

Module 1

Characteristics of Database approach

- **Self-describing nature of a database system:**
 - A DBMS **catalog** stores the description of a particular database (e.g. data structures, types, and constraints).
 - The description is called **meta-data**.
 - This allows the DBMS software to work with different database applications.
- **Insulation between programs and data:**
 - Called **program-data independence**.
 - Allows changing data structures and storage organization 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.
 - Programs refer to the data model constructs rather than data storage details.
- **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 multi-user transaction processing:**
 - Allowing a set of **concurrent users** to retrieve from and to update the database.
 - *Concurrency control* within the DBMS guarantees that each **transaction** is correctly executed or aborted.
 - *Recovery* subsystem ensures each completed transaction has its effect permanently recorded in the database.
 - **OLTP** (Online Transaction Processing) is a major part of database applications. This allows hundreds of concurrent transactions to execute per second.

Advantages of Using 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 (e.g. indexes) 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 from the stored data using deductive and active rules.

Data Models

- **Data Model:**
 - A set of concepts to describe the **structure** of a database, the **operations** for manipulating these structures, and certain **constraints** that the database should obey.
- **Data Model Structure and Constraints:**
 - Constructs are used to define the database structure.
 - Constructs typically include **elements** (and their **data types**) as well as groups of elements (e.g. **entity**, **record**, **table**), and **relationships** among such groups.
 - Constraints specify some restrictions on valid data; these constraints must be enforced at all times.
- **Data Model Operations:**
 - These operations are used for specifying database *retrievals* and *updates* by referring to the constructs of the data model.
 - Operations on the data model may include **basic model operations** (e.g. generic insert, delete, update) and **user-defined operations** (e.g. compute_student_gpa, update_inventory).

Types 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. These are usually specified in an ad-hoc manner through DBMS design and administration manuals.

■ Implementation (representational) data models:

- Provide concepts that fall between the above two, used by many commercial DBMS implementations (e.g. relational data models used in many commercial systems).

Schema and Instances

■ Database Schema:

- Description of a database.
- Includes descriptions of the database structure, data types, and the constraints on the database.

■ Schema Diagram:

- An **illustrative** display of (most aspects of) a database schema.

■ Schema Construct:

- A **component** of the schema or an object within the schema, e.g., STUDENT, COURSE.

■ Database State:

- The actual data stored in a database at a **particular moment in time**. This includes the collection of all the data in the database.
- Also called database instance (or occurrence or snapshot).
 - The term *instance* is also applied to individual database components, e.g. *record instance*, *table instance*, *entity instance*.

■ Database State:

- Refers to the **content** of a database at a moment in time.

■ Initial Database State:

- Refers to the database state when it is initially loaded into the system.

■ Valid State:

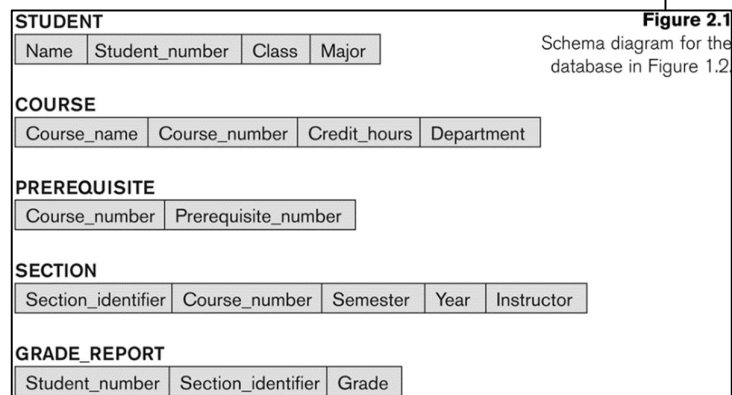
- A state that satisfies the structure and constraints of the database.

■ Distinction

- The **database schema** changes very infrequently.
- The **database state** changes every time the database is updated.

■ Schema is also called **intension**.

■ State is also called **extension**.



