Notes on: Fundamentals of Database Systems

Notes from Textbooks for CIS 530 – Introduction to Database Systems and Processing

Textbooks:

- **Fundamentals of Database Systems**, 7th edition; by Ramez Elmasri and Shamkant B. Navanthe. Published by Pearson, 2017. ISBN-13: 978-0-13-397077-7
- **Database Systems: Design, Implementation, and Management**, 13th edition; by Carlos Coronel and Steven Morris. Published by Cengage, 2022. ISBN-13: 978-1-337-62790-0

The book by Elmasri and Navanthe is the actual textbook for the course, but I am finding that the book by Coronel and Morris is much better written, so I am capturing some information from there as well.

Contents

Ch. 1: Databases and Database Users

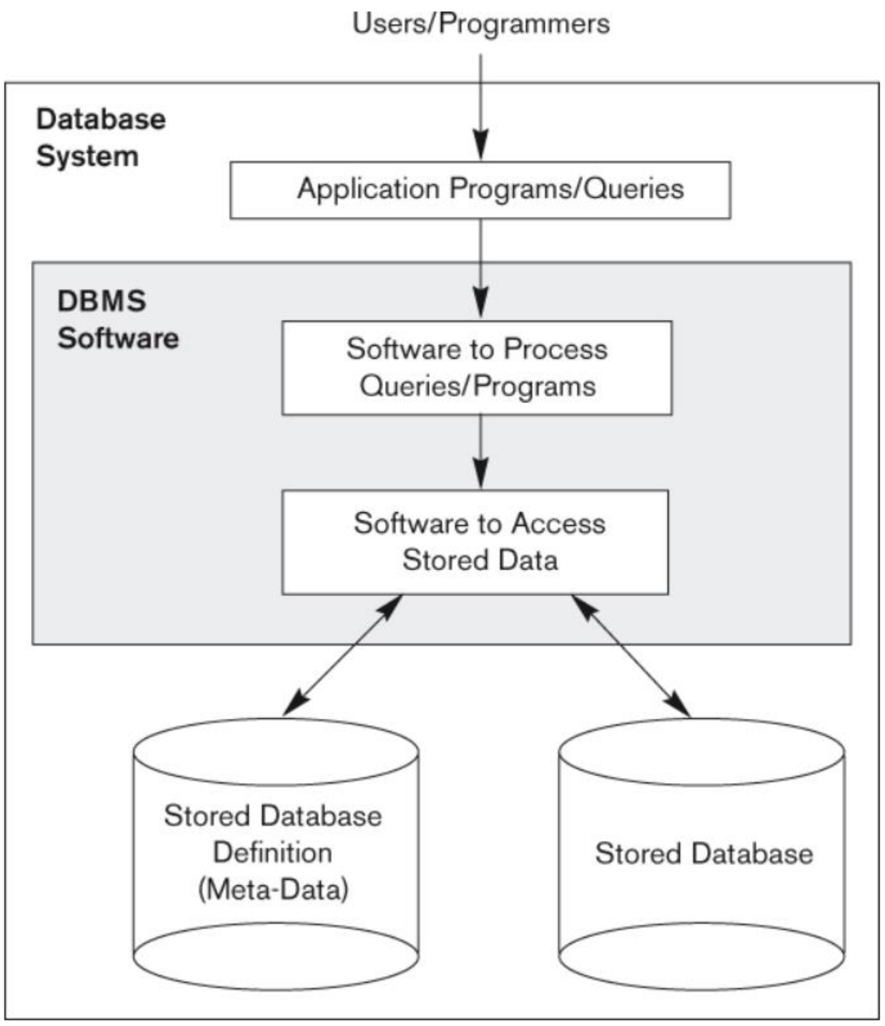
1.1 Introduction

- **Data –** known facts that can be recorded and have implicit meaning.
- **Database –**
    - A logically coherent collection of related data
    - Built and populated for a specific purpose

A database represents some aspect of the real world, sometimes called the **miniworld**.

- **Database Management System (DBMS) –**
    - Computerized system that enables users to create and maintain a database
    - General-purpose software system that facilitates the processes of defining, constructing, manipulating, and sharing databases.

Figure 1: Simplified Database System Environment


Users/Programmers
Database System
Application Programs/Queries
DBMS Software
Software to Process Queries/Programs
Software to Access Stored Data
Stored Database Definition (Meta-Data)
Stored Database

1.3 Characteristics of the Database Approach

Database system contains

- The database itself
- A complete definition of the database structure and constraints – **meta-data**.

A **data model** is a type of **data abstraction** that hides storage and implementation details.

Different types of users of the database may require different **views –** a subset of the database.

