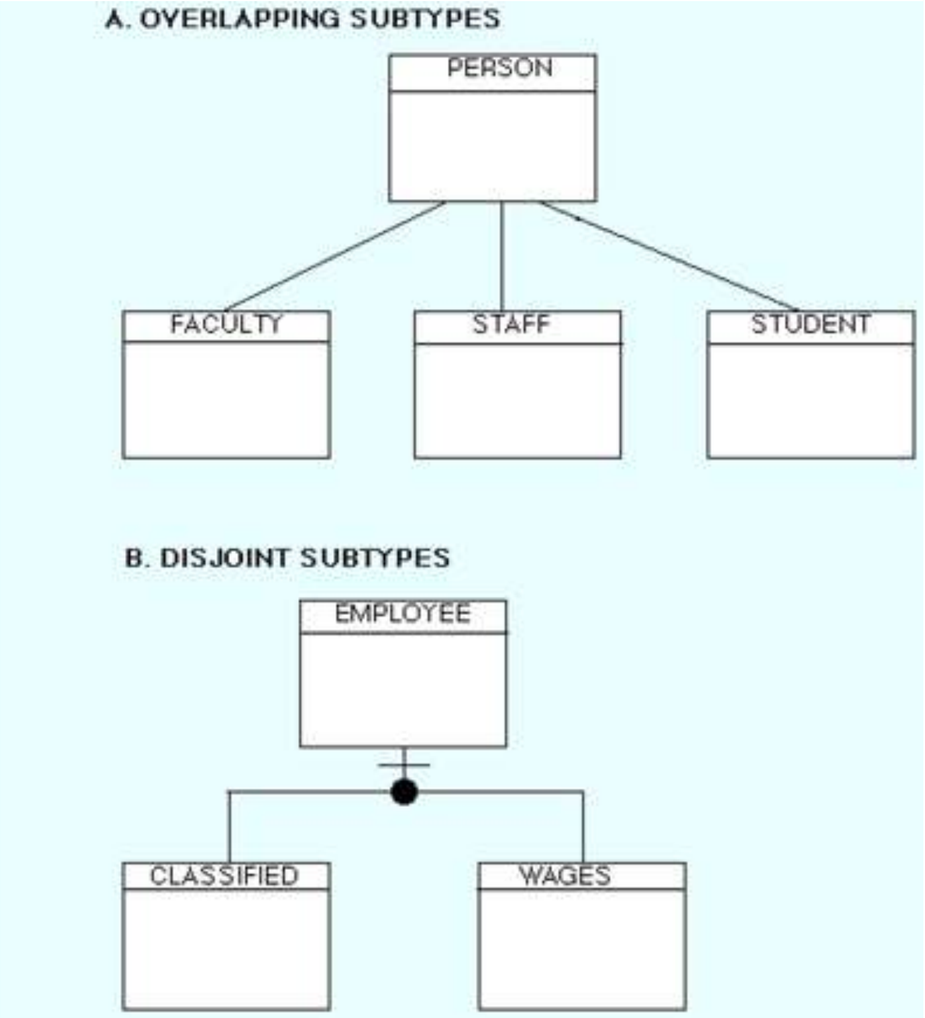


Types of Hierarchies

- Can either be **Overlapping** or **Disjoint.**
- In an **Overlapping** hierarchy an entity instance can be part of multiple subtypes.
    - Example: To represent people at a university we have **Supertype** entity **PERSON** which has three Subtypes, **FACULTY, STAFF**, and **STUDENT**.
- It is quite possible for an individual to be in more than one subtype, a staff member who is also registered as a student

