Database Management Systems

Module 1
Introduction to Database Modelling and Relational Algebra

Presidency University, Bengaluru

PRESIDENCY UNIVERSITY

Private University Estd. in Karnataka State by Act No. 41 of 2013

Contents

- Introduction
- Simplified database system environment
- Typical DBMS Functionality
- Main Characteristics of the Database Approach
- Database Users
- Advantages of Using the Database Approach
- Data Models
- History of Data Models
- Three-Schema Architecture
- DBMS Languages
- DBMS Interfaces
- Database system environment

Basic Definitions

- **Database**: A collection of related data.
- **Data**: Known facts that can be recorded and have an implicit meaning.
- **Mini-world**: A database represents some aspect of the real world, sometimes called the **miniworld** or **the universe of discourse (UoD).** Changes to the miniworld are reflected in the database.
  For example, student grades and transcripts at a university.
- **Database Management System (DBMS)**: A software package/ system to facilitate the creation and maintenance of a computerized database.
- **Database System**: The DBMS software together with the data itself. Sometimes, the applications are also included.

- A **database management system (DBMS)** is a collection of programs that enables users to create and maintain a database.
- The DBMS is a *general-purpose software system* that facilitates the processes of *defining, constructing, manipulating,* and *sharing* databases among various users and applications.
- **Defining** a database involves specifying the data types, structures, and constraints of the data to be stored in the database.
- The database definition or descriptive information is also stored by the DBMS in the form of a database catalog or dictionary; it is called **meta-data**.
- **Constructing** the database is the process of storing the data on some storage medium that is controlled by the DBMS.

- **Manipulating** a database includes functions such as querying the database to retrieve specific data, updating the database to reflect changes in the miniworld, and generating reports from the data.
- **Sharing** a database allows multiple users and programs to access the database simultaneously.
- An **application program** accesses the database by sending queries or requests for data to the DBMS.
- A **query** typically causes some data to be retrieved; a **transaction** may cause some data to be read and some data to be written into the database.

What is a File system?

- A file system is a technique of arranging the files in a storage medium like a hard disk, pen drive, DVD, etc.
- It helps you to organize the data and allows easy retrieval of files when they are required.
- It mostly consists of different types of files like mp3, mp4, txt, doc, etc. that are grouped into directories.
- A file system enables you to handle the way of reading and writing data to the storage medium.
- It is directly installed into the computer with the Operating systems such as Windows and Linux.

KEY DIFFERENCES

- A file system is a software that manages and organizes the files in a storage medium, whereas DBMS is a software application that is used for accessing, creating, and managing databases.
- The file system doesn't have a crash recovery mechanism on the other hand, DBMS provides a crash recovery mechanism.
- Data inconsistency is higher in the file system. On the contrary Data inconsistency is low in a database management system.
- File system does not provide support for complicated transactions, while in the DBMS system, it is easy to implement complicated transactions using SQL.
- File system does not offer concurrency, whereas DBMS provides a concurrency facility.

Simplified database system environment

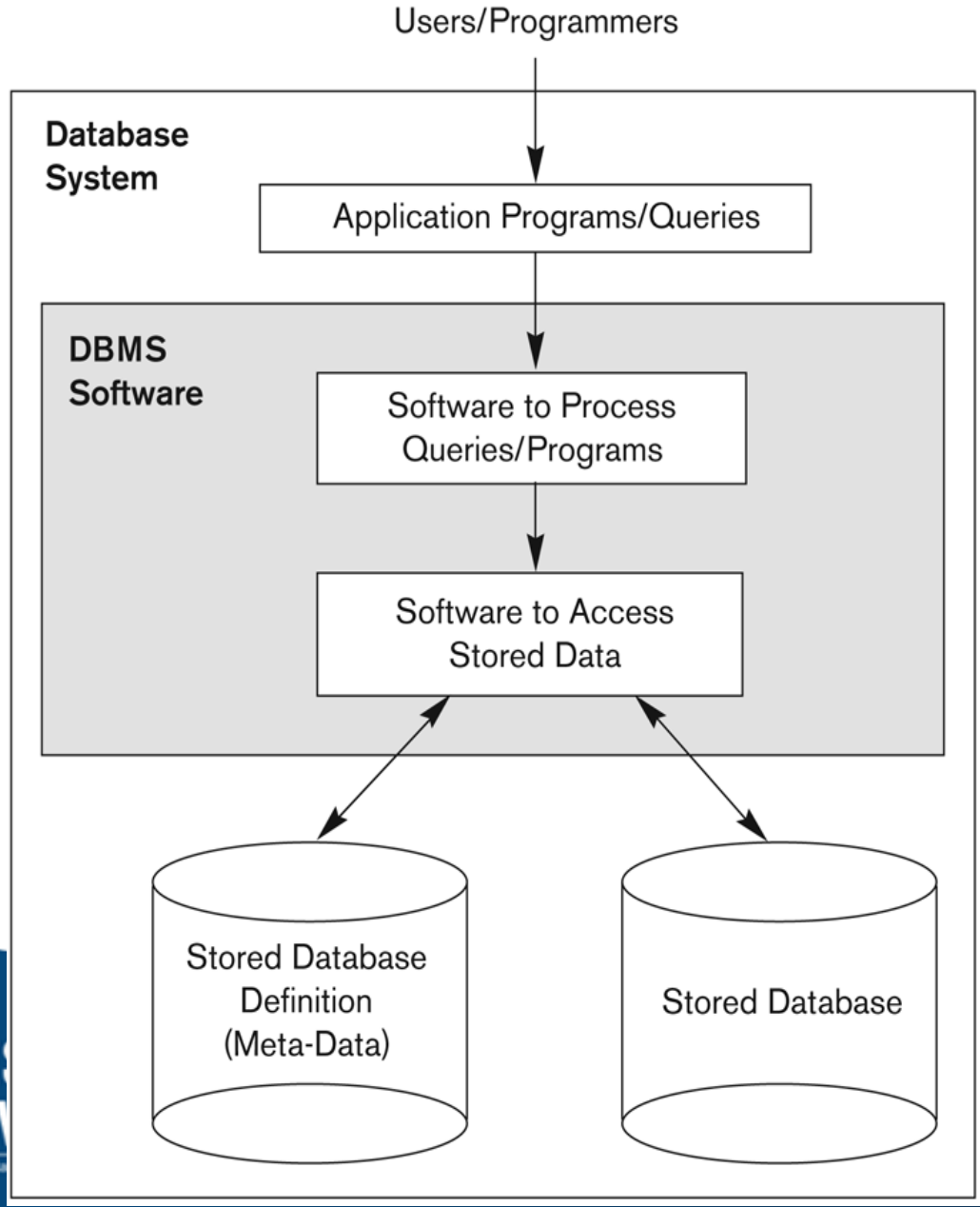


Figure 1.1
A simplified database system environment.

Typical DBMS Functionality

- Define a database : in terms of data types, structures and constraints
- Construct or Load the Database on a secondary storage medium
- Manipulating the database : querying, generating reports, insertions, deletions and modifications to its content
- Concurrent Processing and Sharing by a set of users and programs – yet, keeping all data valid and consistent

Other features:

- Protection or Security measures to prevent unauthorized access
- "Active" processing to take internal actions on data
- Presentation and Visualization of data

Example of a Database (with a Conceptual Data Model)

- **Mini-world -example**: Part of a UNIVERSITY environment.
- **Some mini-world *entities***:
  - STUDENTs
  - COURSEs
  - SECTIONs (of COURSEs)
  - (academic) DEPARTMENTs
  - INSTRUCTORs

Note: The above could be expressed in the ENTITY-RELATIONSHIP data model.

Example of a Database (with a Conceptual Data Model)

- **Some mini-world *relationships***:
  - SECTIONs *are of* specific COURSEs
  - STUDENTs *take* SECTIONs
  - COURSEs *have* prerequisite COURSEs
  - INSTRUCTORs *teach* SECTIONs
  - COURSEs *are offered by* DEPARTMENTs
  - STUDENTs *major in* DEPARTMENTs

Note: The above could be expressed in the *ENTITY-RELATIONSHIP* data model.

STUDENT

Name	Student_number	Class	Major
Smith	17	1	CS
Brown	8	2	CS

COURSE

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

SECTION

Section_identifier	Course_number	Semester	Year	Instructor
85	MATH2410	Fall	07	King
92	CS1310	Fall	07	Anderson
102	CS3320	Spring	08	Knuth
112	MATH2410	Fall	08	Chang
119	CS1310	Fall	08	Anderson
135	CS3380	Fall	08	Stone

GRADE_REPORT

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

PREREQUISITE

Course_number	Prerequisite_number
CS3380	CS3320
CS3380	MATH2410
CS3320	CS1310

Figure 1.2
A database that stores student and course information.

Main Characteristics of the Database Approach

- Self-describing nature of a database system: A DBMS **catalog** stores the *description* of the database. The description is called **meta-data**). This allows the DBMS software to work with different databases.
- Insulation between programs and data: Called **program-data independence**. Allows changing data storage structures and operations without having to change the DBMS access programs.
- Data Abstraction: A **data model** is used to hide storage details and present the users with a *conceptual view* of the database.

Main Characteristics of the Database Approach

- Support of multiple views of the data: Each user may see a different view of the database, which describes *only* the data of interest to that user.
- Sharing of data and multiuser transaction processing : allowing a set of concurrent users to retrieve and to update the database. Concurrency control within the DBMS guarantees that each **transaction** is correctly executed or completely aborted. OLTP (Online Transaction Processing) is a major part of database applications.

Types of Databases and Database Applications

- Numeric and Textual Databases
- Multimedia Databases
- Geographic Information Systems (GIS)
- Data Warehouses
- Real-time and Active Databases

Database Users

- **Actors on the scene**
  - Database Administrator
  - Database Designers
  - End users
    - Casual end users
    - Naive or parametric end users
    - Sophisticated end users
    - Stand alone users
  - System analysts and application programmers (Software Engineers)
- **Workers behind the scene**
  - DBMS system designers and implementers
  - Tool Developers
  - Operators and maintenance personnel

Database Users

Users may be divided into those who actually use and control the content (called "Actors on the Scene") and those who enable the database to be developed and the DBMS software to be designed and implemented (called "Workers Behind the Scene").

Actors on the scene

- **Database administrators:** responsible for authorizing access to the database, for coordinating and monitoring its use, acquiring software, and hardware resources, controlling its use and monitoring efficiency of operations.
- **Database Designers:** responsible to define the content, the structure, the constraints, and functions or transactions against the database. They must communicate with the end-users and understand their needs.
- **End-users:** they use the data for queries, reports and some of them actually update the database content.

Categories of End-users

- **Casual** : access database occasionally when needed
- **Naïve or Parametric** : they make up a large section of the end-user population. They use previously well-defined functions in the form of "canned transactions" against the database. Examples are bank-tellers or reservation clerks who do this activity for an entire shift of operations.
- **Sophisticated** : these include business analysts, scientists, engineers, others thoroughly familiar with the system capabilities. Many use tools in the form of software packages that work closely with the stored database.
- **Stand-alone** : mostly maintain personal databases using ready-to-use packaged applications. An example is a tax program user that creates his or her own internal database.

Workers behind the Scene

- **DBMS system designers and implementers** design and implement the DBMS modules and interfaces as a software package.
- **Tool developers** design and implement **tools—**the software packages that facilitate database modeling and design, database system design, and improved performance.
- **Operators and maintenance personnel** (system administration personnel) are responsible for the actual running and maintenance of the hardware and software environment for the database system.

Advantages of Using the Database Approach

- Controlling redundancy in data storage and in development and maintenance efforts.
- Sharing of data among multiple users.
- Restricting unauthorized access to data.
- Providing persistent storage for program Objects .
- Providing Storage Structures for efficient Query Processing
- Providing backup and recovery services.
- Providing multiple interfaces to different classes of users.
- Representing complex relationships among data.
- Enforcing integrity constraints on the database.
- Drawing Inferences and Actions using rules

Additional Implications of Using the Database Approach

- Potential for enforcing standards: this is very crucial for the success of database applications in large organizations Standards refer to data item names, display formats, screens, report structures, meta-data (description of data) etc.
- Reduced application development time: incremental time to add each new application is reduced.
- Flexibility to change data structures: database structure may evolve as new requirements are defined.
- Availability of up-to-date information – very important for on-line transaction systems such as airline, hotel, car reservations.
- Economies of scale: by consolidating data and applications across departments wasteful overlap of resources and personnel can be avoided.

Extending Database Capabilities

- **New functionality is being added to DBMSs in the following areas:**
  - Scientific Applications
  - Image Storage and Management
  - Audio and Video data management
  - Data Mining
  - Spatial data management
  - Time Series and Historical Data Management

When not to use a DBMS

- **Main inhibitors (costs) of using a DBMS**:
  - High initial investment and possible need for additional hardware.
  - Overhead for providing generality, security, concurrency control, recovery, and integrity functions.
- **When a DBMS may be unnecessary:**
  - If the database and applications are simple, well defined, and not expected to change.
  - If there are stringent real-time requirements that may not be met because of DBMS overhead.
  - If access to data by multiple users is not required.
- **When no DBMS may suffice:**
  - If the database system is not able to handle the complexity of data because of modeling limitations
  - If the database users need special operations not supported by the DBMS.

How much did we learn??

1. What is a database?

   collection of related data

2. What is a DBMS?

   A collection of programs which enables the user to create and maintain a database.

3. What are the properties of a Database?

   some aspects of real world (miniworld or the universe of discourse (UoD).
   Logically coherent collection of data with some inherent meaning.
   designed, built and populated with data for a specific purpose.

4. What are the characteristics of database approach?

   Self-describing nature of a database system

   program-data independence

   Support of multiple views of the data

   Sharing of Data & Multi-user Transaction processing

How much did we learn??

5. What is the role of a Database Administrator?

   Managing the database system, authorizing access, coordinating & monitoring the usage.

6. Who are the different end users?

   Casual end users

   Naive or parametric end users

   Sophisticated end users

   Standalone users

7. What are the advantages of DBMS?

Chapter 2

What do we learn here?

- **Data Models, Schemas, and Instances**
- **Three-Schema Architecture and Data Independence**
- **DBMS languages and Interfaces**
- **The Database System Environment**
- **Centralized and Client/Server Architectures for DBMSs**
- **Classification of Database Management Systems**

Data Models

- **Data Model**: A set of concepts to describe the *structure* of a database, and certain *constraints* that the database should obey.
- **Data Model Operations**: Operations for specifying database retrievals and updates by referring to the concepts of the data model. Operations on the data model may include *basic operations* and *user-defined operations*.

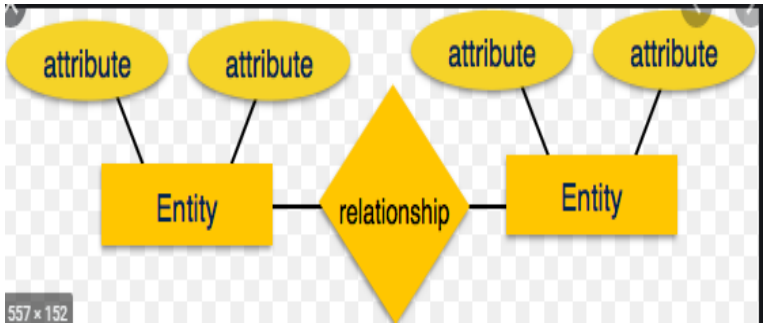
Categories of data models

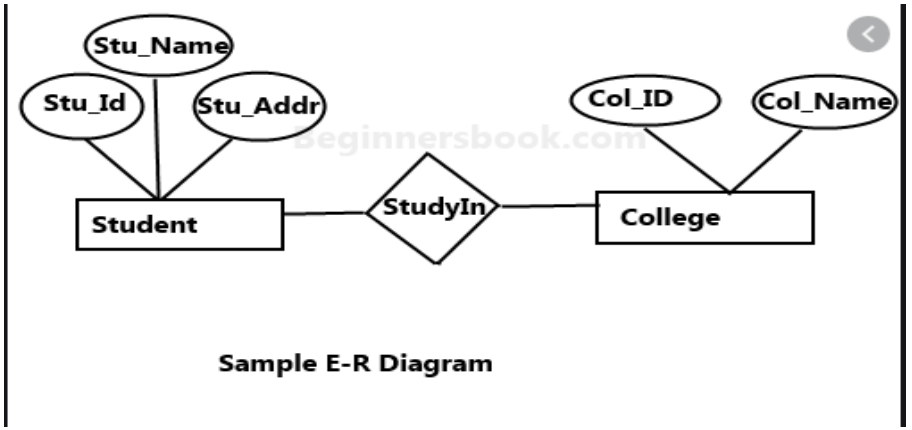
- **Conceptual** (**high-level**, **semantic**) data models: Provide concepts that are close to the way many users *perceive* data. (Also called **entity-based** or **object-based** data models.)
- **Physical** (**low-level**, **internal**) data models: Provide concepts that describe details of how data is stored in the computer.
- **Implementation** (**representational**) data models: Provide concepts that fall between the above two, balancing user views with some computer storage details.

