CS 5200 Database Management Systems

Fall 2022

Database Management Systems: An Architectural View Lecture 2 September 16, 2022

Prof. Scott Valcourt

s.valcourt@northeastern.edu

603-380-2860 (cell)

Portland, ME

The Roux Institute
at Northeastern University

What's Happening?

- Software Installation
  - MariaDB
  - Phpadmin/HeidiSQL
  - Chinook Database
- Homework Assignment 1
- Read Chapter 1 from the textbook

INTRODUCTION

- **Entity-relationship (ER) modeling** - conceptual database modeling technique
  - Enables the structuring and organizing of the requirements collection process
  - Provides a way to graphically represent the requirements
- **ER diagram (ERD)** - the result of ER modeling
  - Serves as a blueprint for the database

ENTITIES

- **Entities -** constructs that represent what the database keeps track of
  - The basic building blocks of an ER diagram
  - Represent various real world notions, such as people, places, objects, events, items, and other concepts
  - Within one ERD, each entity must have a different name

ENTITIES

Two entities

ENTITIES

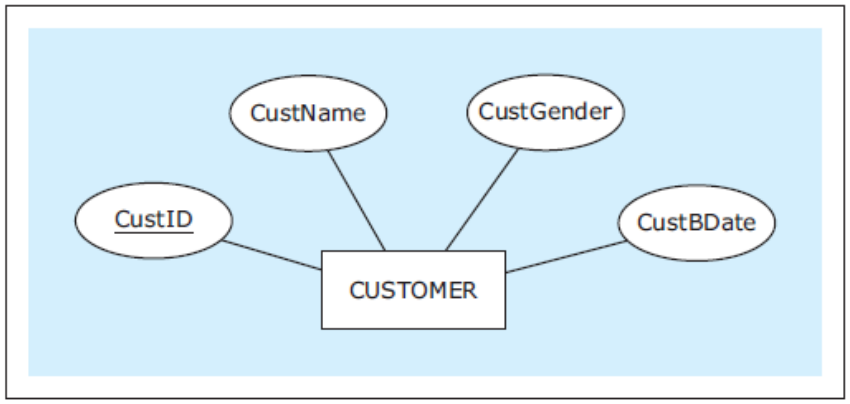
- **Entity instances (entity members)** - occurrences of an entity
  - Entities themselves are depicted in the ER diagrams while entity instances are not
  - Entity instances are eventually recorded in the database that is created based on the ER diagram

ATTRIBUTES

- **Attribute -** depiction of a characteristic of an entity
  - Represents the details that will be recorded for each entity instance
  - Within one entity, each attribute must have a different name
- **Unique Attribute** - attribute whose value is different for each entity instance
  - Every regular entity must have at least one unique attribute

ATTRIBUTES

An entity with attributes



RELATIONSHIPS

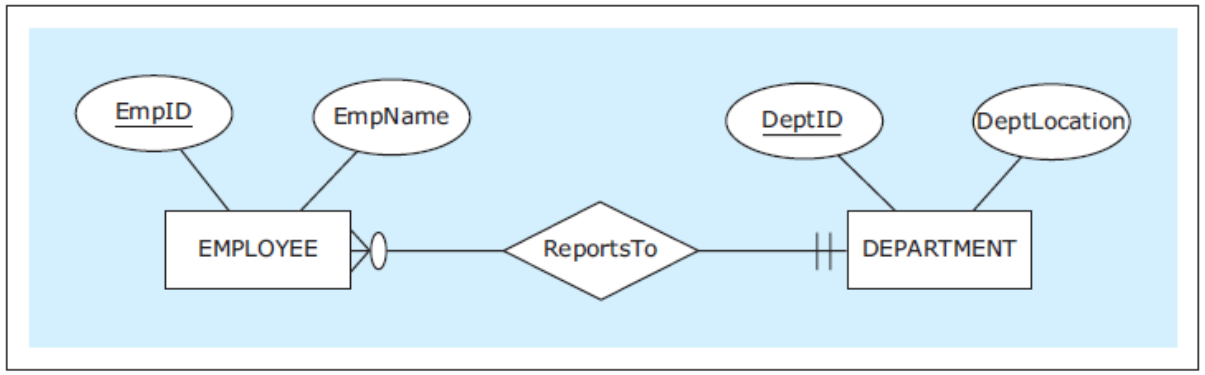
- **Relationship -** ER modeling construct depicting how entities are related
  - Within an ER diagram, each entity must be related to at least one other entity via a relationship

RELATIONSHIPS

- **Cardinality constraints -** depict how many instances of one entity can be associated with instances of another entity
  - **Maximum cardinality**
    - **One** (represented by a straight bar: I)
    - **Many** (represented by a crow's foot symbol)
  - **Minimum cardinality (participation)**
    - **Optional** (represented by a circular symbol: 0)
    - **Mandatory** (represented by a straight bar: I)

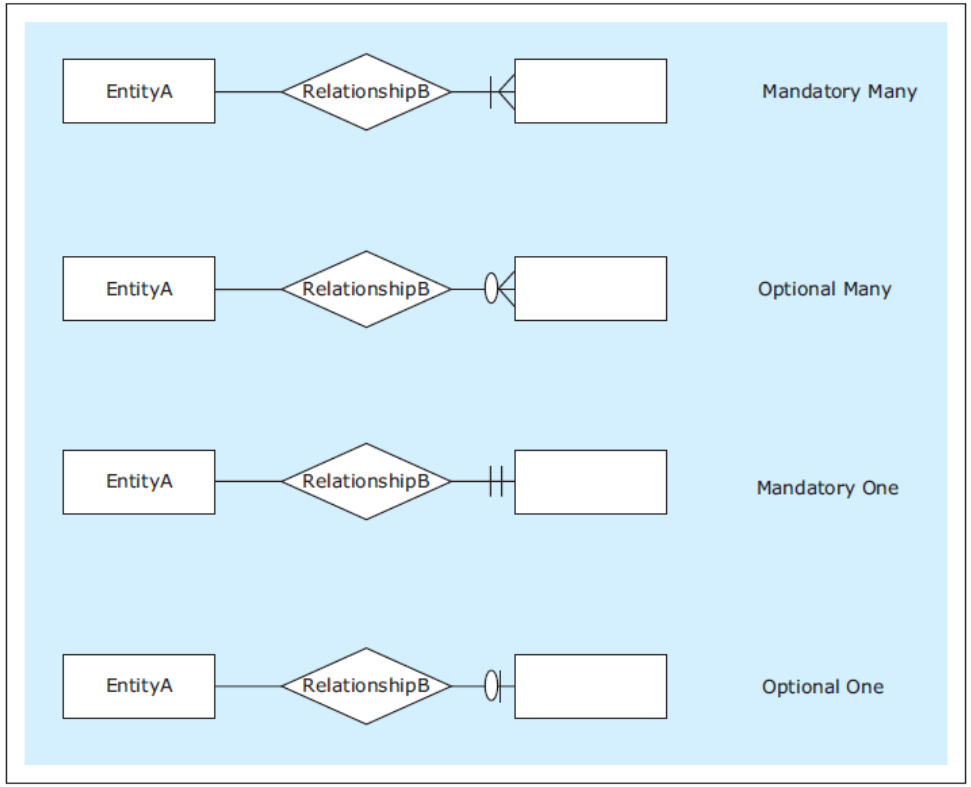
RELATIONSHIPS

A relationship between two entities



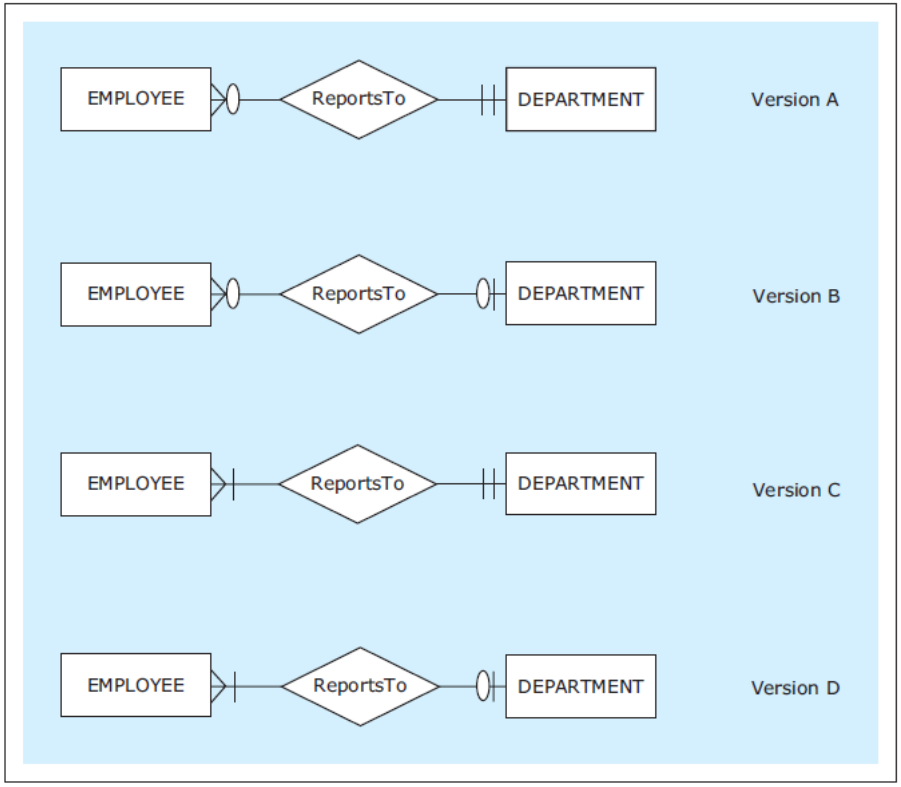
RELATIONSHIPS

Four possible cardinality constraints



RELATIONSHIPS

Several possible versions of the relationship ReportsTo

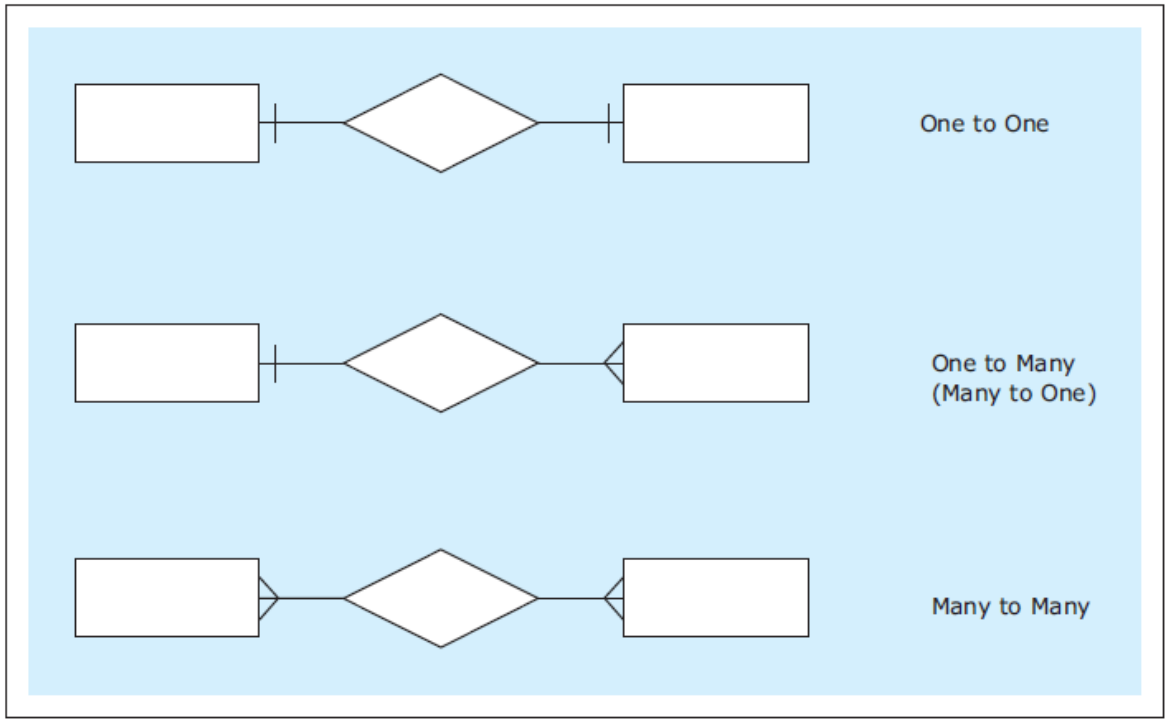


RELATIONSHIPS

- Types of Relationships (maximum cardinality-wise)
  - One-to-one relationship (1:1)
  - One-to-many relationship (1:M)
  - Many-to-many relationship (M:N)

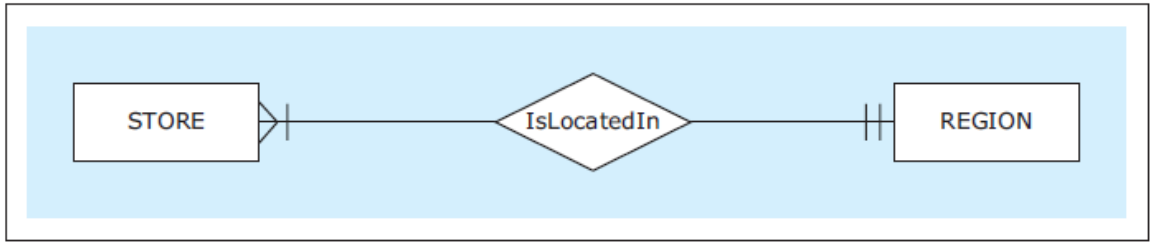
RELATIONSHIPS

Three types of relationships (maximum cardinality-wise)