Transaction – an access – reading (**query**), writing, or updating of database records – of the database by an executing program or process.

1.6 Advantages of Using the DBMS Approach

1. Controlling redundancy of the data
    a. **Data normalization –** each logical data item stored in only one place
    b. Sometimes necessary to use controlled redundancy
2. Provides security and authorization
3. Provides persistent storage for databases
4. Efficient execution of queries and updates.
5. Provides backup and recovery
6. Provides multiple user interfaces (views and GUIs)
7. Represents complex relationships among data
8. Enforces **integrity constraints**
    a. Data type constraints
    b. **Referential integrity –** maintains correct relationships between related records
    c. Key / uniqueness constraint

Ch. 2: Database System Concepts and Architecture

2.1 Data Models, Schemas, and Instances

Data abstraction – suppression of details of data organization and storage; highlighting the essential features

Data model –

- A collection of concepts that can be used to describe the structure of a database
- Provides the means to achieve data abstraction

Entity – Relationship (ER) Model

- **Entity –** represents a real-world object or concept that is described in the database
- **Attribute** -- property of interest that further describes an entity
- **Relationship –** an association among entities

Schema – a description of the database (distinct from the database itself)

- specified during database design
- not expected to change frequently

The data in the database at a particular moment in time is called:

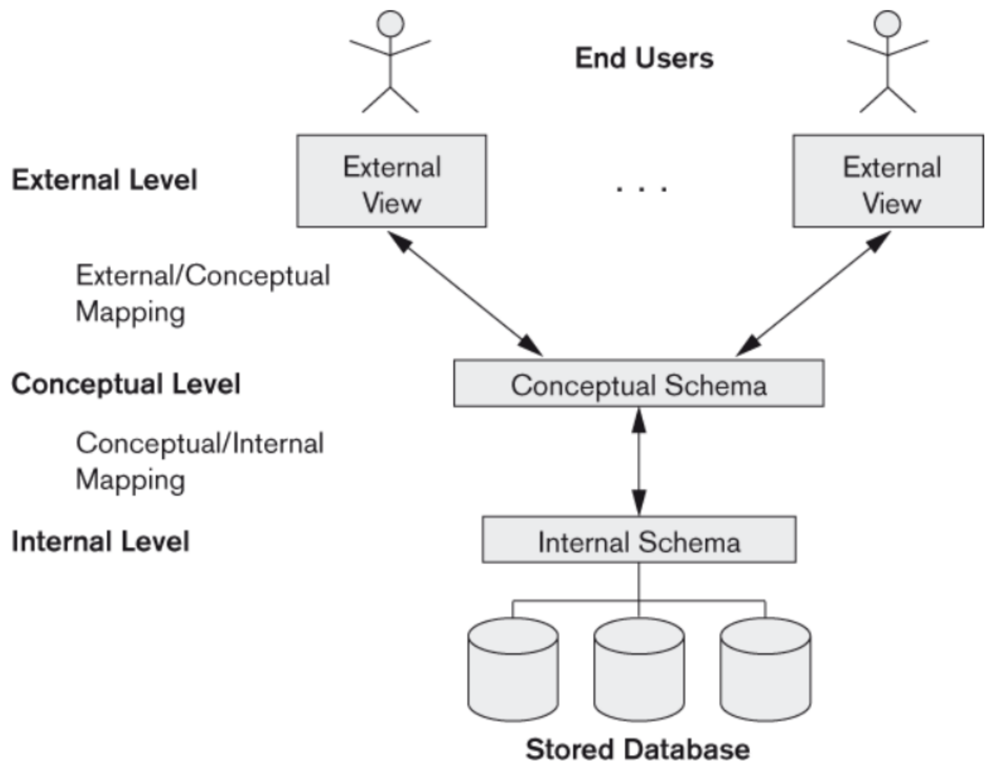
- **Database state**
- **Database instances**

2.2 Three-Schema Architecture and Data Independence

Three-Schema Architecture – architecture for database systems

1. **Internal schema –** describes physical storage structure
2. **Conceptual schema –** describes database structure for the users; entities, data types, relationships, constraints, user operations
3. **External schema, or user views –** the subset of the database that a particular user group is interested in.

Figure 2: The Three-Schema Architecture



The three-schema architecture achieves

- Logical data independence – allows conceptual schema to change without having to change external schemas or application programs.
- Physical data independence – allows internal schema to change without having to change the conceptual schema.