High-level or Conceptual Data Model

- The conceptual or high-level data model uses the concepts such as entities, attributes and relationships.
- Provide concepts that are close to the way many users perceive data.
- An **entity represents a real-world object or concept, such as an employee or a project** from the mini-world that is described in the database.
- An **attribute represents some** property of interest that further describes an entity, such as the employee's name or salary
- A **relationship among two or more entities represents an association among** the entities, for example, a works-on relationship between an employee and a project.
- Example (a popular high-level conceptual data model) : Entity- Relationship Model (ER Model)

High-level or Conceptual Data Model

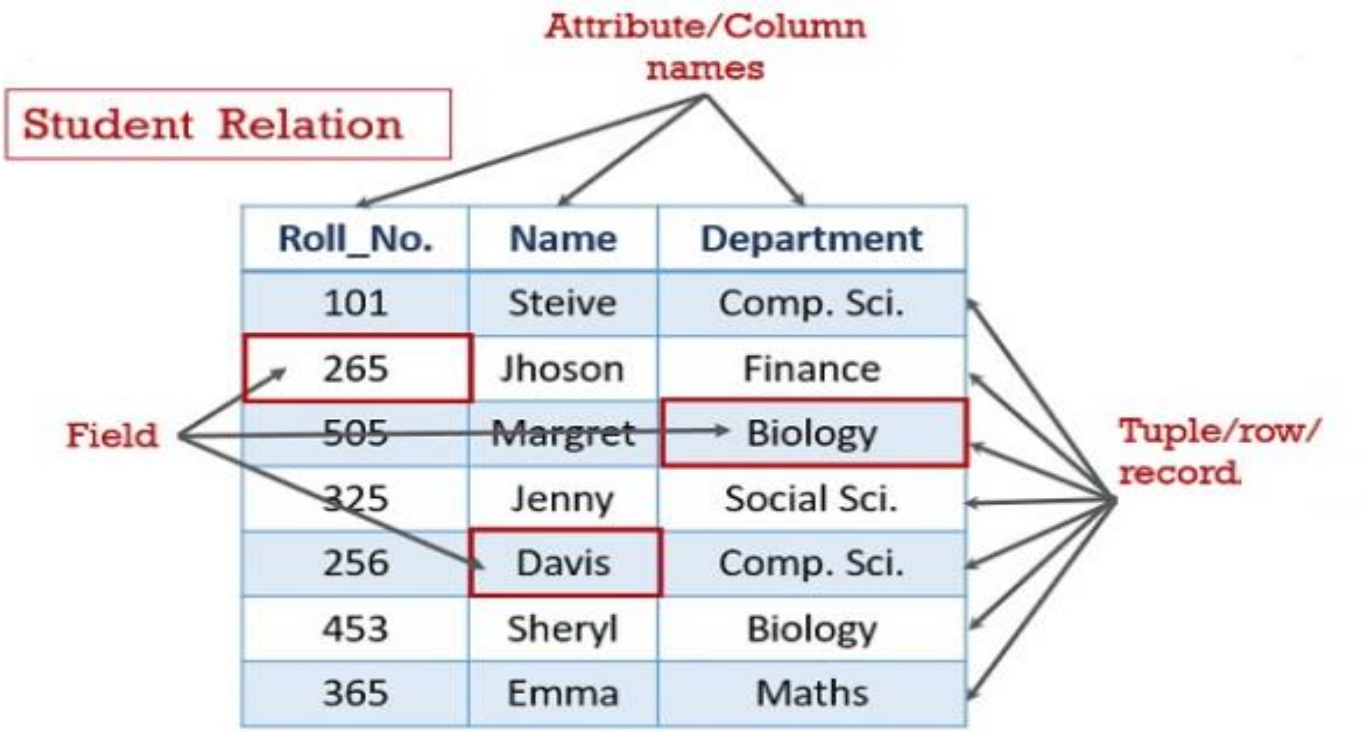




Representational or Implementation Data Model

- Representational or implementation data model represents data by using record structures.
- They are also called record-based data models.
- These are the models used most frequently in traditional commercial DBMSs. These include the widely used relational data model, as well as the so-called legacy data models—the network and hierarchical models—that have been widely used in the past.

Representational or Implementation Data Model



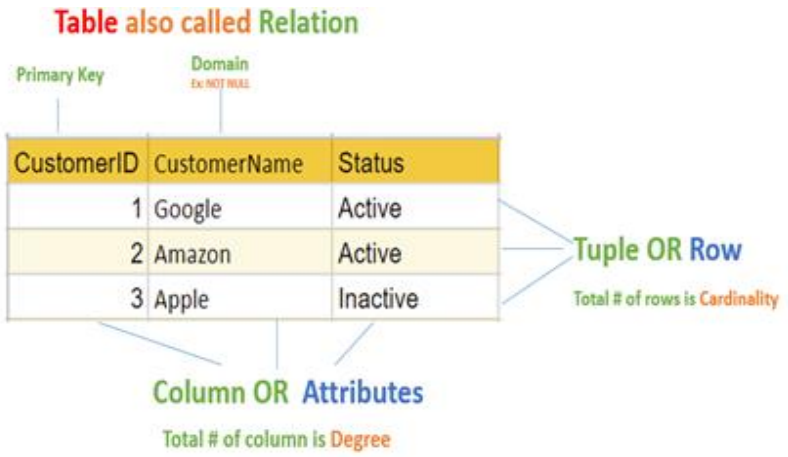
Low-level or Physical Data Model

- Physical data model describe how data is stored as files in the computer by representing information such as record formats, record orderings, and access paths.
- An **access path is a search structure that makes the search for particular database** records efficient, such as indexing or hashing.
- An **index is an example of an** access path that allows direct access to data using an index term or a keyword.

History of Data Models

Relational Model:

- proposed in 1970 by E.F. Codd (IBM), first commercial system in 1981-82. Now in several commercial products (DB2, ORACLE, SQL Server, SYBASE, INFORMIX).
- The relational model represents the database as a collection of relations. A relation is nothing but a table of values. Every row in the table represents a collection of related data values. These rows in the table denote a real-world entity or relationship.



CustomerID	CustomerName	Status
1	Google	Active
2	Amazon	Active
3	Apple	Inactive

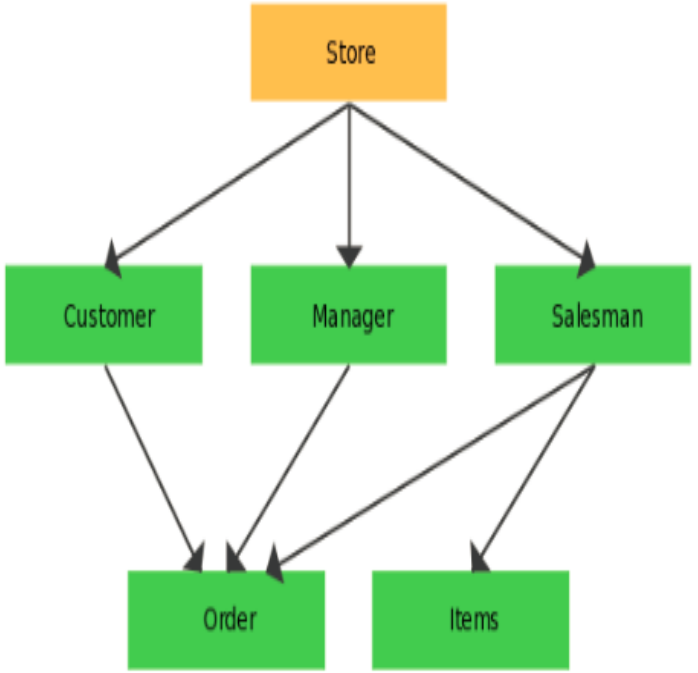
Customer

Billing

InvoiceNo	CustomerID	Amount
1	1	$100
2	1	$200
3	2	$150

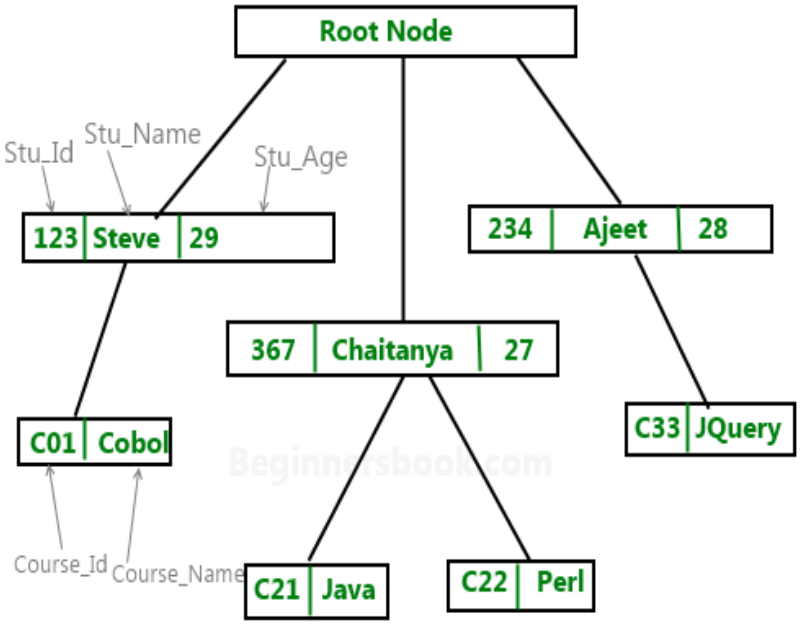
Network model

- Is a database model that is designed as a flexible approach to representing objects and their relationships. A unique feature of the network model is its schema, which is viewed as a graph where relationship types are arcs and object types are nodes.
- the first one to be implemented by Honeywell in 1964-65 (IDS System) Adopted heavily due to the support by CODASYL (CODASYL - DBTG report of 1971). Later implemented in a large variety of systems - IDMS (Cullinet - now CA), DMS 1100 (Unisys), IMAGE (H.P.) VAX -DBMS (Digital Equipment Corp.).



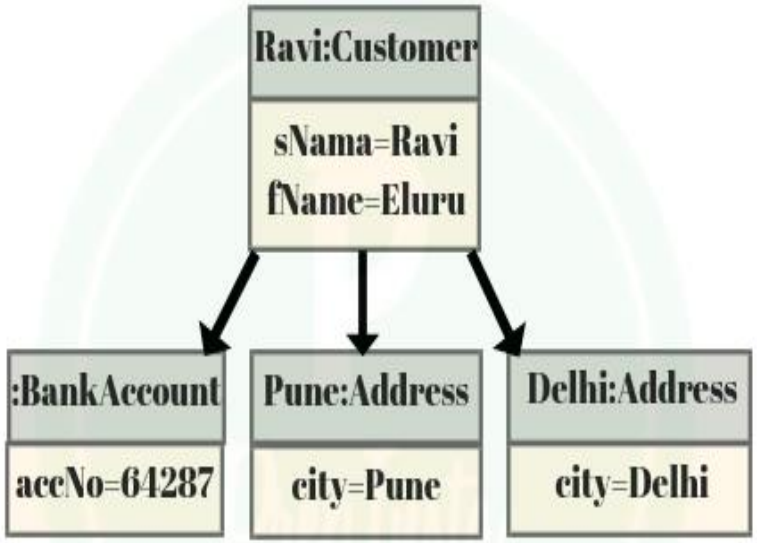
Hierarchical database model

- is a data model in which the data are organized into a tree-like structure. The data are stored as records which are connected to one another through links. A record is a collection of fields, with each field containing only one value. The type of a record defines which fields the record contains.
- The hierarchical database model mandates that each child record has only one parent, whereas each parent record can have one or more child records. In order to retrieve data from a hierarchical database the whole tree needs to be traversed starting from the root node.
- implemented in a joint effort by IBM and North American Rockwell around 1965. Resulted in the IMS family of systems. The most popular model.



Object Oriented (OO) Data **Model**

- Increasingly complex real-world problems demonstrated a need for a data **model** that more closely represented the real world. In the **object oriented** data **model** (OODM), both data and their relationships are contained in a single structure known as an **object**.



Object relational model

- is a combination of a Object oriented database model and a Relational database model. So, it supports objects, classes, inheritance etc. just like Object Oriented models and has support for data types, tabular structures etc. like Relational data model.
- One of the major goals of Object relational data model is to close the gap between relational databases and the object oriented practices frequently used in many programming languages such as C++, C#, Java etc.
- Both Relational data models and Object oriented data models are very useful. But it was felt that they both were lacking in some characteristics and so work was started to build a model that was a combination of them both. Hence, Object relational data model was created as a result of research that was carried out in the 1990's.

Schemas versus Instances

- **Database Schema**: A database schema is the skeleton structure that represents the logical view of the entire database. It defines how the data is organized and how the relations among them are associated. It formulates all the constraints that are to be applied on the data.
- The *description* of a database. Includes descriptions of the database structure and the constraints that should hold on the database.
- **Schema Diagram**: A diagrammatic display of (some aspects of) a database schema.
- **Schema Construct**: A component of the schema or an object within the schema, e.g., STUDENT, COURSE.
- **Database Instance**: The actual data stored in a database at a *particular moment in time*. Also called **database state** (or **occurrence**).

Database Schema Vs. Database State

- **Database State:** Refers to the content of a database at a moment in time.
- **Initial Database State:** Refers to the database when it is loaded
- **Valid State:** A state that satisfies the structure and constraints of the database.
- **Distinction**
  - The **database schema** changes *very infrequently*. The **database state** changes *every time the database is updated.*
  - **Schema** is also called **intension**, whereas **state** is called **extension**.

Database Schema

- The term schema is used to represent the plan of the database.
- Description of the database is database schema.
- The actual data present in the database may change from time to time but the plan does not change.

Database State

- The data in the database at a particular moment in time is called database state. It is also called **instance / snapshot.** It is also called current set of occurrences.

Schema Diagram(conventions for representing schemas as Diagrams)

- A displayed schema is called schema diagram

Example of a Database Schema

STUDENT

Name	Student_number	Class	Major

COURSE

Course_name	Course_number	Credit_hours	Department

PREREQUISITE

Course_number	Prerequisite_number

SECTION

Section_identifier	Course_number	Semester	Year	Instructor

GRADE_REPORT

Student_number	Section_identifier	Grade

Figure 2.1
Schema diagram for the database in Figure 1.2.

Example of a database state

COURSE

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

SECTION

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

GRADE_REPORT

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

PREREQUISITE

Course_number	Prerequisite_number
CS3380	CS3320
CS3380	MATH2410
CS3320	CS1310

Figure 1.2
A database that stores student and course information.

BOOK

ISBN	Book_title	Category	Price	Copyright_date	Year	Page_count	P_ID

PUBLISHER

P_ID	Pname	Address	State	Phone	Email_id

AUTHOR

A_ID	Aname	City	State	Zip	Phone	Url

AUTHOR_BOOK

A_ID	ISBN

REVIEW

R_ID	ISBN	Rating

Three-Schema Architecture

- Proposed to support DBMS characteristics of:

  - **Program-data independence**.

    Data Independence is the property of DBMS that helps you to change the Database schema at one level of a database system without requiring to change the schema at the next higher level.

  - Support of **multiple views** of the data.

Three Schema Architecture and Data Independence

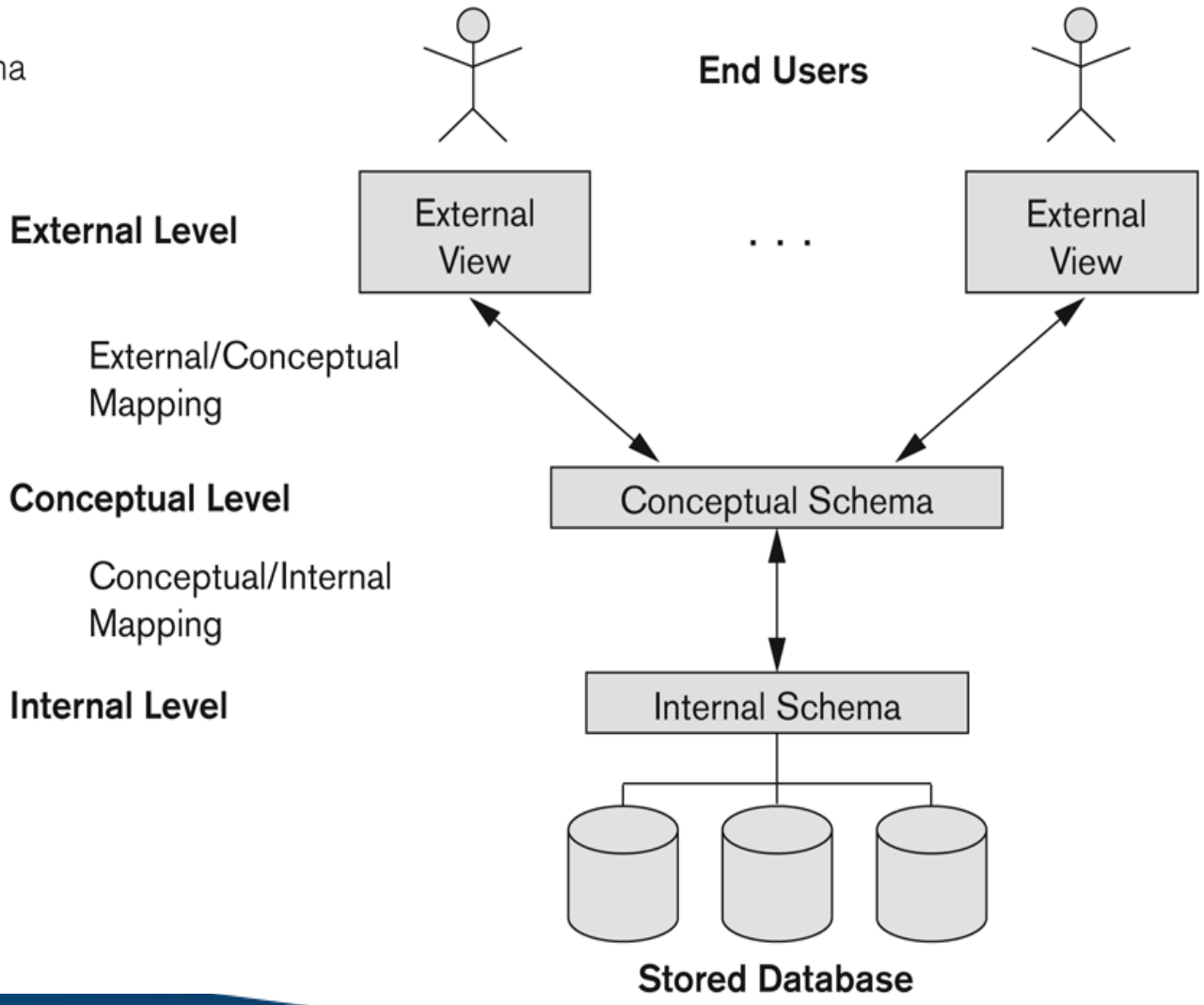
- Three Schema Architecture is an architecture for db systems that helps to achieve the characteristics self-describing, insulation between program and data, support of multiple-user views of database approach.
- Goal of three schema architecture – is to separate the user applications from the physical database.
- Schemas can be defined at the following 3 levels:
- Internal Schema: The internal level has an internal schema which describes the physical storage structure of the database. It describes the complete details of data storage and access paths for the database. The Internal Schema uses the physical data model.
- Conceptual Schema: The conceptual level has the conceptual schema. This level hides the details of physical storage structure and concentrates on describing entities, datatypes, relationships, user operations and constraints. A representational data model is used to describe the conceptual schema when database is implemented based on conceptual schema design in a high-level data model.