Types of Hierarchies

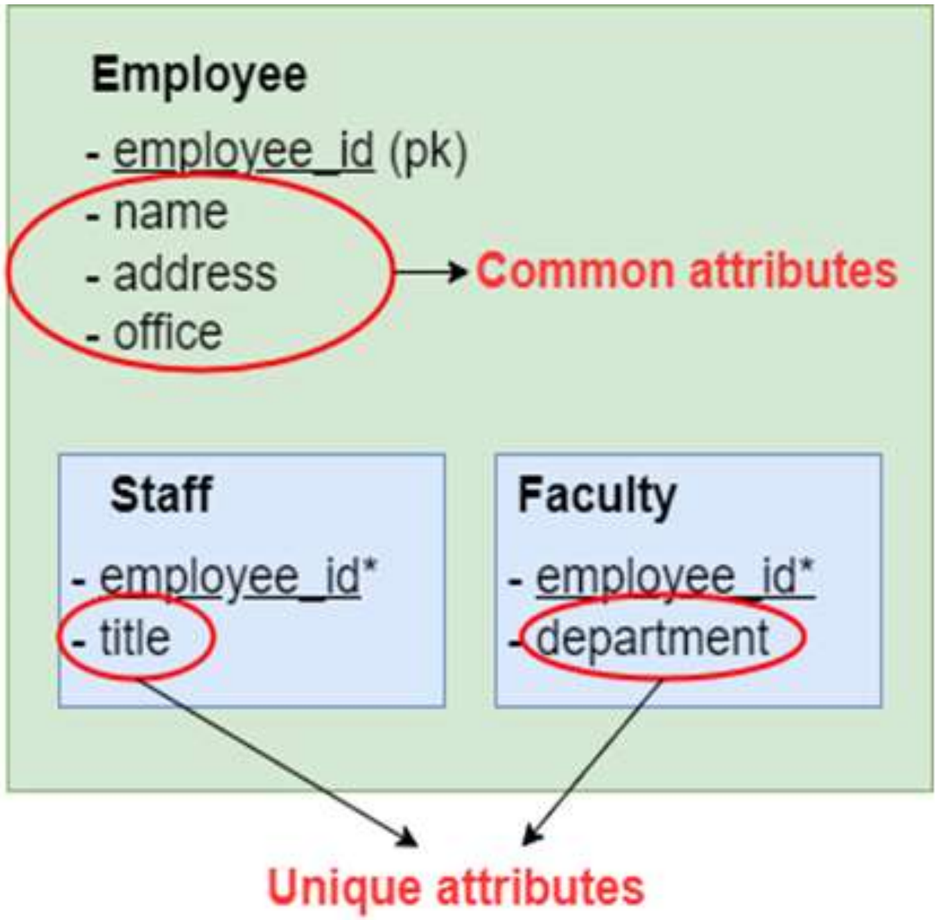
- In a **Disjoint** hierarchy, an entity instance can be in only one **Subtype**.
  - Example: Entity **EMPLOYEE**, may have two subtypes, **CLASSIFIED** and **WAGES**.
- An employee may be one type or the other but not both.

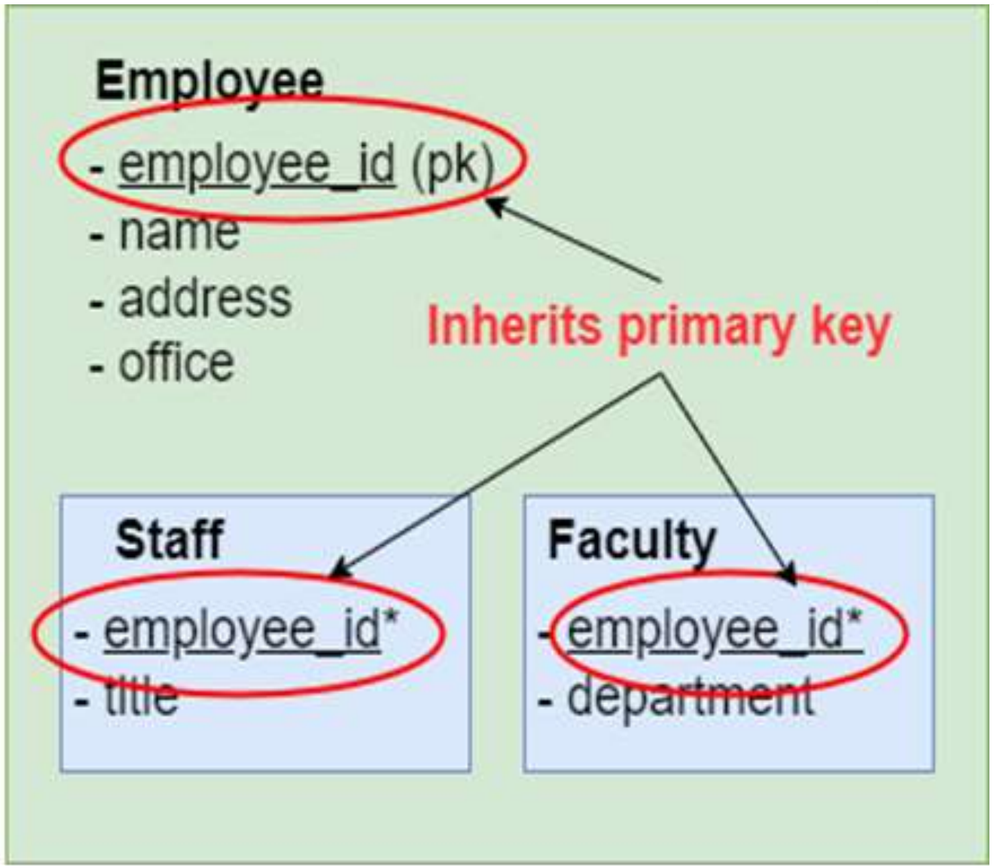


Rule

- Primary rule of generalization hierarchies is that each instance of the **Supertype** entity must appear in at least one **Subtype**; likewise, an instance of the **Subtype** must appear in **Supertype**.
- **Subtypes** can be a part of only one generalization hierarchy. That is, a **Subtype** cannot be related to more than one **Supertype**.
- Generalization hierarchies may be nested by having the **Subtype** of one hierarchy be the **Supertype** for another.
- **Subtypes** may be parent entity in a relationship but not the child. If this were allowed, the subtype would inherit two primary keys.

Creating Generalisation Hierarchies





Rules of Generalisation Hierarchies

ITAHARI INTERNATIONAL COLLEGE

islington college (इस्लिङ्टन कलेज)

informatics college • pokhara

Thank you

End of Lecture