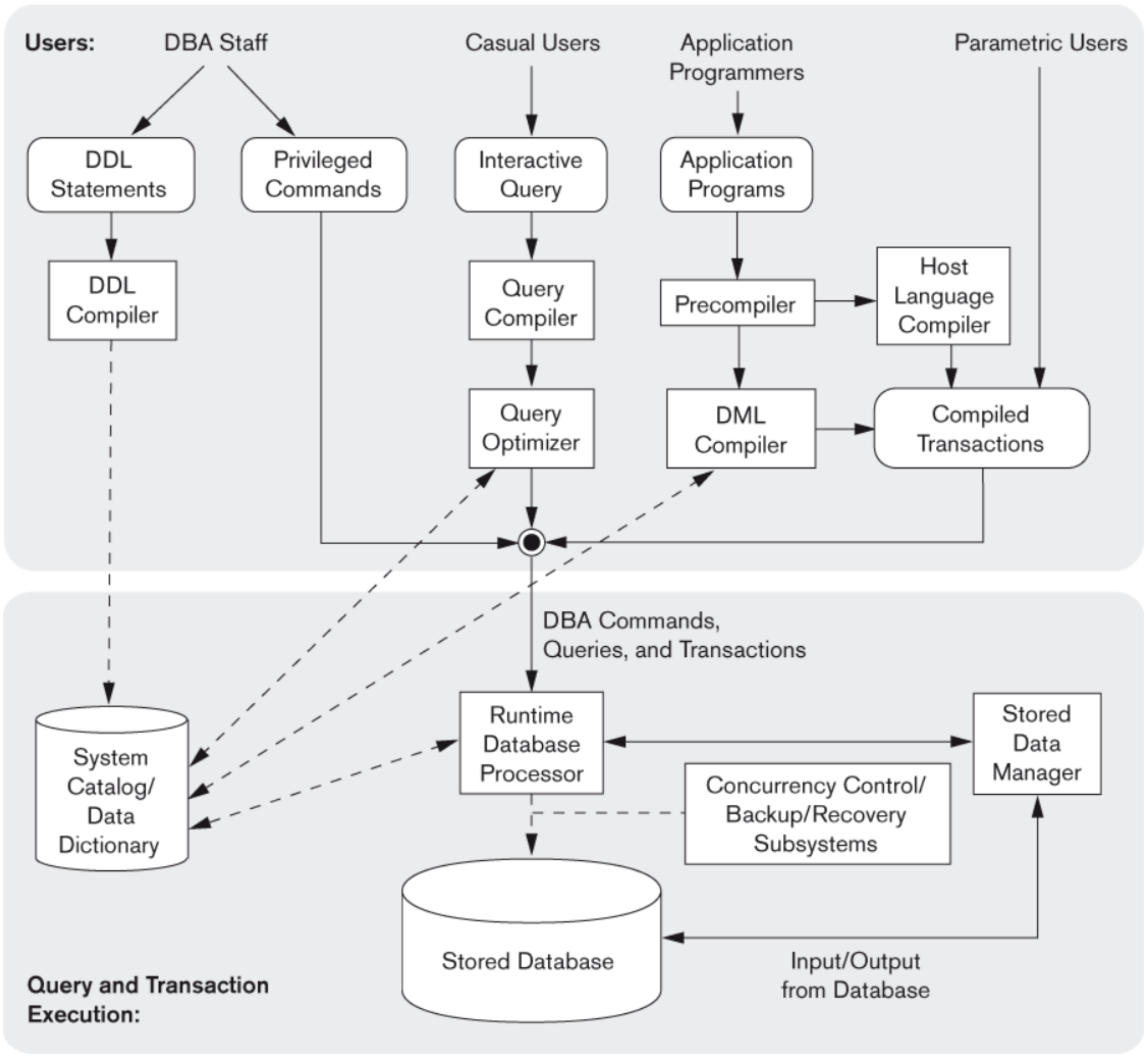
2.3 Database Languages and Interfaces

Two primary types of languages used to define, populate, and query a database:

- **Data Definition Language (DDL)** – used by database designers to define the conceptual and logical (or internal) schema of the database.
- **Data Manipulation Language (DML)** – operations to insert, delete, retrieve, and modify the data in the database.

2.4 The Database System Environment

Figure 3: Components of a DBMS and their Interactions



2.5 Centralized and Client / Server Architectures for DBMSs

Figure 4: Physical Two-Tier Client / Server Architecture

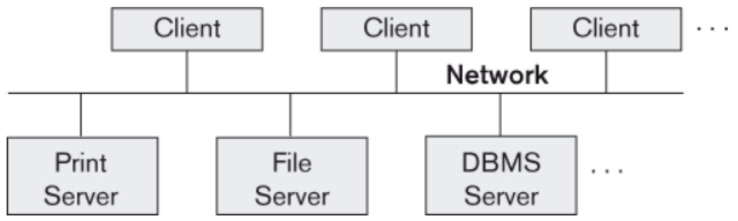


Figure 2.5 Logical two-tier client/server architecture.

