Database Management Systems I (DBMS I)

BCSE1-412

Chapter 1

Databases And Database Users

Topics to be covered

- Types of Database Applications
- Basic Definitions
- Example of a Database
- Main Characteristics of Database Technology
- Database Users
- Advantages of Database Technology
- Extending Database Functionality
- When Not to Use a DBMS

Types of Databases and Database Applications

Numeric and Textual Databases

Multimedia Databases

Geographic Information Systems (GIS)

Data Warehouses

Real-time and Active Databases

Basic Definitions

Database: A collection of related data.

Data: Known facts that can be recorded and have an implicit meaning.

Mini-world: Some part of the real world about which data is stored in a 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.

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 for the example: Part of a UNIVERSITY environment.

Some mini-world *entities*:
- STUDENTs
- COURSEs
- SECTIONs (of COURSEs)
- (academic) DEPARTMENTs
- INSTRUCTORs

Some mini-world *relationships*:

- SECTIONs *are of* specific COURSEs
- STUDENTs *take* SECTIONs
- COURSEs *have* prerequisite COURSEs
- INSTRUCTORs *teach* SECTIONs
- COURSEs *are offered by* DEPARTMENTs
- STUDENTs *major in* DEPARTMENTs

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

Figure 1.1 A simplified database system environment, illustrating the concepts and terminology discussed in Section 1.1.

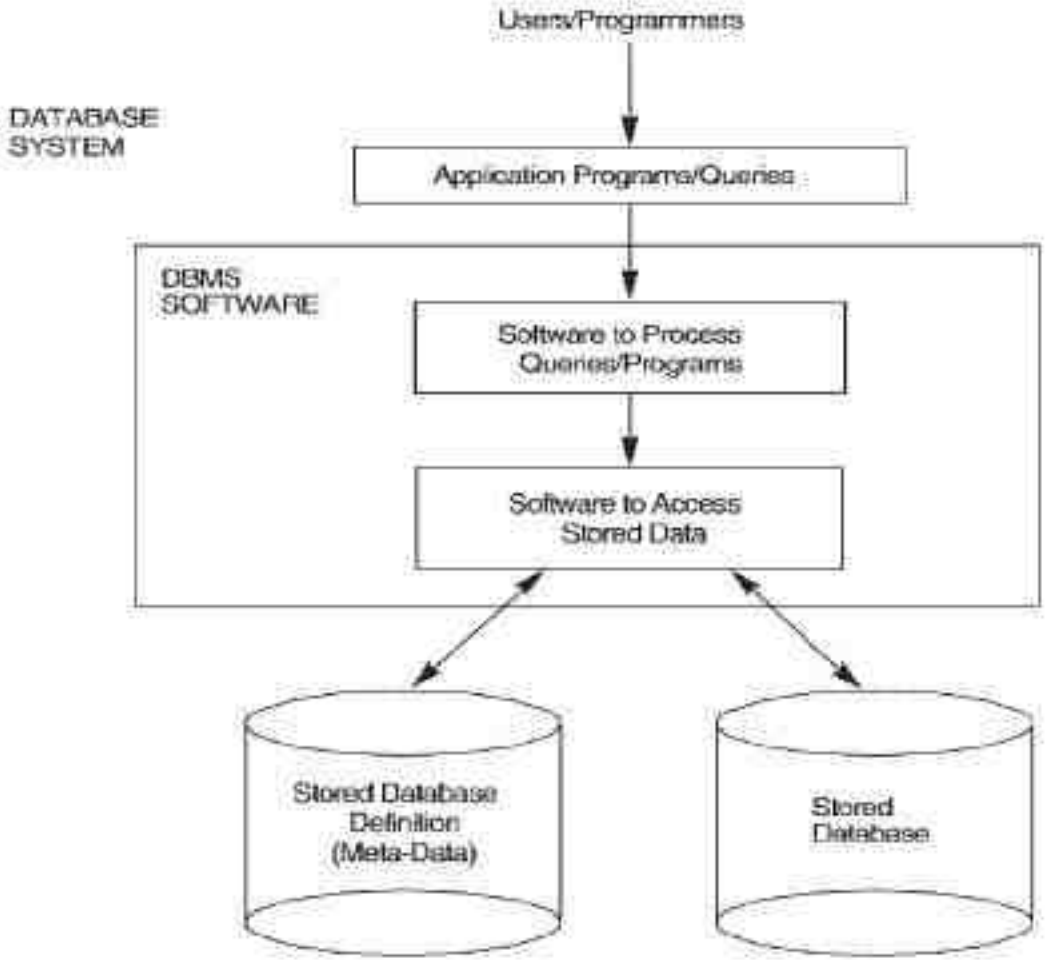


Figure 1.2 An example of a database that stores student records and their grades.

STUDENT	Name	StudentNumber	Class	Major
	Smith	17	1	CS
	Brown	8	2	CS

COURSE	CourseName	CourseNumber	CreditHours	Department
	Intro to Computer Science	CS1310	4	CS
	Data Structures	CS3320	4	CS
	Discrete Mathematics	MATH2410	3	MATH
	Database	CS3380	3	CS

SECTION	SectionIdentifier	CourseNumber	Semester	Year	Instructor
	85	MATH2410	Fall	98	King
	92	CS1310	Fall	98	Anderson
	102	CS3320	Spring	99	Knuth
	112	MATH2410	Fall	99	Chang
	119	CS1310	Fall	99	Anderson
	135	CS3380	Fall	99	Stone

GRADE_REPORT	StudentNumber	SectionIdentifier	Grade
	17	112	B
	17	119	C
	8	85	A
	8	92	A
	8	102	B
	8	135	A

PREREQUISITE	CourseNumber	PrerequisiteNumber
	CS3380	CS3320
	CS3380	MATH2410
	CS3320	CS1310

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.

- 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.

Figure 1.4 Two views derived from the example database shown in Figure 1.2. (a) The student transcript view. (b) The course prerequisite view.

(a)

TRANSCRIPT	StudentName	Student Transcript				
		CourseNumber	Grade	Semester	Year	SectionId
	Smith	CS1310	C	Fall	99	119
		MATH2410	B	Fall	99	112
	Brown	MATH2410	A	Fall	98	85
		CS1310	A	Fall	98	92
		CS3320	B	Spring	99	102
		CS3380	A	Fall	99	135

(b)

PREREQUISITES	CourseName	CourseNumber	Prerequisites
	Database	CS3380	CS3320
			MATH2410
	Data Structures	CS3320	CS1310

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 co-ordinating 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.

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

Historical Development of Database Technology

Early Database Applications: The Hierarchical and Network Models were introduced in mid 1960's and dominated during the seventies. A bulk of the worldwide database processing still occurs using these models.

Relational Model based Systems: The model that was originally introduced in 1970 was heavily researched and experimented with in IBM and the universities. Relational DBMS Products emerged in the 1980's.

Object-oriented applications: OODBMSs were introduced in late 1980's and early 1990's to cater to the need of complex data processing in CAD and other applications. Their use has not taken off much.

Data on the Web and E-commerce Applications: Web contains data in HTML (Hypertext markup language) with links among pages. This has given rise to a new set of applications and E-commerce is using new standards like XML (eXtended Markup Language).

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.

Chapter 2

Database System Concepts And Architecture

Topics to be covered

1 Data Models

1A. History of data Models

1B. Network Data Model

1C. Hierarchical Data Model

2 Schemas versus Instances

3 Database Schema vs. Database State

4 Three-Schema Architecture

5 Data Independence

6 DBMS Languages

7 DBMS Interfaces

8 Database System Environment

9 Classification of DBMSs

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.)

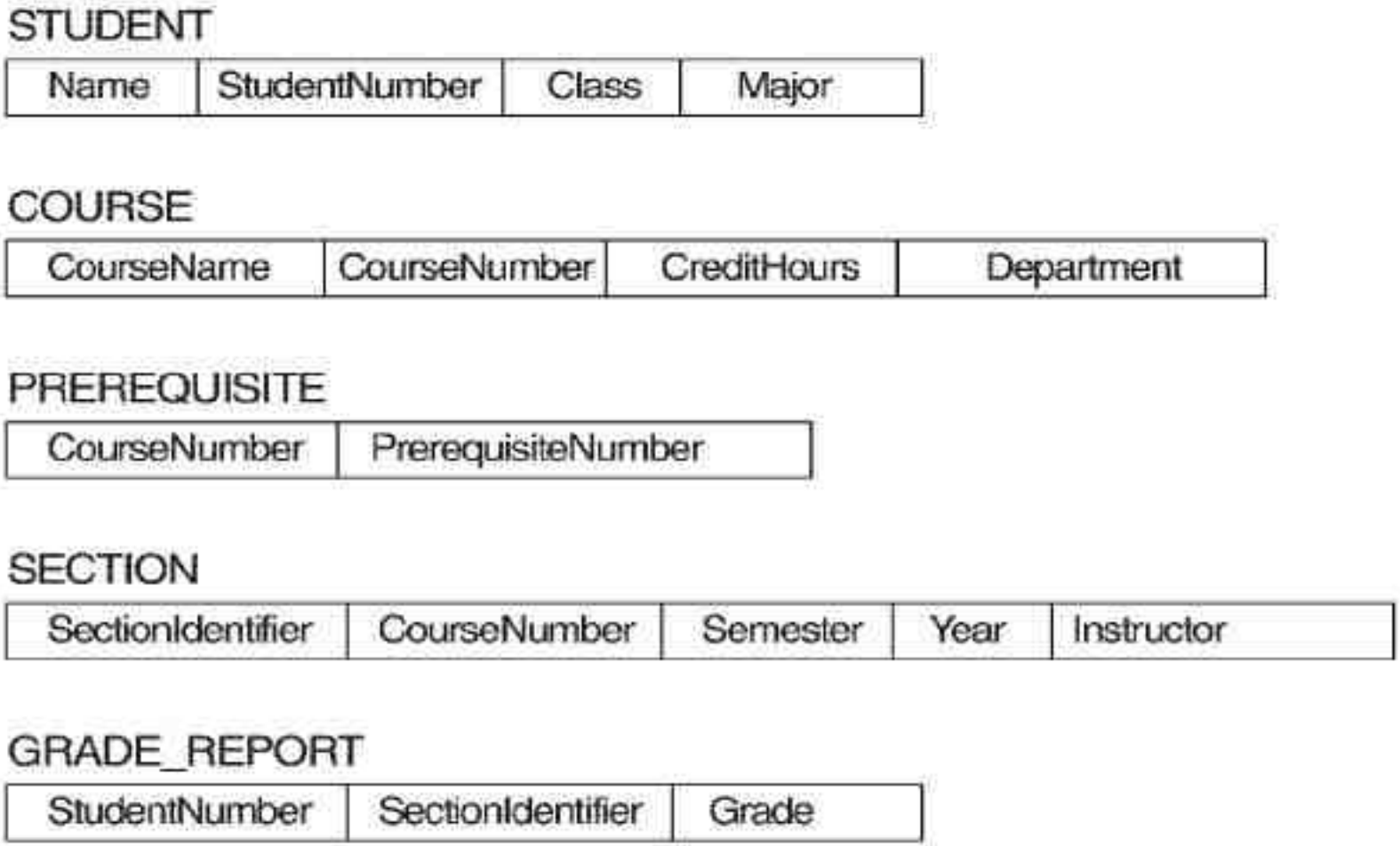
- **Physical** (**low-level**, **internal**) data models: Provide concepts that describe details of how data is stored in the computer.

- **Implementation** (**representational**) data models: Provide concepts that fall between the above two, balancing user views with some computer storage details.

HISTORY OF DATA MODELS

- Relational Model: proposed in 1970 by E.F. Codd (IBM), first commercial system in 1981-82.
- Network Model: the first one to be implemented by Honeywell in 1964-65 (IDS System). Later implemented in a large variety of systems -DMS 1100 (Unisys), IMAGE (H.P.), VAX -DBMS (Digital Equipment Corp.).
- Hierarchical Data Model: a joint effort by IBM and North American Rockwell around 1965. Resulted in the IMS family of systems. The most popular model.
- Object-oriented Data Model(s): comprises models of persistent O-O Programming Languages such as C++
- Object-Relational Models: Most Recent Trend. Started with Informix Universal Server. Exemplified in the latest versions of Oracle-10i, DB2, and SQL Server etc. systems.

Figure 2.1 Schema diagram for the database of Figure 1.2.