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

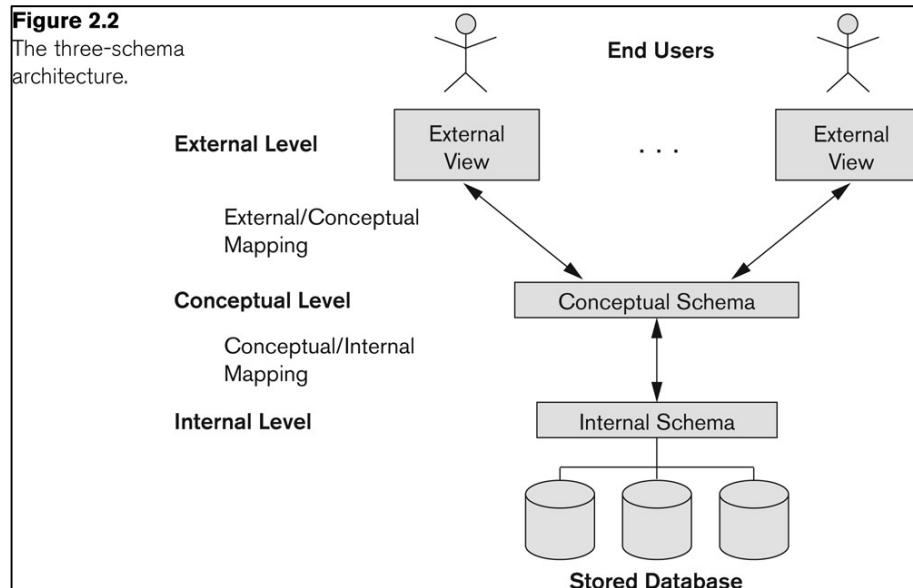
SECTION				
Section_identifier	Course_number	Semester	Year	Instructor
85	MATH2410	Fall	04	King
92	CS1310	Fall	04	Anderson
102	CS3320	Spring	05	Knuth
112	MATH2410	Fall	05	Chang
119	CS1310	Fall	05	Anderson
135	CS3380	Fall	05	Stone

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.

Three-Schema Architecture



- Proposed to support DBMS characteristics of:
 - **Program-data independence.**
 - Support of **multiple views** of the data.
- Defines DBMS schemas at **three** levels:
 - **Internal schema** at the internal level to describe physical storage structures and access paths (e.g. indexes).
 - Typically uses a **physical** data model.
 - **Conceptual schema** at the conceptual level to describe the structure and constraints for the whole database for a community of users.
 - Uses a **conceptual** or an **implementation** data model.
 - **External schemas** at the external level to describe the various user views.
 - Usually uses the same data model as the conceptual schema.
- Mappings among schema levels are needed to transform requests and data.
 - Programs refer to an external schema and are mapped by the DBMS to the internal schema for execution.
 - Data extracted from the internal DBMS level is reformatted to match the user's external view (e.g. formatting the results of an SQL query for display in a Web page)

Data Independence

- **Logical Data Independence:**
 - The capacity to change the conceptual schema without having to change the external schemas and their associated application programs.
- **Physical Data Independence:**
 - The capacity to change the internal schema without having to change the conceptual schema.
 - For example, the internal schema may be changed when certain file structures are reorganized, or new indexes are created to improve database performance.
- 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.