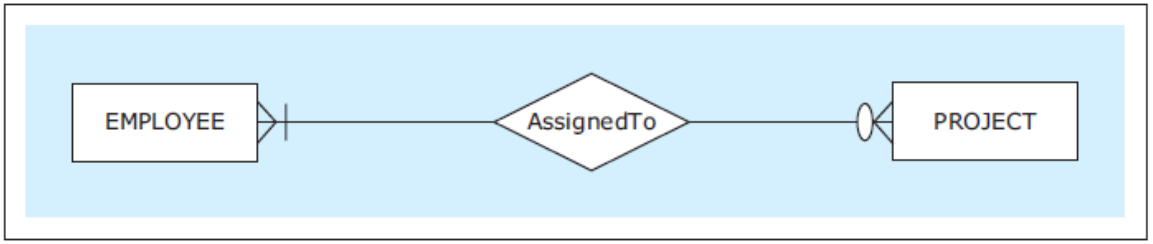


RELATIONSHIPS

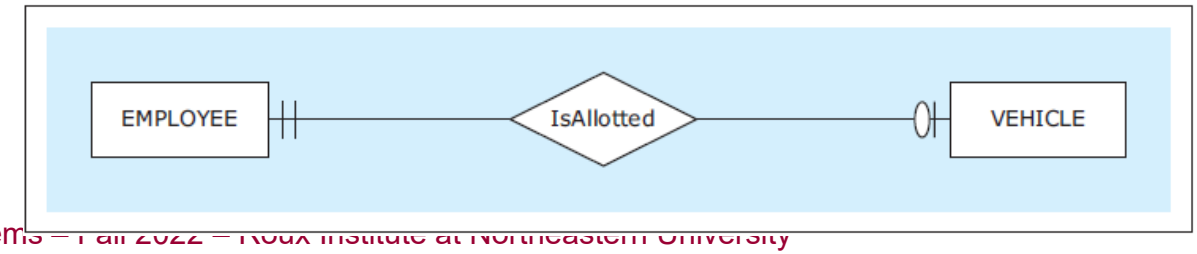
A 1:M Relationship



A M:N Relationship



A 1:1 Relationship

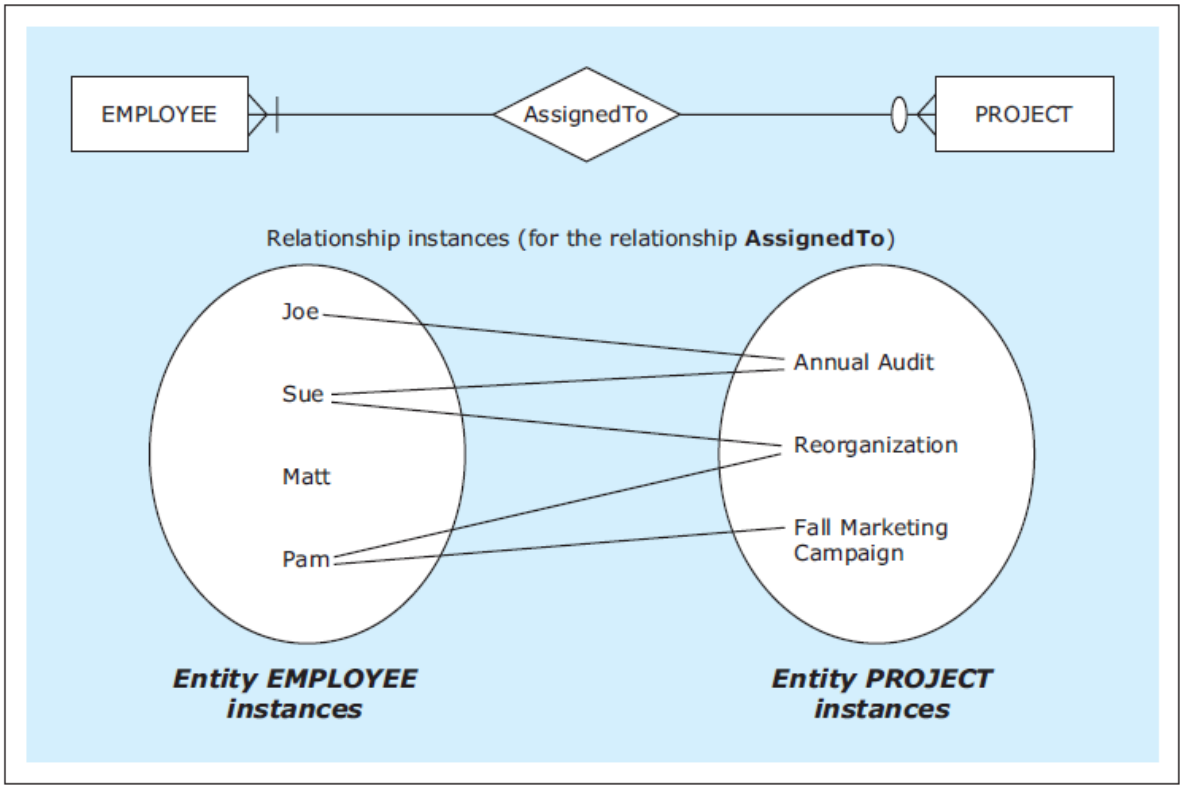


RELATIONSHIPS

- **Relationship instances** - occurrences of a relationship
  - Occur when an instance of one entity is related to an instance of another entity via a relationship
  - Relationship themselves are depicted in the ER diagrams while relationship instances are not
  - Relationship instances are eventually recorded in the database that is created based on the ER diagram

RELATIONSHIPS

A relationship and its instances

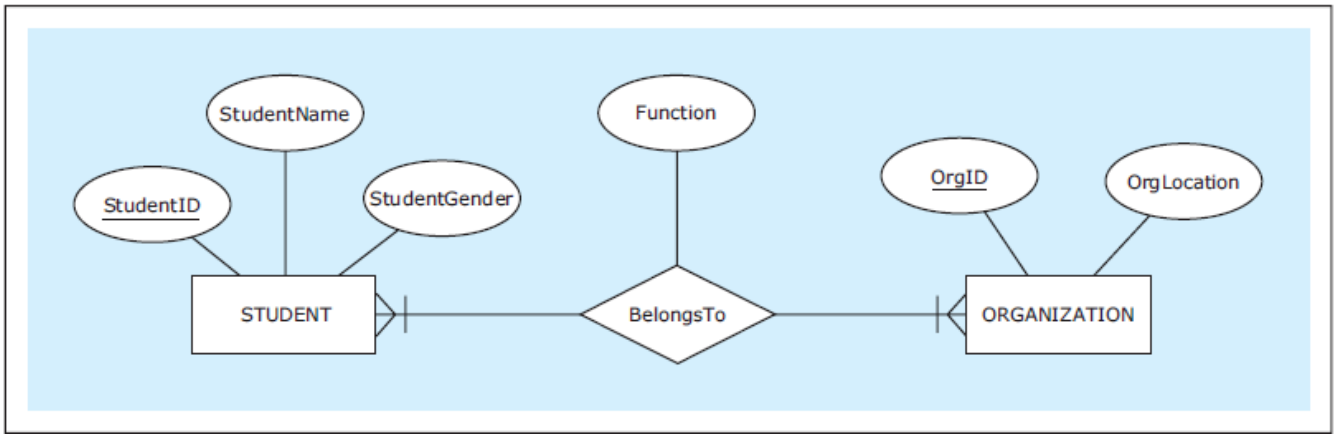


RELATIONSHIPS

- **Relationship attributes**
  - In some cases M:N relationships can actually have attributes of their own

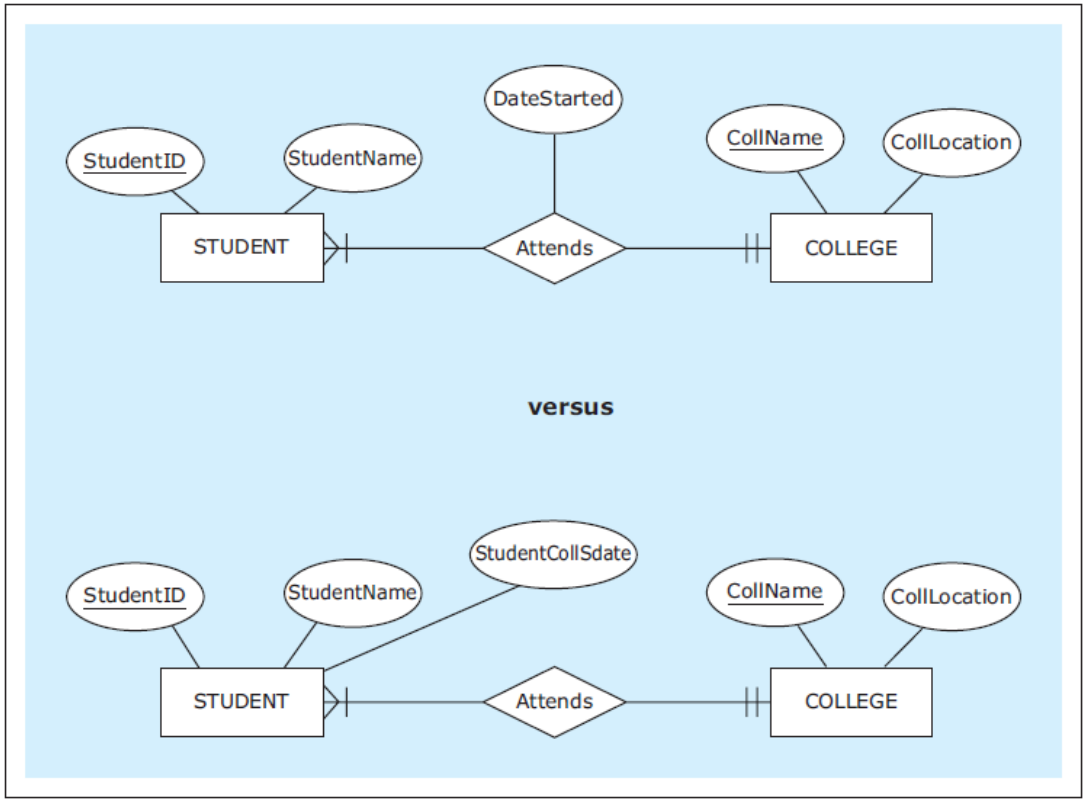
RELATIONSHIPS

A M:N relationship with an attribute

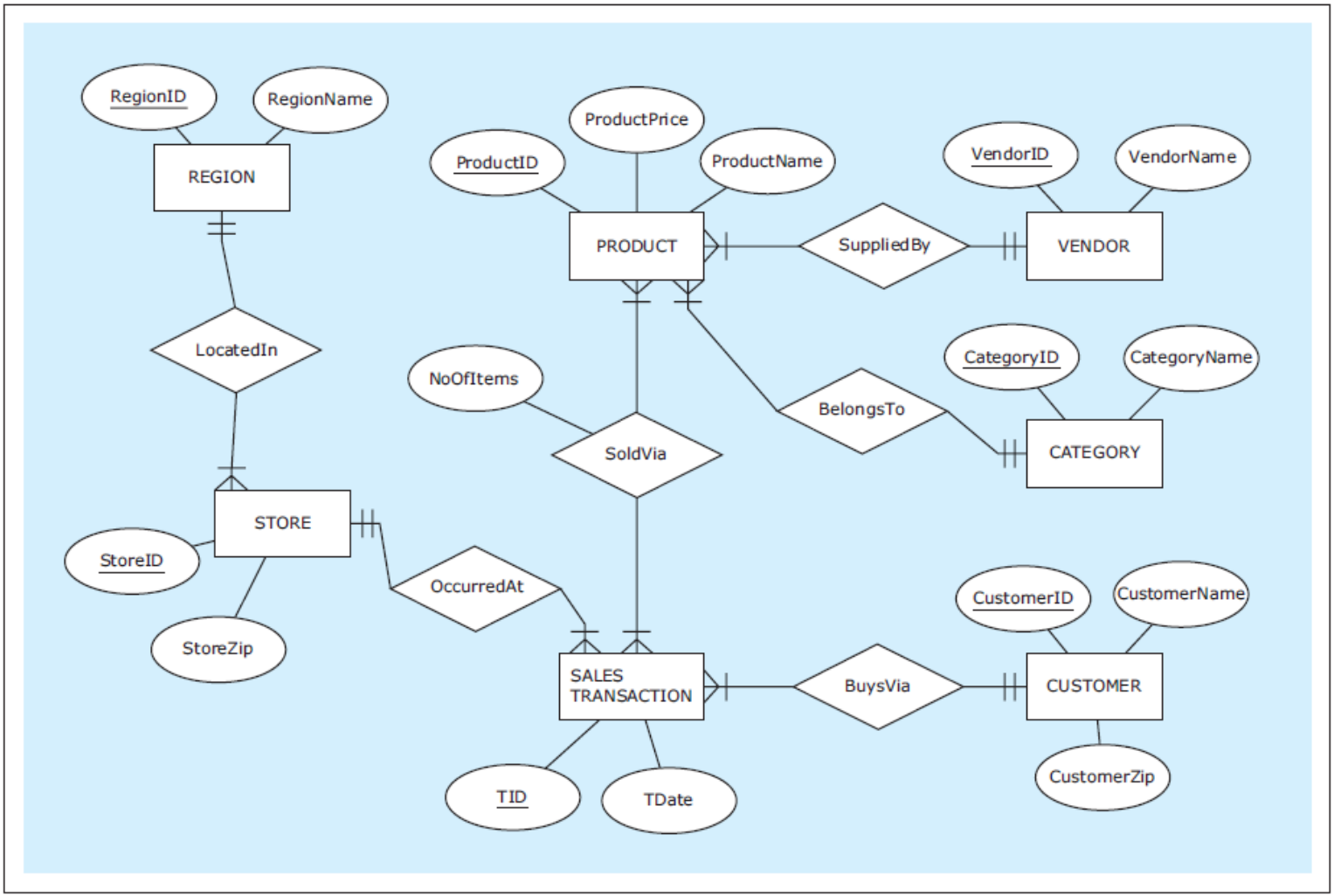


RELATIONSHIPS

A 1:M relationship with and without an attribute



ER diagram example: ZAGI Retail Company Sales Department Database

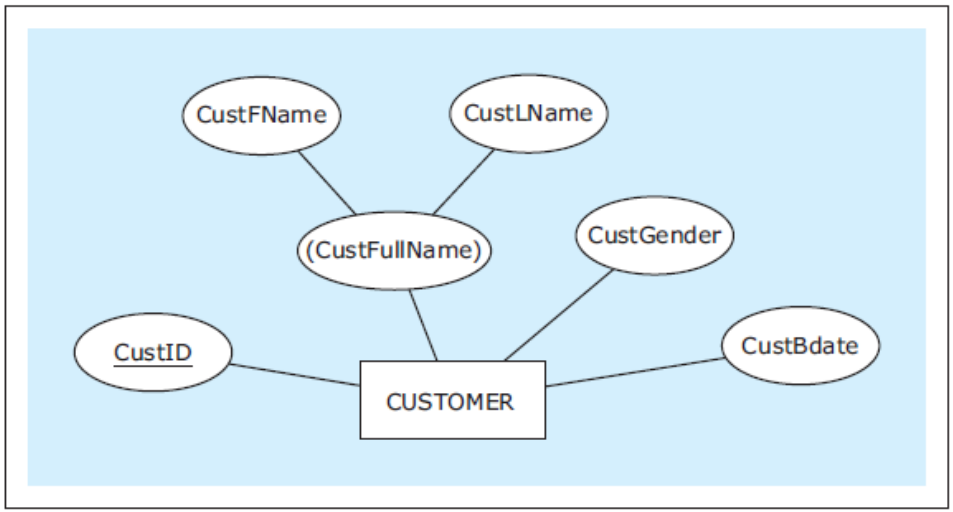


ATTRIBUTES

- **Composite attribute –** attribute that is composed of several attributes
  - Not an additional attribute of an entity
  - Its purpose is to indicate a situation in which a collection of attributes has an additional meaning, besides the individual meanings of each attribute