HIERARCHICAL MODEL

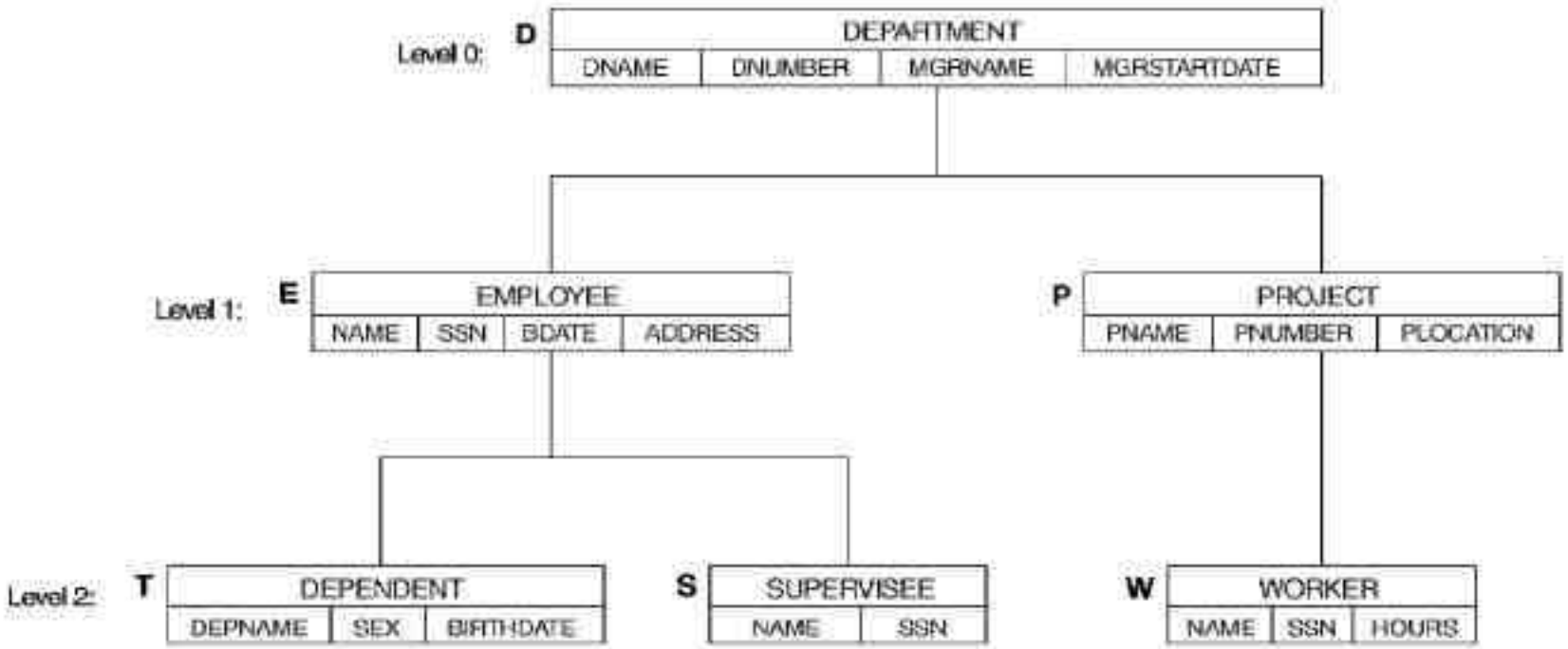
ADVANTAGES:

- Hierarchical Model is simple to construct and operate on
- Corresponds to a number of natural hierarchically organized domains - e.g., assemblies in manufacturing, personnel organization in companies
- Language is simple

DISADVANTAGES:

- Navigational and procedural nature of processing
- Database is visualized as a linear arrangement of records
- Little scope for "query optimization"

Figure D.4 A hierarchical schema for part of the COMPANY database.



NETWORK MODEL

ADVANTAGES:

• Network Model is able to model complex relationships and represents semantics of add/delete on the relationships.

• Can handle most situations for modeling using record types and relationship types.

• Language is navigational; uses constructs like FIND, FIND member, FIND owner, FIND NEXT within set, GET etc. Programmers can do optimal navigation through the database.

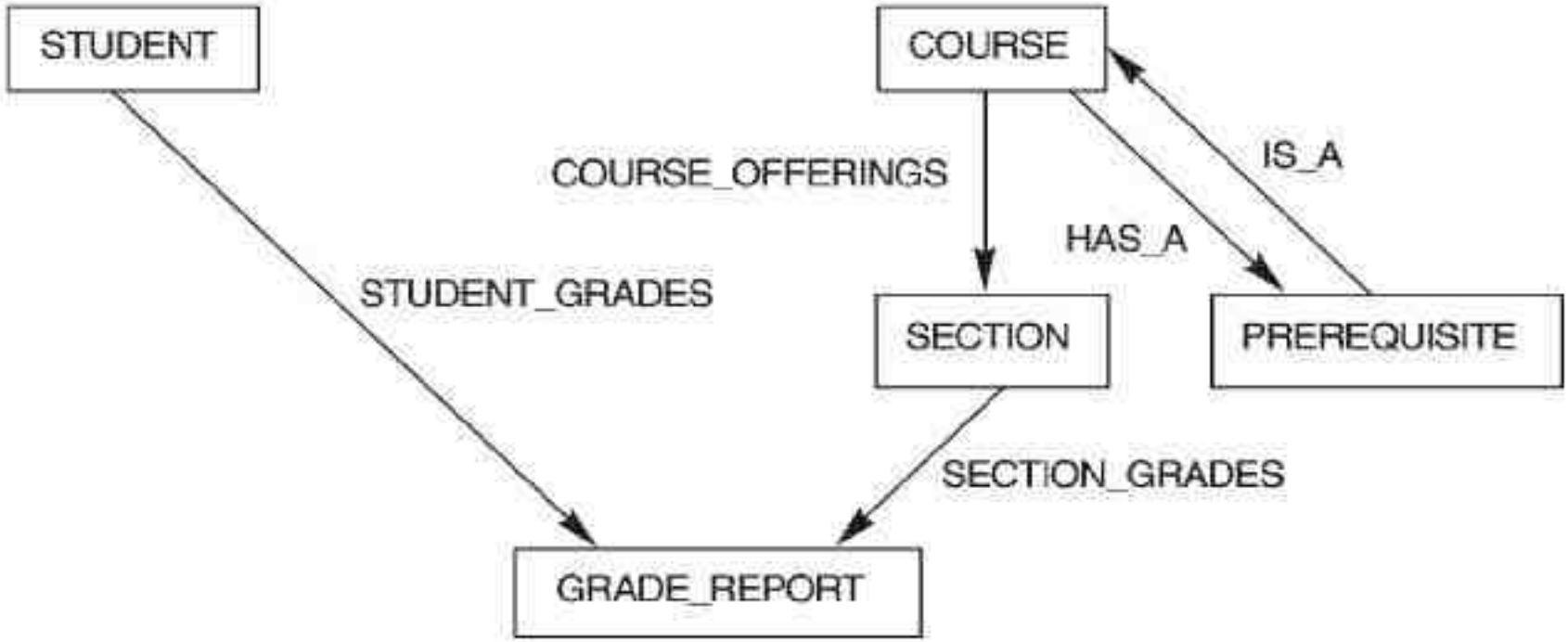
DISADVANTAGES:

• Navigational and procedural nature of processing

• Database contains a complex array of pointers that thread through a set of records.

• Little scope for automated "query optimization"

Figure 2.4 The schema of Figure 2.1 in the notation of the network data model.



Schemas versus Instances

Database Schema: 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*. Th **database state** changes *every time the database is updated.*
- **Schema** is also called **intension**, whereas **state** is calle **extension**.

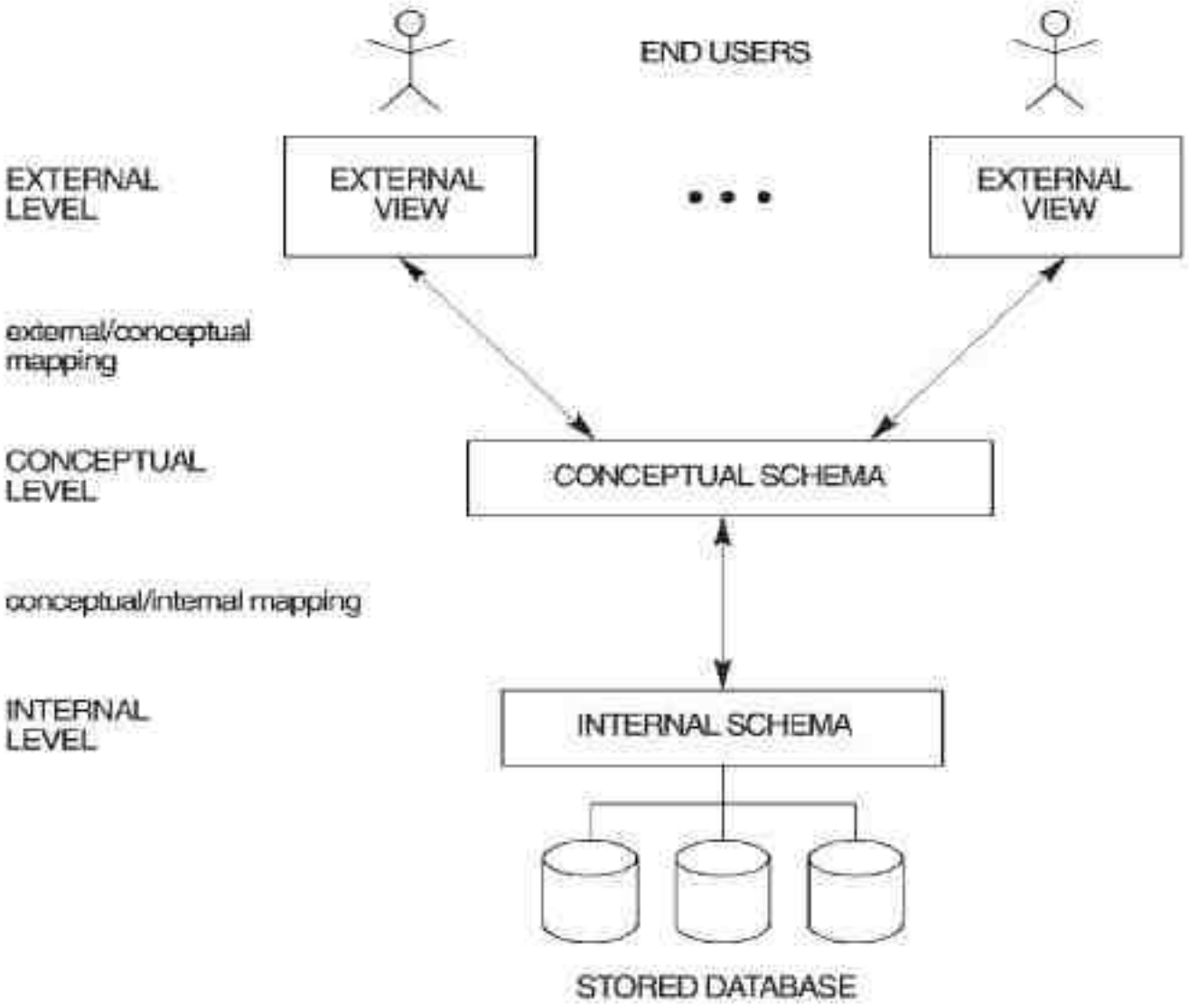
Three-Schema Architecture

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. 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 level.

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.

Figure 2.2 Illustrating the three-schema architecture.



Data Independence

Logical Data Independence: The capacity to change the conceptual schema without having to change the external schemas and their application programs.

Physical Data Independence: The capacity to change the internal schema without having to change the conceptual schema.

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

DBMS Languages

Data Definition Language (**DDL**): Used by 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**).

High Level or **Non-procedural Languages:** e.g., SQL, are *set-oriented* and specify what data to retrieve than how to retrieve. Also called *declarative* languages.

Low Level or **Procedural Languages:** record-at-a-time; they specify *how* to retrieve data and include constructs such as looping.