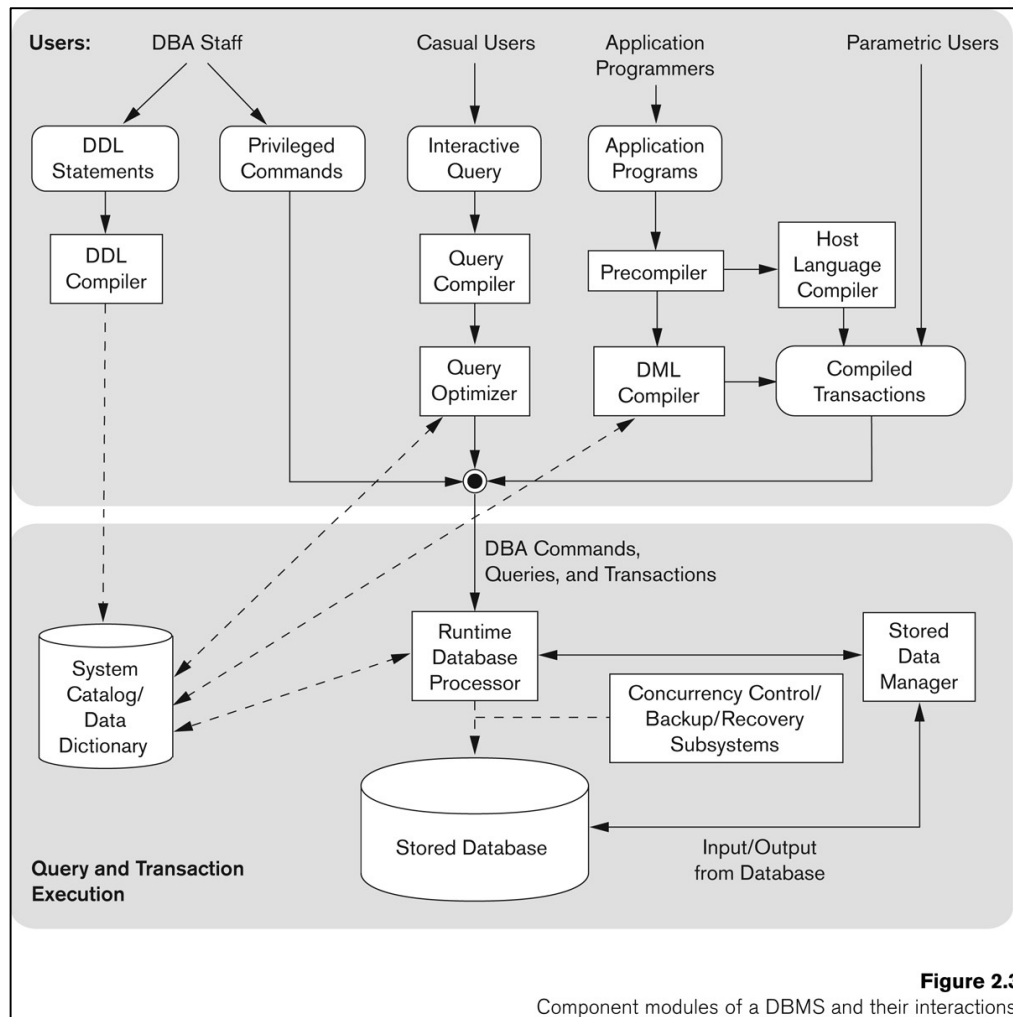
Database Languages

- Data Definition Language (DDL)
- Data Manipulation Language (DML)
 - High-Level or Non-procedural Languages: These include the relational language SQL.
 - May be used in a standalone way or may be embedded in a programming language.
 - Low Level or Procedural Languages:
 - These must be embedded in a programming language.
- **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.
 - SDL is typically realized via DBMS commands provided to the DBA and database designers.
- **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, C++, or Java.
 - A library of functions can also be provided to access the DBMS from a programming language.
 - Alternatively, stand-alone DML commands can be applied directly (called a *query language*).
- **Types of DML:**
 - **High Level or Non-procedural Language:**
 - For example, the SQL relational language
 - Are “set”-oriented and specify what data to retrieve rather than how to retrieve it.
 - Also called **declarative** languages.
 - **Low Level or Procedural Language:**
 - Retrieve data one record-at-a-time;
 - Constructs such as looping are needed to retrieve multiple records, along with positioning pointers.