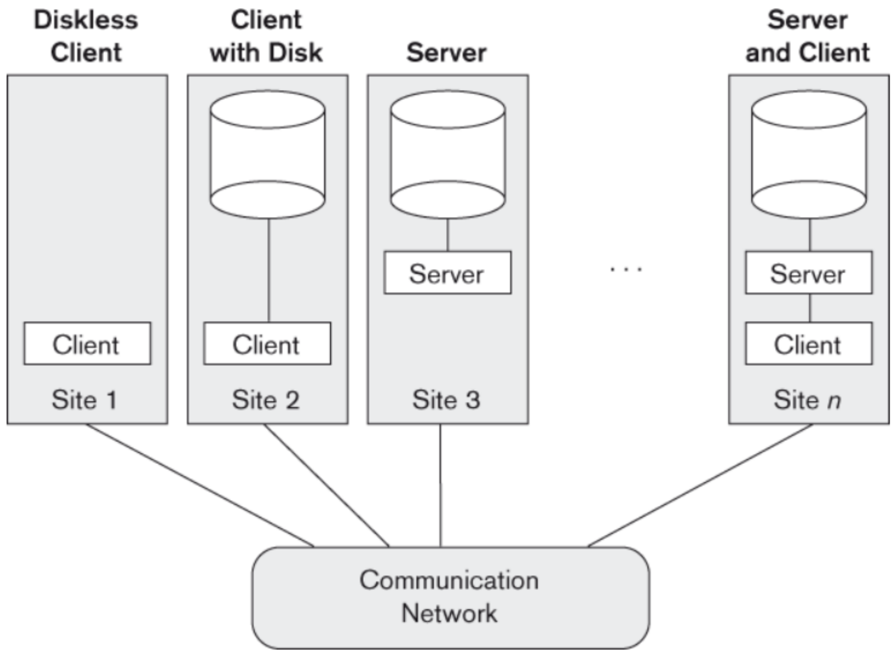
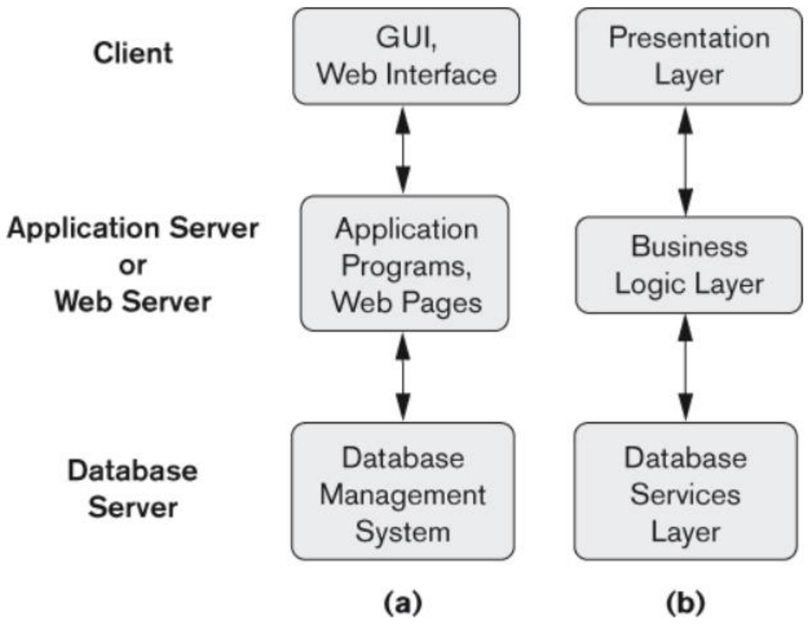


Figure 5: Logical Three-Tier Client / Server Architecture, with Two Commonly Used Nomenclatures



In a two-tier architecture, the DBMS server is often called a **query server** or **transaction server**, or in an RDBMS an **SQL server**.

The three-tier architecture adds an intermediate layer (application server or web server) between the client and database server. It plays the role of running application programs that access data from the database server, and then renders it visually for the client.