DBMS Interfaces

- Stand-alone query language interfaces.
- Programmer interfaces for embedding DML i programming languages:
  - Pre-compiler Approach
  - Procedure (Subroutine) Call Approach
- User-friendly interfaces:

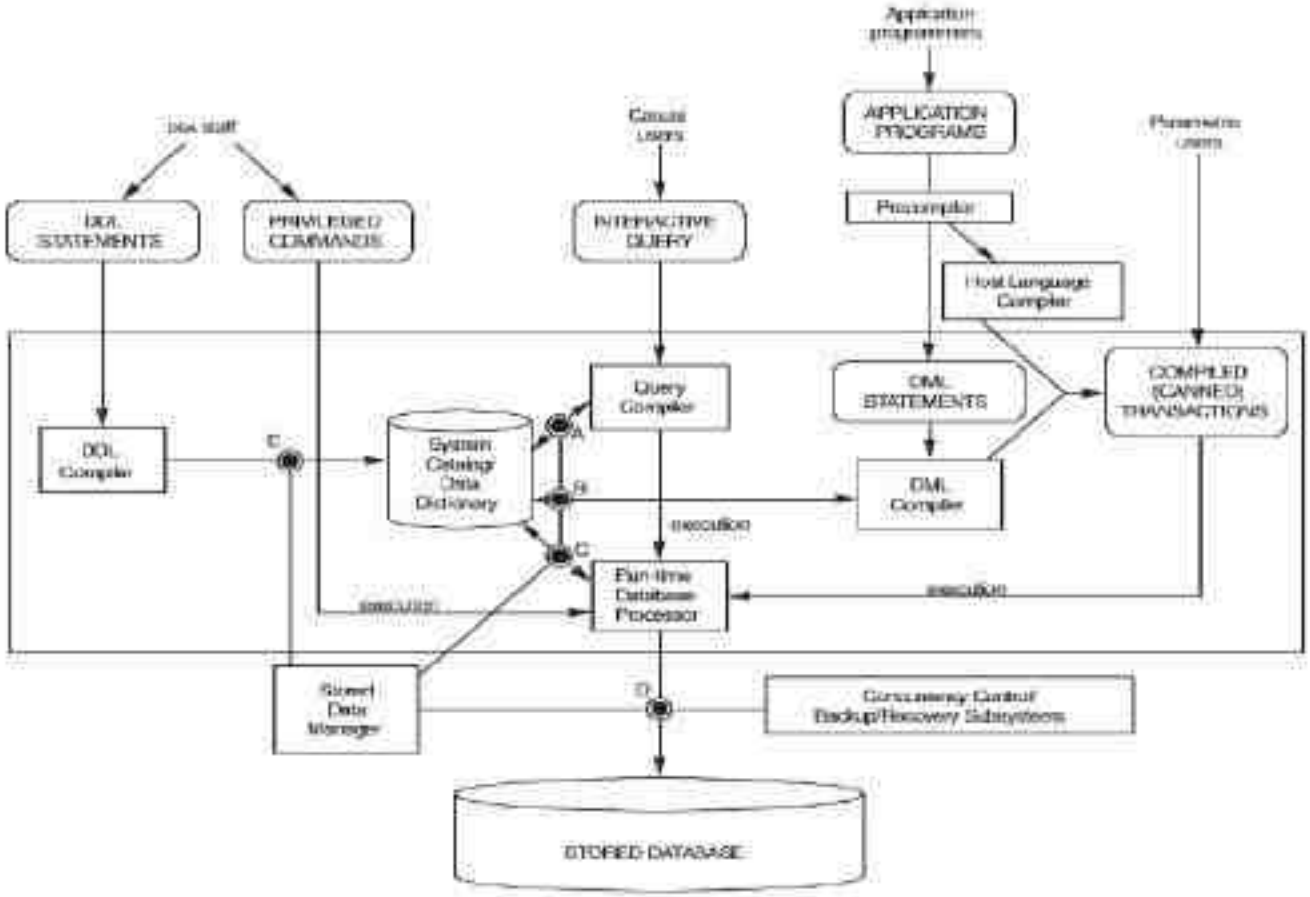
- Menu-based, popular for browsing on the web
- Forms-based, designed for naïve users
- Graphics-based (Point and Click, Drag and Drop etc.)
- Natural language: requests in written English
- Combinations of the above

Others:

- Speech as 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

Database System Environment

Figure 2.3 Typical component modules of a DBMS. Dotted lines show accesses that are under the control of the stored data manager.



Classification of DBMSs

Based on the data model used:

- Traditional: Relational, Network, Hierarchical.
- Emerging: Object-oriented, Object-relational.

Other classifications:

- Single-user (typically used with micro- computers) vs. multi-user (most DBMSs).
- Centralized (uses a single computer with one database) vs. distributed (uses multiple computers, multiple databases)

Distributed Database Systems *have now come to be known as client server based database systems because they do not support a totally distributed environment, but rather a set of database servers supporting a set of clients.*

Chapter 3

Data Modeling Using the Entity-Relationship (ER) Model

Chapter Outline

- Example Database Application (COMPANY)
- ER Model Concepts
  - Entities and Attributes
  - Entity Types, Value Sets, and Key Attributes
  - Relationships and Relationship Types
  - Weak Entity Types
  - Roles and Attributes in Relationship Types
- ER Diagrams - Notation
- ER Diagram for COMPANY Schema
- Alternative Notations – UML class diagrams, others

Example COMPANY Database

- Requirements of the Company (oversimplified for illustrative purposes)
  - 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.
  - Each department *controls* a number of PROJECTs. Each project has a name, number and is located at a single location.

Example COMPANY Database (Cont.)

- 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 employee.

ER Model Concepts

- **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 a 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 (1)

- **Simple**
  - Each entity has a single atomic value for the attribute. For example, SSN or Sex.
- **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)

- 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 SET corresponding to the ENTITY TYPE CAR

CAR

Registration(RegistrationNumber, State), VehicleID, Make, Model, Year, {Color}

car_1

((ABC 123, TEXAS), TK629, Ford Mustang, convertible, 1999, {red, black})

car_2

((ABC 123, NEW YORK), WP9872, Nissan 300ZX, 2-door, 2002, {blue})

car_3

((VSY 720, TEXAS), TD729, Buick LeSabre, 4-door, 2003, {white, blue})

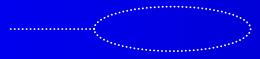
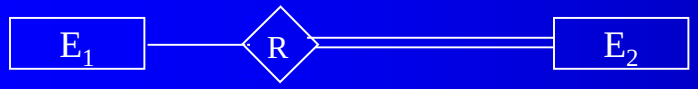
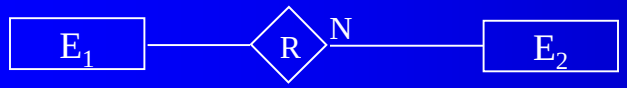
.

.

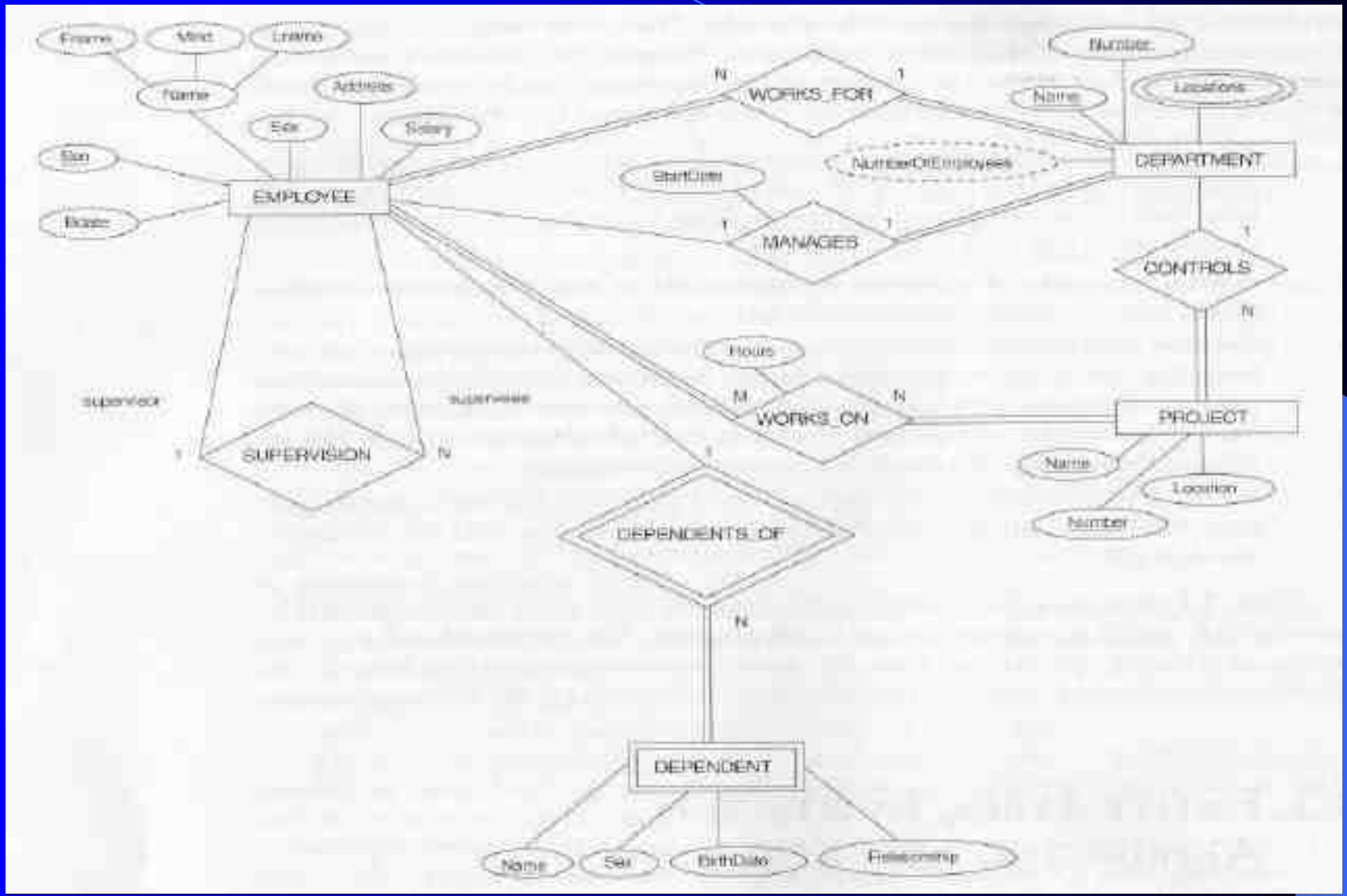
.

SUMMARY OF ER-DIAGRAM

NOTATION FOR ER SCHEMAS

Symbol	Meaning
	ENTITY TYPE
	WEAK ENTITY TYPE
	RELATIONSHIP TYPE
	IDENTIFYING RELATIONSHIP TYPE
	ATTRIBUTE
	KEY ATTRIBUTE
	MULTIVALUED ATTRIBUTE
	COMPOSITE ATTRIBUTE
	DERIVED ATTRIBUTE
	TOTAL PARTICIPATION OF E_2 IN R
	CARDINALITY RATIO 1:N FOR E_1:E_2 IN R
	STRUCTURAL CONSTRAINT (min, max) ON PARTICIPATION OF E IN R