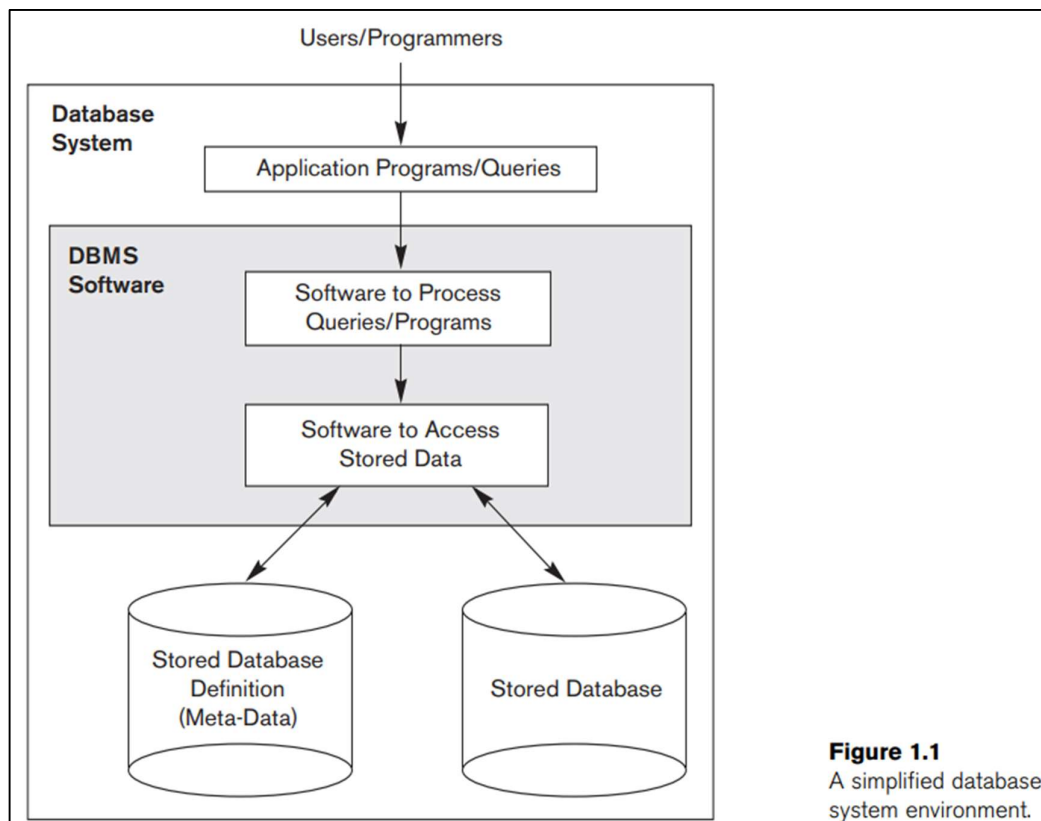
DBMS Interfaces

- Stand-alone query language interfaces
 - Example: Entering SQL queries at the DBMS interactive SQL interface (e.g. SQL*Plus in ORACLE)
- Programmer interfaces for embedding DML in programming languages.
- User-friendly interfaces
 - Menu-based, forms-based, graphics-based, etc.
- Programmer interfaces for embedding DML in a programming language:
 - **Embedded Approach:** e.g. embedded SQL (for C, C++, etc.), SQLJ (for Java)
 - **Procedure Call Approach:** e.g. JDBC for Java, ODBC for other programming languages
 - **Database Programming Language Approach:** e.g. ORACLE has PL/SQL, a programming language based on SQL; language incorporates SQL and its data types as integral components.
 - Menu-based, popular for browsing on the web.
 - Forms-based, designed for naïve users.
 - Graphics-based
 - Natural language: requests in written English
 - Combinations of the above. For example, both menus and forms are used extensively in Web interfaces.
 - Speech as Input and Output
 - Parametric interfaces, e.g., bank tellers using function keys.
- Interfaces for the DBA:
 - Creating user accounts, granting authorizations
 - Setting system parameters
 - Changing schemas or access paths