DBMS Architecture

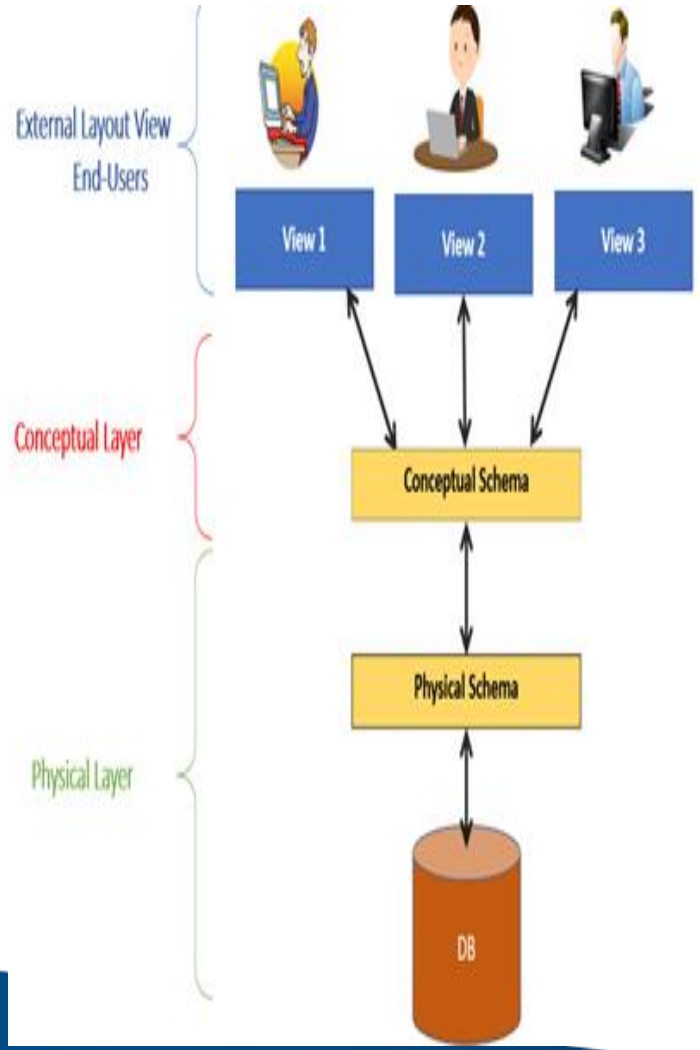
- External Schema or User Views: The external or view level has external schema. Each external schema describes the part of the database that a particular user group is interested in and hides the rest of the database from the user group.
- The three schemas are only descriptions of data; the actual data is stored at the physical level only.

Three-Schema Architecture

Figure 2.2
The three-schema architecture.

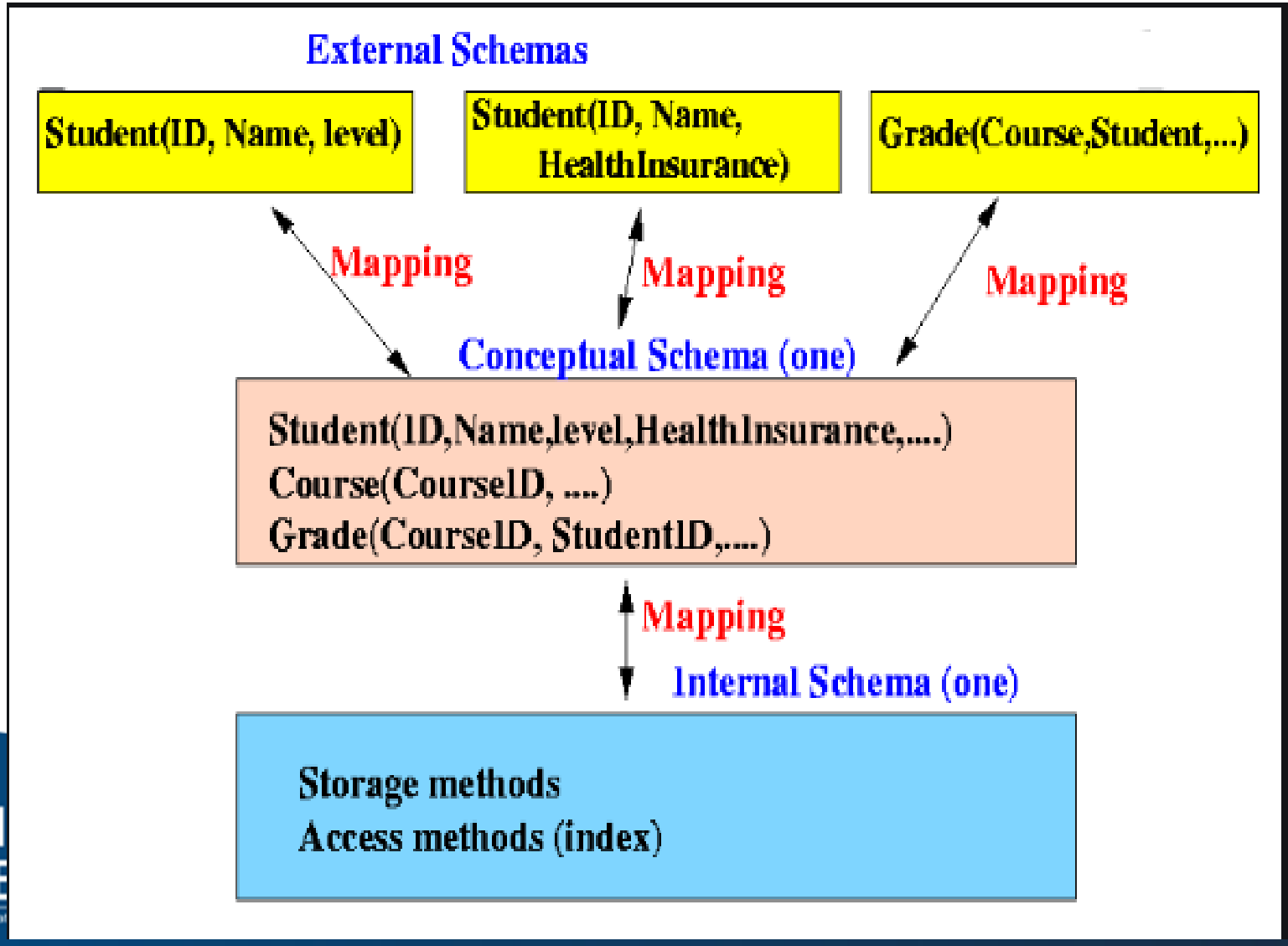


Ex:



Type of Schema	Implementation
External Schema	**View 1**: Course info(cid:int,cname:string) **View 2**: studeninfo(id:int. name:string)
Conceptual Shema	Students(id: int, name: string, login: string, age: integer) Courses(id: int, cname.string, credits:integer) Enrolled(id: int, grade:string)
Physical Schema	•Relations stored as unordered files. •Index on the first column of Students.

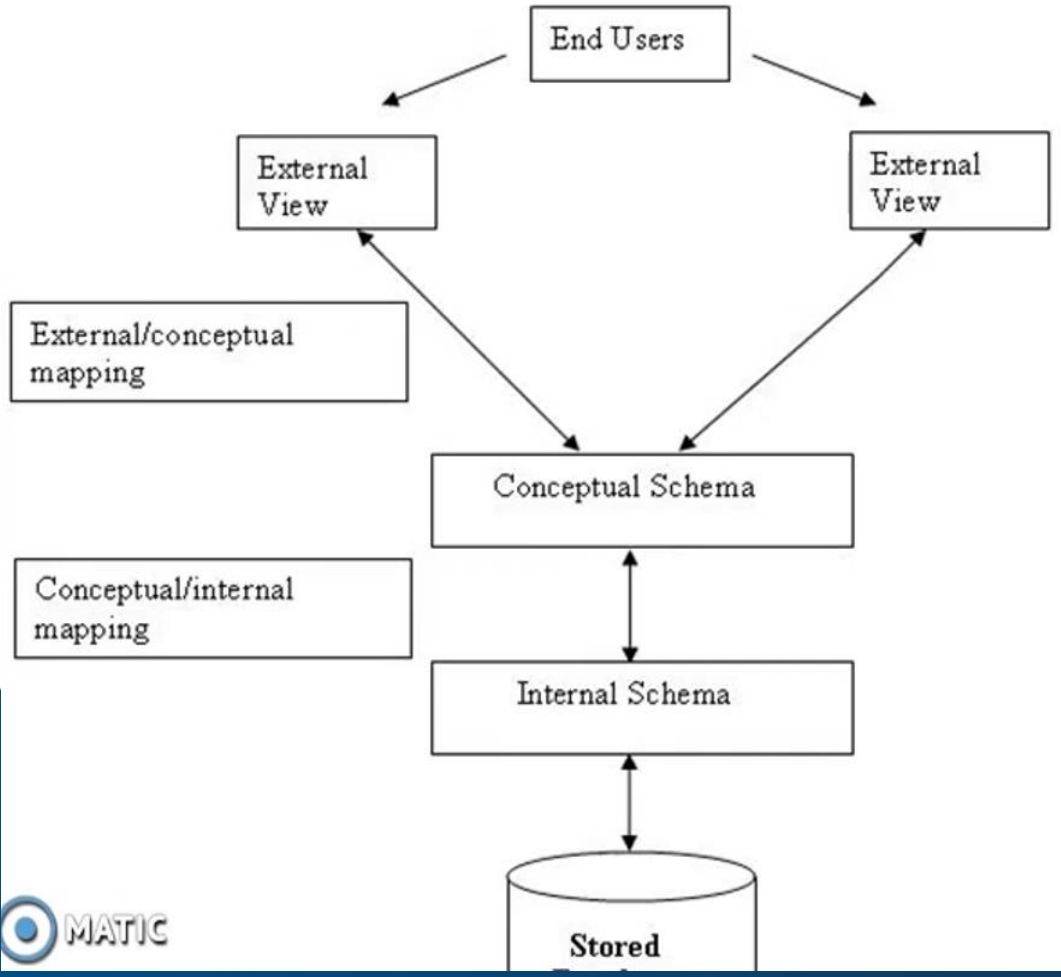
DBMS Architecture



Three-Schema Architecture- An Example

The Bank ATM example

The Three-Schema Architecture



1. Insert card in to machine
2. Provide card details pin etc
3. Specify amount to be withdrawn.
4. Machine does processing.
5. Transaction completed, database updated.

DBMS Architecture

- The DBMS must transform a request specified on an external schema into a request against the conceptual schema, and then into a request on the internal schema for processing over the stored database.
- If the request is a database retrieval, the data extracted from the stored database must be reformatted to match the user's external view.
- The processes of transforming requests and results between levels are called **mappings.**

Data Independence

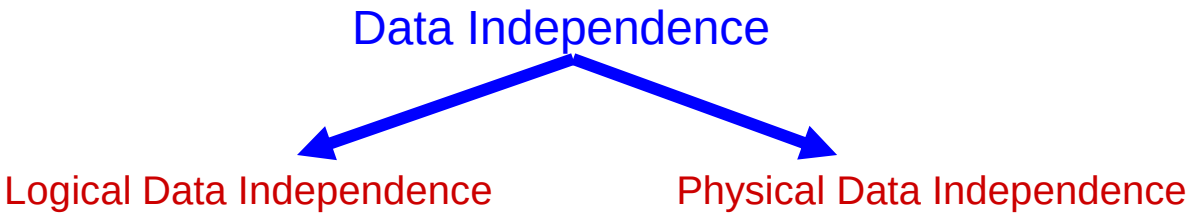
When a schema at a lower level is changed, only the **mappings** between this schema and higher-level schemas need to be changed in a DBMS that fully supports data independence.

The higher-level schemas themselves are *unchanged*. Hence, the application programs need not be changed since they refer to the external schemas.

Data Independence

The three schema architecture can be used to explain the concept of Data Independence.

- Data Independence can be defined as the capacity to change the schema at one level of a database without having to change it at the other level.



Data independence

Logical data independence

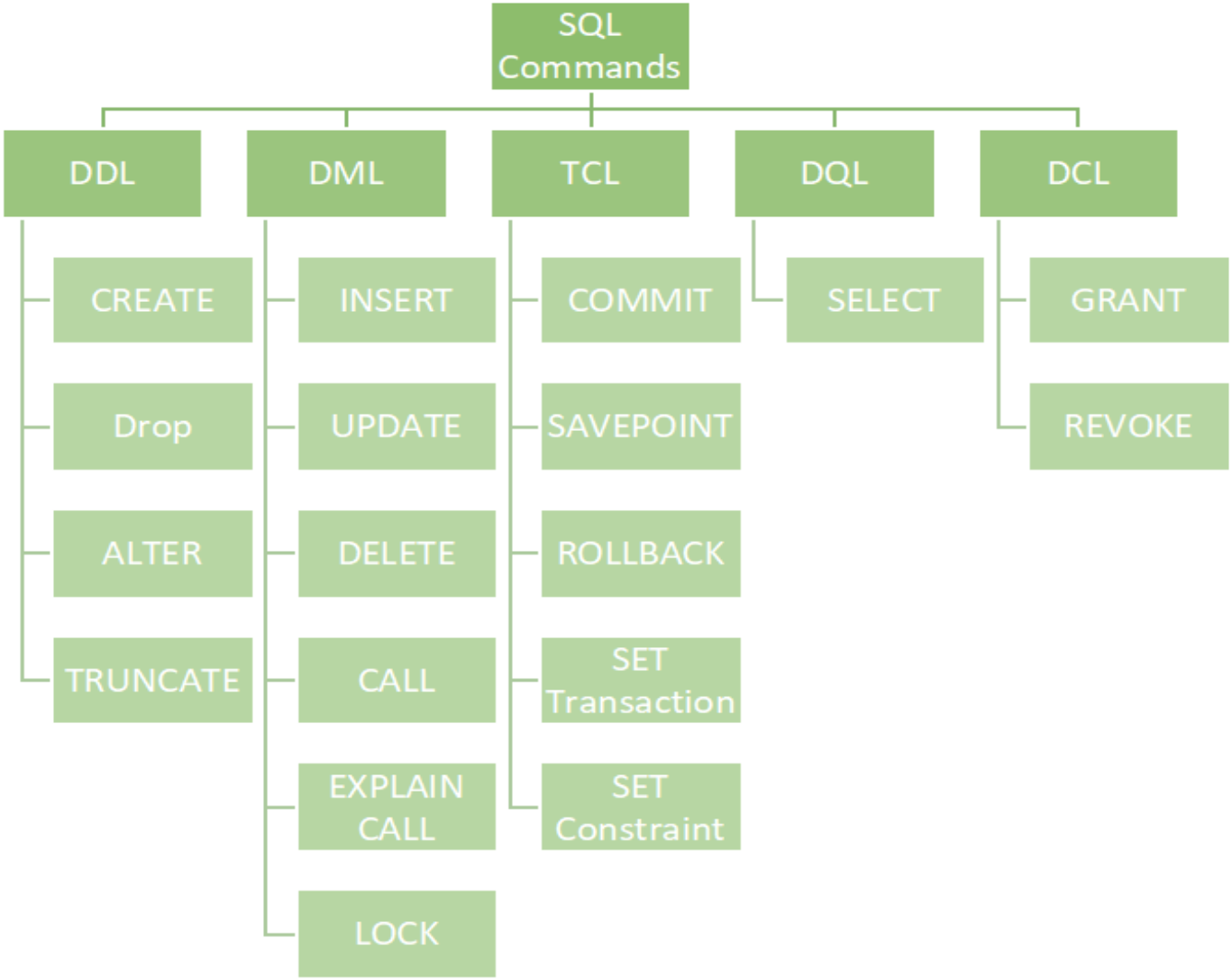
- It is the capacity to change the conceptual schema without having to change external schemas or application programs.
- Usually done when logical structure of database is altered.

Physical data independence

- It is the capacity to change the internal schema without having to change the conceptual schema. Hence, the external schemas need not be changed as well.
- The ability to modify the physical schema without causing application programs to be rewritten
- Modifications at this level are usually to improve the performance of retrieval or update.

DBMS Languages

- **Data Definition Language** (**DDL**): Used by the DBA and database designers to specify the *conceptual schema* of a database. In many DBMSs, the DDL is also used to define internal and external schemas (views). In some DBMSs, separate **storage definition language** (**SDL**) and **view definition language** (**VDL**) are used to define internal and external schemas.
- **Data Manipulation Language** (**DML**): Used to specify database retrievals and updates.
  - DML commands (**data sublanguage**) can be *embedded* in a general-purpose programming language (**host language**), such as COBOL, C or an Assembly Language.
  - Alternatively, *stand-alone* DML commands can be applied directly (**query language**).


SQL Commands
DDL
DML
TCL
DQL
DCL
CREATE
Drop
ALTER
TRUNCATE
INSERT
UPDATE
DELETE
CALL
EXPLAIN CALL
LOCK
COMMIT
SAVEPOINT
ROLLBACK
SET Transaction
SET Constraint
SELECT
GRANT
REVOKE

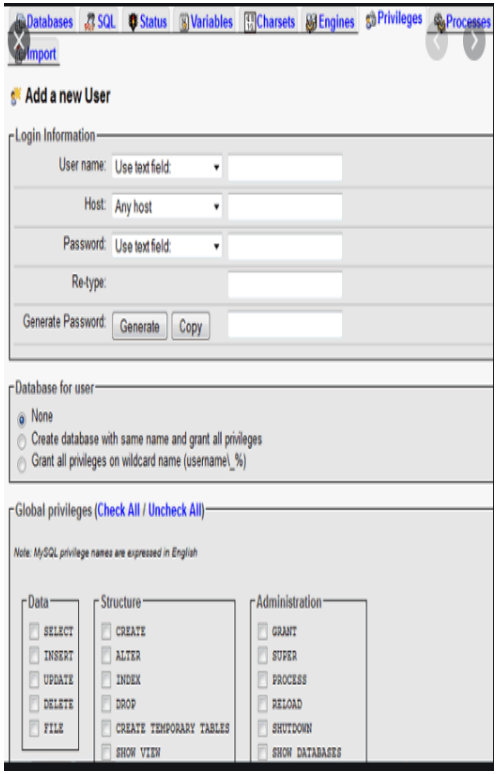
DBMS Interfaces

- Stand-alone query language interfaces.
- Programmer interfaces for embedding DML in programming languages:
  - Pre-compiler Approach
  - Procedure (Subroutine) Call Approach
- User-friendly interfaces:
  - Menu-based, popular for browsing on the web
  - Forms-based, designed for naïve users
  - Graphics-based (Point and Click, Drag and Drop etc.)
  - Natural language: requests in written English
  - Combinations of the above

Other DBMS Interfaces

- Speech Input and Output
- Web Browser as an interface
- Parametric interfaces (e.g., bank tellers) using function keys.
- Interfaces for the DBA:
  - Creating accounts, granting authorizations
  - Setting system parameters
  - Changing schemas or access path

DBMS Interfaces



Database System Utilities

- To perform certain functions such as:
  - *Loading* data stored in files into a database. Includes data conversion tools.
  - *Backing up* the database periodically on tape.
  - *Reorganizing* database file structures.
  - *Report generation* utilities.
  - *Performance monitoring* utilities.
  - Other functions, such as *sorting*, *user monitoring*, *data compression*, etc.