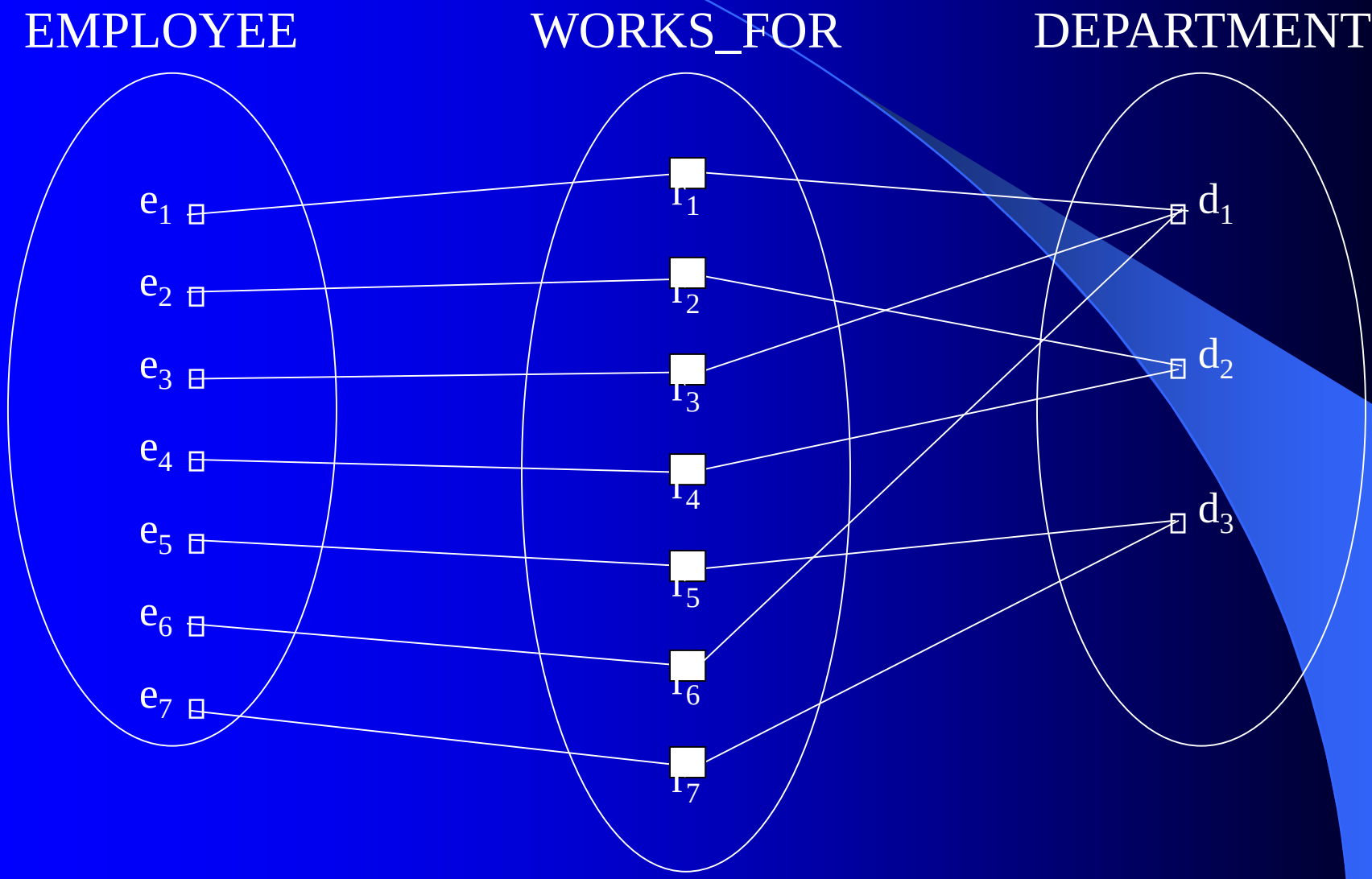
ER DIAGRAM – Entity Types are: EMPLOYEE, DEPARTMENT, PROJECT, DEPENDENT



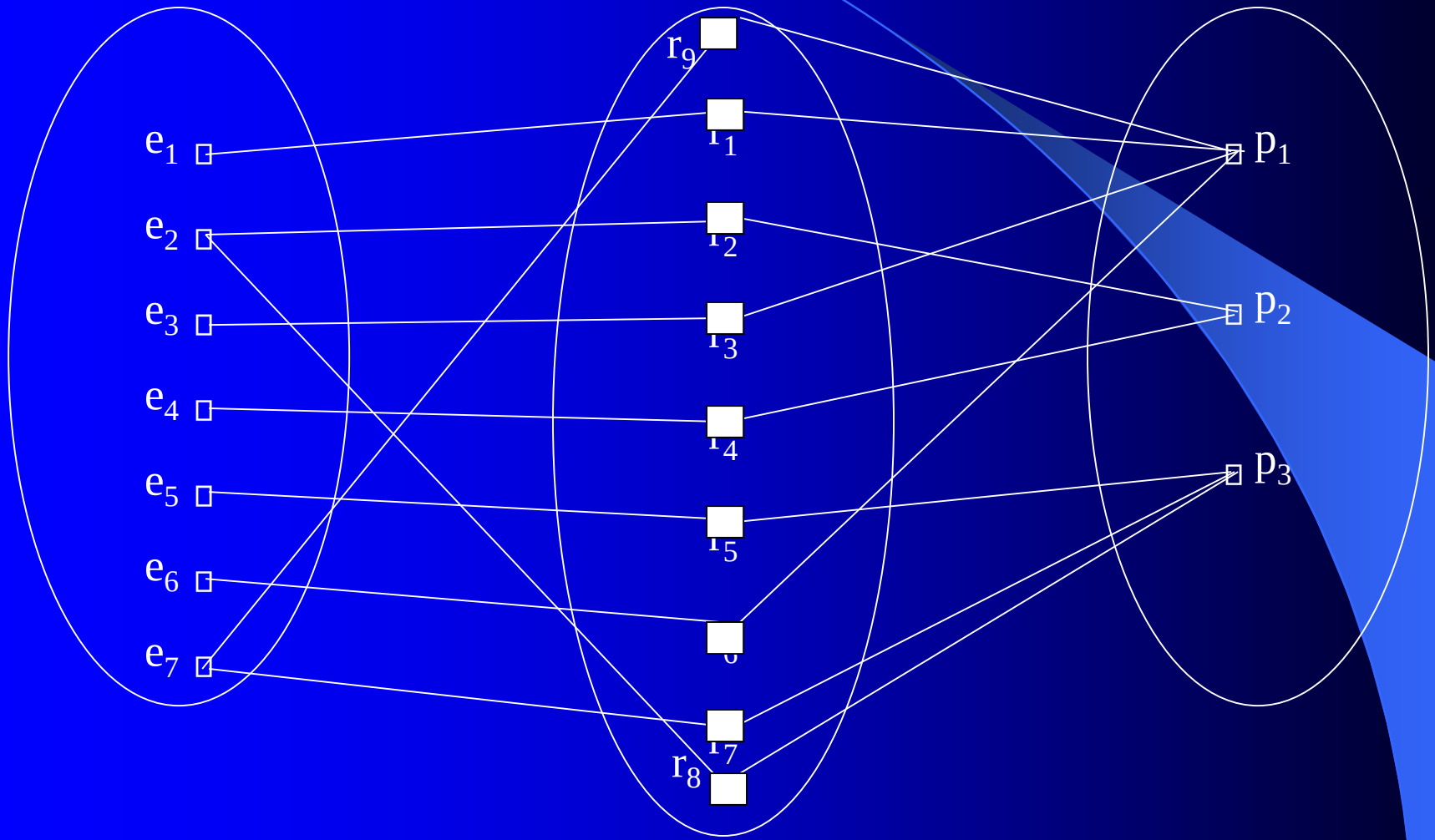
Relationships and Relationship Types (1)

- 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.
- The degree of a relationship type is the number of participating entity types. Both MANAGES and WORKS_ON are binary relationships.

Example relationship instances of the WORKS_FOR relationship between EMPLOYEE and DEPARTMENT



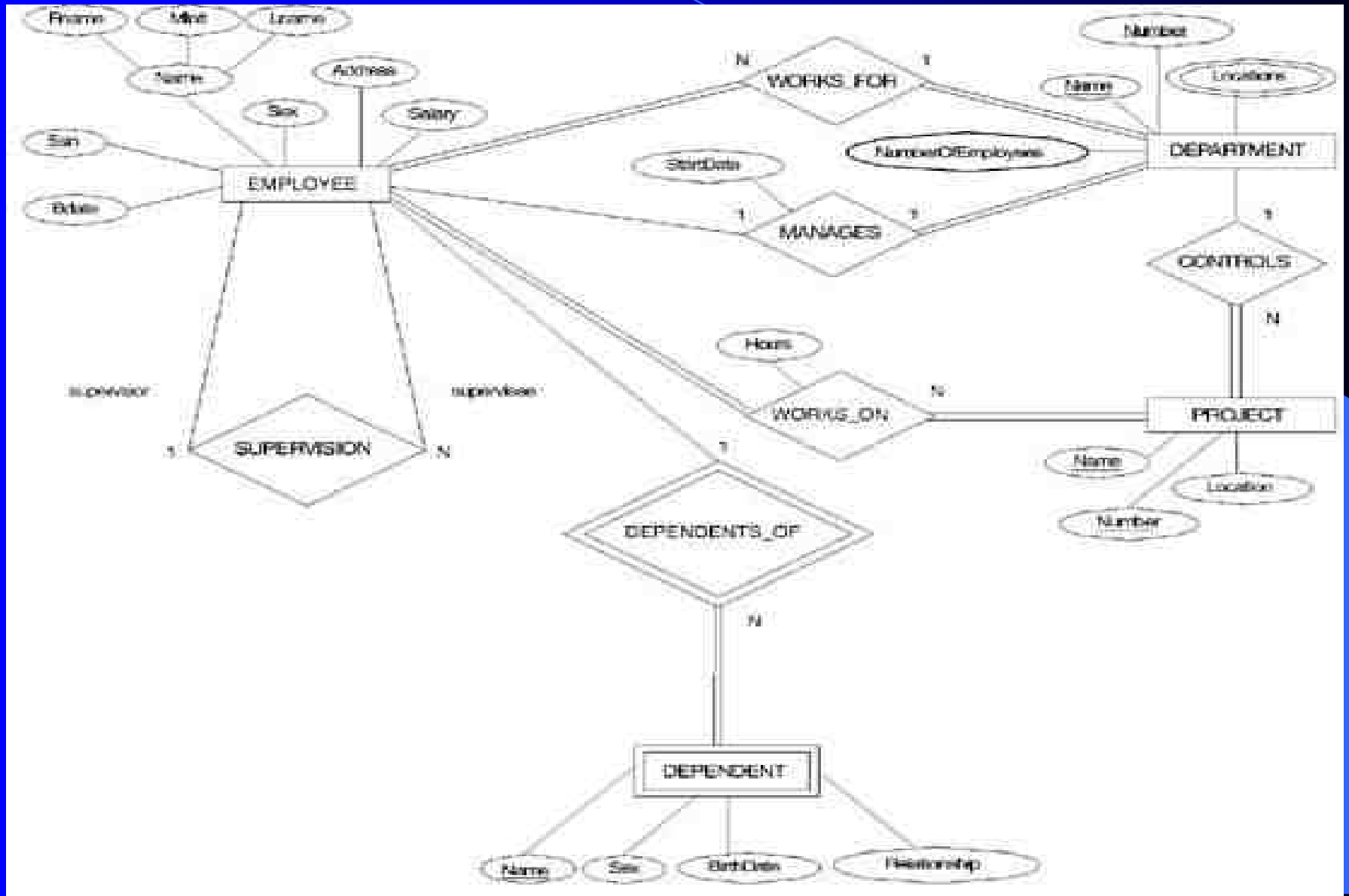
Example relationship instances of the WORKS_ON relationship between EMPLOYEE and PROJECT



Relationships and Relationship Types (2)

- More than one relationship type can exist with the same participating entity types. For example, MANAGES and WORKS_FOR are distinct relationships between EMPLOYEE and DEPARTMENT, but with different meanings and different relationship instances.