Typical DBMS Component Modules



Simplified Database System Environment



Entity & Relationships

- **Entity:** An entity is referred to as an object or thing that exists in the real world.
 - For example, customer, car, pen, etc. Entities are stored in the database, and they should be distinguishable, i.e., they should be easily identifiable from the group.
 - An entity has some attributes which depict the entity's characteristics. For example, an entity "Student" has attributes such as "Student_roll_no", "Student_name", "Student_subject", and "Student_marks".
- **Entity Type:** A collection of entities with general characteristics is known as an entity type.
 - For example, a database of a corporate company has entity types such as employees, departments, etc. In DBMS, every entity type contains a set of attributes that explain the entity.
- **Entity Set:** An entity set is a group of entities of the same entity type.
 - For example, an entity set of students, an entity set of motorbikes, an entity of smartphones, an entity of customers, etc.
- **Relationships:** Associations between entities are called relationships.
 - For example, an employee works for an organization. Here "works for" is a relation between the entity's employee and organization.

Types of Relationships

- **One-to-One Relationship**
 - When a single instance of an entity is associated with a single instance of another entity, then it is called a one-to-one relationship.
 - For example, a person has only one passport and a passport is given to one person.
- **One-to-Many Relationships**
 - When a single instance of an entity is associated with more than one instance of another entity, then it is called one-to-many relationships.
 - For example – a customer can place many orders, but an order cannot be placed by many customers.
- **Many-to-One Relationship**
 - When more than one instance of an entity is associated with a single instance of another entity, then it is called many to one relationship.
 - For example – many students can study in a single college, but a student cannot study in many colleges at the same time.
- **Many-to-Many Relationships**
 - When more than one instance of an entity is associated with more than one instance of another entity, then it is called many-to-many relationships.
 - For example, a STUDENT can be assigned to many projects and a project can be assigned to many students.

Degree of Relationship

- Degree of relationship is the number of entity sets that are participated (associated) in that relationship.
- That is, the number of entity sets that are connected through the relationship in question is called the degree of relationship.
- Based on the degree, the relationships may be identified as unary (degree 1), binary (degree 2), ternary (degree 3) and so on.
- The number of entity sets that participate in a relationship set is termed as the degree of that relationship set.
- **Unary Relationship Set**
 - It is a relationship set where only one entity set participates in a relationship set.
- **Binary Relationship Set**
 - It is a relationship set where two entity sets participate in a relationship set.
- **Ternary Relationship Set**
 - It is a relationship set where three entity sets participate in a relationship set.
- **N-ary Relationship Set**
 - It is a relationship set where 'N' entity sets participate in a relationship set.

Role Names and Recursive Relationships

- Each entity type that participates in a relationship type plays a particular role in the relationship.
- The role name signifies the **role that a participating entity from the entity type plays in each relationship instance**, and it helps to explain what the relationship means.
 - For example, in the WORKS_FOR relationship type, EMPLOYEE plays the role of employee or worker, and DEPARTMENT plays the role of department or employer.
- Role names are not technically necessary in relationship types where all the participating entity types are distinct since each participating entity type name can be used as the role name.
- However, in some cases the same entity type participates more than once in a relationship type in different roles.
- In such cases the role name becomes essential for distinguishing the meaning of the **role that each participating entity plays**. Such relationship types are called **recursive relationships** or **self-referencing relationships**.
- The **SUPERVISION** relationship type relates an **employee** to a **supervisor**, where both employee and supervisor entities are members of the same **EMPLOYEE** entity set. Hence, the **EMPLOYEE entity type** participates twice in SUPERVISION: once in the role of supervisor (or boss), and once in the role of supervisee (or subordinate).