2.6 Classification of Database Management Systems

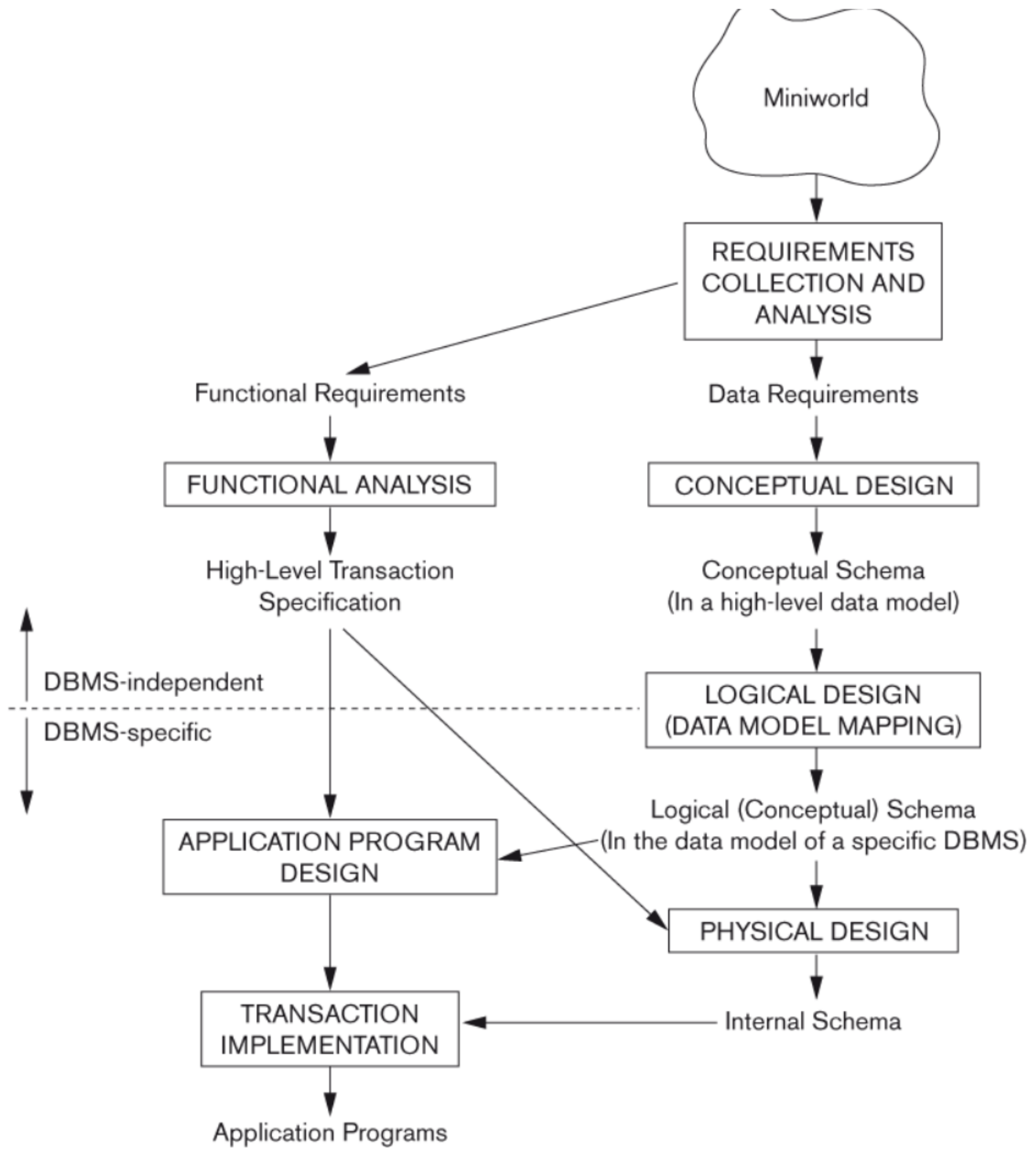
DBMSs can be classified in several ways:

- Data model
  - Relational data model
  - Object data model
  - Key-value models / NOSQL (Not Only SQL) systems
    - Document-based models
    - XML / Tree-structured models
- Number of users: single-user vs. multiuser systems
- Number of sites over which database is distributed: centralized vs. distributed
- Cost

Ch. 3: Data Modeling Using the Entity-Relationship (ER) Model

3.1 Database Design

Figure 6: Simplified Diagram Illustrating the Main Phases of Database Design


Miniworld
REQUIREMENTS COLLECTION AND ANALYSIS
Functional Requirements
Data Requirements
FUNCTIONAL ANALYSIS
CONCEPTUAL DESIGN
High-Level Transaction Specification
Conceptual Schema (In a high-level data model)
DBMS-independent
DBMS-specific
LOGICAL DESIGN (DATA MODEL MAPPING)
Logical (Conceptual) Schema (In the data model of a specific DBMS)
APPLICATION PROGRAM DESIGN
PHYSICAL DESIGN
TRANSACTION IMPLEMENTATION
Internal Schema
Application Programs

Steps in database design process

1. **Requirements collection and analysis –** concisely written set of users' requirements, including functional requirements
2. **Conceptual design –** leads to a conceptual schema, with detailed descriptions of the entity types, relationships, and constraints
3. **Logical design or data model mapping –** conceptual schema transformed into the implementation data model

4. **Physical design** – specify internal storage structures, file organizations, indexes, access paths

A conceptual schema for a database can be displayed by means of a graphical notation known as an **ER diagram**.