ER DIAGRAM – Relationship Types are: WORKS_FOR, MANAGES, WORKS_ON, CONTROLS, SUPERVISION, DEPENDENTS_OF



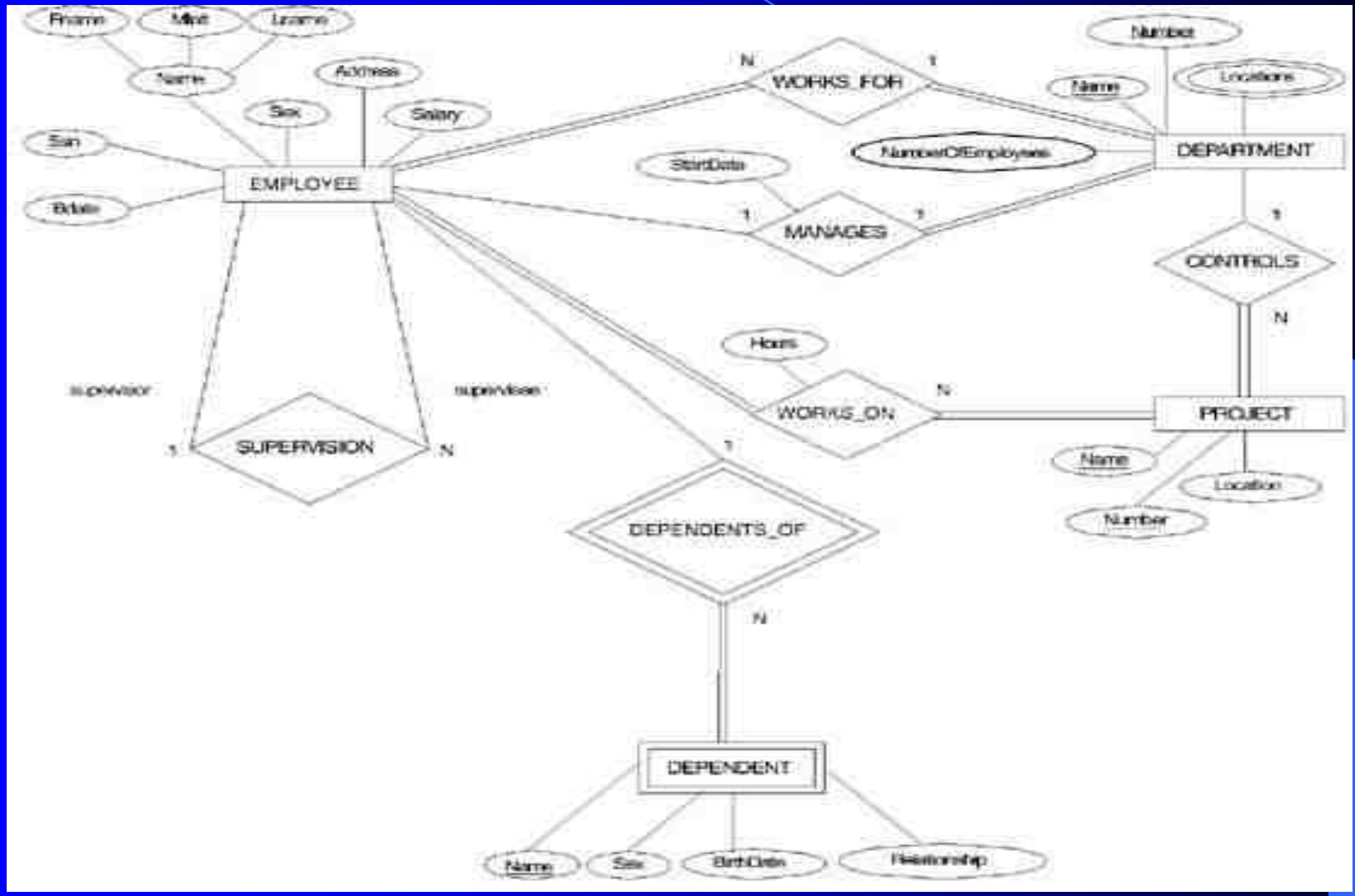
Weak Entity Types

- 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:

Suppose that a DEPENDENT entity is identified by the dependent's first name and birhtdate, *and* the specific EMPLOYEE that the dependent is related to. DEPENDENT is a weak entity type with EMPLOYEE as its identifying entity type via the identifying relationship type DEPENDENT_OF

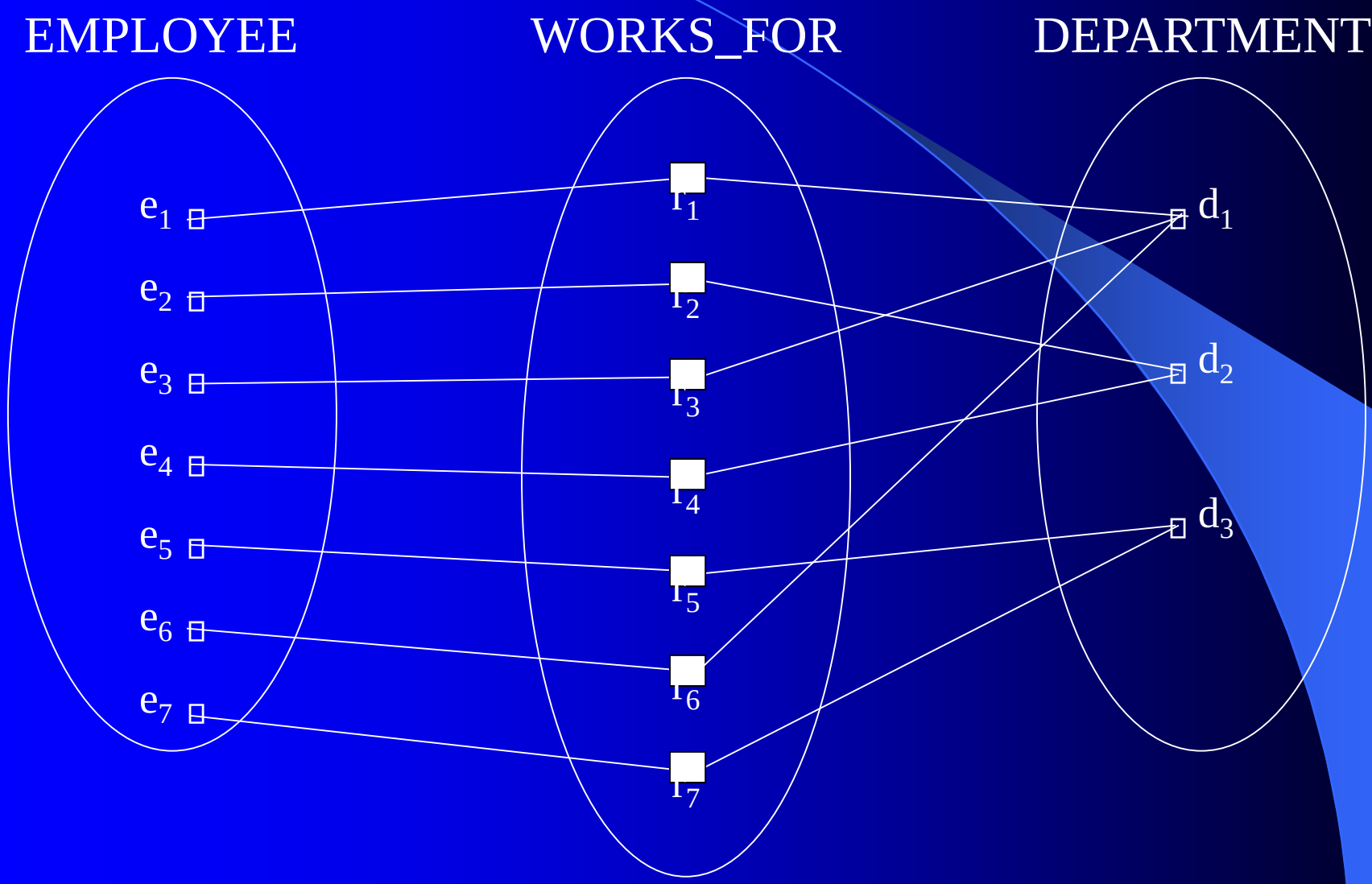
Weak Entity Type is: DEPENDENT
Identifying Relationship is: DEPENDENTS_OF



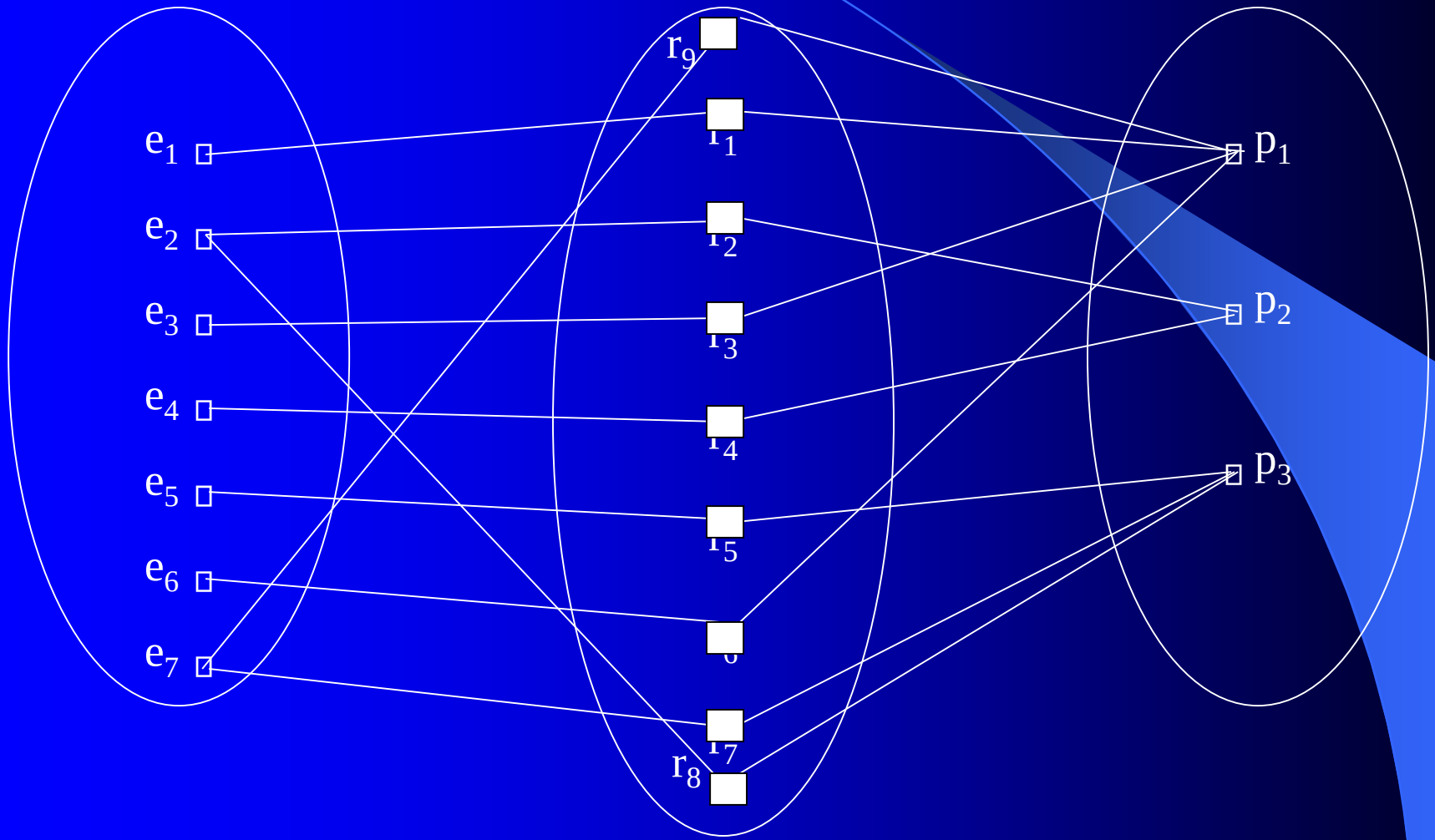
Constraints on Relationships

- Constraints on Relationship Types
  - (Also known as ratio constraints)
  - Maximum Cardinality
    - One-to-one (1:1)
    - One-to-many (1:N) or Many-to-one (N:1)
    - Many-to-many
  - Minimum Cardinality (also called participation constraint or existence dependency constraints)
    - zero (optional participation, not existence-dependent)
    - one or more (mandatory, existence-dependent)