Constraints on Relationship Types

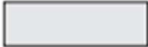

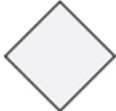




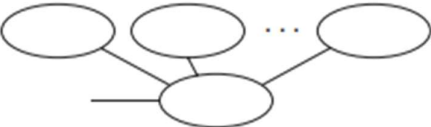

- There are two main types of relationship constraints:
 - Cardinality ratio
 - Participation
- **Cardinality of a Relationship**
 - Relationship cardinalities specify how many of each entity type is allowed.
 - The cardinality ratio for a binary relationships specifies the maximum number of relationship instances that an entity can participate in.
 - Relationships can have four connectivity's as given below.
 - One-to-one (1:1) relationship
 - One-to-many (1:N) relationship
 - Many-to-one (M:1) relationship
 - Many-to-many (M:N) relationship
 - The minimum and maximum values of this connectivity is called the cardinality of the relationship
 - Example for Cardinality – One-to-One (1:1)
- **Relationship Participation**
 - The participation constraint specifies whether the existence of an entity depends on its being related to another entity via the relationship type.
 - This constraint specifies the minimum number of relationship instances that each entity can participate in and is sometimes called the minimum cardinality constraint.
 - **Total Participation:** In total participation, every entity instance will be connected through the relationship to another instance of the other participating entity types.
 - Total participation is also called existence dependency.
 - **Partial Participation:** Only a few instances of an entity participates in a relationship.
 - Consider the relationship - Employee is head of the department. Here all employees will not be the head of the department. Only one employee will be the head of the department.
 - In other words, the employee entity's participation is partial in the said relationship.
 - However, each department will be headed by some employee. So, the department entity's participation is total in the said relationship.

Weak Entity type

- An entity that cannot be uniquely identified by its own attributes and relies on the relationship with other entity is called weak entity.
- The weak entity is represented by a double rectangle.
- For example – a bank account cannot be uniquely identified without knowing the bank to which the account belongs, so bank account is a weak entity.
- For any weak entity set, following restrictions must be held.
 - The owner entity set, and the weak entity set must participate in a one-to-many relationship set, which is called as the 'Identifying Relationship Set' of the weak entity set.
 - The weak entity set must have total participation in the identifying relationship set.