3.3 Entity Types, Entity Sets, Attributes, and Keys

The ER model describes data as

- Entities – things or objects in the real world with an independent existence
- Attributes – particular properties that describe an entity
- Relationships

Attribute Types

- **Simple** or atomic – not divisible into smaller pieces
- **Composite** – can be divided into multiple simple attributes (e.g. a name (first, middle, last) or an address (street, city, state, zip code)
- **Single-valued** vs. **multivalued** (e.g. colors attribute for a car model, or college_degrees attribute for a person)
- **Stored** vs. **derived**
- **Complex attributes** – attributes consisting of nested composite (noted by grouping components within parentheses ()) and multivalued attributes (noted by placing within braces {}). For example

```
{Address_phone(
      {Phone(Area_code, Phone_number),
       Address(Street_address(Number,Street,Apartment_number),
            City, State, Zip)
       }
 )
}
```

Entity Types, Sets, Keys, Value Sets

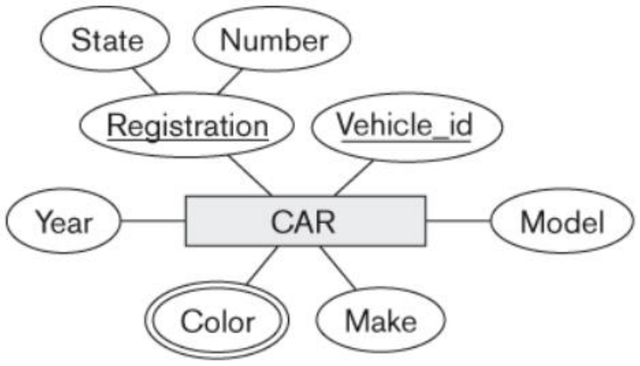
Entity type defines collection (or set) of entities that have the same attributes (for example, EMPLOYEE, DEPARTMENT).

Entity set is the collection of all entities of a particular entity type in the database at a specific point in time.

Strong entities have one or more attributes with values that are distinct for each individual entity in the entity set – these are called **key attributes**. **Weak entities** have no key attribute.

Each attribute has a **domain** – the set of values that may be assigned to the attribute for each individual entity.

Figure 7: Example of an ER Diagram for a Single Entity and Its Attributes



As shown in Figure 7, in an ER diagram

- An Entity type is shown as a rectangular box enclosing the entity type name.
- Attributes are enclosed in ovals, connected by lines to their entity type.
- Composite attributes are connected to their component attributes by straight lines.
- Multivalued attributes are displayed in double ovals.
- **Key attributes** have their names underlined inside its oval.

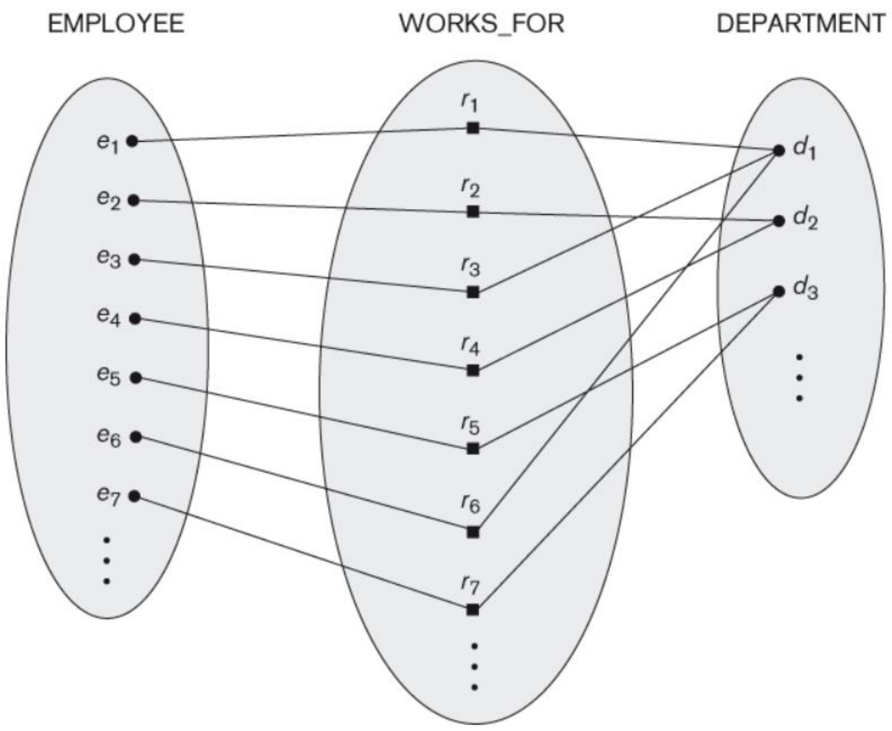
3.4 Relationship Types, Relationship Sets, Roles, and Structural Constraints

Whenever an attribute of one entity type refers to another entity type, some relationship exists. In the ER model, these references are represented as **relationships**, rather than attributes.