Data Modelling using Entities and Relationships

ER Diagram

- The ER or (Entity Relational Model) is a high-level conceptual data model diagram. Entity-Relation model is based on the notion of real-world entities and the relationship between them.
- Helps you to define terms related to entity relationship modeling
- Provide a preview of how all your tables should connect, what fields are going to be on each table
- Helps to describe entities, attributes, relationships
- ER diagrams are translatable into relational tables which allows you to build databases quickly
- ER diagrams can be used by database designers as a blueprint for implementing data in specific software applications
- The database designer gains a better understanding of the information to be contained in the database with the help of ERP diagram
- ERD is allowed you to communicate with the logical structure of the database to users

Components of the ER Diagram

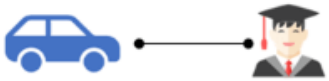
This model is based on three basic concepts:

- Entities Attributes Relationships

Entity

Person,place,object,event or concept about which data is to be maintained

Example: Car, Student



Relation

Association between the instances of one or more entity types

Example: Blue Car Belongs to Student Jack

Attribute

Property or characteristic of an entity

Example: Color of car Entity Name of Student Entity

Example COMPANY Database

We need to create a database schema design based on the following (simplified) R**equirements** of the COMPANY Database:

- The company is organized into DEPARTMENTs. Each department has a name, number and an employee who *manages* the department. We keep track of the start date of the department manager. A department may have several locations.
- Each department *controls* a number of PROJECTs. Each project has a unique name, unique number and is located at a single location.
- We store each EMPLOYEE's social security number, address, salary, sex, and birthdate.
  - Each employee *works for* one department but may *work on* several projects.
  - We keep track of the number of hours per week that an employee currently works on each project.
  - We also keep track of the *direct supervisor* of each employee.

- Each employee may have a number of DEPENDENTs.
  - For each dependent, we keep track of their name, sex, birthdate, and relationship to the employee.
- **Entities and Attributes**
- Entities are specific objects or things in the mini-world that are represented in the database.
  - For example the EMPLOYEE John Smith, the Research DEPARTMENT, the ProductX PROJECT
- Attributes are properties used to describe an entity.
  - For example an EMPLOYEE entity may have the attributes Name, SSN, Address, Sex, BirthDate
- A specific entity will have a value for each of its attributes.
  - For example a specific employee entity may have Name='John Smith', SSN='123456789', Address ='731, Fondren, Houston, TX', Sex='M', BirthDate='09-JAN-55‘
- Each attribute has a *value set* (or data type) associated with it – e.g. integer, string, subrange, enumerated type, …

Types of Attributes

Types of Attributes	Description
Simple attribute	Simple attributes can't be divided any further. For example, a student's contact number. It is also called an atomic value.
Composite attribute	It is possible to break down composite attribute. For example, a student's full name may be further divided into first name, second name, and last name.
Derived attribute	This type of attribute does not include in the physical database. However, their values are derived from other attributes present in the database. For example, age should not be stored directly. Instead, it should be derived from the DOB of that employee.
Multivalued attribute	Multivalued attributes can have more than one values. For example, a student can have more than one mobile number, email address, etc.

Types of Attributes Examples

- Simple
  - Each entity has a single atomic value for the attribute. For example, SSN.
- Composite
  - The attribute may be composed of several components. For example, Address (Apt#, House#, Street, City, State, ZipCode, Country) or Name (FirstName, MiddleName, LastName). Composition may form a hierarchy where some components are themselves composite.
- Multi-valued
  - An entity may have multiple values for that attribute. For example, Color of a CAR or PreviousDegrees of a STUDENT. Denoted as {Color} or {PreviousDegrees}.

Types of Attributes (2)

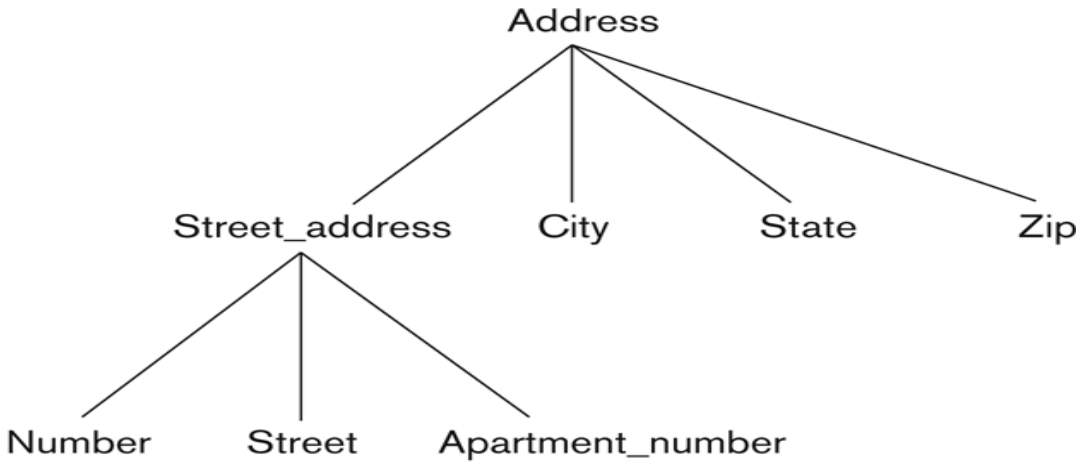


Figure 3.4
A hierarchy of composite attributes.

- In general, composite and multi-valued attributes may be nested arbitrarily to any number of levels although this is rare.
- For example, PreviousDegrees of a STUDENT is a composite multi-valued attribute denoted by {PreviousDegrees (College, Year, Degree, Field)}.

Entity Types and Key Attributes

- Entities with the same basic attributes are grouped or typed into an entity type.
  - For example, the EMPLOYEE entity type or the PROJECT entity type.
- An attribute of an entity type for which each entity must have a unique value is called a key attribute of the entity type.
  - For example, SSN of EMPLOYEE.
- A key attribute may be composite.
  - For example, VehicleTagNumber is a key of the CAR entity type with components (Number, State).
- An entity type may have more than one key.
  - For example, the CAR entity type may have two keys:
  - VehicleIdentificationNumber (popularly called VIN) and
  - VehicleTagNumber (Number, State), also known as license_plate number.

Entity Type CAR with two keys and a corresponding Entity Set

(a)

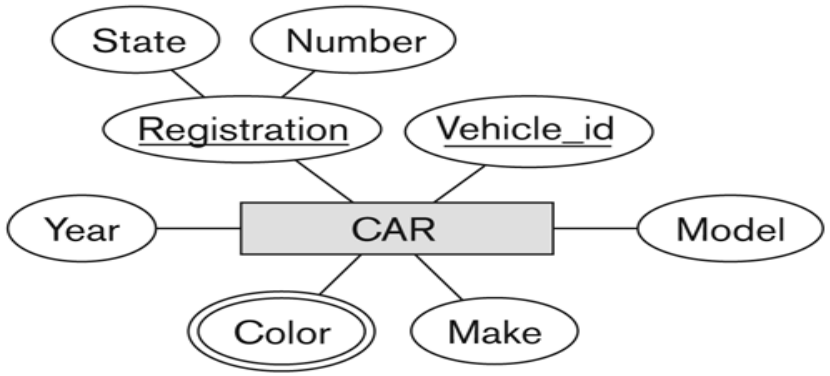


Figure 3.7
The CAR entity type with two key attributes, Registration and Vehicle_id. (a) ER diagram notation. (b) Entity set with three entities.

(b)

CAR
Registration (Number, State), Vehicle_id, Make, Model, Year, {Color}

CAR_1
((ABC 123, TEXAS), TK629, Ford Mustang, convertible, 2004 {red, black})

CAR_2
((ABC 123, NEW YORK), WP9872, Nissan Maxima, 4-door, 2005, {blue})

CAR_3
((VSY 720, TEXAS), TD729, Chrysler LeBaron, 4-door, 2002, {white, blue})

Entity Set

- Each entity type will have a collection of entities stored in the database is called the **entity set**
- Previous slide shows three CAR entity instances in the entity set for CAR
- Same name (CAR) used to refer to both the entity type and the entity set
- Entity set is the current *state* of the entities of that type that are stored in the database

Initial Design of Entity Types for the COMPANY Database Schema

- Based on the requirements, we can identify four initial entity types in the COMPANY database:
  - DEPARTMENT
  - PROJECT
  - EMPLOYEE
  - DEPENDENT
- Their initial design is shown on the following slide
- The initial attributes shown are derived from the requirements description

Initial Design of Entity Types:
EMPLOYEE, DEPARTMENT, PROJECT, DEPENDENT

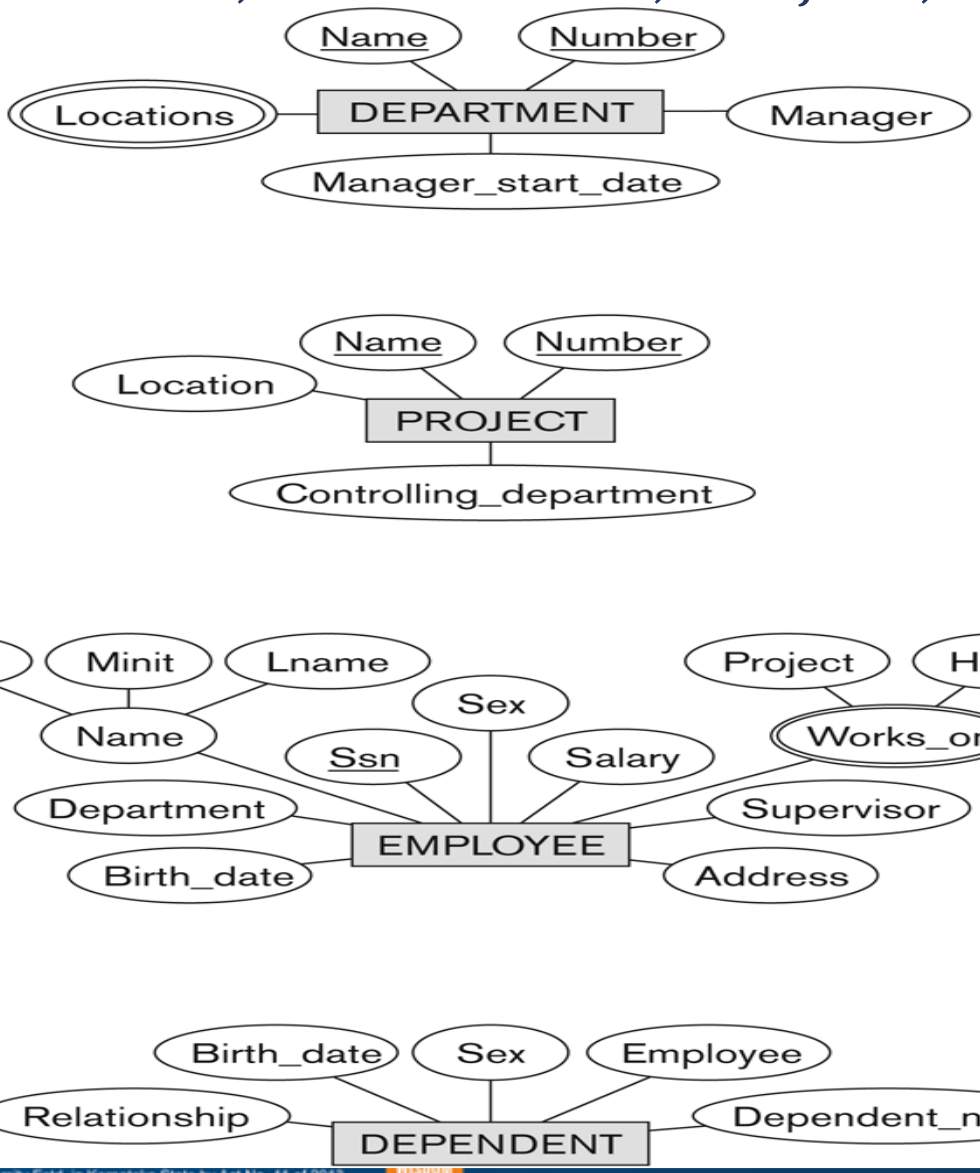


Figure 3.8
Preliminary design of entity types for the COMPANY database. Some of the shown attributes will be refined into relationships.

Refining the initial design by introducing **relationships**

- ER model has **three** main concepts:
  - Entities (and their entity types and entity sets)
  - Attributes (simple, composite, multivalued)
  - Relationships (and their relationship types and relationship sets)
- A **relationship** relates two or more distinct entities with a specific meaning.
  - For example, EMPLOYEE John Smith *works on* the ProductX PROJECT, or EMPLOYEE Franklin Wong *manages* the Research DEPARTMENT.
- Relationships of the same type are grouped or typed into a **relationship type**.
  - For example, the WORKS_ON relationship type in which EMPLOYEEs and PROJECTs participate, or the MANAGES relationship type in which EMPLOYEEs and DEPARTMENTs participate.

Relationship instances of the WORKS_FOR N:1 relationship between EMPLOYEE and DEPARTMENT

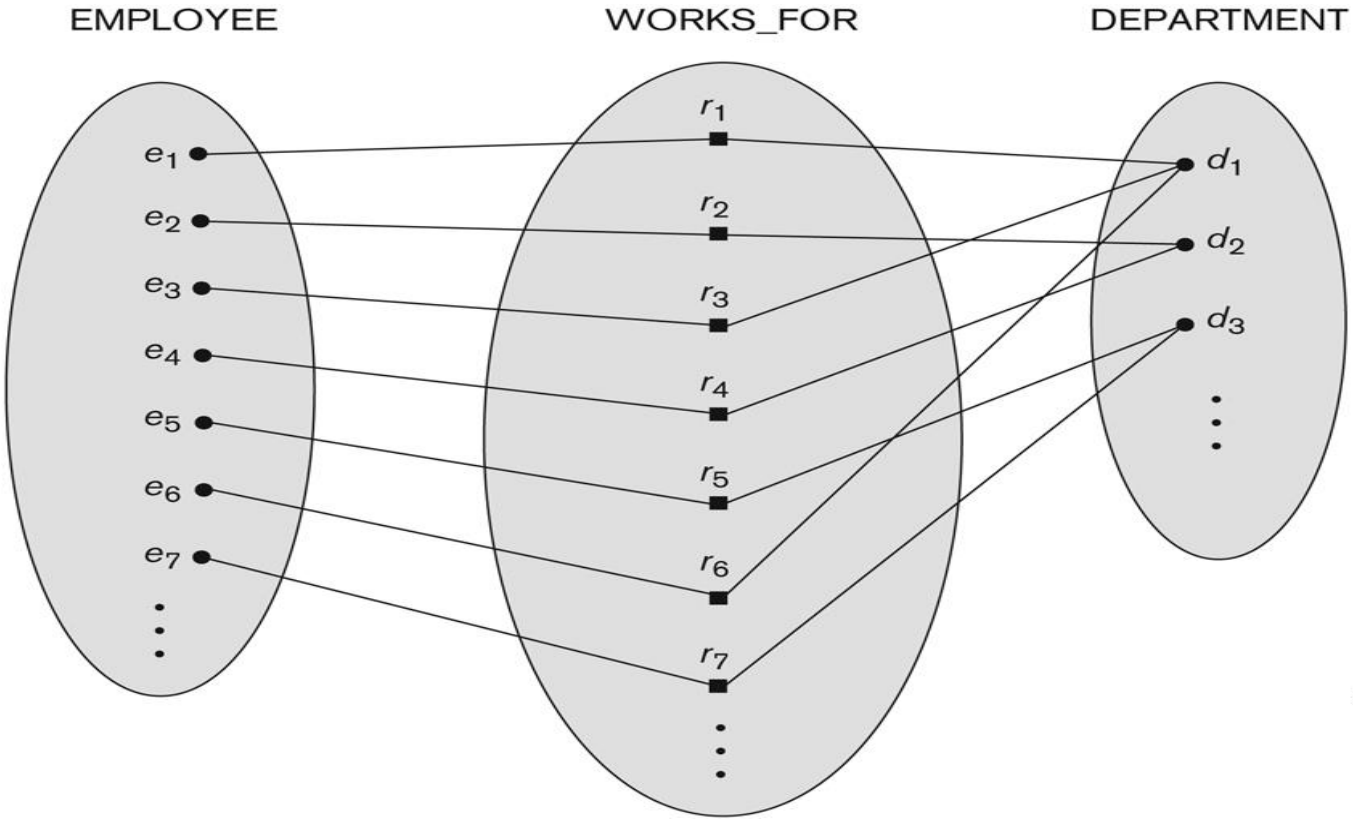


Figure 3.9
Some instances in the WORKS_FOR relationship set, which represents a relationship type WORKS_FOR between EMPLOYEE and DEPARTMENT.

Relationship instances of the M:N WORKS_ON relationship between EMPLOYEE and PROJECT

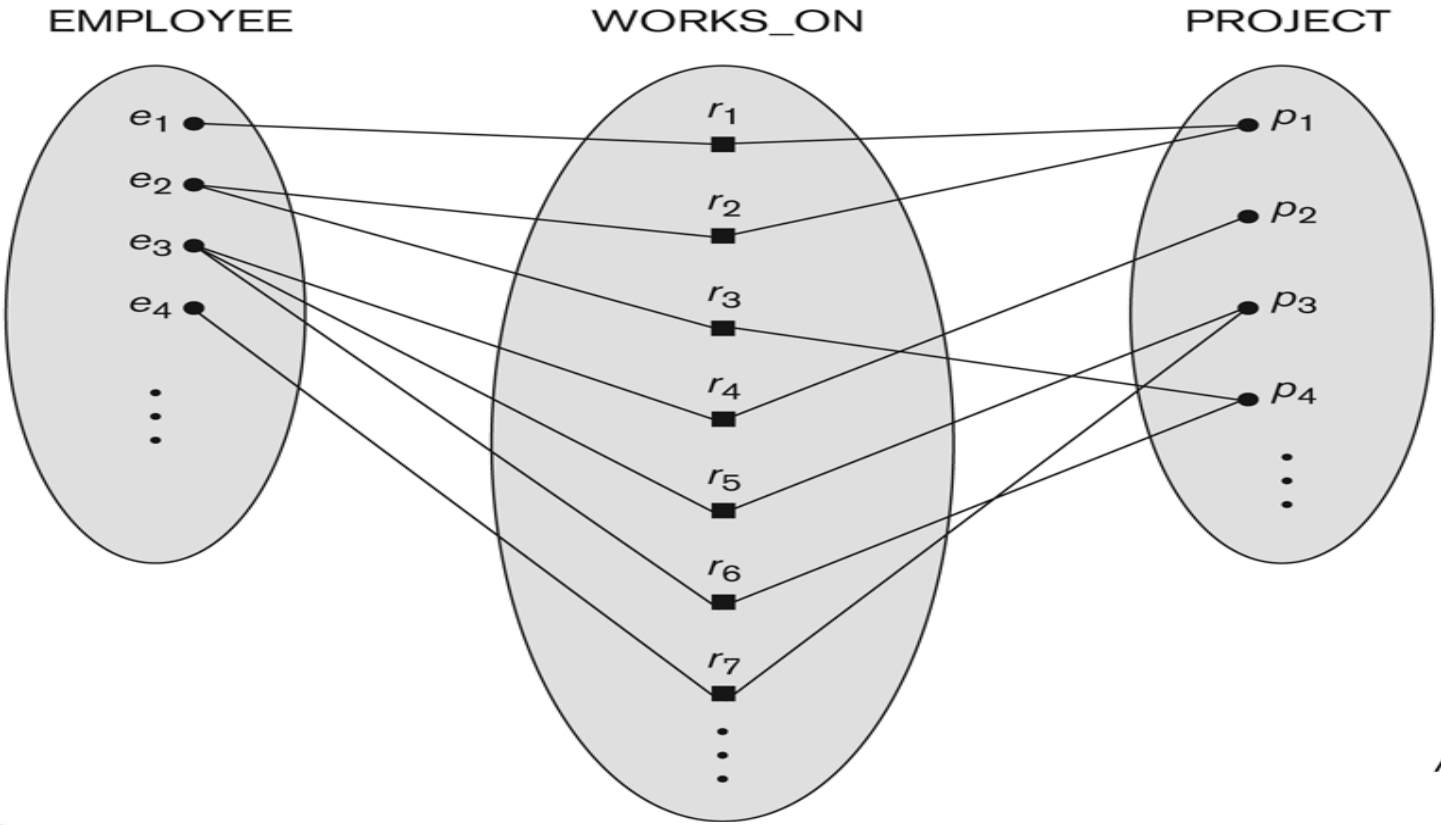


Figure 3.13
An M:N relationship, WORKS_ON.