Many-to-one (N:1) RELATIONSHIP



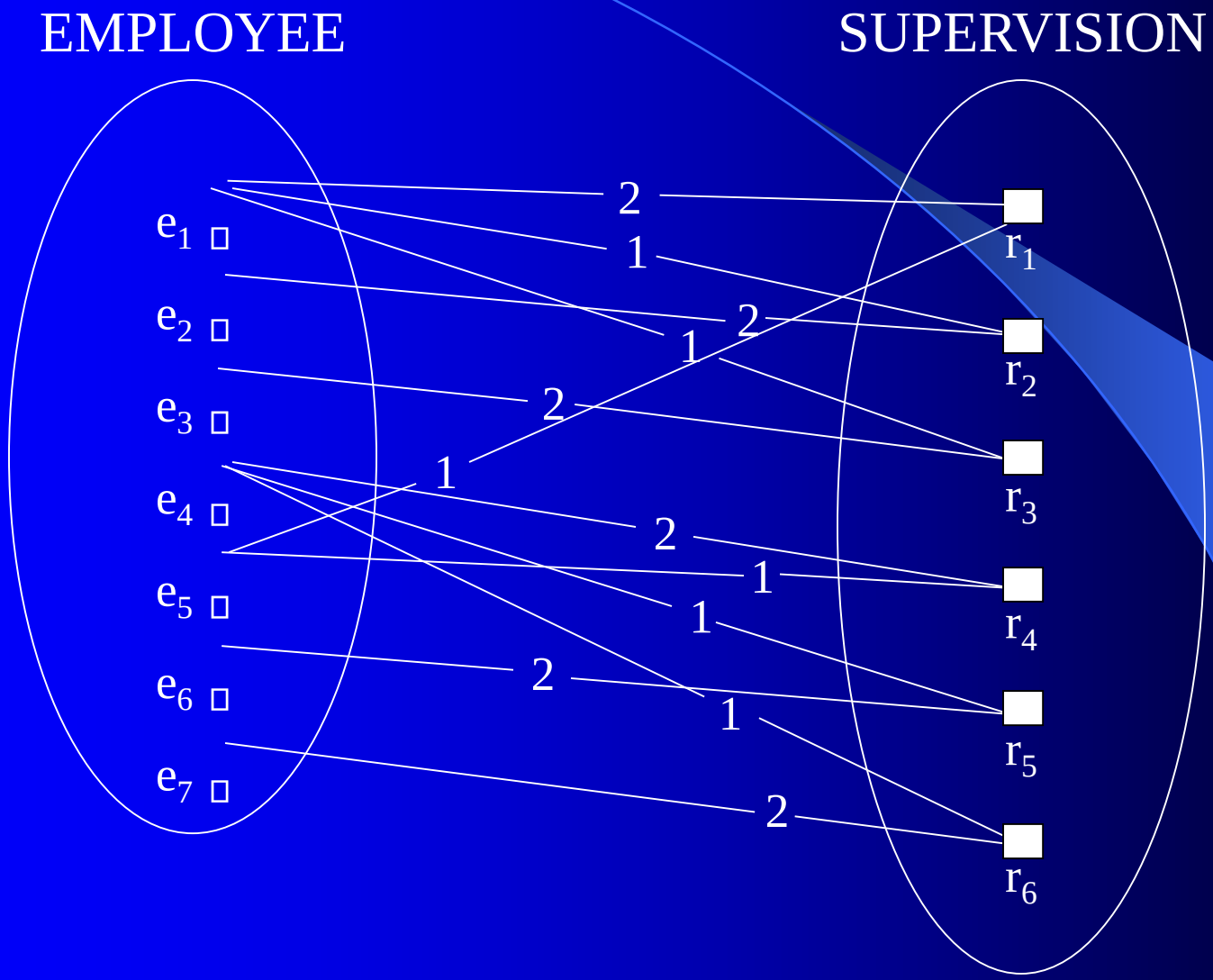
Many-to-many (M:N) RELATIONSHIP



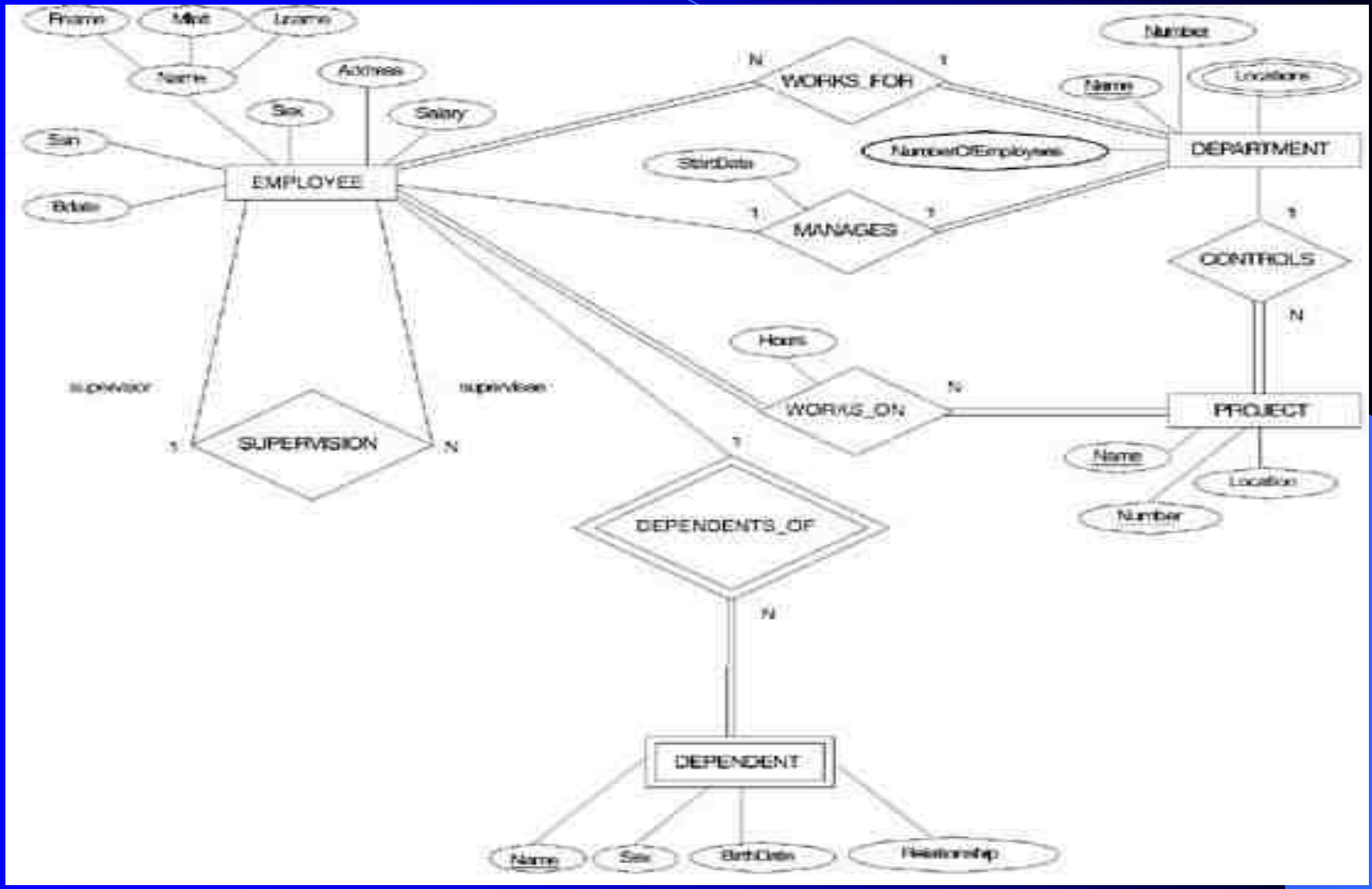
Relationships and Relationship Types (3)

- We can also have a **recursive** relationship type.
- Both participations are same entity type in different roles.
- For example, SUPERVISION relationships between EMPLOYEE (in role of supervisor or boss) and (another) EMPLOYEE (in role of subordinate or worker).
- In following figure, first role participation labeled with 1 and second role participation labeled with 2.
- In ER diagram, need to display role names to distinguish participations.

A RECURSIVE RELATIONSHIP SUPERVISION



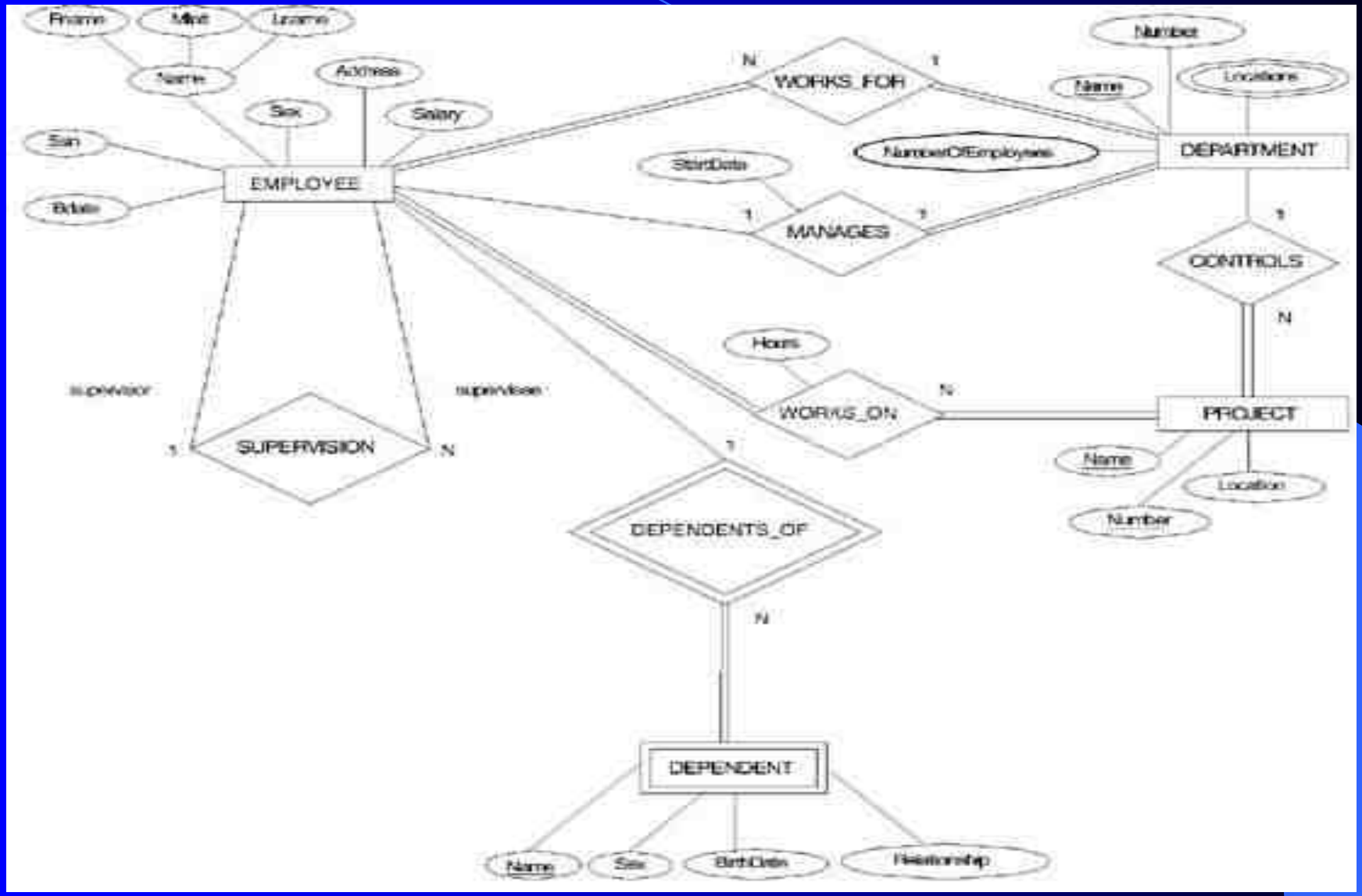
Recursive Relationship Type is: SUPERVISION (participation role names are shown)



Attributes of Relationship types

- A relationship type can have attributes; for example, HoursPerWeek of WORKS_ON; its value for each relationship instance describes the number of hours per week that an EMPLOYEE works on a PROJECT.

Attribute of a Relationship Type is: Hours of WORKS_ON



Structural Constraints – one way to express semantics of relationships

Structural constraints on relationships:

- **Cardinality ratio** (of a binary relationship): 1:1, 1:N, N:1, or M:N

  SHOWN BY PLACING APPROPRIATE NUMBER ON THE LINK.

- **Participation constraint** (on each participating entity type): total (called *existence dependency*) or partial.

  SHOWN BY DOUBLE LINING THE LINK

NOTE: These are easy to specify for Binary Relationship Types.