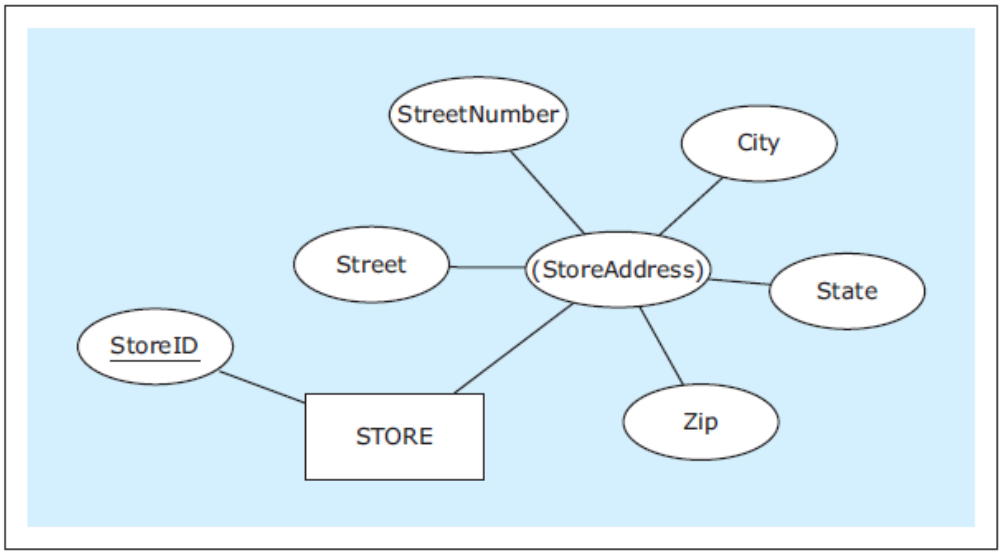
ATTRIBUTES

An entity with a composite attribute



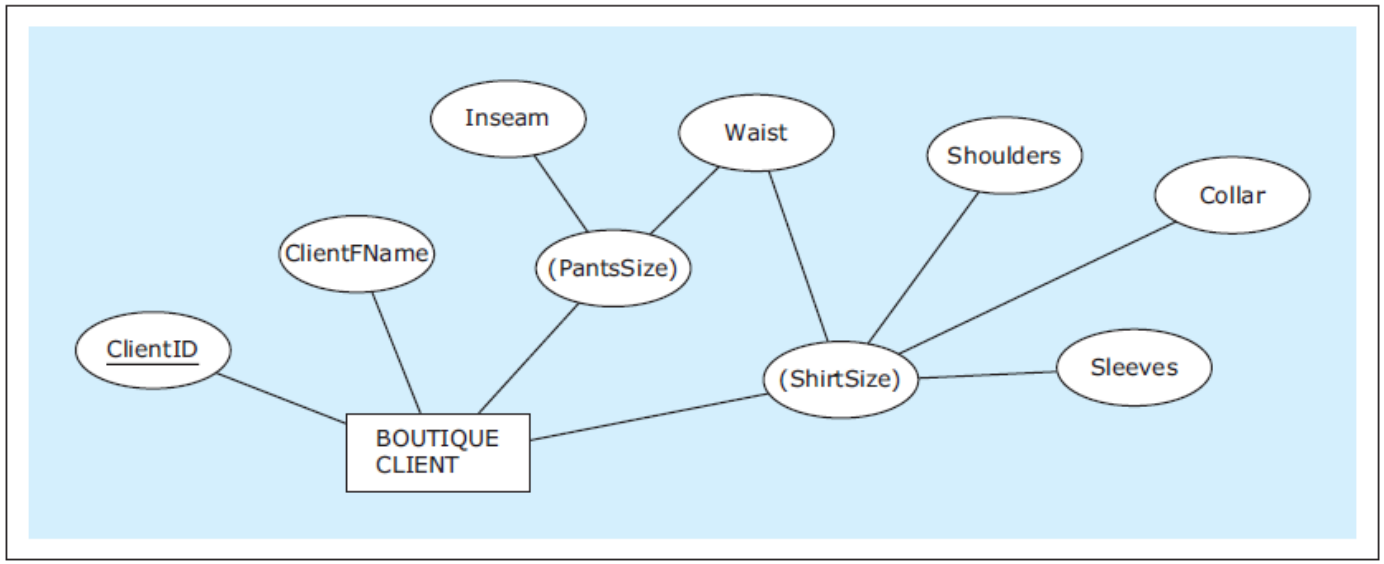
ATTRIBUTES

Another entity with a composite attribute



ATTRIBUTES

Composite attributes sharing components

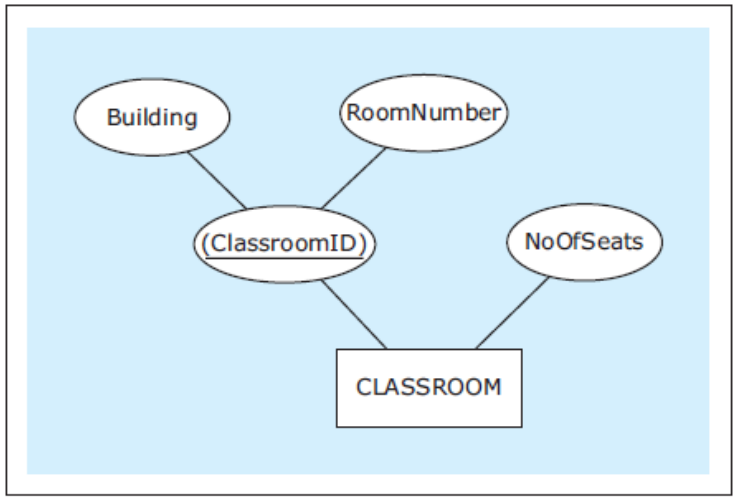


ATTRIBUTES

- **Composite unique attribute –** attribute that is composed of several attributes and whose value is different for each entity instance

ATTRIBUTES

An entity with a unique composite attribute

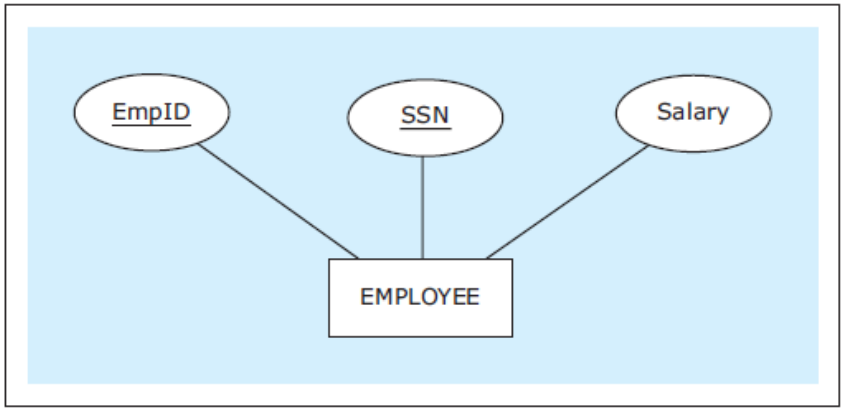


ATTRIBUTES

- **Multiple unique attributes (candidate keys) -** when an entity has more than one unique attribute each unique attribute is also called a candidate key

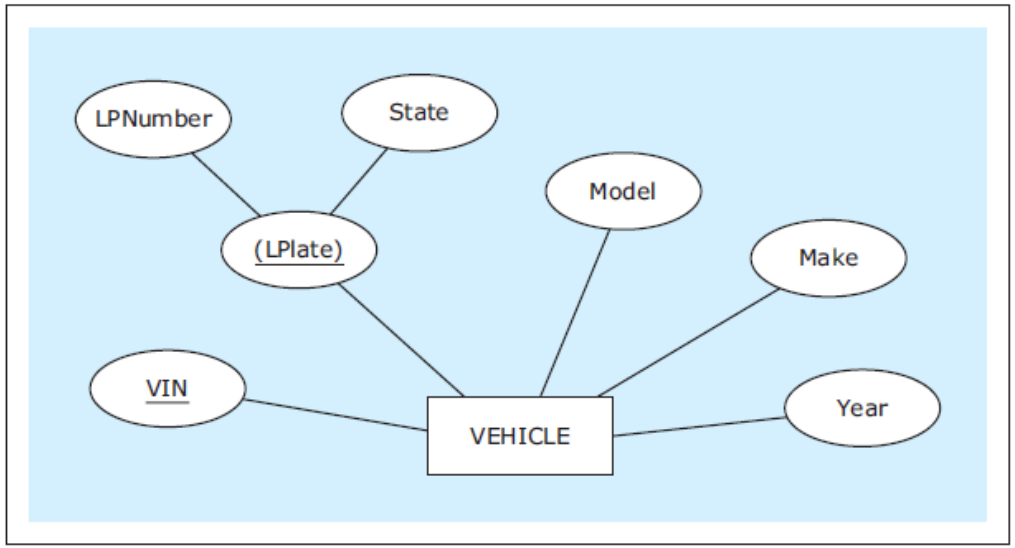
ATTRIBUTES

An entity with multiple unique attributes (candidate keys)



ATTRIBUTES

An entity with a regular and composite candidate key

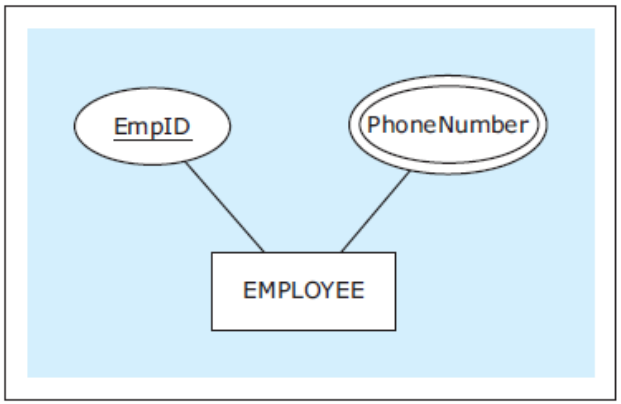


ATTRIBUTES

- **Multivalued attribute -** attribute for which instances of an entity can have multiple values for the same attribute

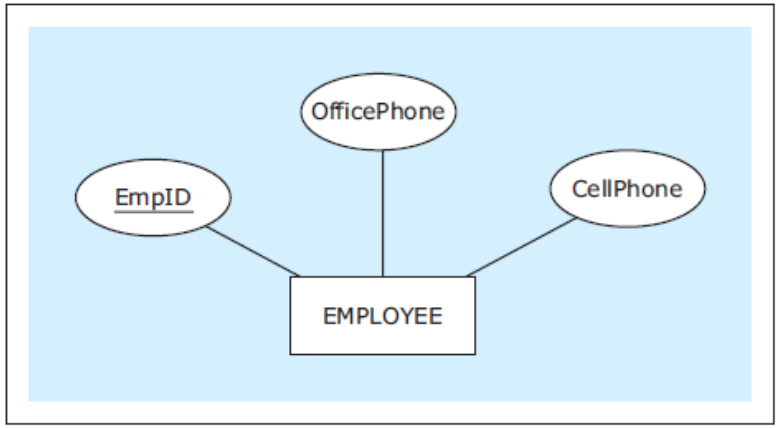
ATTRIBUTES

A multivalued attribute



ATTRIBUTES

A scenario that does not use multivalued attributes

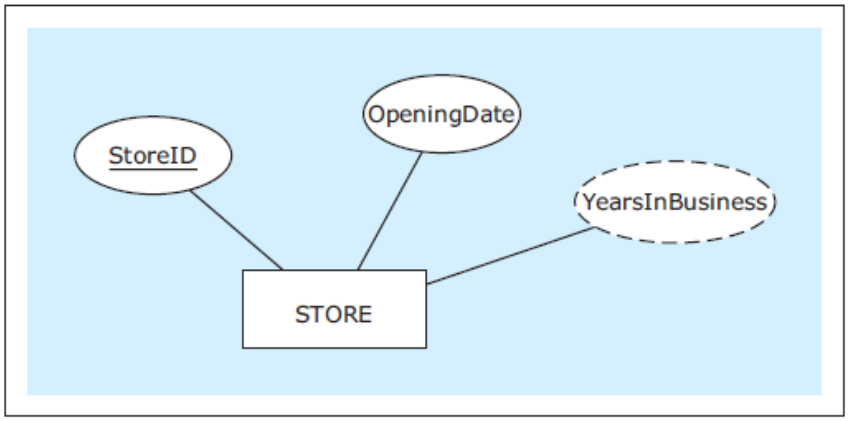


ATTRIBUTES

- **Derived attribute -** attribute whose values are calculated and not permanently stored in a database