A **relationship type** R among n entity types $E_1, E_2, \ldots, E_n$ defines a **relationship set** – a set of associations among a subset of the entity types, called **relationship instances**.

We can define a relationship set as a subset of the Cartesian product of the entity sets $E_1 \times E_2 \times \ldots \times E_n$, where each of the entity types is said to **participate** in the relationship type R.

Figure 8: Example of a Relationship Type -- EMPLOYEE and DEPARTMENT Participate in WORKS_FOR



In ER diagrams, **relationship types are displayed as diamond-shaped boxes**, connected by straight lines to the participating entity type boxes.

Degree of a relationship type – number of participating entity types. Binary is most common, but unary (recursive relationships) and ternary also occur.

Constraints on Binary Relationship Types

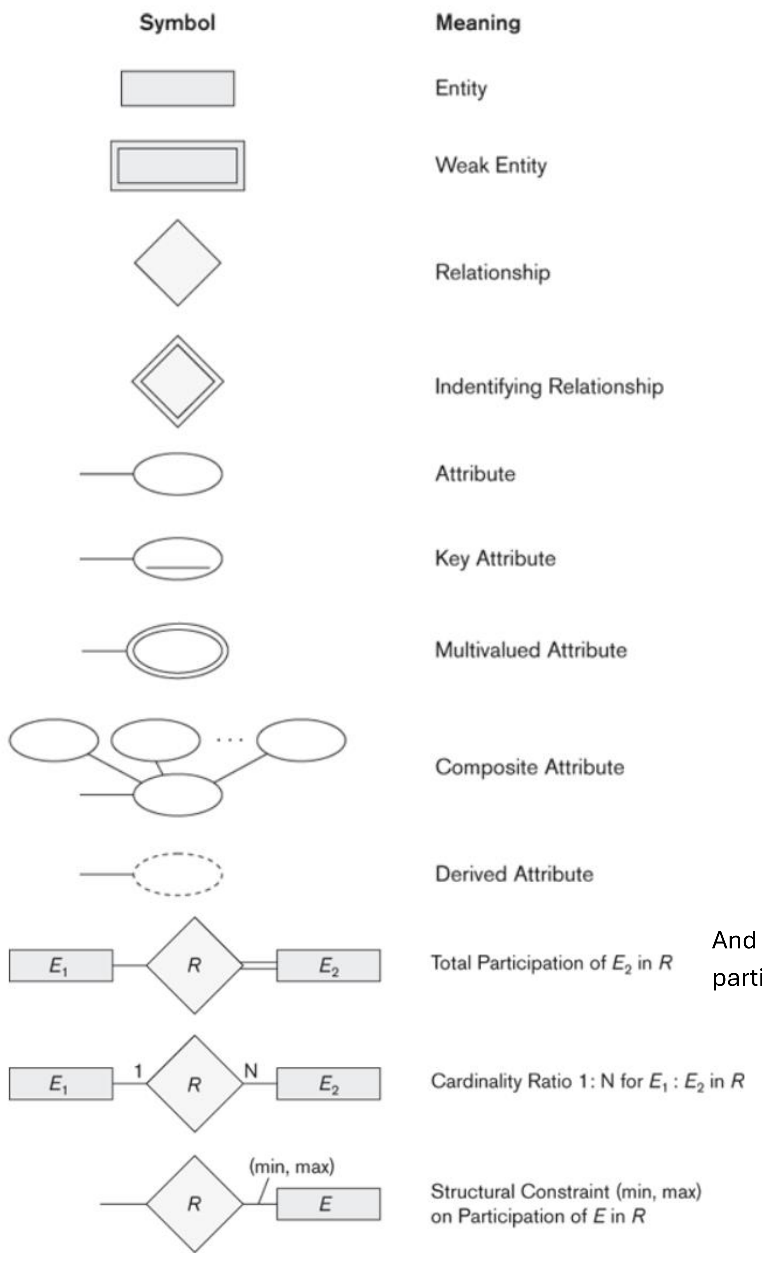
Two main types of binary relationship constraints:

- **Cardinality ratio** – maximum number of relationship instances that an entity can participate in.
    - Example: In the WORKS_FOR relationship, EMPLOYEE : DEPARTMENT has a cardinality ratio of N : 1; each employee can work in only one department, but any one department can have multiple employees working for it.
    - Example: the MANAGES relationship between EMPLOYEE and DEPARTMENT is 1:1; each employee can manage at most 1 department, while each department has at most one manager.
    - For a binary relationship, **possible ratios are: 1:1, 1:N, N:1, and M:N**.
- **Participation constraint** – the minimum number of relationship instances that each entity can participate in.