- Relationship Type:
  - Is the schema description of a relationship
  - Identifies the relationship name and the participating entity types
  - Also identifies certain relationship constraints
- Relationship Set:
  - The current set of relationship instances represented in the database
  - The current *state* of a relationship type
- In ER diagrams, we represent the *relationship type* as follows:
  - Diamond-shaped box is used to display a relationship type
  - Connected to the participating entity types via straight lines

Refining the COMPANY database schema by introducing relationships

- By examining the requirements, six relationship types are identified
- All are *binary* relationships(degree 2)
- Listed below with their participating entity types:
  - WORKS_FOR (between EMPLOYEE, DEPARTMENT)
  - MANAGES (also between EMPLOYEE, DEPARTMENT)
  - CONTROLS (between DEPARTMENT, PROJECT)
  - WORKS_ON (between EMPLOYEE, PROJECT)
  - SUPERVISION (between EMPLOYEE (as subordinate),
  - EMPLOYEE (as supervisor))
  - DEPENDENTS_OF (between EMPLOYEE, DEPENDENT)

ER DIAGRAM

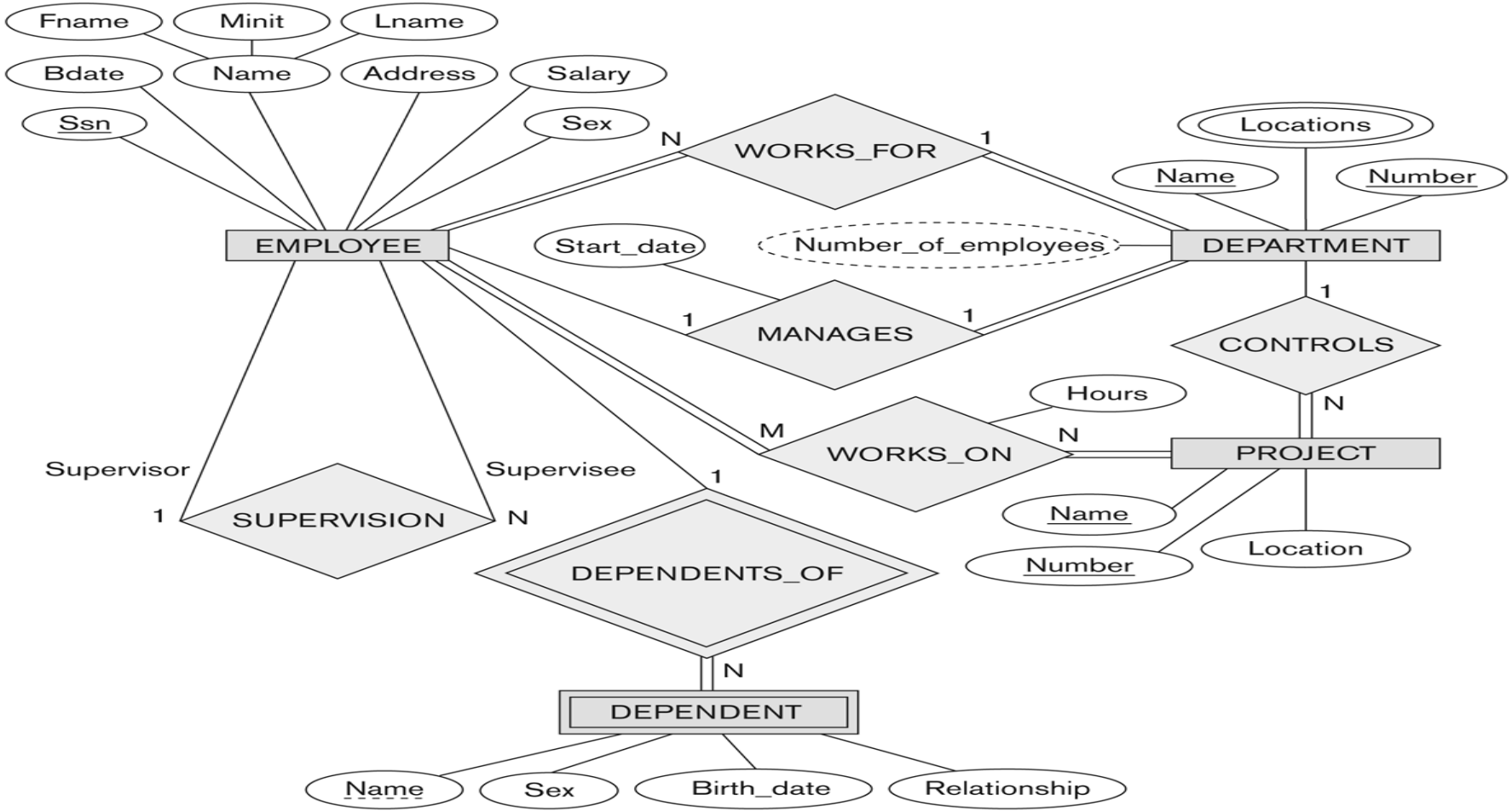


Figure 3.2
An ER schema diagram for the COMPANY database. The diagrammatic notation is introduced gradually throughout this chapter.

Recursive Relationship Type

- An relationship type whose with the same participating entity type in **distinct roles**
  - Example: the SUPERVISION relationship
- EMPLOYEE participates twice in two distinct roles:
  - supervisor (or boss) role
  - supervisee (or subordinate) role
- Each relationship instance relates two distinct EMPLOYEE entities:
  - One employee in *supervisor* role
  - One employee in *supervisee* role

Weak Entity Types

- An entity that does not have a key attribute
- A weak entity must participate in an identifying relationship type with an owner or identifying entity type
- Entities are identified by the combination of:
  - A partial key of the weak entity type
  - The particular entity they are related to in the identifying entity type
- **Example:**
  - A DEPENDENT entity is identified by the dependent's first name, *and* the specific EMPLOYEE with whom the dependent is related
  - Name of DEPENDENT is the *partial key*
  - DEPENDENT is a *weak entity type*
  - EMPLOYEE is its identifying entity type via the identifying relationship type DEPENDENT_OF

Constraints on Relationships

- Constraints on Relationship Types(Also known as ratio constraints)
- Cardinality Ratio (specifies *maximum* participation)
  - One-to-one (1:1)
  - One-to-many (1:N) or Many-to-one (N:1)
  - Many-to-many (M:N)
- Existence Dependency Constraint (specifies *minimum*
- participation) (also called participation constraint)
  - zero (optional participation, not existence-dependent)
  - one or more (mandatory participation, existence-dependent)

Many-to-one (N:1) RELATIONSHIP

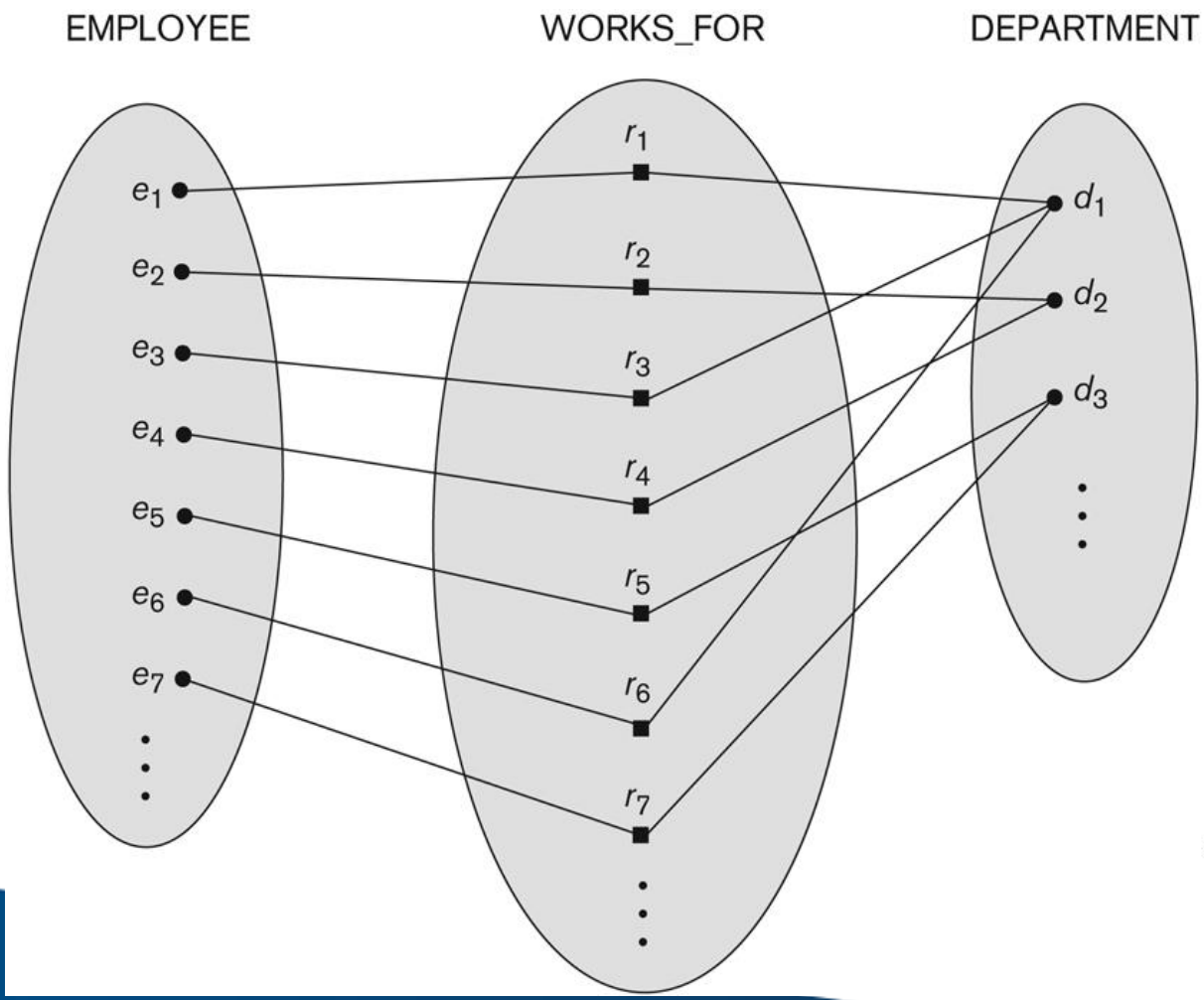


Figure 3.9
Some instances in the WORKS_FOR relationship set, which represents a relationship type WORKS_FOR between EMPLOYEE and DEPARTMENT.

Many-to-many (M:N) RELATIONSHIP

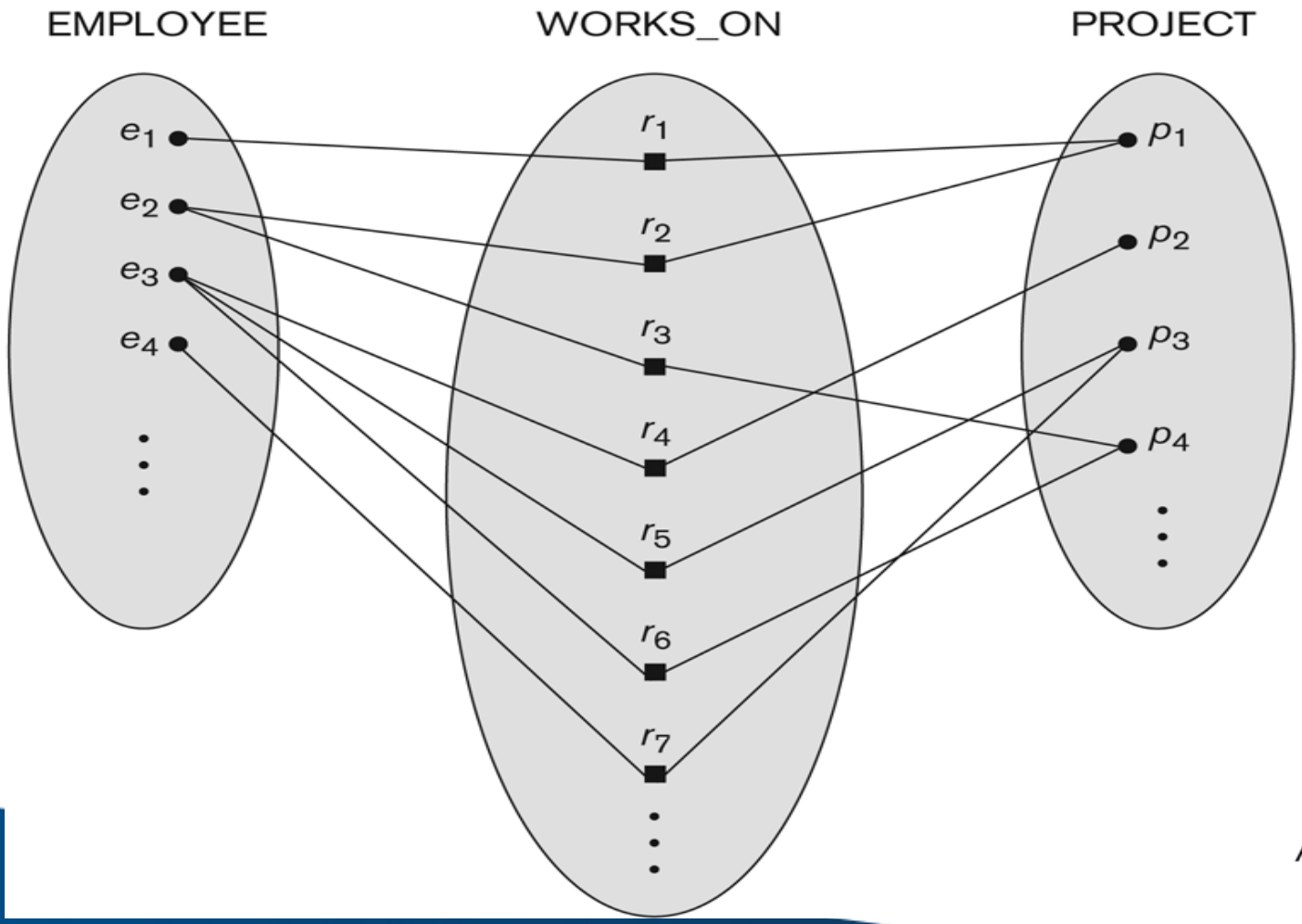


Figure 3.13
An M:N relationship, WORKS_ON.

A RECURSIVE RELATIONSHIP SUPERVISION

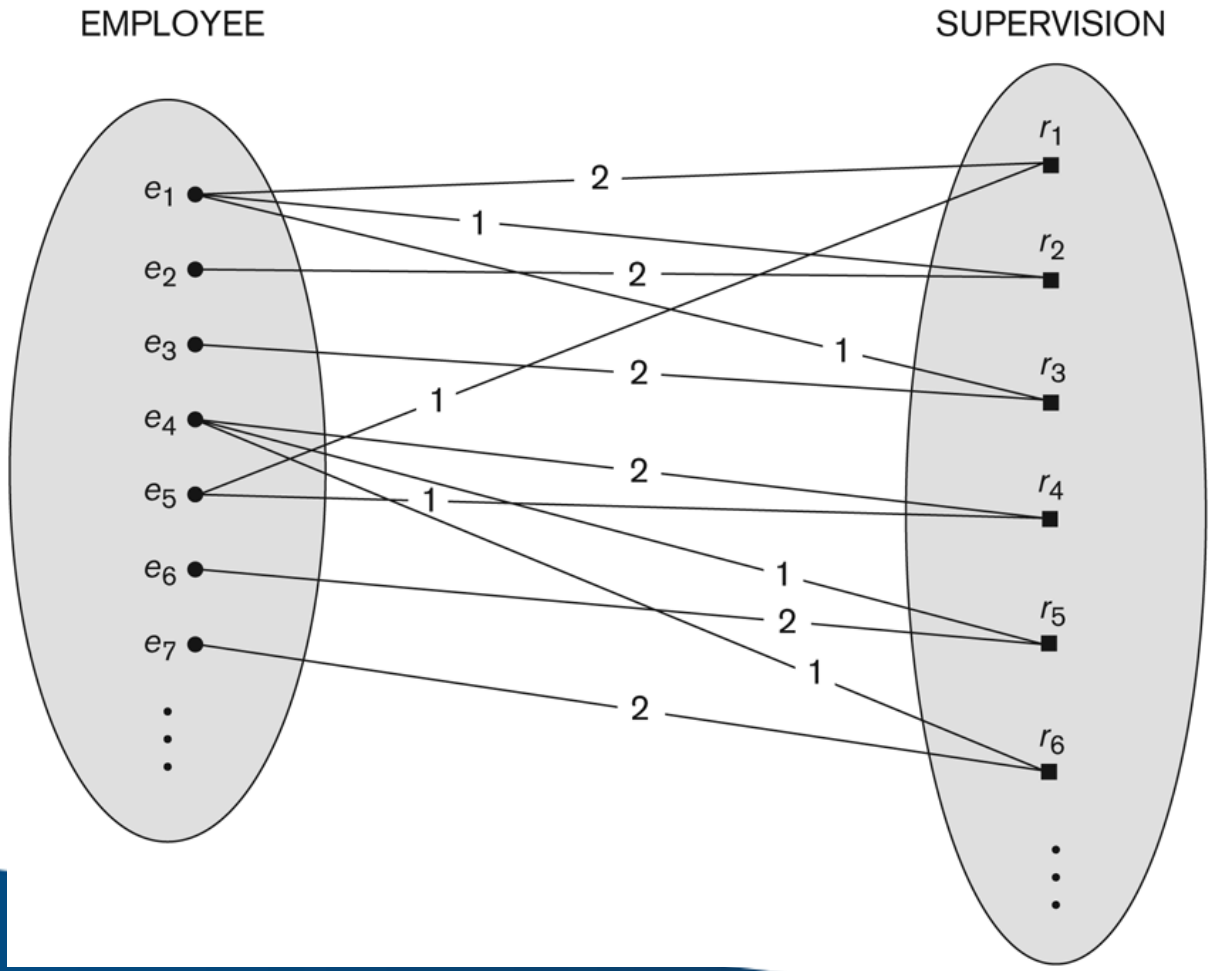
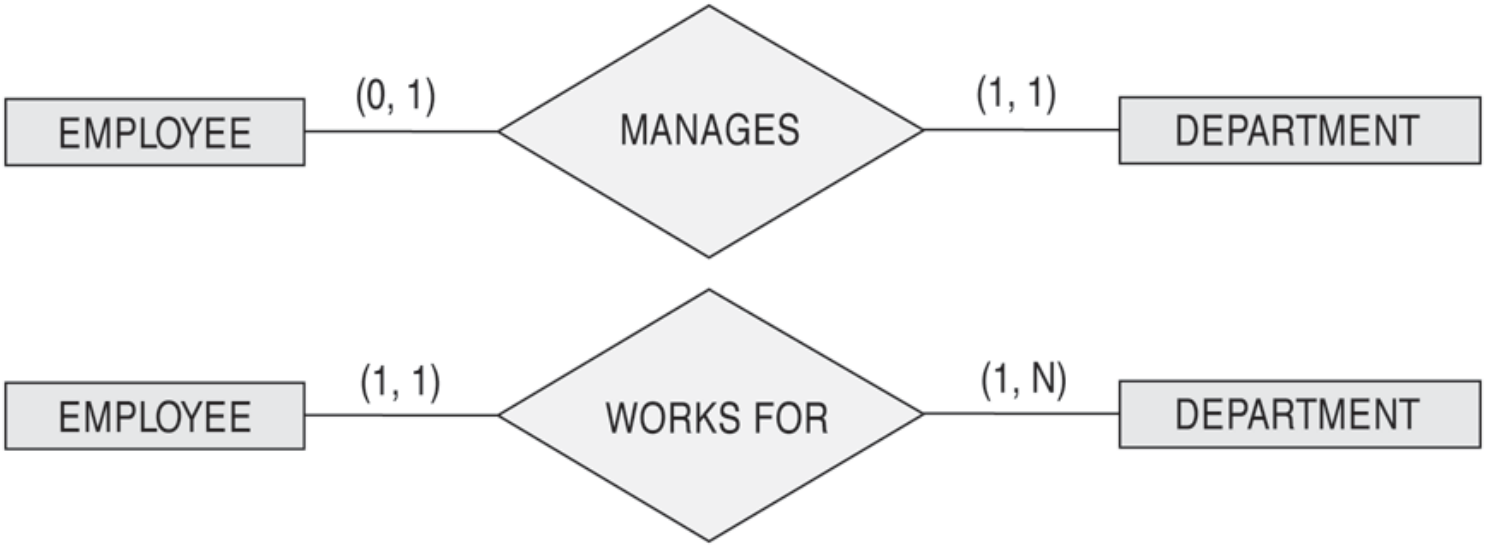