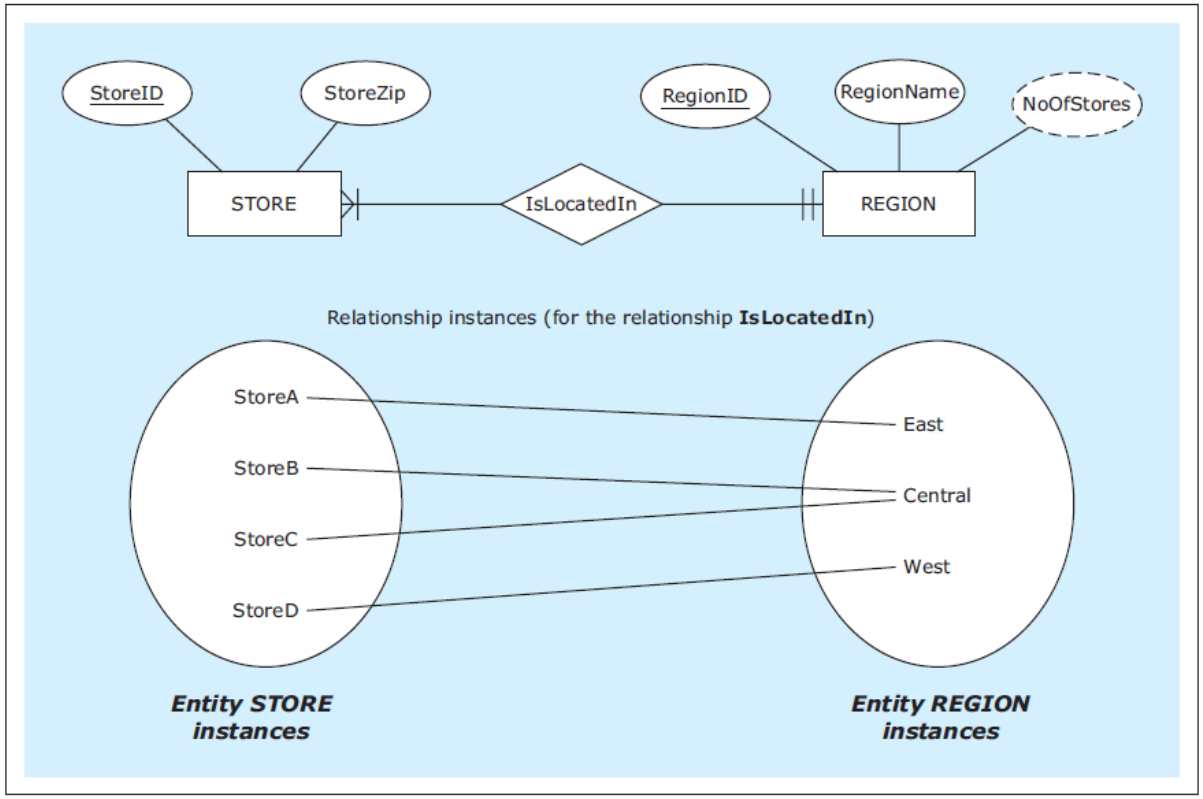
ATTRIBUTES

A derived attribute example



ATTRIBUTES

Another derived attribute example

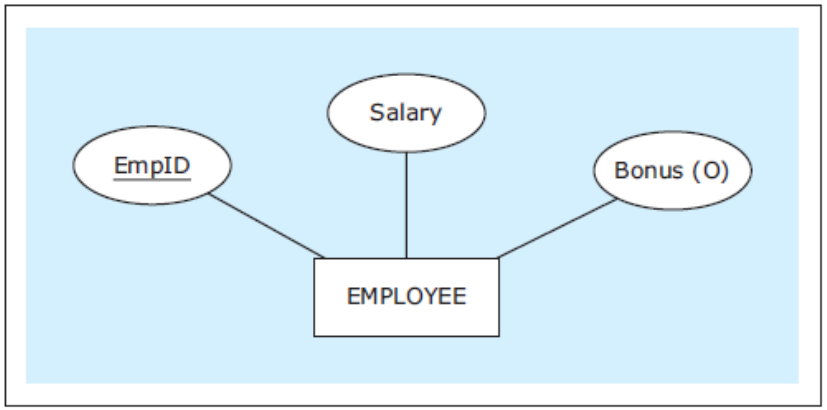


ATTRIBUTES

- **Optional attribute -** attribute that is allowed to not have a value

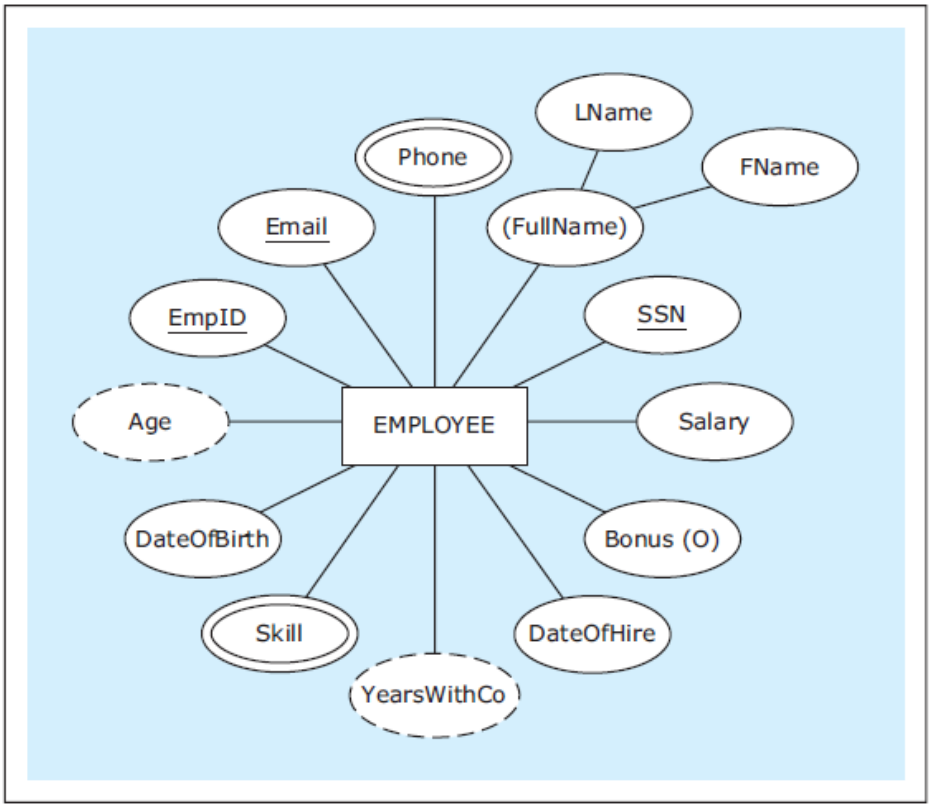
ATTRIBUTES

An optional attribute example



ATTRIBUTES

EXAMPLE: An entity with various types of attributes

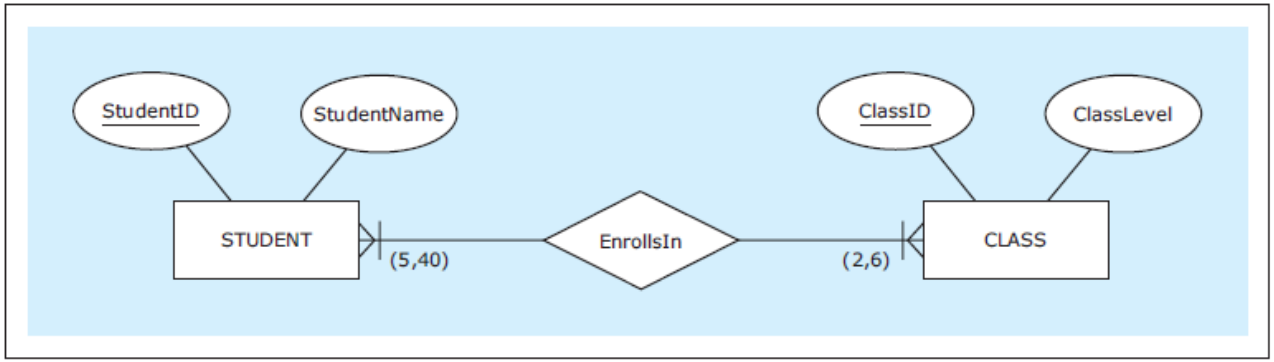


RELATIONSHIPS

- **Exact minimum and maximum cardinality in relationships**
  - In some cases the exact minimum and/or maximum cardinality in relationships is known in advance
  - Exact minimum/and or maximum cardinalities can be depicted in ER diagrams

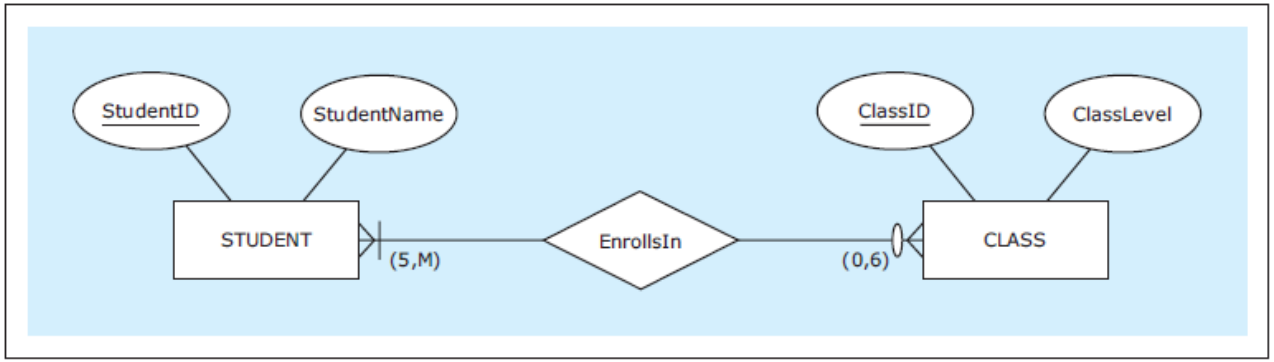
RELATIONSHIPS

A relationship with specific minimum and maximum cardinalities



RELATIONSHIPS

A relationship with a mixture of specific and non-specific cardinalities

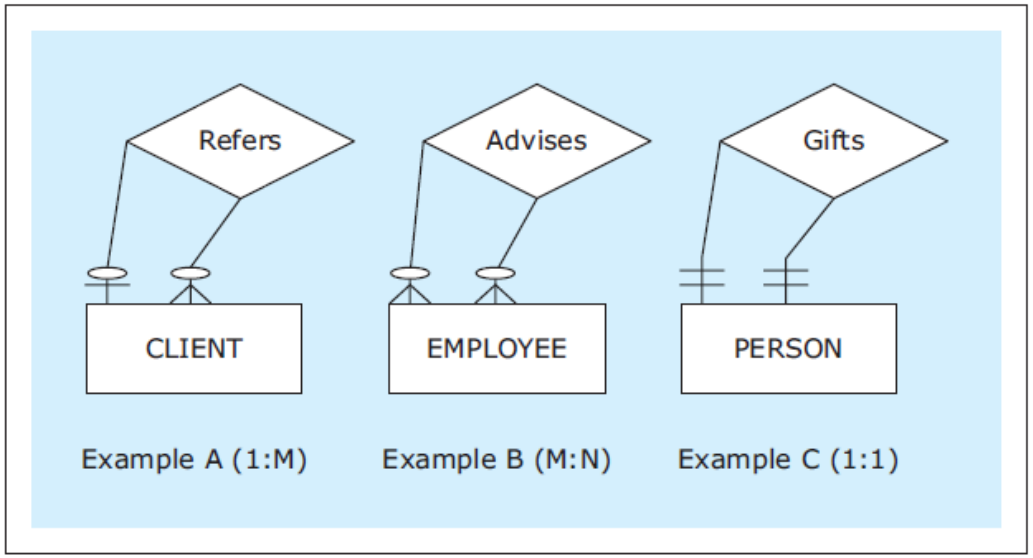


RELATIONSHIPS

- **Degree of a relationship -** reflects how many entities are involved in the relationship
- **Binary relationship** - relationship between two entities (*degree 2 relationship*)
- **Unary relationship (recursive relationship)** - occurs when an entity is involved in a relationship with itself (*degree 1 relationship*)