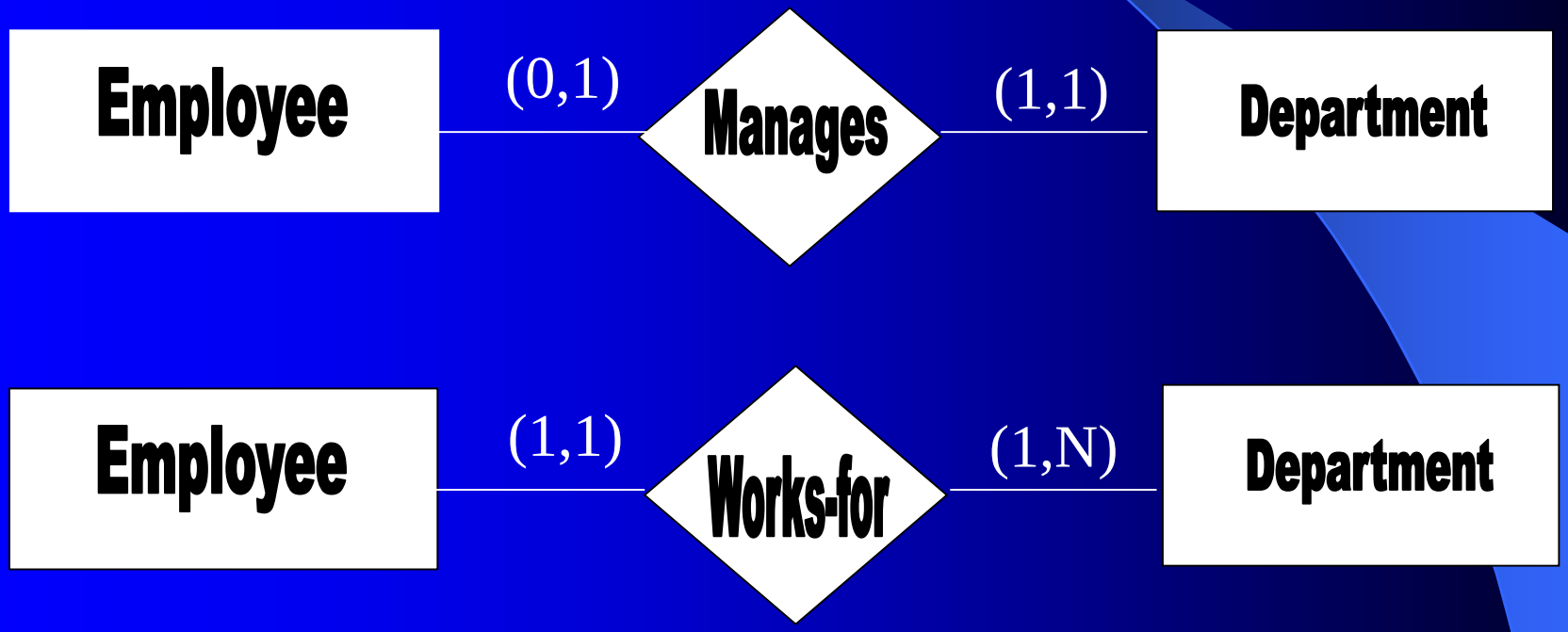
Alternative (min, max) notation for relationship structural constraints:

- 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
- Must have min≤max, min≥0, max ≥1
- Derived from the knowledge of mini-world constraints

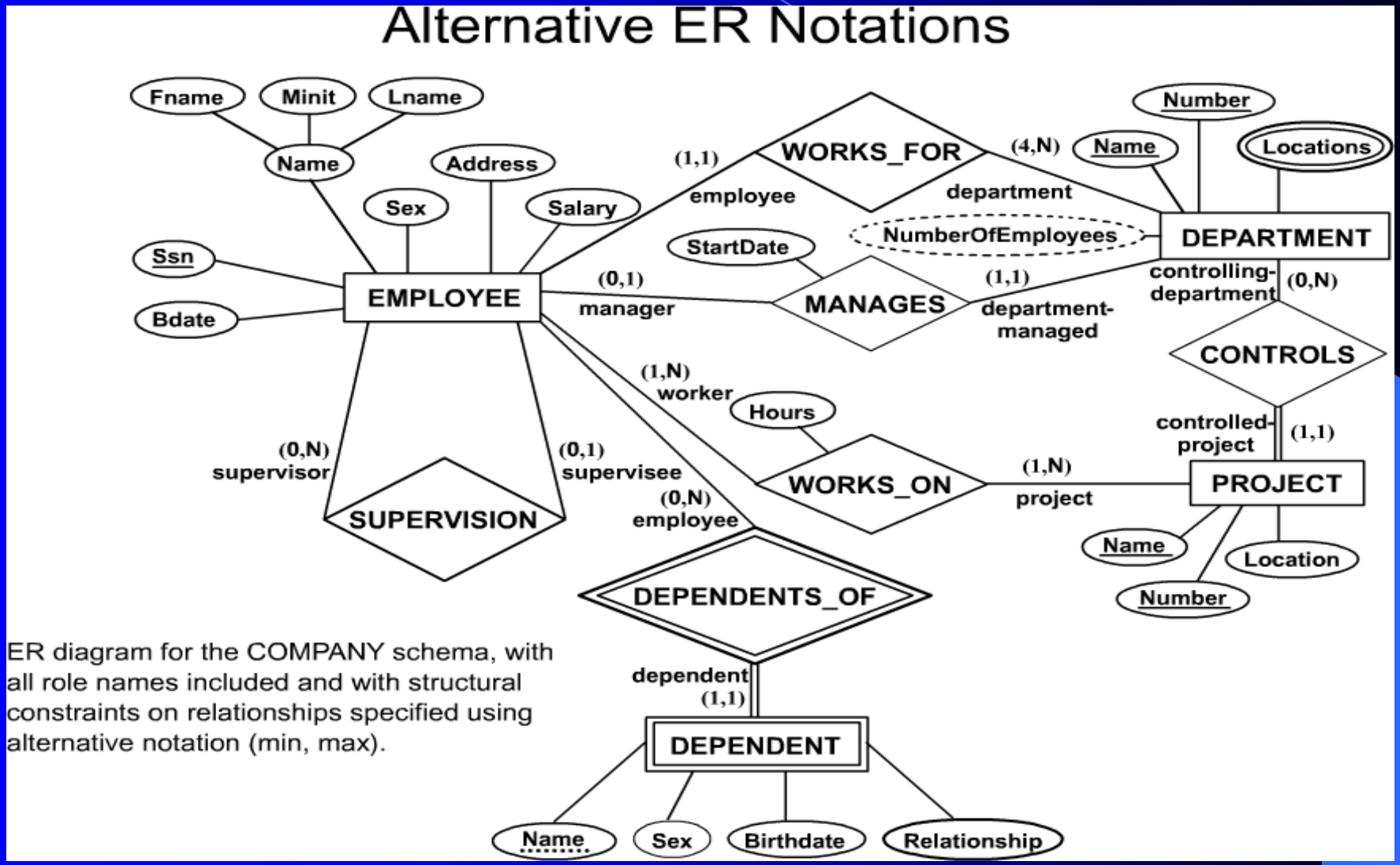
Examples:

- 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 relationship constraints



COMPANY ER Schema Diagram using (min, max) notation



ER diagram for the COMPANY schema, with all role names included and with structural constraints on relationships specified using alternative notation (min, max).

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

Data Modeling Tools

A number of popular tools that cover conceptual modeling and mapping into relational schema design. Examples: ERWin, S- Designer (Enterprise Application Suite), ER- Studio, etc.

POSITIVES: serves as documentation of application requirements, easy user interface - mostly graphics editor support

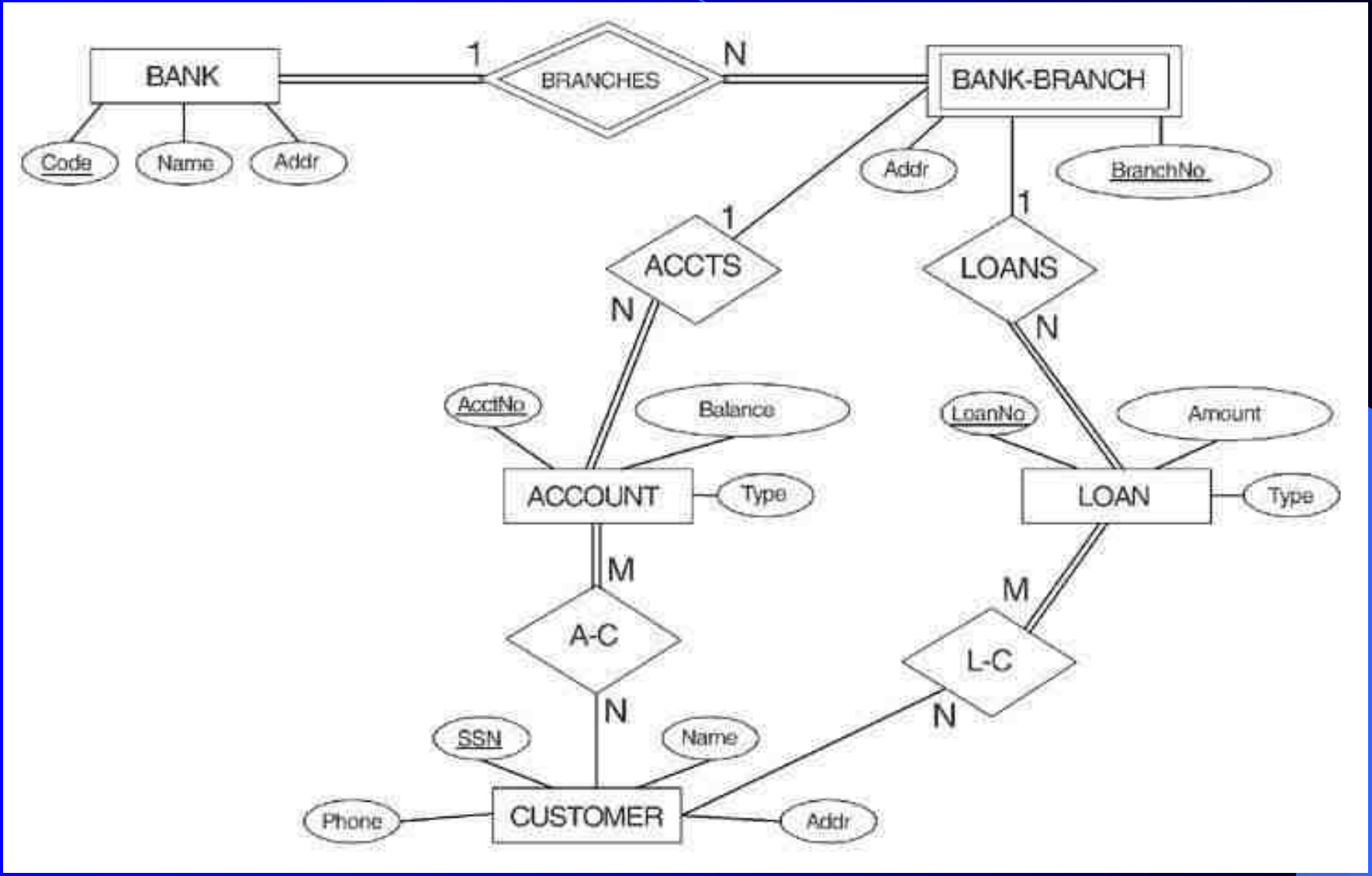
Problems with Current Modeling Tools

- DIAGRAMMING
  - Poor conceptual meaningful notation.
  - To avoid the problem of layout algorithms and aesthetics of diagrams, they prefer boxes and lines and do nothing more than represent (primary-foreign key) relationships among resulting tables.(a few exceptions)
- METHODOLGY
  - lack of built-in methodology support.
  - poor tradeoff analysis or user-driven design preferences.
  - poor design verification and suggestions for improvement.

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 and space and security management
Oracle	Developer 2000 and Designer 2000	Database modeling, application development
Popkin Software	System Architect 2001	Data modeling, object modeling, process modeling, structured analysis/design
Platinum Technology	Platinum Enterprice Modeling Suite: Erwin, BPWin, Paradigm Plus	Data, process, and business component modeling
Persistence Inc.	Pwertier	Mapping from O-O to relational model
Rational	Rational Rose	Modeling in UML and application generation in C++ and JAVA
Rogue Ware	RW Metro	Mapping from O-O to relational model
Resolution Ltd.	Xcase	Conceptual modeling up to code maintenance
Sybase	Enterprise Application Suite	Data modeling, business logic modeling
Visio	Visio Enterprise	Data modeling, design and reengineering Visual Basic and Visual C++

ER DIAGRAM FOR A BANK DATABASE



Chapter 4

The Relational Data Model and Relational Database Constraints, The Relational Algebra and Relational Calculus

Chapter Outline

- Relational Model Concepts
- Relational Model Constraints and Relational Database Schemas
- Update Operations and Dealing with Constraint Violations
- Basic Relational Algebra Operations
- Example of Queries in Relational Algebra
- The Tuple Relational Calculus
- The Domain Relational Calculus

Relational Model Concepts

- The relational Model of Data is based on the concept of a Relation.
- A Relation is a mathematical concept based on the ideas of sets.
- The strength of the relational approach to data management comes from the formal foundation provided by the theory of relations.