Figure 3.11
A recursive relationship SUPERVISION between EMPLOYEE in the *supervisor* role (1) and EMPLOYEE in the *subordinate* role (2).

Alternative (min, max) notation for relationship structural constraints:

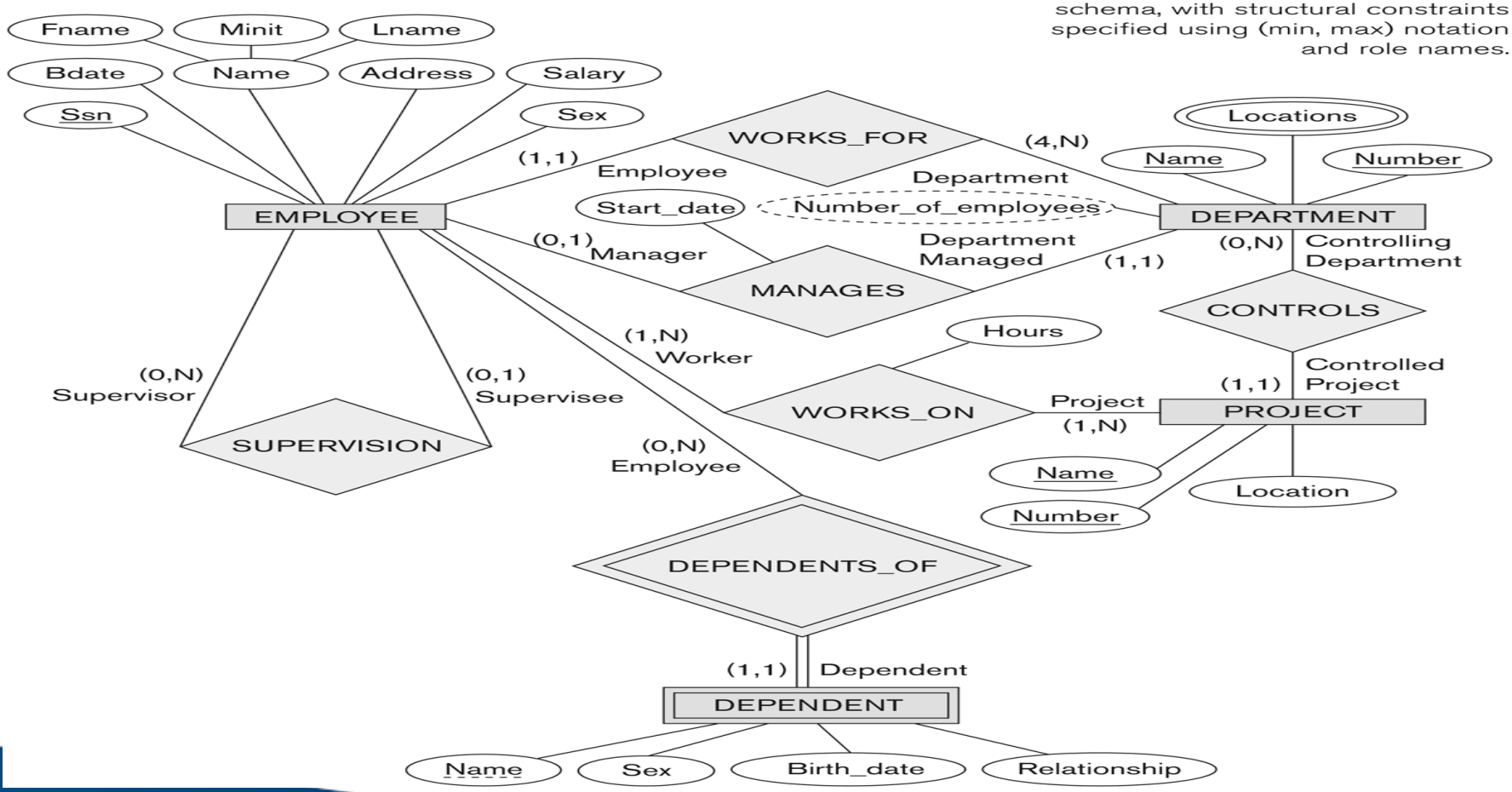
- Specified on each participation of an entity type E in a relationship type R
- Specifies that each entity e in E participates in at least *min* and at most *max* relationship instances in R
- Default(no constraint): min=0, max=n (signifying no limit)
- Must have min<=max, min>=0, max >=1
- Derived from the knowledge of mini-world constraints
- Examples
  - A department has exactly one manager and an employee can manage at most one department.
    - Specify (0,1) for participation of EMPLOYEE in MANAGES
    - Specify (1,1) for participation of DEPARTMENT in MANAGES
  - An employee can work for exactly one department but a department can have any number of employees.
    - Specify (1,1) for participation of EMPLOYEE in WORKS_FOR
    - Specify (0,n) for participation of DEPARTMENT in WORKS_FOR

The (min,max) notation for relationship constraints



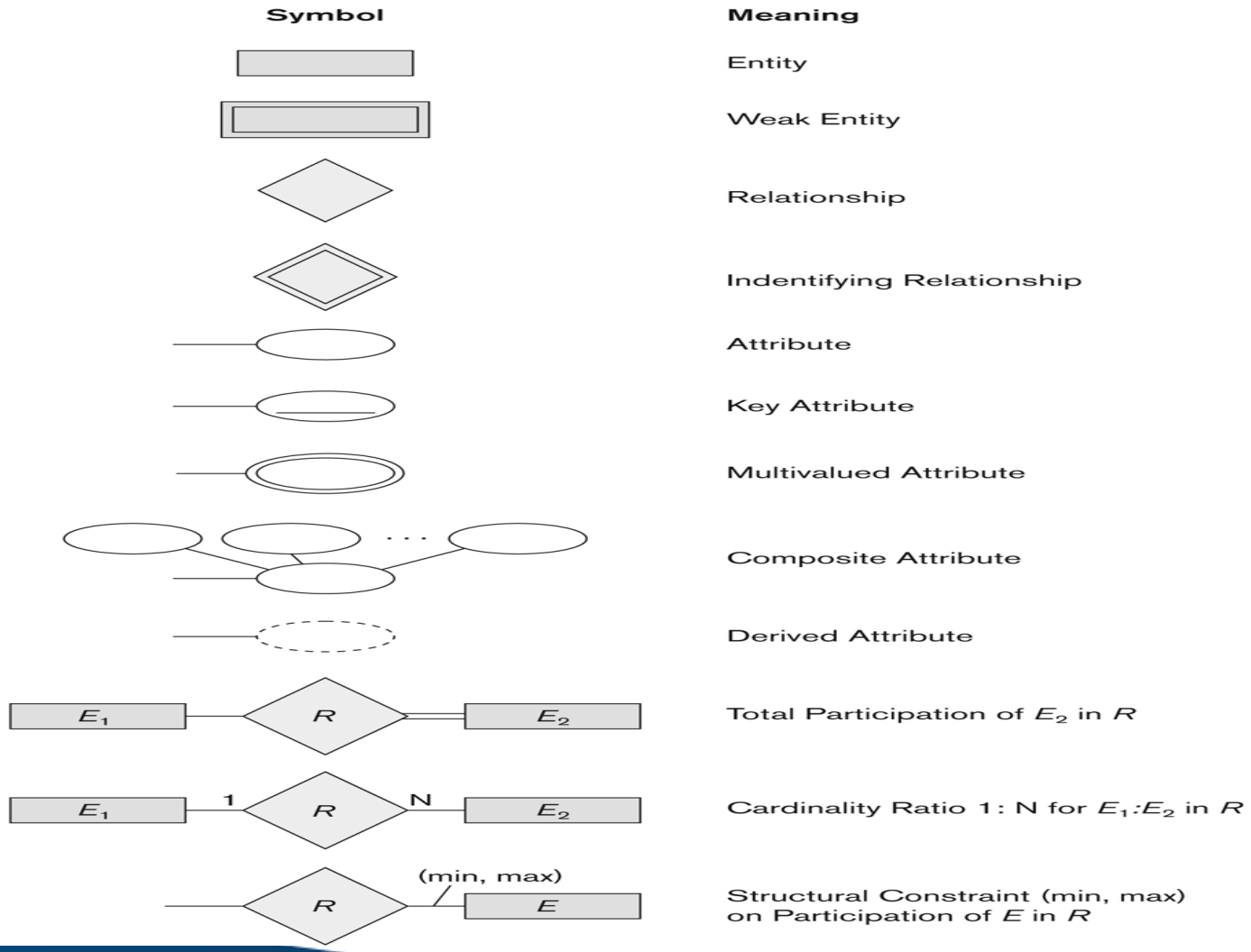
COMPANY ER Schema Diagram using (min, max) notation

Figure 3.15
ER diagrams for the company schema, with structural constraints specified using (min, max) notation and role names.



Summary of notation for ER diagrams

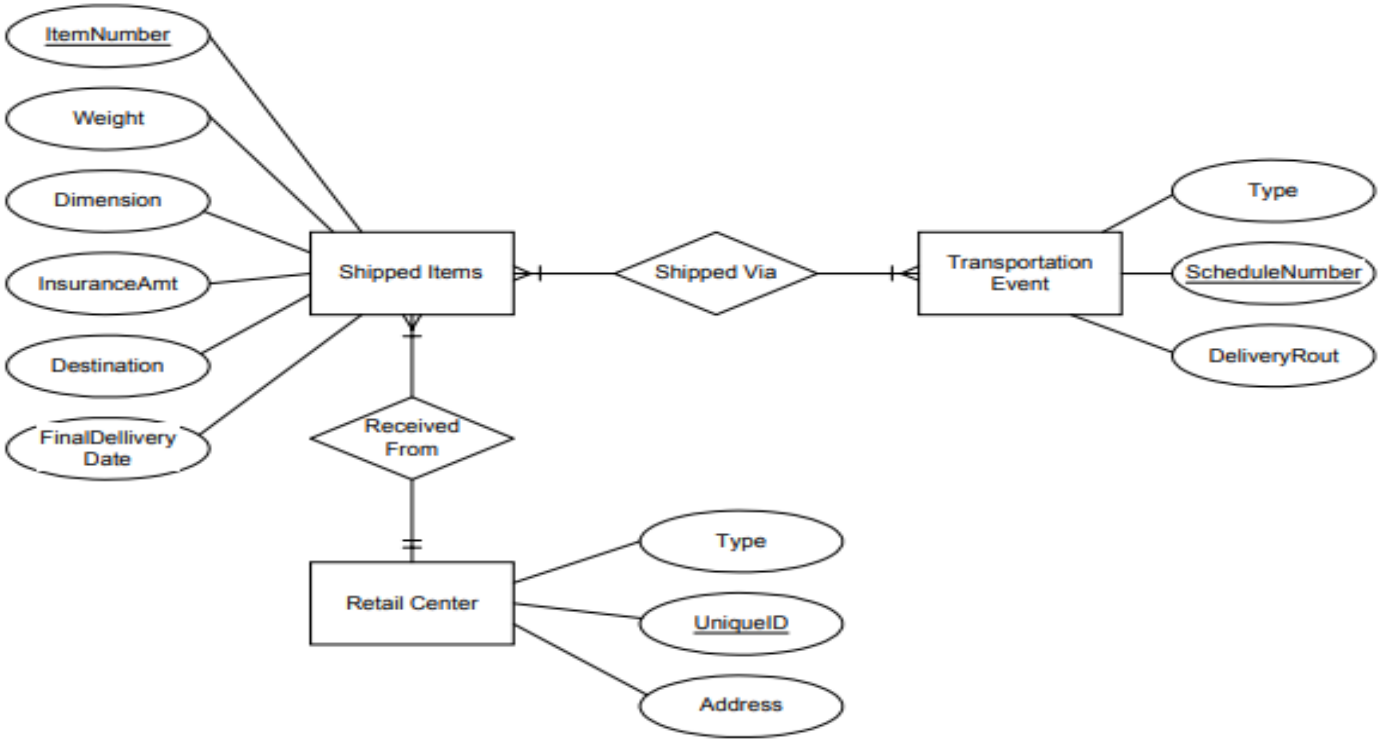
Figure 3.14
Summary of the notation for ER diagrams.



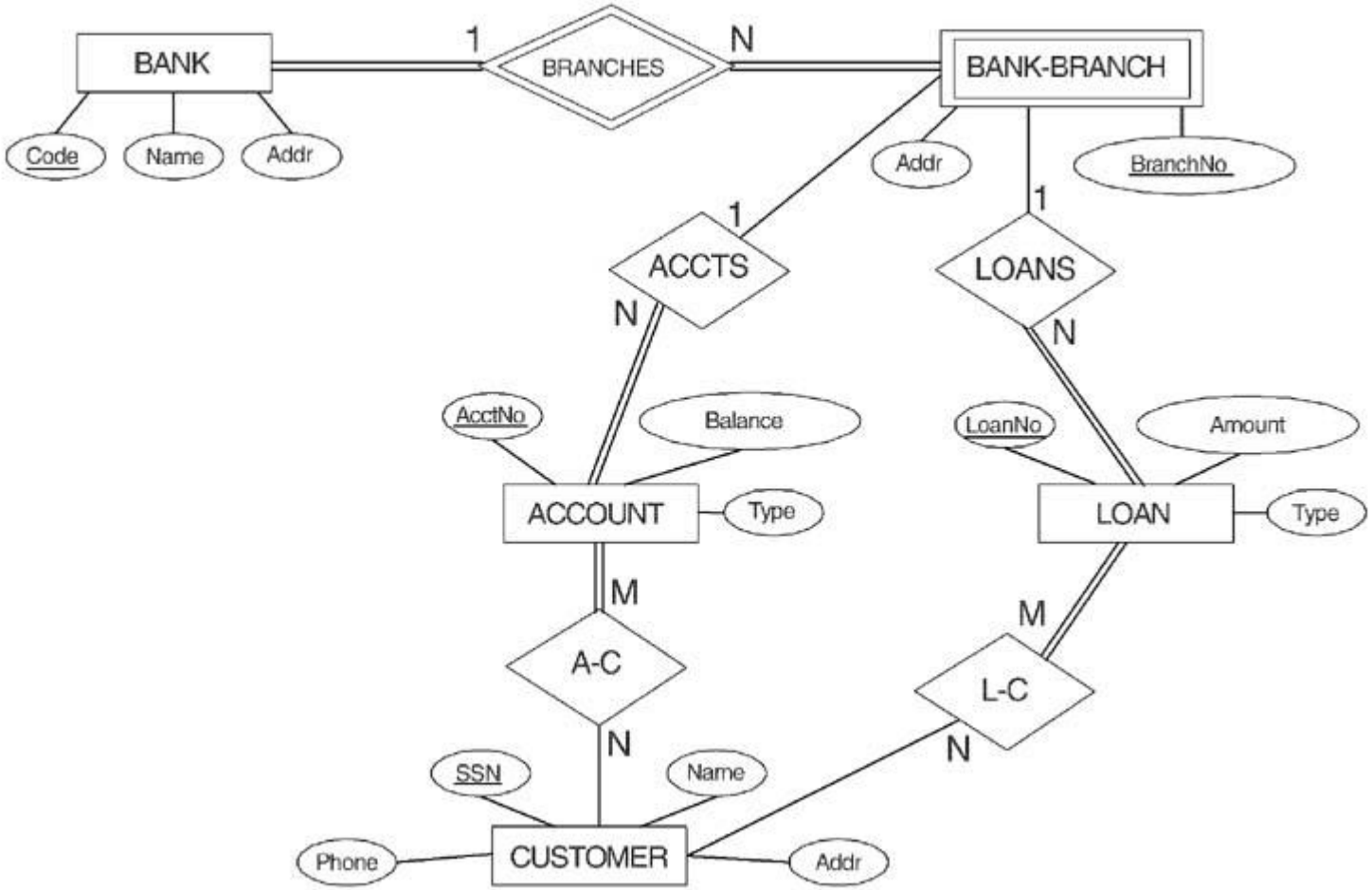
UPS prides itself on having up-to-date information on the processing and current location of each shipped item. To do this, UPS relies on a company-wide information system. Shipped items are the heart of the UPS product tracking information system. Shipped items can be characterized by item number (unique), weight, dimensions, insurance amount, destination, and final delivery date. Shipped items are received into the UPS system at a single retail center. Retail centers are characterized by their type, uniqueID, and address. Shipped items make their way to their destination via one or more standard UPS transportation events (i.e., flights, truck deliveries). These transportation events are characterized by a unique scheduleNumber, a type (e.g, flight, truck), and a deliveryRoute.