RELATIONSHIPS

Unary relationship examples

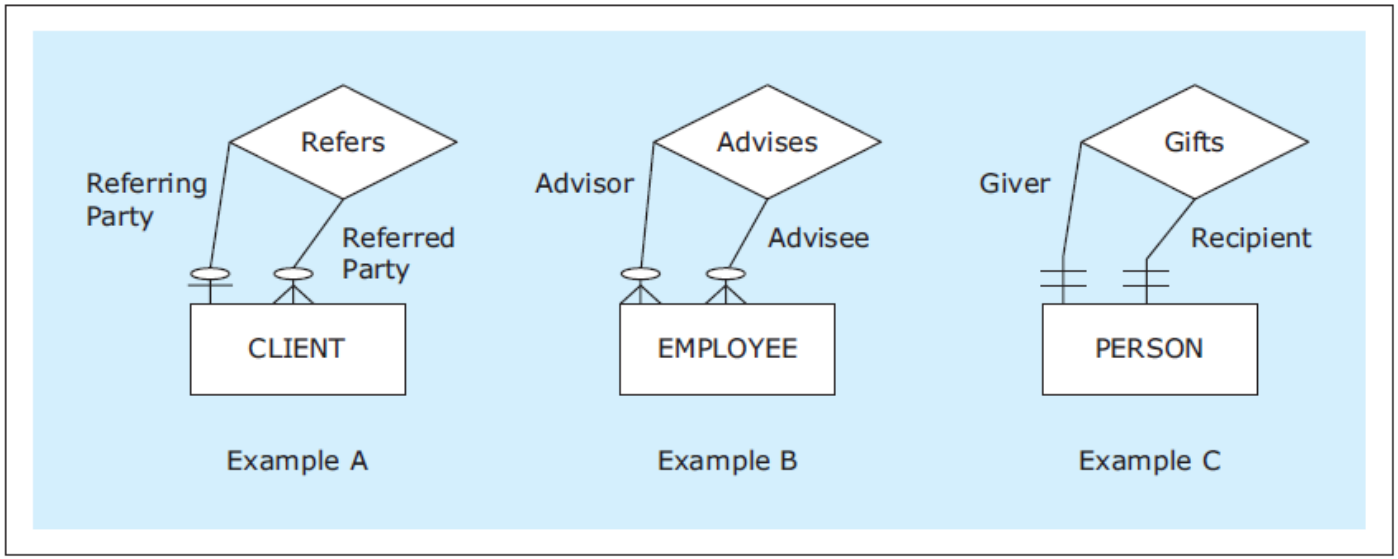


RELATIONSHIPS

- **Relationship roles -** additional syntax that can be used in ER diagrams at the discretion of a data modeler to clarify the role of each entity in a relationship

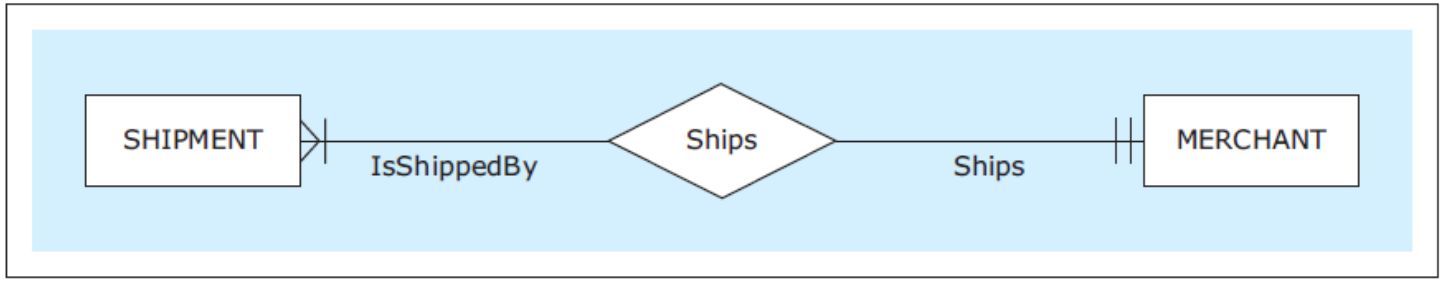
RELATIONSHIPS

Unary relationships with role names



RELATIONSHIPS

A binary relationship with role names

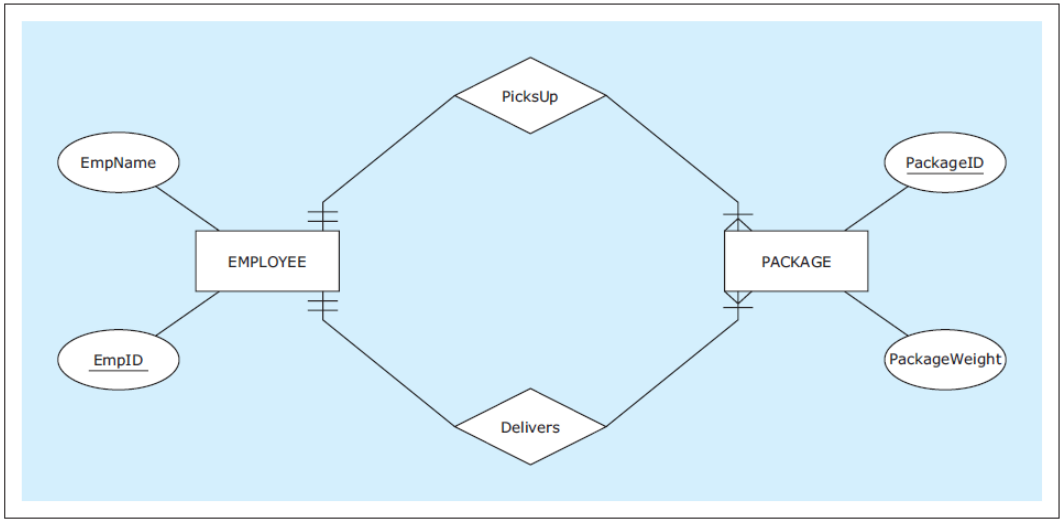


RELATIONSHIPS

- **Multiple relationships between same entities**
  - Same entities in an ER diagram can be related via more than one relationship

RELATIONSHIPS

Multiple relationships between the same entities



WEAK ENTITY

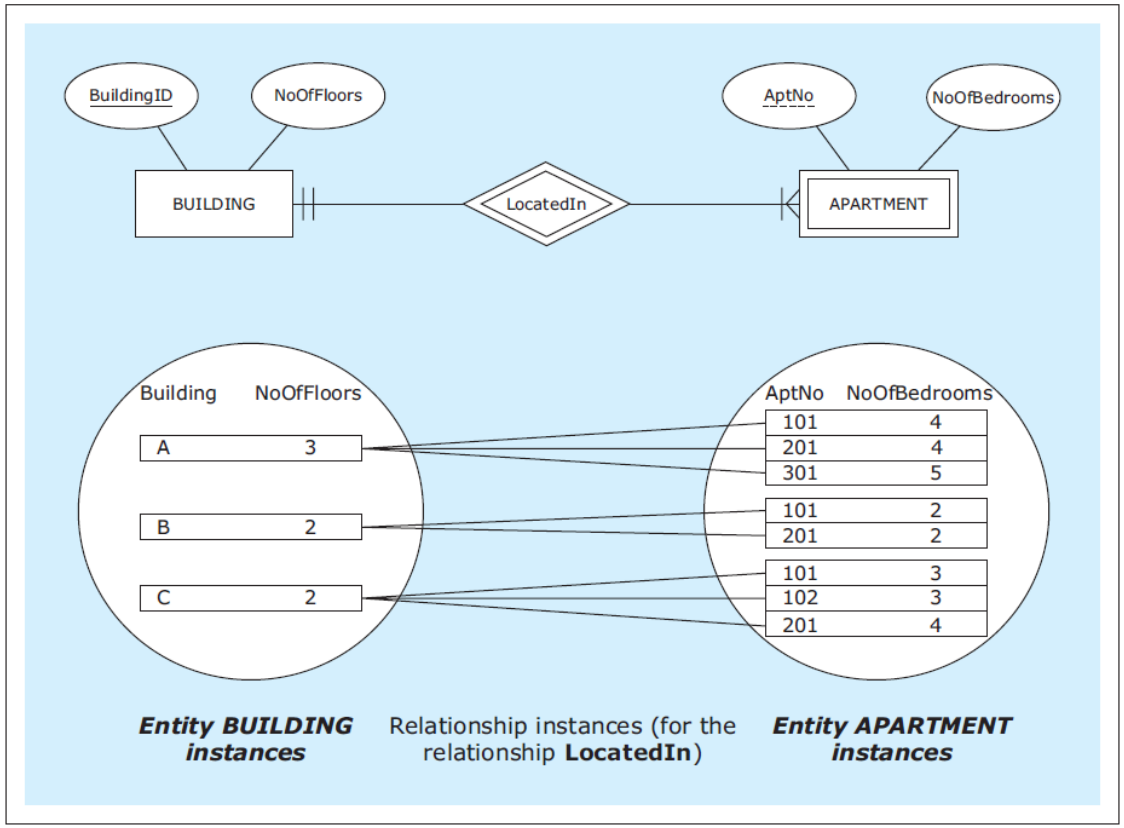
- **Weak entity -** ER diagram construct depicting an entity that does not have a unique attribute of its own
- **Owner entity** - entity whose unique attribute provides a mechanism for identifying instances of a weak entity
- **Identifying relationship** - relationship between a weak entity and its owner entity in which each instance of a weak entity is associated with exactly one instance of an owner entity
  - Each weak entity must be associated with its owner entity via an identifying relationship
  - Unique attribute from the owner entity uniquely identifies every instance of the weak entity via an identifying relationship

WEAK ENTITY

- **Partial key** - attribute of a weak entity that combined with the unique attribute of the owner entity uniquely identifies the weak entity's instances
  - Combination of the partial key and the unique attribute from the owner entity uniquely identifies every instance of the weak entity