Relational Model Concepts

- The model was first proposed by Dr. E.F. Codd of IBM in 1970 in the following paper: "A Relational Model for Large Shared Data Banks," Communications of the ACM, June 1970.

The above paper caused a major revolution in the field of Database management and earned Ted Codd the coveted ACM Turing Award.

INFORMAL DEFINITIONS

- RELATION: A table of values
  - A relation may be thought of as a **set of rows**.
  - A relation may alternately be though of as a **set of columns**.
  - Each row represents a fact that corresponds to a real-world **entity** or **relationship**.
  - Each row has a value of an item or set of items that uniquely identifies that row in the table.
  - Sometimes row-ids or sequential numbers are assigned to identify the rows in the table.
  - Each column typically is called by its column name or column header or attribute name.

FORMAL DEFINITIONS

- A **Relation** may be defined in multiple ways.
- The **Schema** of a Relation: *R* (A1, A2, ...An)

  Relation schema *R* is defined over **attributes** A1, A2, ...An

  For Example -

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

  Here, CUSTOMER is a relation defined over the four attributes Cust-id, Cust-name, Address, Phone#, each of which has a **domain** or a set of valid values. For example, the domain of Cust-id is 6 digit numbers.

FORMAL DEFINITIONS

- A **tuple** is an ordered set of values
- Each value is derived from an appropriate domain.
- Each row in the CUSTOMER table may be referred to as a tuple in the table and would consist of four values.
- **<632895, "John Smith", "101 Main St. Atlanta, GA 30332", "(404) 894-2000">** is a tuple belonging to the CUSTOMER relation.
- A relation may be regarded as a ***set of tuples*** (rows).
- Columns in a table are also called attributes of the relation.

FORMAL DEFINITIONS

- A **domain** has a logical definition: e.g., “USA_phone_numbers” are the set of 10 digit phone numbers valid in the U.S.
- A domain may have a data-type or a format defined for it. The USA_phone_numbers may have a format: (ddd)-ddd-dddd where each d is a decimal digit. E.g., Dates have various formats such as monthname, date, year or yyyy-mm-dd, or dd mm,yyyy etc.
- An attribute designates the **role** played by the domain. E.g., the domain Date may be used to define attributes “Invoice-date” and “Payment-date”.

FORMAL DEFINITIONS

- The relation is formed over the cartesian product of the sets; each set has values from a domain; that domain is used in a specific role which is conveyed by the attribute name.
- For example, attribute Cust-name is defined over the domain of strings of 25 characters.
- Formally,

  Given $R(A_1, A_2, ..., A_n)$

  $r(R) \subset dom(A_1) \times dom(A_2) \times ....\times dom(A_n)$
- R: schema of the relation
- r of R: a specific "value" or population of R.
- R is also called the **intension** of a relation
- r is also called the **extension** of a relation

FORMAL DEFINITIONS

- Let S1 = {0,1}
- Let S2 = {a,b,c}

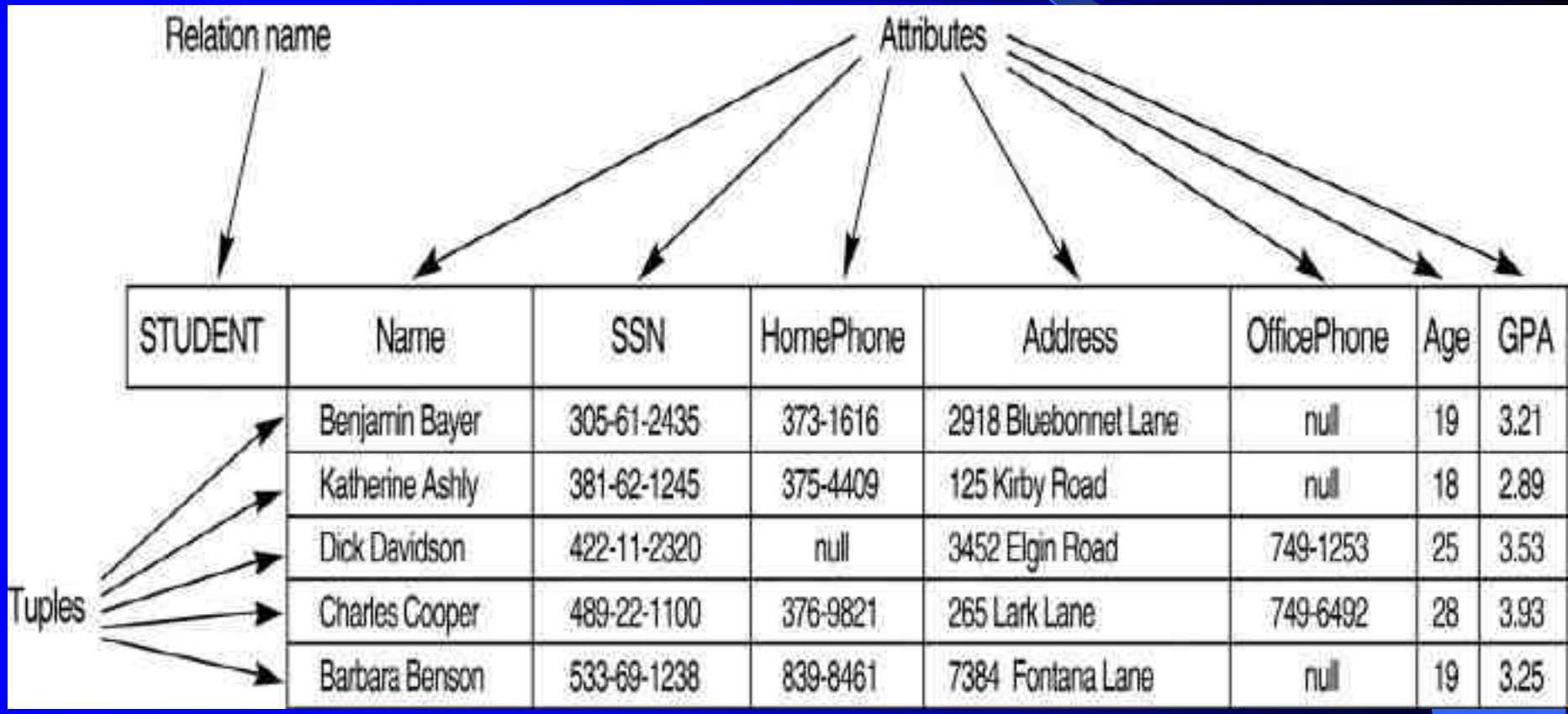
- Let $R \subset S1 \times S2$

- Then for example: r(R) = {<0,a> , <0,b> , <1,c> }
  is one possible "state" or "population" or "extension" r of the relation R, defined over domains S1 and S2. It has three tuples.

DEFINITION SUMMARY

Informal Terms		Formal Terms
Table		Relation
Column		Attribute/Domain
Row		Tuple
Values in a column		Domain
Table Definition		Schema of a Relation
Populated Table		Extension

Example - **Figure 5.1**



STUDENT	Name	SSN	HomePhone	Address	OfficePhone	Age	GPA
	Benjamin Bayer	305-61-2435	373-1616	2918 Bluebonnet Lane	null	19	3.21
	Katherine Ashly	381-62-1245	375-4409	125 Kirby Road	null	18	2.89
	Dick Davidson	422-11-2320	null	3452 Elgin Road	749-1253	25	3.53
	Charles Cooper	489-22-1100	376-9821	265 Lark Lane	749-6492	28	3.93
	Barbara Benson	533-69-1238	839-8461	7384 Fontana Lane	null	19	3.25

CHARACTERISTICS OF RELATIONS

- **Ordering of tuples in a relation r(R)**: The tuples are *not* considered to be ordered, even though they appear to be in the tabular form.
- **Ordering of attributes in a relation schema R** (and of values within each tuple): We will consider the attributes in $R(A_1, A_2, ..., A_n)$ and the values in $t=<v_1, v_2, ..., v_n>$ to be *ordered* .

  (However, a more general *alternative definition* of relation does not require this ordering).
- **Values in a tuple**: All values are considered *atomic* (indivisible). A special **null** value is used to represent values that are unknown or inapplicable to certain tuples.

CHARACTERISTICS OF RELATIONS

- Notation:

- We refer to **component values** of a tuple t by $t[A_i] = v_i$ (the value of attribute A_i for tuple t). Similarly, $t[A_u, A_v, ..., A_w]$ refers to the subtuple of t containing the values of attributes $A_u, A_v, ..., A_w$, respectively.

CHARACTERISTICS OF RELATIONS- Figure 5.2

STUDENT	Name	SSN	HomePhone	Address	OfficePhone	Age	GPA
	Dick Davidson	422-11-2320	null	3452 Elgin Road	749-1253	25	3.53
	Barbara Benson	533-69-1238	839-8461	7384 Fontana Lane	null	19	3.25
	Charles Cooper	489-22-1100	376-9821	265 Lark Lane	749-6492	28	3.93
	Katherine Ashly	381-62-1245	375-4409	125 Kirby Road	null	18	2.89
	Benjamin Bayer	305-61-2435	373-1616	2918 Bluebonnet Lane	null	19	3.21

Relational Integrity Constraints

- Constraints are *conditions* that must hold on *all* valid relation instances. There are three main types of constraints:
  1. **Key** constraints
  2. **Entity integrity** constraints
  3. **Referential integrity** constraints

Key Constraints

- **Superkey** of R: A set of attributes SK of R such that no two tuples *in any valid relation instance r(R)* will have the same value for SK. That is, for any distinct tuples t1 and t2 in r(R), t1[SK] ≠ t2[SK].
- **Key** of R: A "minimal" superkey; that is, a superkey K such that removal of any attribute from K results in a set of attributes that is not a superkey.

Example: The CAR relation schema:

CAR(State, Reg#, SerialNo, Make, Model, Year)

has two keys Key1 = {State, Reg#}, Key2 = {SerialNo}, which are also superkeys. {SerialNo, Make} is a superkey but *not* a key.

- If a relation has *several* **candidate keys**, one is chosen arbitrarily to be the **primary key**. The primary key attributes are *underlined*.

Key Constraints

Figure 7.4 5.4 : CAR relation with two candidate keys: LicenseNumber and EngineSerialNumber.

CAR	LicenseNumber	EngineSerialNumber	Make	Model	Year
	Texas ABC-739	A69352	Ford	Mustang	96
	Florida TVP-347	B43696	Oldsmobile	Cutlass	99
	New York MPO-22	X83554	Oldsmobile	Delta	95
	California 432-TFY	C43742	Mercedes	190-D	93
	California RSK-629	Y82935	Toyota	Camry	98
	Texas RSK-629	U028365	Jaguar	XJS	98

Entity Integrity

- **Relational Database Schema**: A set S of relation schemas that belong to the same database. S is the *name* of the **database**.

$S = \{R_1, R_2, ..., R_n\}$

- **Entity Integrity**: The *primary key attributes* PK of each relation schema R in S cannot have null values in any tuple of r(R). This is because primary key values are used to *identify* the individual tuples.

$t[PK] \neq$ null for any tuple t in r(R)

- Note: Other attributes of R may be similarly constrained to disallow null values, even though they are not members of the primary key.

Referential Integrity

- A constraint involving *two* relations (the previous constraints involve a *single* relation).
- Used to specify a *relationship* among tuples in two relations: the **referencing relation** and the **referenced relation**.
- Tuples in the *referencing relation* R_1 have attributes FK (called **foreign key** attributes) that reference the primary key attributes PK of the *referenced relation* R_2. A tuple t_1 in R_1 is said to **reference** a tuple t_2 in R_2 if $t_1[FK] = t_2[PK]$.
- A referential integrity constraint can be displayed in a relational database schema as a directed arc from R_1.FK to R_2.

Referential Integrity Constraint

Statement of the constraint

The value in the foreign key column (or columns) FK of the the **referencing relation** R_1 can be either:

(1) a value of an existing primary key value of the corresponding primary key PK in the **referenced relation** R_2, or..

(2) a null.

In case (2), the FK in R_1 should not be a part of its own primary key.