Please create an Entity Relationship diagram that captures this information about the UPS system. Be certain to indicate identifiers and cardinality constraints.

Solutions:


ItemNumber
Weight
Dimension
InsuranceAmt
Destination
FinalDellivery Date
Shipped Items
Shipped Via
Transportation Event
Type
ScheduleNumber
DeliveryRout
Received From
Retail Center
Type
UniqueID
Address

ER DIAGRAM FOR A BANK DATABASE



Relationships of Higher Degree

- Relationship types of degree 2 are called **binary**
- Relationship types of degree 3 are called **ternary** and of degree n are called **n-ary**
- In general, an n-ary relationship *is not* equivalent to n binary relationships
- Higher-order relationships discussed further in Chapter 4

Example of a ternary relationship

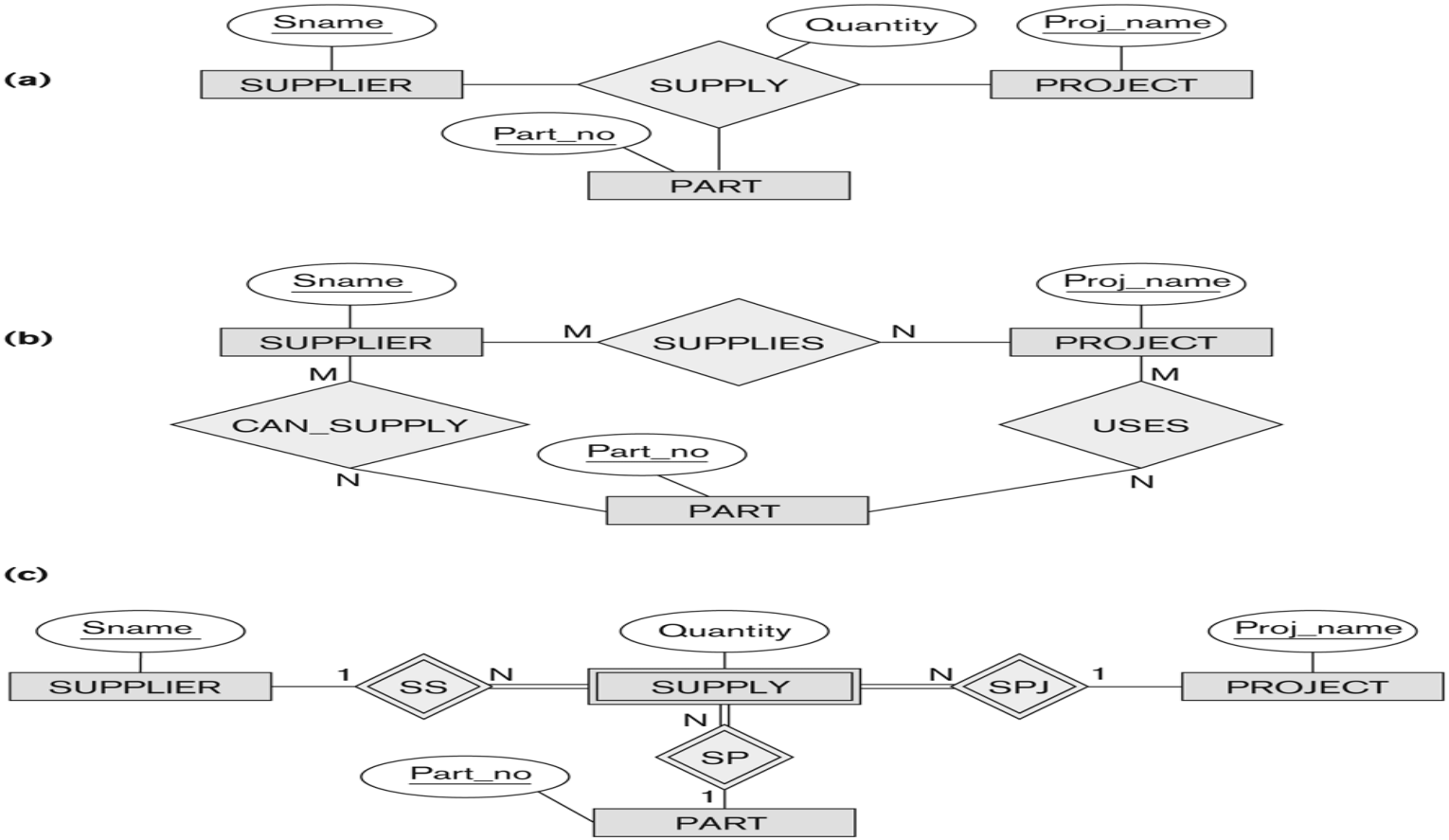


Figure 3.17
Ternary relationship types. (a) The SUPPLY relationship. (b) Three binary relationships not equivalent to SUPPLY. (c) SUPPLY represented as a weak entity type.

Some of the Currently Available Automated Database Design Tools

COMPANY	TOOL	FUNCTIONALITY
Embarcadero Technologies	ER Studio	Database Modeling in ER and IDEF1X
	DB Artisan	Database administration, space and security management
Oracle	Developer 2000/Designer 2000	Database modeling, application development
Popkin Software	System Architect 2001	Data modeling, object modeling, process modeling, structured analysis/design
Platinum (Computer Associates)	Enterprise Modeling Suite: Erwin, BPWin, Paradigm Plus	Data, process, and business component modeling
Persistence Inc.	Pwertier	Mapping from O-O to relational model
Rational (IBM)	Rational Rose	UML Modeling & application generation in C++/JAVA
Resolution Ltd.	Xcase	Conceptual modeling up to code maintenance

Query Languages

- Language in which user requests information from the database.
- Categories of languages
  - Procedural
  - Non-procedural, or declarative
- “Pure” languages:
  - Relational algebra
  - Tuple relational calculus
  - Domain relational calculus
- Pure languages form underlying basis of query languages that people use.

Relational Algebra

- The relational algebra is a *procedural* query language. Relational algebra is the basic set of operations for the relational model
- It consists of a set of operations that take one or two relations as input and produce a new relation as their result.
- Six basic operators
  - select: σ
  - project: Π
  - union: $\cup$
  - set difference: –
  - Cartesian product: x
  - rename: ρ

Relational Algebra Overview

- **Relational Algebra consists of several groups of operations**
- Unary Relational Operations
  - SELECT (symbol: σ (sigma))
  - PROJECT (symbol: π (pi))
  - RENAME (symbol: ρ (rho))
- Relational Algebra Operations From Set Theory
  - UNION ($\cup$), INTERSECTION ($\cap$), DIFFERENCE (or MINUS, –)
  - CARTESIAN PRODUCT (**x**)
- Binary Relational Operations
  - JOIN (several variations of JOIN exist)
  - DIVISION
- Additional Relational Operations
  - OUTER JOINS, OUTER UNION
  - AGGREGATE FUNCTIONS (These compute summary of information: for example, SUM, COUNT, AVG, MIN, MAX)

Select Operation

- The **select** operation selects tuples that satisfy a given predicate.
- Notation: $\sigma_p(r)$
- p is called the **selection predicate**
- The selection condition acts as a **filter**
- comparisons are done using =, ≠, <, ≤, >, and ≥ in the selection predicate.
- we can combine several predicates into a larger predicate by using the connectives *and* (∧), *or* (∨), and *not* (¬)

Example of selection:

1. To select tuples of the *instructor* who is in the "Physics" department

$$\sigma_{dept\_name="Physics"}(instructor)$$

2. find all instructors with salary greater than 90,000

$$\sigma_{salary>90000}(instructor)$$

3. to find the instructors in Physics with a salary greater than $90,000

$$\sigma_{dept\ name = \text{"Physics"} \wedge salary > 90000}(instructor)$$

Project Operation

- Project is used to display the required attributes from a relation.
- Notation: $\prod_{A_1, A_2, \ldots, A_k}(r)$
  where A_1, A_2 are attribute names and r is a relation name.
- The result is defined as the relation of k columns obtained by erasing the columns that are not listed
- Duplicate rows removed from result, since relations are sets

Example:

- To list all instructors' *ID*, *name*, and *salary* attributes of *instructor*

$$\prod_{ID,\ name,\ salary}(instructor)$$

- Find the name of all instructors in the Physics department

$$\prod name\ (\sigma\ dept\ name = \text{“Physics”}\ (instructor))$$

RENAME

- **The RENAME operator is denoted by ρ (rho)**
- **The general RENAME operation ρ can be expressed by any of the following forms:**
- $\rho_{S(B1, B2, \ldots, Bn)}(R)$ changes both:
  - the relation name to S, *and*
  - the column (attribute) names to B1, B1, …..Bn
- $\rho_S(R)$ changes:
  - the *relation name* only to S
- $\rho_{(B1, B2, \ldots, Bn)}(R)$ changes:
  - the *column (attribute) names* only to B1, B1, …..Bn

Union Operation

- Notation: $r \cup s$
- Defined as:

$$r \cup s = \{t \mid t \in r \text{ or } t \in s\}$$

- The result of this operation, denoted by $R \cup S$, is a relation that includes all tuples that are either in R or in S or in both R and S. Duplicate tuples are eliminated.
- For $r \cup s$ to be valid (r and s should be **union compatible**).
  1. *r, s* must have the *same* **arity** (same number of attributes)
  2. The attribute domains must be **compatible** (example: 2nd column of *r* deals with the same type of values as does the 2nd column of *s*)
- Example: to find all courses taught in the Fall 2009 semester, or in the Spring 2010 semester, or in both

$$\Pi_{course\_id}(\sigma_{semester=\text{“Fall”} \wedge year=2009}(section)) \cup$$
$$\Pi_{course\_id}(\sigma_{semester=\text{“Spring”} \wedge year=2010}(section))$$

Set Difference Operation

- Notation $r - s$
- Defined as:
- $$r - s = \{t \mid t \in r \textbf{ and } t \notin s\}$$

- Set differences must be taken between **compatible** relations.
  - r and s must have the same arity
  - attribute domains of r and s must be compatible
- Example: to find all courses taught in the Fall 2009 semester, but not in the Spring 2010 semester

$$\Pi_{course\_id}(\sigma_{semester=\text{“Fall”} \wedge year=2009}(section)) -$$
$$\Pi_{course\_id}(\sigma_{semester=\text{“Spring”} \wedge year=2010}(section))$$

Set-Intersection Operation

- Notation: $r \cap s$
- Defined as:
- $r \cap s = \{ t \mid t \in r \textbf{ and } t \in s \}$
- The result of this operation, denoted by $r \cap$ s, is a relation that includes all tuples that are in both *r* and s.
- Assume:
  - *r*, *s* have the *same arity*
  - attributes of *r* and *s* are compatible
- Example: to find all courses taught in the Fall 2009 semester, or in the Spring 2010 semester, or in both

$$\prod_{course\_id} (\sigma_{semester=\text{“Fall”} \wedge year=2009} (section)) \cap \prod_{course\_id} (\sigma_{semester=\text{“Spring”} \wedge year=2010} (section))$$

Set operations example

The set operations UNION, INTERSECTION, and MINUS. (a) Two union-compatible relations. (b) STUDENT ∪ INSTRUCTOR. (c) STUDENT ∩ INSTRUCTOR. (d) STUDENT – INSTRUCTOR. (e) INSTRUCTOR – STUDENT.

(a) **STUDENT**

Fn	Ln
Susan	Yao
Ramesh	Shah
Johnny	Kohler
Barbara	Jones
Amy	Ford
Jimmy	Wang
Ernest	Gilbert

INSTRUCTOR

Fname	Lname
John	Smith
Ricardo	Browne
Susan	Yao
Francis	Johnson
Ramesh	Shah

(b)

Fn	Ln
Susan	Yao
Ramesh	Shah
Johnny	Kohler
Barbara	Jones
Amy	Ford
Jimmy	Wang
Ernest	Gilbert
John	Smith
Ricardo	Browne
Francis	Johnson

(c)

Fn	Ln
Susan	Yao
Ramesh	Shah

(d)

Fn	Ln
Johnny	Kohler
Barbara	Jones
Amy	Ford
Jimmy	Wang
Ernest	Gilbert

(e)

Fname	Lname
John	Smith
Ricardo	Browne
Francis	Johnson

Cartesian-Product Operation

The **Cartesian-product** operation, denoted by a cross (×), allows us to combine information from any two relations.
Cartesian product of relations $r1$ and $r2$ is written as $r1 \times r2$.
Ex: to find the names of all instructors in the Physics department together with the *course id* of all courses they taught.

$$\Pi_{name,\ course\_id}\left(\sigma_{instructor.ID = teaches.ID}\left(\sigma_{dept\_name = \text{“Physics”}}\left(instructor \times teaches\right)\right)\right)$$

Join / Cartesian Product

- Binary Operation between two relation A and B
- The operator generates all possible combination between all tuples of A and B
- Denoted by ‘× ‘
- Synonym as ‘cross join’ . e.g

A

P	Q
p1	q1
p2	q2

B

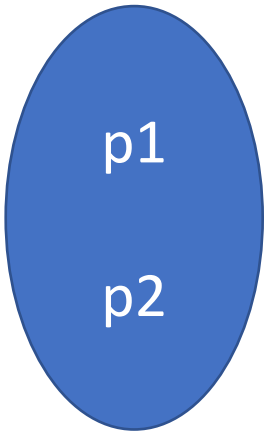
M	N
m1	n1
m2	n2
m3	n3

A × B

P	Q	M	N
p1	q1	m1	n1
p1	q1	m2	n2
p1	q1	m3	n3
p2	q2	m1	n1
p2	q2	m2	n2
p2	q2	m3	n3

Join / Cartesian Product

- A

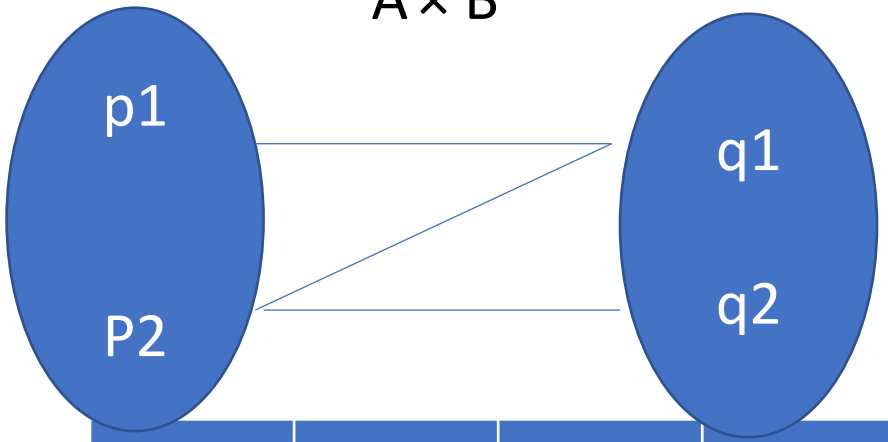


P	Q
p1	q1
p2	q2

B

M	N
m1	n1
m2	n2
m3	n3

A × B



P	Q	M	N
p1	q1	m1	n1
p1	q1	m2	n2
p1	q1	m3	n3
p2	q2	m1	n1
p2	q2	m2	n2
p2	q2	m3	n3

Join / Cartesian Product

Properties of Join operation

Given two relation A and B with :

degree(A) = m degree(B)=n

cardinality(A)= c1 cardinality(B)=c2

Then

degree(A×B) = degree(A)+ degree(B)=>m+n

cardinality(A×B) = cardinality(c1) * cardinality(c2) => c1*c2

Join in relational Algebra

Join is a combination of a Cartesian product followed by a selection process.

A Join operation pairs two tuples from different relations, if and only if a given join condition is satisfied.

Various forms of join operation are:

- Inner Joins:
  - Theta join
  - EQUI join
  - Natural join
- Outer join:
  - Left Outer Join
  - Right Outer Join
  - Full Outer Join

Inner Join:

In an inner join, only those tuples that satisfy the matching criteria are included, while the rest are excluded.

Theta join(θ)

- The general case of JOIN operation is called a Theta join.
- It is denoted by symbol **θ.**
- Also Known as Conditional Join.
- Used when you want to join two or more relation based on some conditions.

Example

$$P \bowtie_{\theta} Q$$

Customer

Cid	Cname	Age
101	Ajay	20
102	Vijay	19
103	Sita	21

Order

Oid	Oname
101	Pizza
101	Noodles
103	Burger

$$\text{Customer} \bowtie_{\text{Customer.cid>Order.oid}} \text{Order}$$

Cid	Cname	Age	Oid	Oname
102	Vijay	19	101	Pizza
102	Vijay	19	101	Noodles
103	Sita	21	101	Pizza
103	Sita	21	101	Noodles

Equi join(⋈)

- Equijoin is a **special case of conditional join**
- When a theta join uses only equivalence condition, it becomes a equi join.
- As values of two attributes will be equal in result of equijoin, only one attribute will be appeared in result.

Example

$P \bowtie Q$

Customer

Cid	Cname	Age
101	Ajay	20
102	Vijay	19
103	Sita	21

Order

Oid	Oname
101	Pizza
101	Noodles
103	Burger

Customer $\bowtie_{\text{Customer.cid=Order.oid}}$ Order

Cid	Cname	Age	Oname
101	Ajay	20	Pizza
101	Ajay	20	Noodles
103	Sita	21	Burger

πcustomer.cid,cusmtomer.cname,order.oname (σcustomer.cid=order.cid(customerXorder))

Join-Natural Join (⋈)

- Natural join can only be performed if there is a common attribute (column) between the relations.
- The name and type (domain) of the attribute must be same.
- Natural join does not use any comparison operator.
- It does not concatenate the way a Cartesian product does.
- Natural Join will also return the similar attributes only once as their value will be same in resulting relation.
- It is a special case of equijoin in which equality condition hold on all attributes which have same name in relations R and S.