ER Diagram

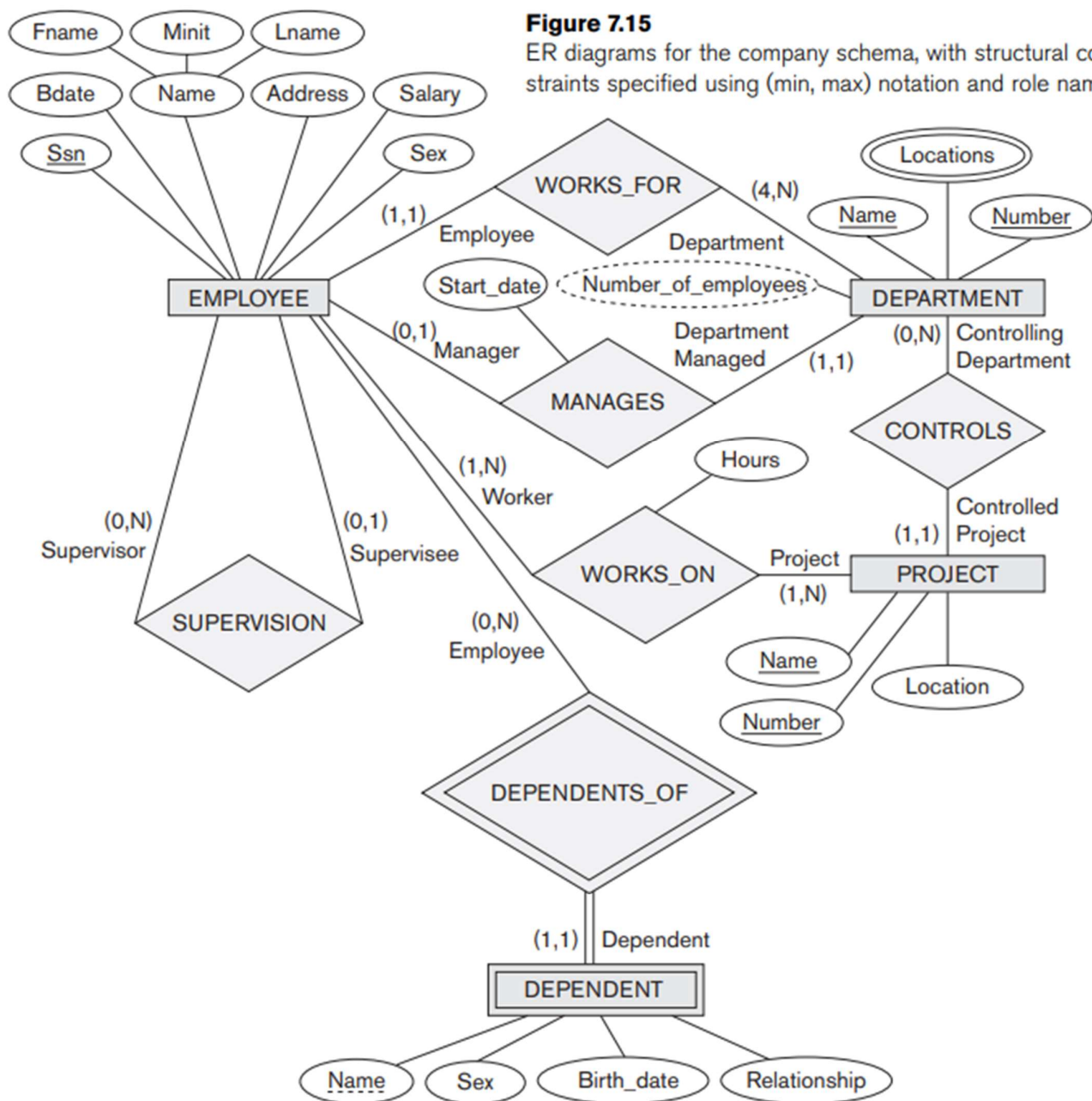
■ Symbols

Symbol	Meaning
	Entity
	Weak Entity
	Relationship
	Identifying Relationship
	Attribute
	Key Attribute
	Multivalued Attribute
	Composite Attribute
	Derived Attribute