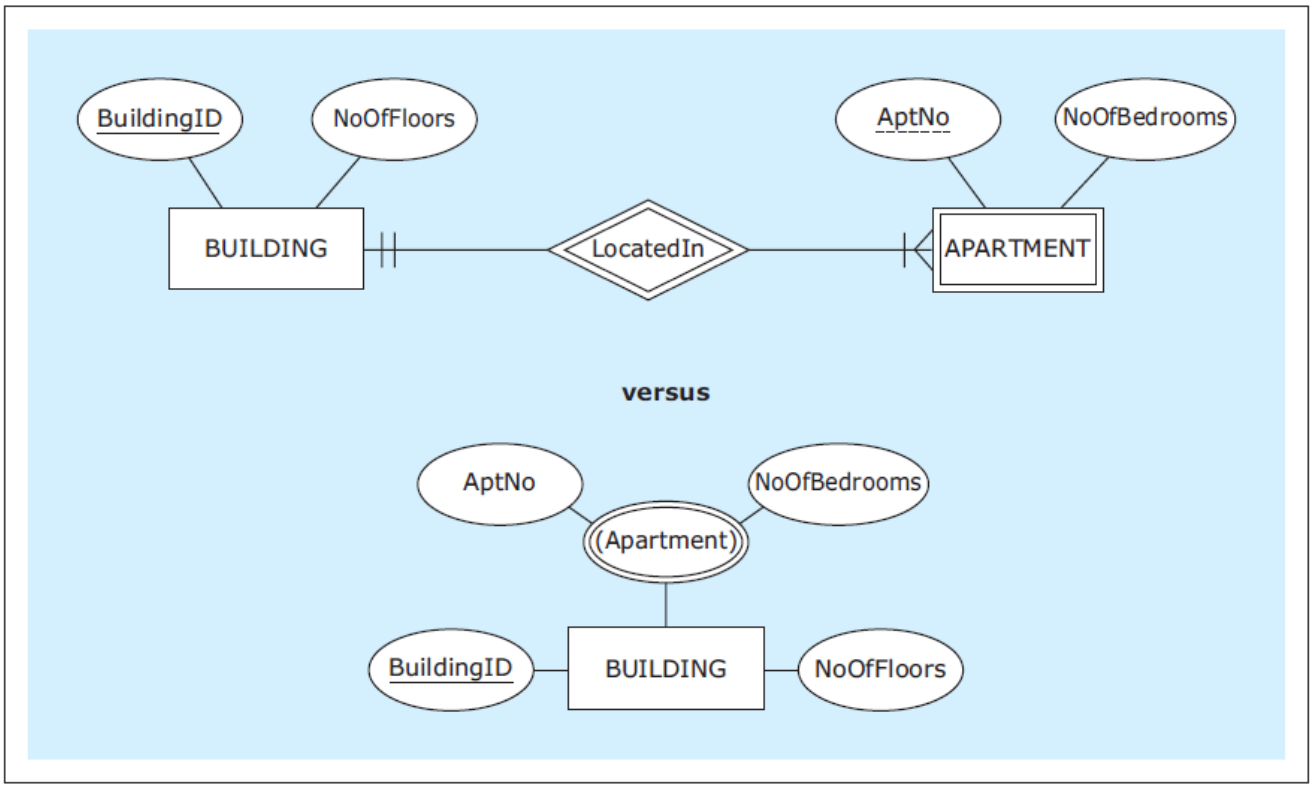
WEAK ENTITY

A weak entity example with entity instances



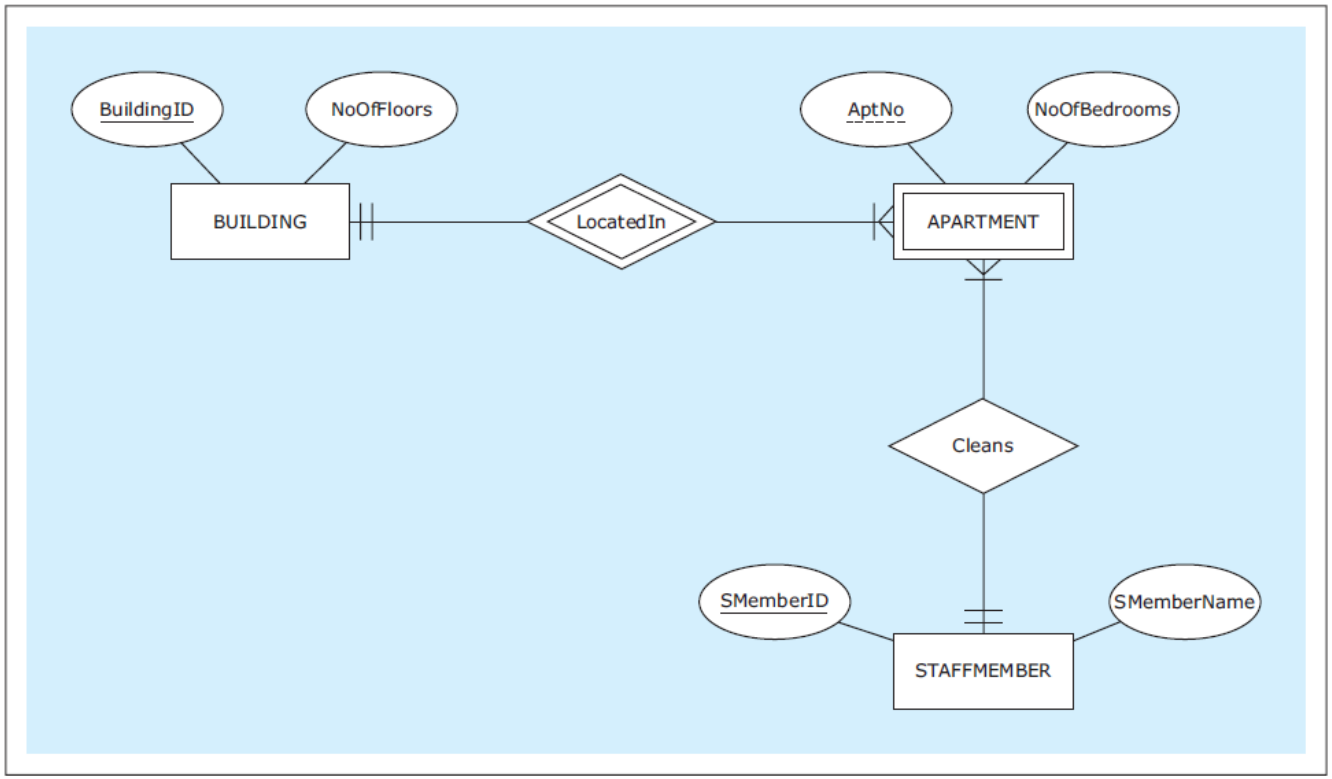
WEAK ENTITY

A weak entity versus a multivalued composite attribute



WEAK ENTITY

A weak entity with an identifying and regular relationship

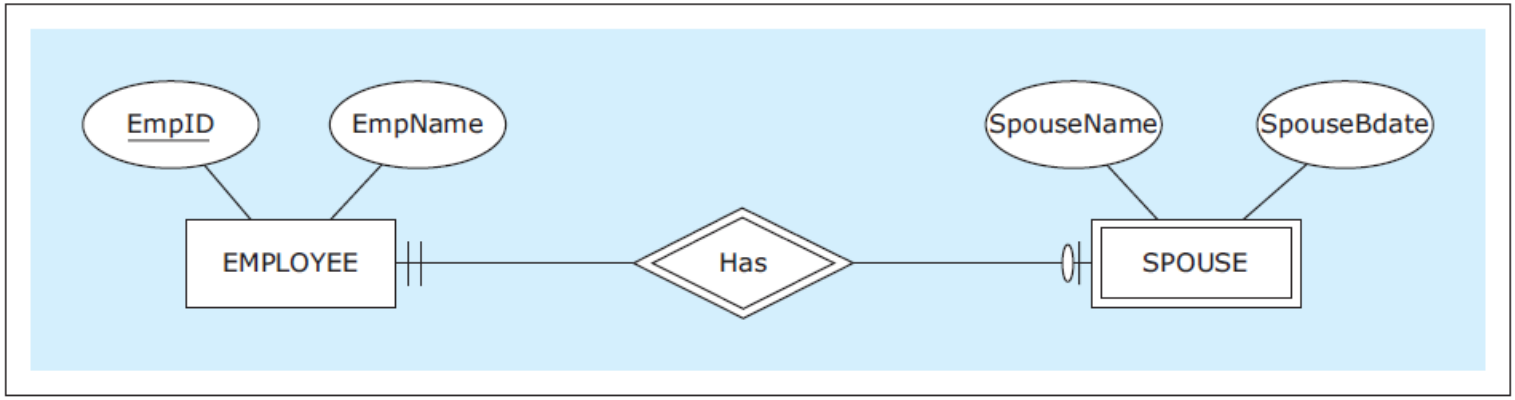


WEAK ENTITY

- Identifying relationship is either 1:M or 1:1 relationship
  - In case of 1:M identifying relationship, a weak entity must have a partial key attribute
  - In case of 1:1 identifying relationship, a weak entity doesn't need to have a partial key attribute

WEAK ENTITY

A weak entity with a 1:1 identifying relationship



NAMING CONVENTIONS FOR ER DIAGRAMS

- Entities and attributes
  - Use singular (rather than plural) nouns
- Relationships
  - Use verbs or verb phrases, rather than nouns

NAMING CONVENTIONS FOR ER DIAGRAMS

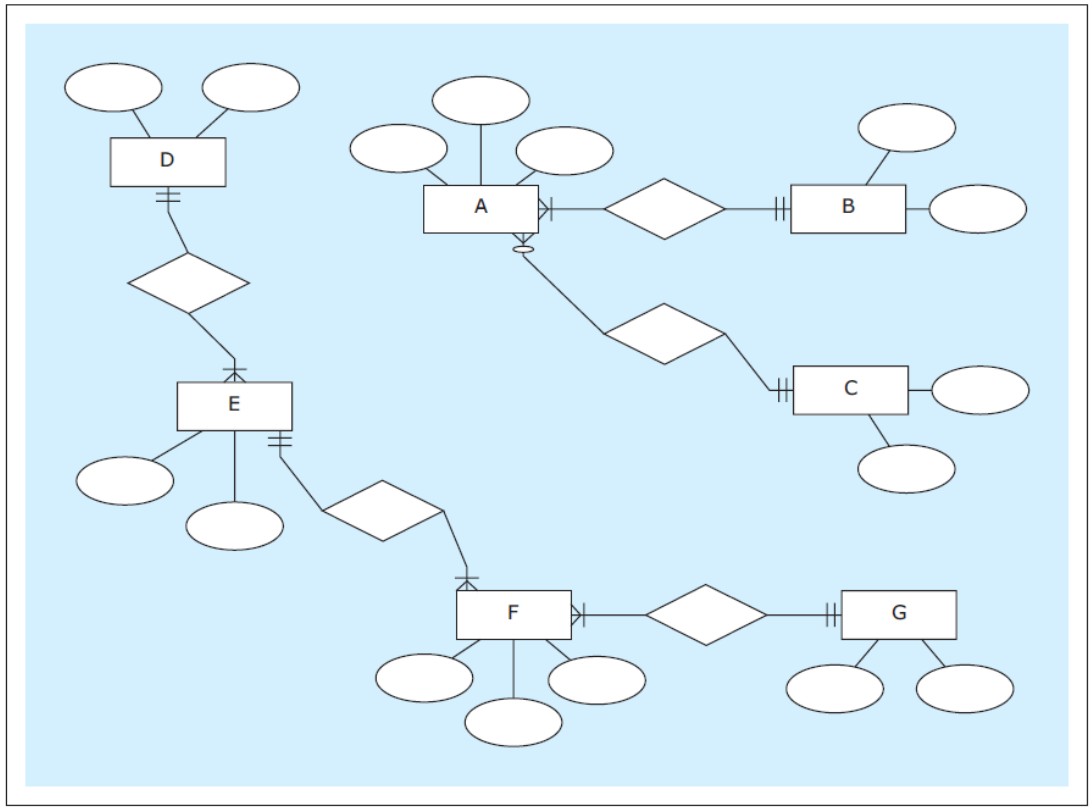
- Names should be as brief as possible, without being too condensed as to obscure the meaning of the construct
- If possible, give all attributes in the entire ER diagram different names

MULTIPLE ER DIAGRAMS

- When depicting multiple ER diagrams, each diagram should be visualized separately
- Instead of multiple ER diagrams in one schema a better choice is to present each ER diagram separately

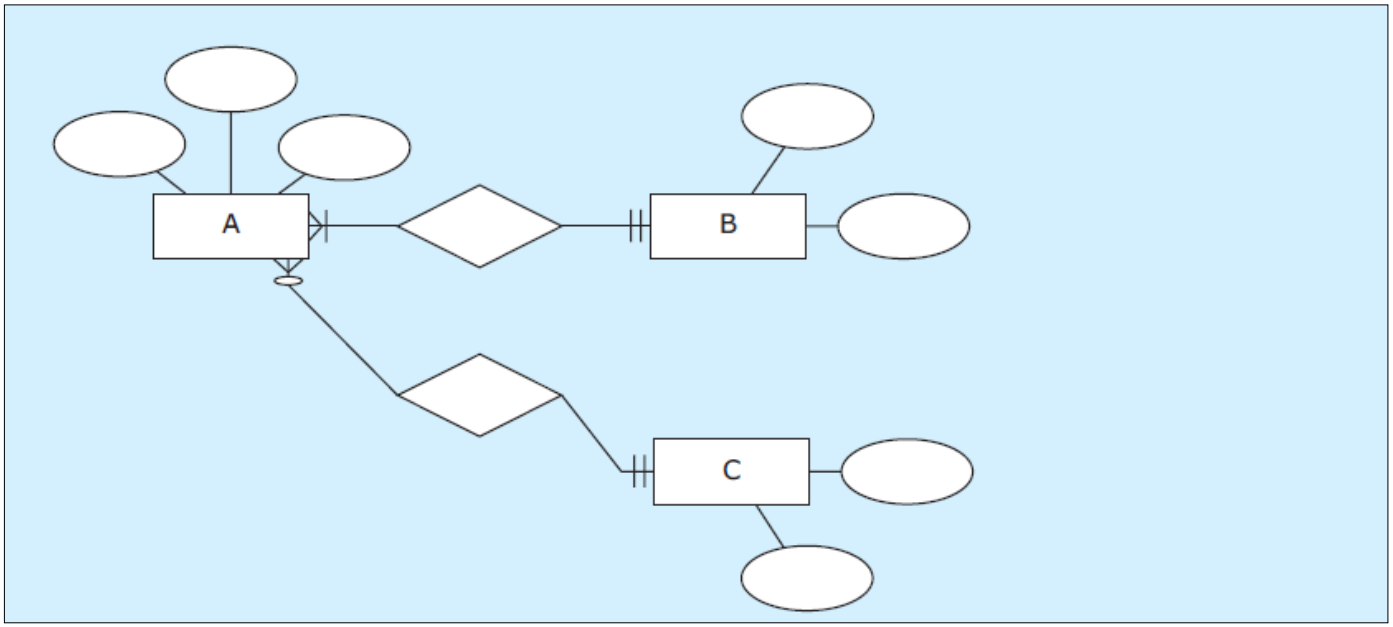
MULTIPLE ER DIAGRAMS

A schema with two separate ER diagrams (potentially misleading)