Customer

Cid	Cname	Age
101	Ajay	20
102	Vijay	19
103	Sita	21
104	Gita	22

Order

Cid	Oname	Cost
101	Pizza	500
103	Noodles	300
108	Burger	99

Customer ⋈ Order

Cid	Cname	Age	Oname	Cost
101	Ajay	20	Pizza	500
103	Sita	21	Noodles	300

$\sigma_{customer.cid=order.cid}$(custome X order)

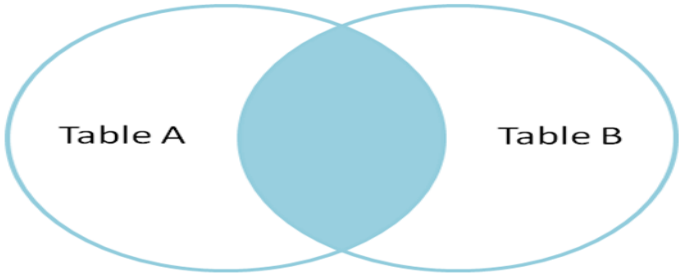
Drawback of Natural Join

- Natural join can only be performed if there is a common attribute (column) between the relations.
- It may lead to data loss if attributes to be match have different names, or some values are present in both the tables.

- Points to Remember

The INNER JOIN keyword selects all rows from both the tables as long as the condition satisfies.

This keyword will create the result-set by combining all rows from both the tables where the condition satisfies i.e value of the common field will be same.



INNER JOIN

OUTER JOIN

A

Id	Name
10	Jay
20	Veer
30	John

B

Name	Marks
Rohan	20
Veer	18
John	14
Sam	13

A ⋈ B

Id	Name	Marks
20	Veer	18
30	John	14

- In an outer join, along with tuples that satisfy the matching criteria, we also include some or all tuples that do not match the criteria.

  Outer join:
- Left Outer Join (A ⟕ B)
- Right Outer Join (A ⟖ B)
- Full Outer Join (A ⟗ B)

OUTER JOIN

A B A ⟕ B

Id	Name
10	Jay
20	Veer
30	John

Name	Marks
Rohan	20
Veer	18
John	14
Sam	13

Id	Name	Marks
20	Veer	18
30	John	14

•In an outer join, along with tuples that satisfy the matching criteria, we also include some or all tuples that do not match the criteria.

Outer join:

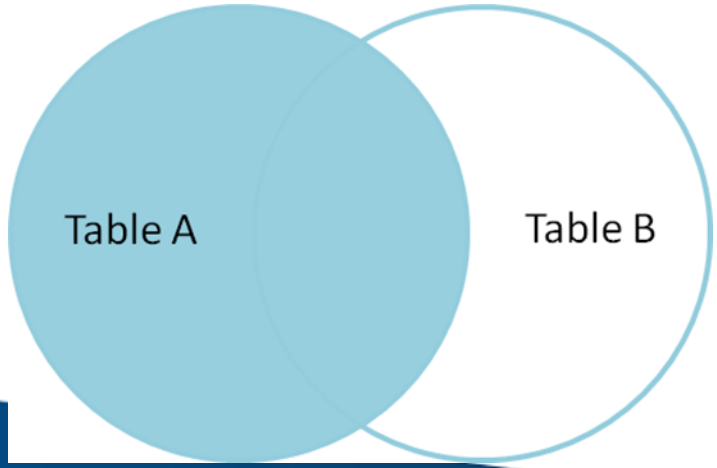
•Left Outer Join (A ⟕ B)

•Right Outer Join (A ⟖ B)

•Full Outer Join (A ⟗ B)

LEFT JOIN (⟕)

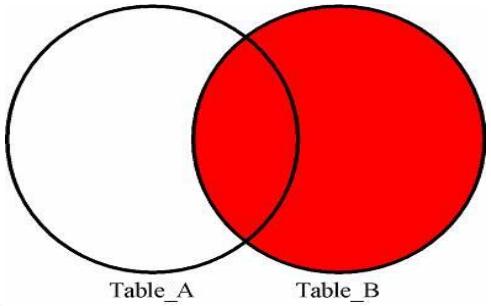
- This join returns all the rows of the table on the left side of the join and matching rows for the table on the right side of join.
- The rows for which there is no matching row on right side, the result-set will contain *null*.
- LEFT JOIN is also known as LEFT OUTER JOIN



id	Name	Marks
10	Jay	NULL
20	Veer	18
30	John	14

RIGHT JOIN(⋈)

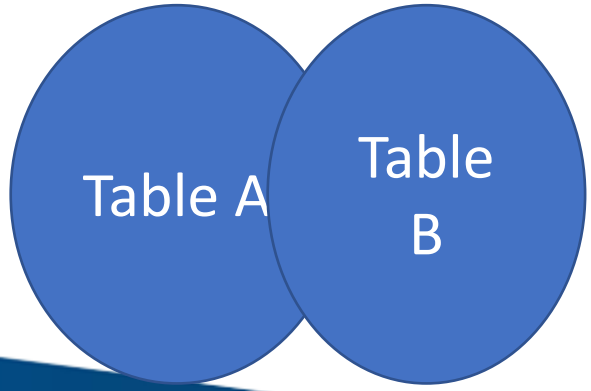
- RIGHT JOIN is similar to LEFT JOIN.
- This join returns all the rows of the table on the right side of the join and matching rows for the table on the left side of join.
- The rows for which there is no matching row on left side, the result-set will contain *null*.
- RIGHT JOIN is also known as RIGHT OUTER JOIN



id	name	marks
Null	Rohan	20
20	Veer	18
30	John	14
Null	Sam	13

FULL JOIN (⋈)

- FULL OUTER JOIN creates the result-set by combining result of both LEFT JOIN and RIGHT JOIN.
- The result-set will contain all the rows from both the tables.
- The rows for which there is no matching, the result-set will contain *NULL* values.



ID	Name	Marks
10	Jay	NULL
20	Veer	18
30	John	14
NULL	Rohan	20
Null	Sam	13

OUTER UNION Operations

- The outer union operation was developed to take the union of tuples from two relations if the relations are *not type compatible*.
- This operation will take the union of tuples in two relations R(X, Y) and S(X, Z) that are **partially compatible**, meaning that only some of their attributes, say X, are type compatible.
- The attributes that are type compatible are represented only once in the result, and those attributes that are not type compatible from either relation are also kept in the result relation T(X, Y, Z).

Division ÷

- Binary Operation between two relation C and B
- Implicitly C is A × B where A is any Relation
- C÷B => (A × B) ÷ B
- The operator '÷ ' splits B from C and produces A
- e.g

C= A × B

P	Q	M	N
p1	q1	m1	n1
p1	q1	m2	n2
p1	q1	m3	n3
p2	q2	m1	n1
p2	q2	m2	n2
p2	q2	m3	n3

B

M	N
m1	n1
m2	n2
m3	n3

C÷B=>

A

P	Q
p1	q1
p2	q2

Division ÷

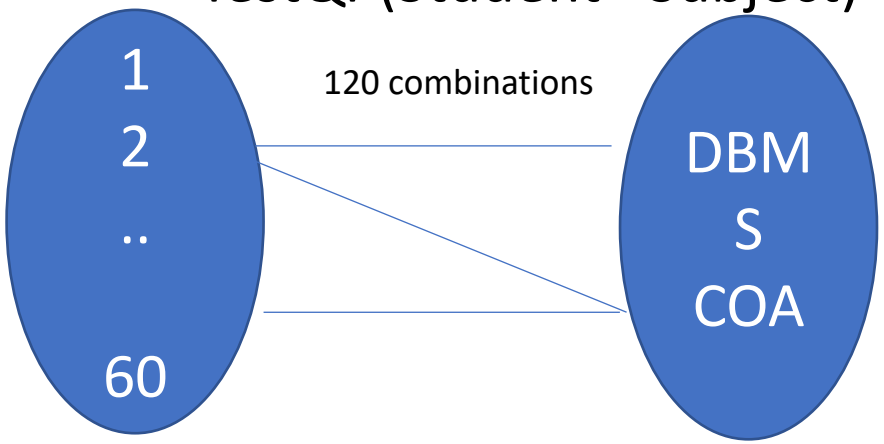
Student

1
2
3
..
60

Subject

DBM
S
COA

TestQP(Student ×Subject)



TestQP(Student ×Subject)

1,DBMS
1,COA
2,DBMS
2,COA
....
60,DBMS
60,COA

Student

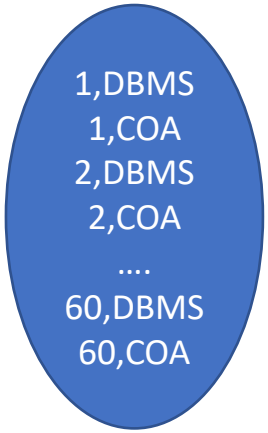
1
2
3
..
60

Subject(TestQP ÷ Student)

DBMS
COA

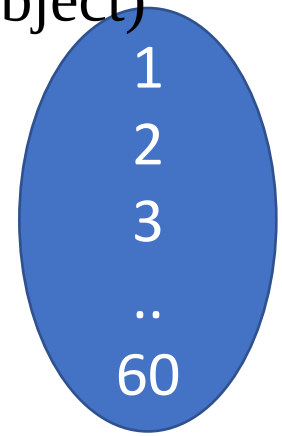
Division ÷

TestQP(Student ×Subject)Subject



Separateses Subject

Student(TestQP ÷ Subject)



Division ÷

Formal examples:

Cases :

Case 1:

Given A,B,C are relations and X,Y are attributes

C(X,Y) ÷ A(X) => B(Y)

C(X,Y) ÷ A(Y) => B(X)

Case 2:

X	Y
X1	Y1
X2	Y2
X1	Y2
X4	y4

÷

Y
Y1
Y2

=

X
X1

Division ÷

Formal examples:

Case 3:

X	Y
X1	Y1
X2	Y2
X1	Y2
X4	y4

÷

X
X1

=

Y
Y1
Y2

Case 4:

X	Y
X1	Y1
X2	Y2
X1	Y2
X4	y4

÷

Y
Y1
Y2
Y3
Y4

=

X
Null

Division ÷

Formal examples:

Case 5:

X	Y
X1	Y1
X2	Y1
X3	Y1
X4	y1

÷

Y
Y1

=

x
X1
X2
X3
X4

Case 6:

X	Y
X1	Y1
X2	Y1
X2	Y2
X1	y2

÷

Y
Y1
Y2

=

X
X1
X2

Recap of Relational Algebra Operations

Table 6.1
Operations of Relational Algebra

Operation	Purpose	Notation
SELECT	Selects all tuples that satisfy the selection condition from a relation R.	$\sigma_{<\text{selection condition}>}(R)$
PROJECT	Produces a new relation with only some of the attributes of R, and removes duplicate tuples.	$\pi_{<\text{attribute list}>}(R)$
THETA JOIN	Produces all combinations of tuples from R_1 and R_2 that satisfy the join condition.	$R_1 \bowtie_{<\text{join condition}>} R_2$
EQUIJOIN	Produces all the combinations of tuples from R_1 and R_2 that satisfy a join condition with only equality comparisons.	$R_1 \bowtie_{<\text{join condition}>} R_2$, OR $R_1 \bowtie_{(<\text{join attributes 1}>),(<\text{join attributes 2}>)} R_2$
NATURAL JOIN	Same as EQUIJOIN except that the join attributes of R_2 are not included in the resulting relation; if the join attributes have the same names, they do not have to be specified at all.	$R_1 *_{<\text{join condition}>} R_2$, OR $R_1 *_{(<\text{join attributes 1}>),(<\text{join attributes 2}>)} R_2$ OR $R_1 * R_2$
UNION	Produces a relation that includes all the tuples in R_1 or R_2 or both R_1 and R_2; R_1 and R_2 must be union compatible.	$R_1 \cup R_2$
INTERSECTION	Produces a relation that includes all the tuples in both R_1 and R_2; R_1 and R_2 must be union compatible.	$R_1 \cap R_2$
DIFFERENCE	Produces a relation that includes all the tuples in R_1 that are not in R_2; R_1 and R_2 must be union compatible.	$R_1 - R_2$
CARTESIAN PRODUCT	Produces a relation that has the attributes of R_1 and R_2 and includes as tuples all possible combinations of tuples from R_1 and R_2.	$R_1 \times R_2$
DIVISION	Produces a relation $R(X)$ that includes all tuples $t[X]$ in $R_1(Z)$ that appear in R_1 in combination with every tuple from $R_2(Y)$, where $Z = X \cup Y$.	$R_1(Z) \div R_2(Y)$

Aggregate Function Operation

- **Use of the Aggregate Functional operation $\mathcal{F}$**
- $\mathcal{F}_{\text{MAX Salary}}$ (EMPLOYEE) retrieves the maximum salary value from the EMPLOYEE relation
- $\mathcal{F}_{\text{MIN Salary}}$ (EMPLOYEE) retrieves the minimum Salary value from the EMPLOYEE relation
- $\mathcal{F}_{\text{SUM Salary}}$ (EMPLOYEE) retrieves the sum of the Salary from the EMPLOYEE relation
- $\mathcal{F}_{\text{COUNT SSN, AVERAGE Salary}}$ (EMPLOYEE) computes the count (number) of employees and their average salary
- Note: count just counts the number of rows, without removing duplicates

Examples of applying aggregate functions and grouping

Figure 6.10

The aggregate function operation.

(a) $\rho_{R(\text{Dno, No\_of\_employees, Average\_sal})}\ (_{\text{Dno}}\ \Im_{\text{COUNT Ssn, AVERAGE Salary}}\ (\text{EMPLOYEE}))$.

(b) $_{\text{Dno}}\ \Im_{\text{COUNT Ssn, AVERAGE Salary}}\ (\text{EMPLOYEE})$.

(c) $\Im_{\text{COUNT Ssn, AVERAGE Salary}}\ (\text{EMPLOYEE})$.

R

(a)

Dno	No_of_employees	Average_sal
5	4	33250
4	3	31000
1	1	55000

(b)

Dno	Count_ssn	Average_salary
5	4	33250
4	3	31000
1	1	55000

(c)

Count_ssn	Average_salary
8	35125

Examples of Queries in Relational Algebra

- **Query 1.** Retrieve the name and address of all employees who work for the 'Research' department.

$$RESEARCH\_DEPT \leftarrow \sigma_{Dname='Research'}(DEPARTMENT)$$
$$RESEARCH\_EMPS \leftarrow (RESEARCH\_DEPT \bowtie_{Dnumber=Dno} EMPLOYEE)$$
$$RESULT \leftarrow \pi_{Fname,\ Lname,\ Address}(RESEARCH\_EMPS)$$

As a single in-line expression, this query becomes:

$$\pi_{Fname,\ Lname,\ Address}\ (\sigma_{Dname='Research'}(DEPARTMENT \bowtie_{Dnumber=Dno}(EMPLOYEE))$$

- **Query 2.** For every project located in ‘Stafford’, list the project number, the controlling department number, and the department manager’s last name, address, and birth date.

$$\text{STAFFORD\_PROJS} \leftarrow \sigma_{\text{Plocation}=\text{'Stafford'}}(\text{PROJECT})$$
$$\text{CONTR\_DEPTS} \leftarrow (\text{STAFFORD\_PROJS} \bowtie_{\text{Dnum}=\text{Dnumber}} \text{DEPARTMENT})$$
$$\text{PROJ\_DEPT\_MGRS} \leftarrow (\text{CONTR\_DEPTS} \bowtie_{\text{Mgr\_ssn}=\text{Ssn}} \text{EMPLOYEE})$$
$$\text{RESULT} \leftarrow \pi_{\text{Pnumber, Dnum, Lname, Address, Bdate}}(\text{PROJ\_DEPT\_MGRS})$$

Query 3. Find the names of employees who work on *all* the projects controlled by department number 5.

$$\text{DEPT5\_PROJS} \leftarrow \rho_{(\text{Pno})}(\pi_{\text{Pnumber}}(\sigma_{\text{Dnum}=5}(\text{PROJECT})))$$
$$\text{EMP\_PROJ} \leftarrow \rho_{(\text{Ssn, Pno})}(\pi_{\text{Essn, Pno}}(\text{WORKS\_ON}))$$
$$\text{RESULT\_EMP\_SSNS} \leftarrow \text{EMP\_PROJ} \div \text{DEPT5\_PROJS}$$
$$\text{RESULT} \leftarrow \pi_{\text{Lname, Fname}}(\text{RESULT\_EMP\_SSNS} * \text{EMPLOYEE})$$

Query 4. Make a list of project numbers for projects that involve an employee whose last name is 'Smith', either as a worker or as a manager of the department that controls the project.

$$SMITHS(Essn) \leftarrow \pi_{Ssn}(\sigma_{Lname='Smith'}(EMPLOYEE))$$
$$SMITH\_WORKER\_PROJS \leftarrow \pi_{Pno}(WORKS\_ON * SMITHS)$$
$$MGRS \leftarrow \pi_{Lname,\ Dnumber}(EMPLOYEE \bowtie_{Ssn=Mgr\_ssn} DEPARTMENT)$$
$$SMITH\_MANAGED\_DEPTS(Dnum) \leftarrow \pi_{Dnumber}(\sigma_{Lname='Smith'}(MGRS))$$
$$SMITH\_MGR\_PROJS(Pno) \leftarrow \pi_{Pnumber}(SMITH\_MANAGED\_DEPTS * PROJECT)$$
$$RESULT \leftarrow (SMITH\_WORKER\_PROJS \cup SMITH\_MGR\_PROJS)$$

Query 5. List the names of all employees with two or more dependents.

$$T1(\text{Ssn, No\_of\_dependents}) \leftarrow {}_{\text{Essn}}\Im_{\text{COUNT Dependent\_name}}(\text{DEPENDENT})$$
$$T2 \leftarrow \sigma_{\text{No\_of\_dependents}>2}(T1)$$
$$\text{RESULT} \leftarrow \pi_{\text{Lname, Fname}}(T2 * \text{EMPLOYEE})$$

Query 6. Retrieve the names of employees who have no dependents.

$$\text{ALL\_EMPS} \leftarrow \pi_{\text{Ssn}}(\text{EMPLOYEE})$$
$$\text{EMPS\_WITH\_DEPS}(\text{Ssn}) \leftarrow \pi_{\text{Essn}}(\text{DEPENDENT})$$
$$\text{EMPS\_WITHOUT\_DEPS} \leftarrow (\text{ALL\_EMPS} - \text{EMPS\_WITH\_DEPS})$$
$$\text{RESULT} \leftarrow \pi_{\text{Lname, Fname}}(\text{EMPS\_WITHOUT\_DEPS} * \text{EMPLOYEE})$$

Query 7. List the names of managers who have at least one dependent.

$$\text{MGRS}(\text{Ssn}) \leftarrow \pi_{\text{Mgr\_ssn}}(\text{DEPARTMENT})$$
$$\text{EMPS\_WITH\_DEPS}(\text{Ssn}) \leftarrow \pi_{\text{Essn}}(\text{DEPENDENT})$$
$$\text{MGRS\_WITH\_DEPS} \leftarrow (\text{MGRS} \cap \text{EMPS\_WITH\_DEPS})$$
$$\text{RESULT} \leftarrow \pi_{\text{Lname, Fname}}(\text{MGRS\_WITH\_DEPS} * \text{EMPLOYEE})$$

Thank YOU