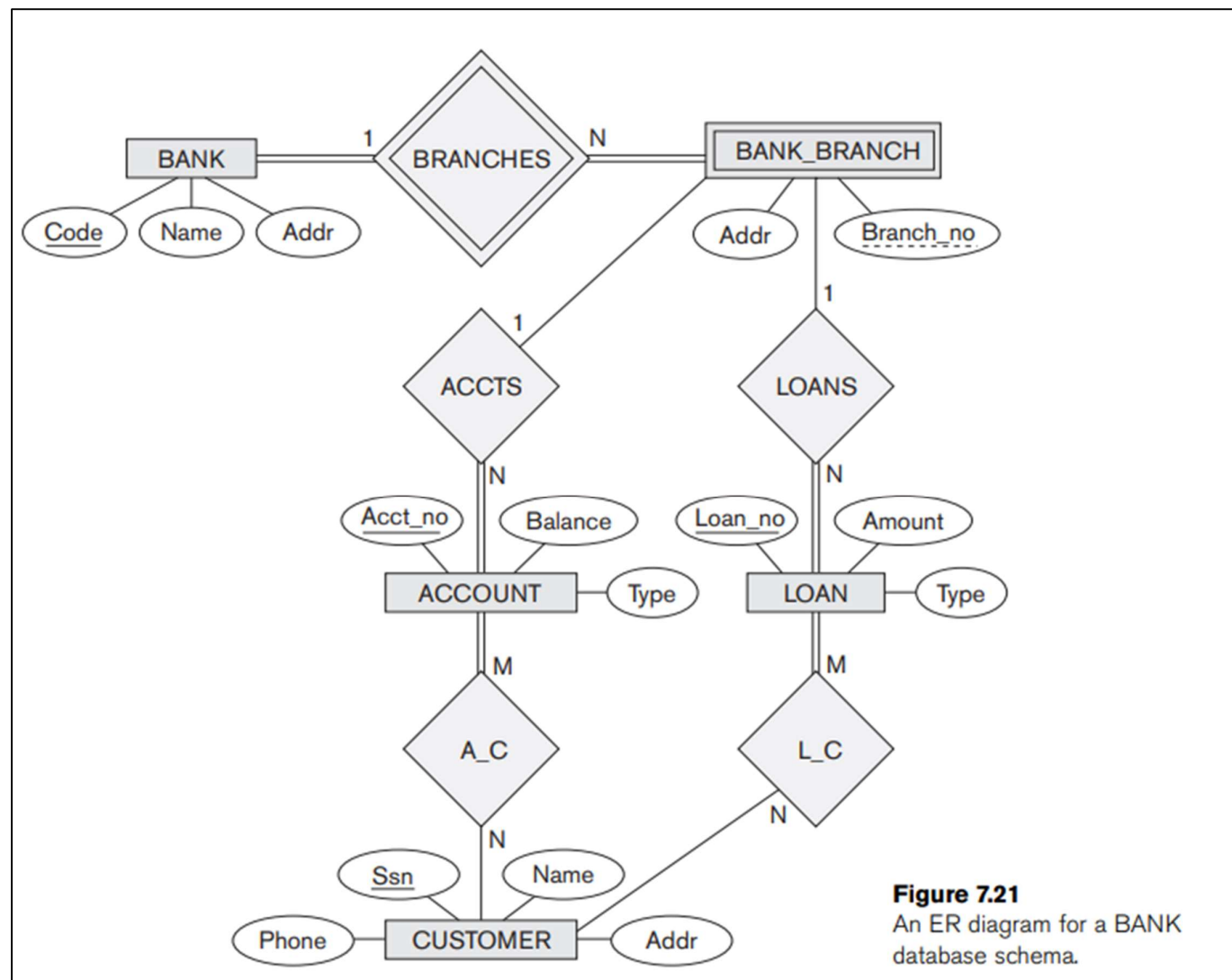
ER Diagram for Company Database

Figure 7.15

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



ER Diagram for Bank Database



Module 2

Relational Model Concepts

- A relation is a mathematical concept based on the ideas of sets.
- A relation is basically a table with rows and columns.
- An attribute is the characteristics of a relation.
- Each row in a table is called a tuple.
- Key of a relation is a unique identifier that is present in each row of a table.
 - For example, in a STUDENT table, a USN is considered a key.
- Each attribute has a domain or a set of valid values it can accept.
 - For example, the domain of a phone number is 10 digit integer value.
- Schema of a relation:
 - Denoted by $R(A_1, A_2, A_3, \dots, A_n)$.
 - R is the name of relation.
 - Attributes of a relation are A_1, A_2, \dots, A_n
 - For example, CUSTOMER(ID, Name, Address, Phone).
- Tuple is an ordered set of values.
 - Each value of a tuple is derived from an appropriate domain.
 - A row in the CUSTOMER relation is a 4-tuple and consists of 4 values.
- A relation is a set of tuples.
- Domain:
 - Attribute Name:
 - Used to interpret the meaning of the data elements corresponding to the attribute.
 - Gives the characteristics of a column header.
- State:
 - The relation state is a subset of the cartesian product of the domains of its attributes.
 - Each domain contains a set of all possible values the attribute can take.
- $r(R)$:
 - A specific state / populated state of a relation R.
 - This is a set of tuples.
 - $r(R) = \{t_1, t_2, \dots, t_n\}$ where each t_i is an n-tuple.
 - $t_1 = \langle v_1, v_2, \dots, v_n \rangle$ where each v_j element of domain (A_j) .

Relational Integrity Constraints

- Constraints are conditions that must hold on to all valid relation states.
- Three main types of relational models.
 - Key Constraints.
 - Entity Integrity Constraints.
 - Referential Integrity Constraints.
- Domain Constraint: Every value in a tuple must be from the domain of its attribute or null (if it is allowed by the attribute).