- **Total participation –** an entity of one type must have at least one relationship with an entity of the second type.
- **Partial participation –** it is allowed for an entity of one type to have no relationships with any entity of the second type.
- Example: each EMPLOYEE must work for a DEPARTMENT, but a DEPARTMENT can have no employees. EMPLOYEE has total participation in the WORKS_FOR relationship, while DEPARTMENT has partial participation.

3.7 ER Diagrams, Naming Conventions, and Design Issues

Figure 9: Summary of Notation for ER Diagrams


Symbol
Meaning
Entity
Weak Entity
Relationship
Indentifying Relationship
Attribute
Key Attribute
Multivalued Attribute
. . .
Composite Attribute
Derived Attribute
E_1
R
E_2
Total Participation of E_2 in R
E_1
1
R
N
E_2
Cardinality Ratio 1 : N for E_1 : E_2 in R
(min, max)
R
E
Structural Constraint (min, max) on Participation of E in R

And partial participation of E_1 in R

Naming conventions for ER diagrams:

- Use all uppercase letters for entity type and relationship type names.
- Capitalize only the initial letter of attribute names.
- Use lower case for role names.

Ch. 5: The Relational Data Model and Relational Database Constraints

5.1 Relational Model Concepts

The relational model represents the database as a collection of **relations**.

- A relation can be thought of as a **table** of values.
- Each row represents a real-world entity – a **tuple** of attribute values.

A **domain**, D, is a set of **atomic** (i.e. indivisible) values. A **data type** is specified for each domain.

A **relation schema** is made up of a relation R and a list of attributes, $A_1, A_2, \dots, A_n$.

$$R(A_1, A_2, \dots, A_n)$$

Each attribute A_i has a domain, $dom(A_i)$. A **relation state**, r, of the relation schema $R(A_1, A_2, \dots, A_n)$ is a set (of size m) of n-tuples, $r = \{t_1, t_2, \dots, t_m\}$, each of which is an ordered list of n values, $t_i = < v_{i1}, v_{i2}, \dots, v_{in} >$, where each value v_{ij}, where $1 \le j \le n$, is an element of $dom(A_j)$.

A relation state can be more formally stated as

$$r(R) \subseteq (dom(A_1)\ \times dom(A_2) \times \dots \times dom(A_n))$$

that is, the relation state is a proper subset of the Cartesian product of the domains of the attributes.

5.2 Relational Model Constraints

Constraints on databases can be divided into three main categories:

- **Inherent model-based constraints**: constraints that are inherent in the data model.
    - Example: duplicate tuples not allowed
- **Schema-based / explicit constraints**: expressed in the schemas, and specified in DDL
- **Application-based constraints**: cannot be expressed in schemas, but must be enforced by application programs.

Domain Constraints

The value of each attribute, A, must be an atomic value $\in dom(A)$.

Key Constraints

Any subset of attributes of a relation, R, which cannot have the same combination of values for any two tuples is called a **superkey**. This is the same as saying that every superkey must be unique.

A **key** of R is a superkey that satisfies two properties:

1. **Uniqueness**: Two distinct tuples in any state of the relation cannot have identical values for all attributes in the key.
2. It is a **minimal** superkey.

A relation may have multiple **candidate keys –** subsets of attributes that satisfy both the uniqueness and minimality properties. It is common to designate one of these as the **primary key**.

Entity Integrity, Referential Integrity, Foreign Keys

Entity integrity constraint: no primary key value can be NULL.

Foreign key: a set of attributes FK in relation R_1 is a foreign key of R_1 that references relation R_2 if it satisfies the following rules:

1. The attributes in FK have the same domain(s) as the attributes of the primary key, PK, of R_2.
2. A value of FK in a tuple $t_1 \in r_1(R_1)$ either occurs as a value of PK for some tuple $t_2 \in r_2(R_2)$ or is NULL.

Referential integrity constraint

- Specified between two relations
- Used to maintain consistency among tuples in the two relations
- Informal definition: a tuple in one relation that refers to another relation must refer to an existing tuple in that relation.
- Formal definition: if the foreign key rules are satisfied between relations R_1 and R_2, then we say that a referential integrity constraint exists between these two relations.