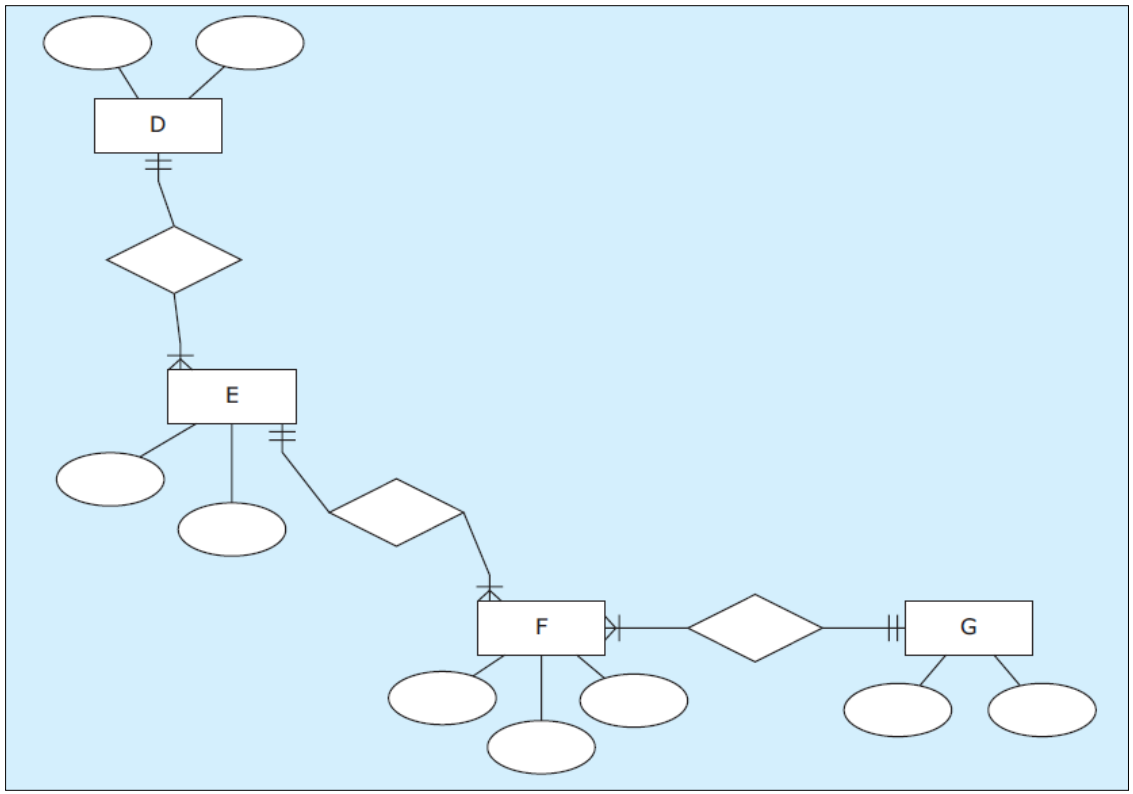
MULTIPLE ER DIAGRAMS

Separate ER diagrams in separate schemas

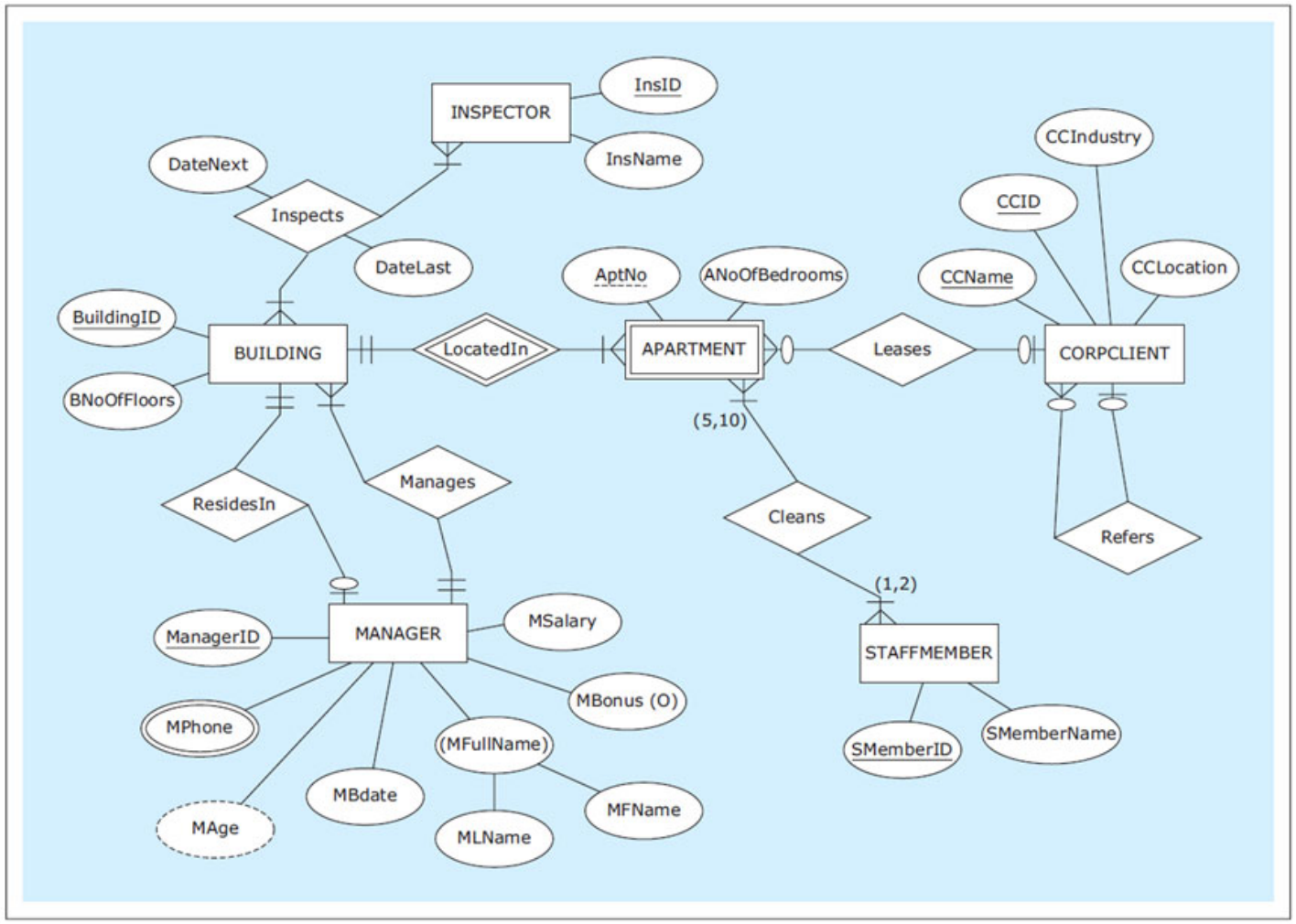


MULTIPLE ER DIAGRAMS

Separate ER diagrams in separate schemas



Another ER diagram example: HAFH Realty Company Property Management Database



DATABASE REQUIREMENTS AND ER MODEL USAGE

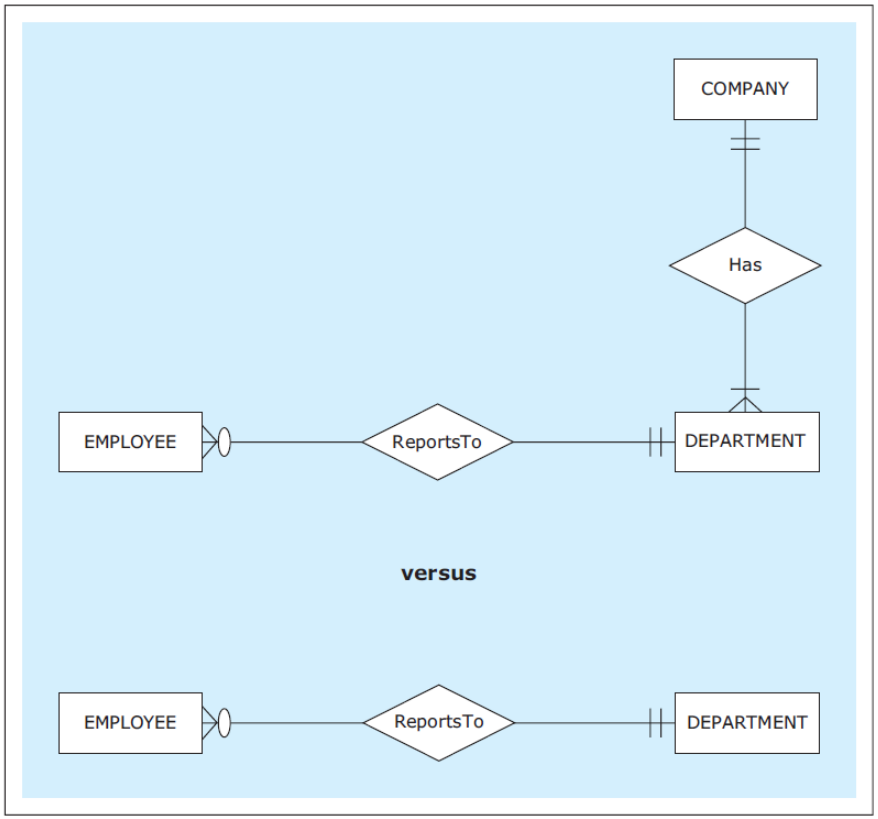
- ER modeling provides a straightforward technique for collecting, structuring, and visualizing requirements
- An understanding of ER modeling is crucial, not just for creating ER models based on the requirements, but also during the requirements collection process itself
- It helps keep the focus on asking or seeking answers to the right questions in order to establish the relevant facts about entities, attributes, and relationships

DATABASE REQUIREMENTS AND ER MODEL USAGE

- One of the common mistakes that beginners make when engaging in ER modeling for the first time is not recognizing the difference between an entity and the ER diagram itself

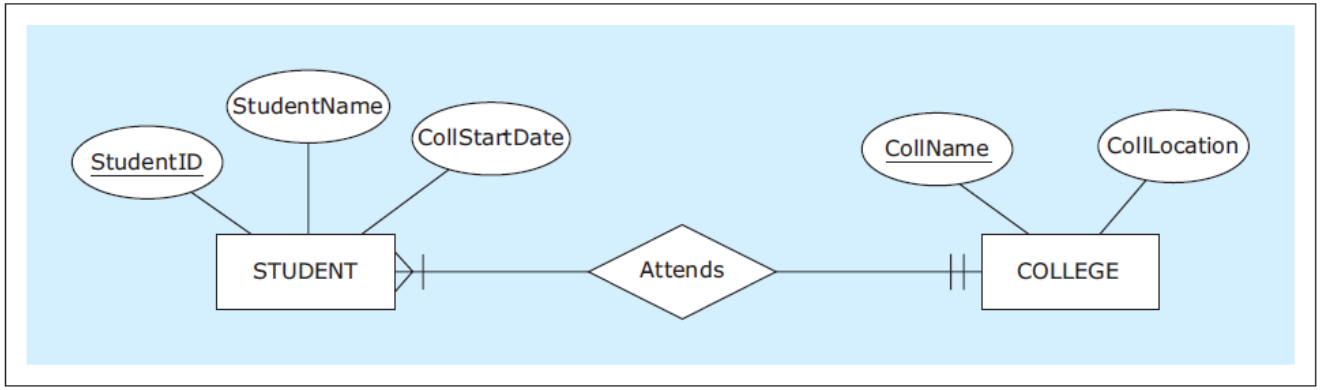
DATABASE REQUIREMENTS AND ER MODEL USAGE

An ER diagram incorrectly and correctly interpreting requirements



DATABASE REQUIREMENTS AND ER MODEL USAGE

An ER diagram incorrectly and correctly interpreting requirements



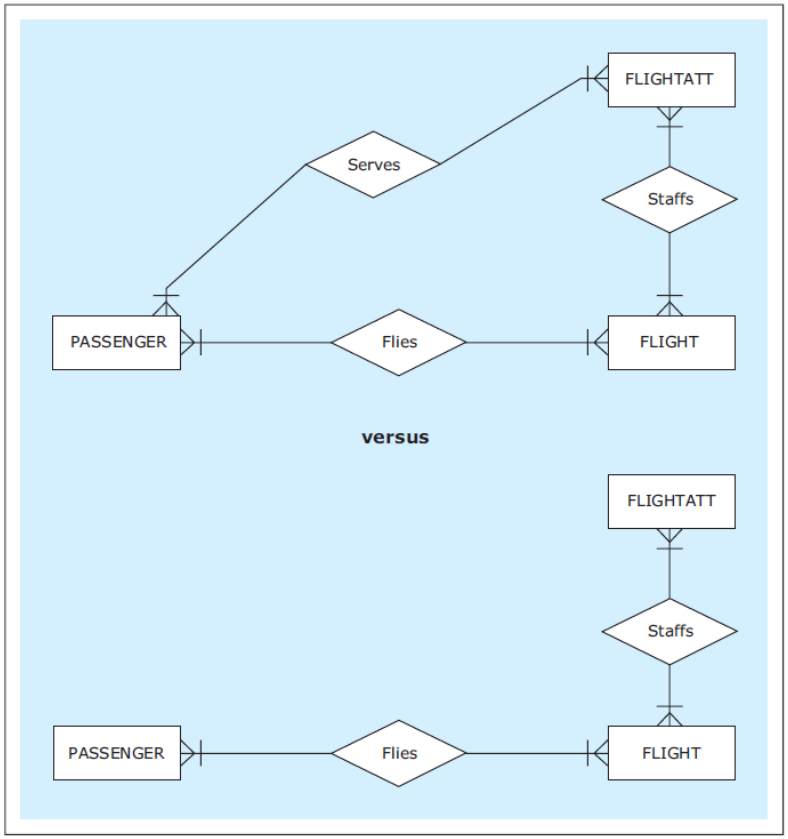
DATABASE REQUIREMENTS AND ER MODEL USAGE

- Another common database requirements collection and ER modeling mistake made by novices is not distinguishing between:

  Modeling of the data that is wanted and can be kept track of
  versus
  Modeling of everything that takes place in an organization

DATABASE REQUIREMENTS AND ER MODEL USAGE

An ER diagram based on unfeasible and proper requirements

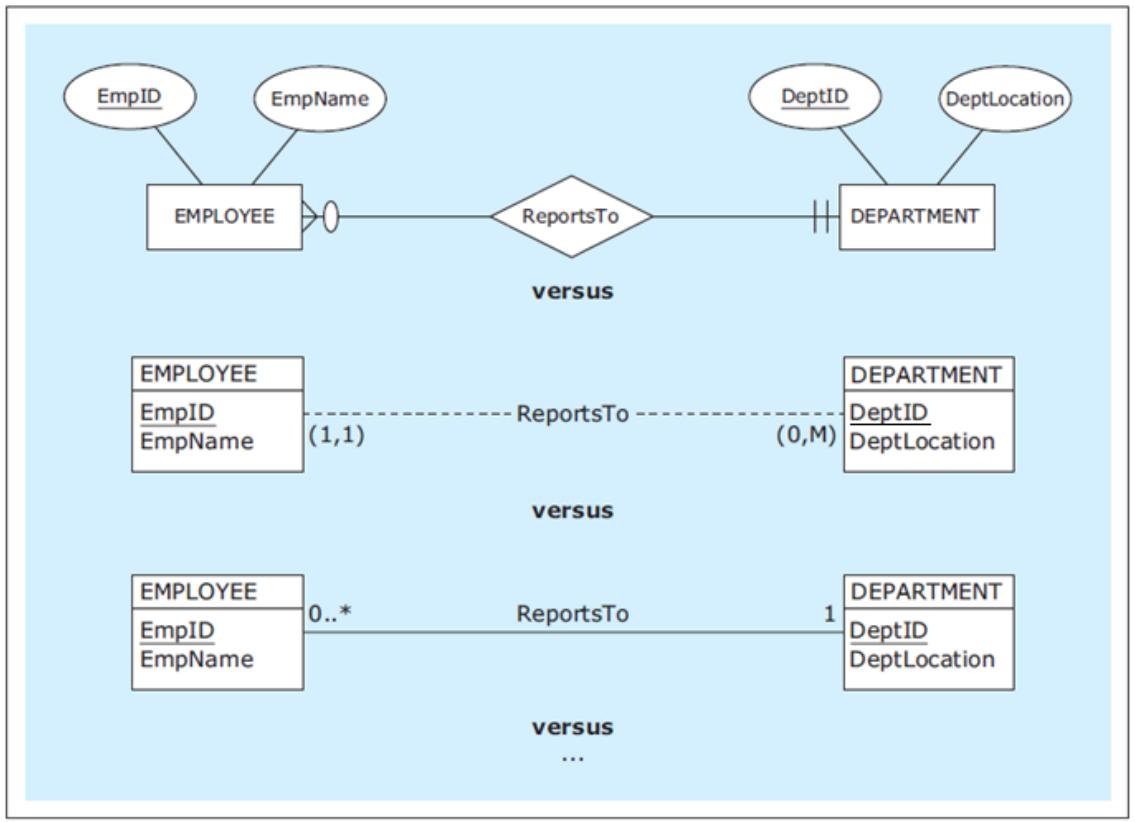


VARIOUS ER NOTATIONS

- There is no universally adopted ER notation to which all database projects conform
- Instead, there is a variety of available ER notations in use
- However, if a designer is familiar with one ER notation, other alternative ER notations are easy to understand and use

VARIOUS ER NOTATIONS

Examples of various ER notations



ERDPlus Software

- https://erdplus.com/

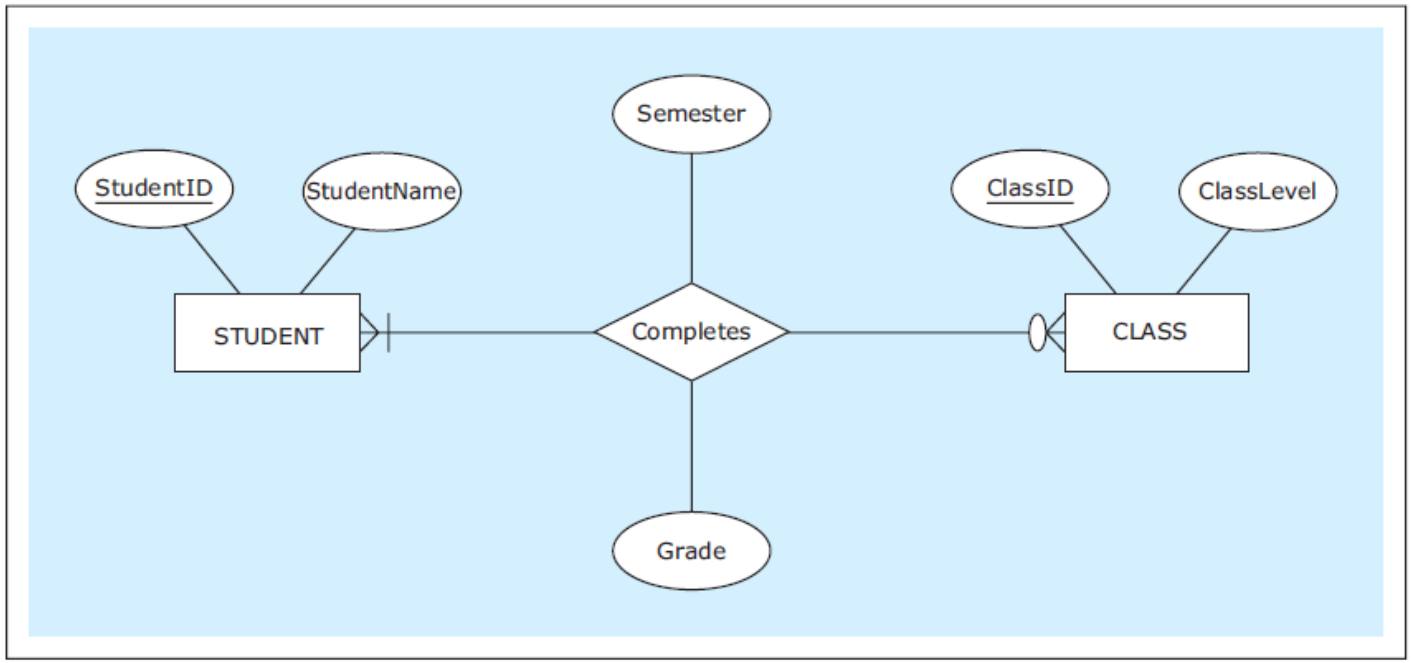
- In-Class Activity
  - Select a domain
  - Create a basic ERD of Entities, Attributes, Relationships

M:N RELATIONSHIPS WITH MULTIPLE INSTANCES BETWEEN THE SAME ENTITIES

- In some cases, M:N relationships can have multiple occurrences between the same instances of involved entities
  - The following examples illustrates such cases

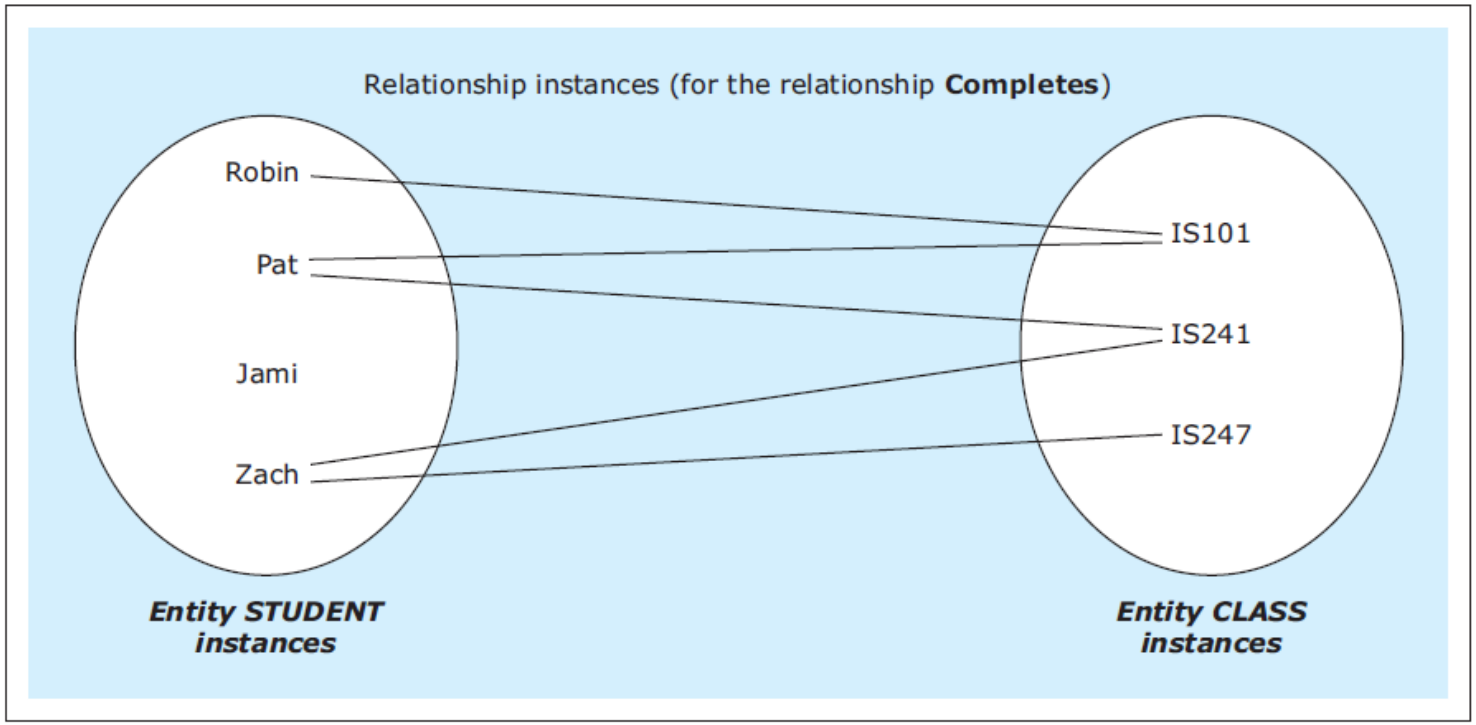
M:N RELATIONSHIPS WITH MULTIPLE INSTANCES BETWEEN THE SAME ENTITIES

An ER diagram for an M:N relationship depicting students completing classes



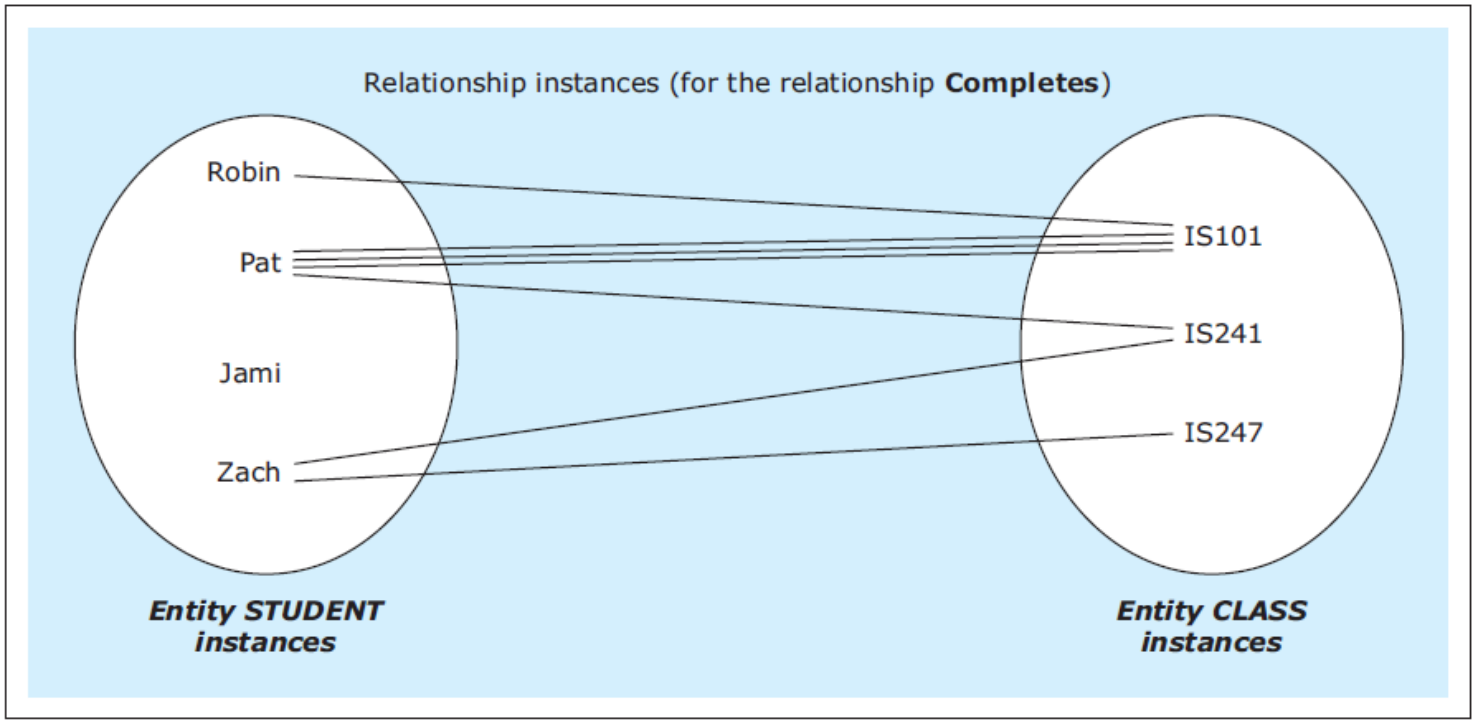
M:N RELATIONSHIPS WITH MULTIPLE INSTANCES BETWEEN THE SAME ENTITIES

Instances of the M:N relationship Completes



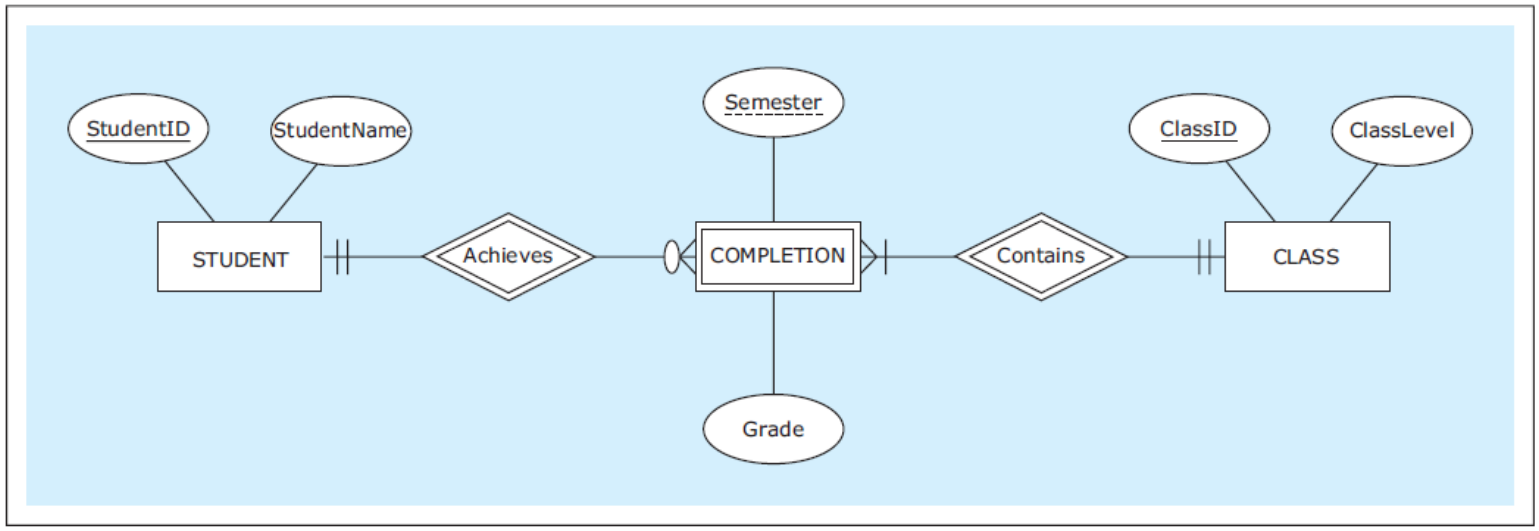
M:N RELATIONSHIPS WITH MULTIPLE INSTANCES BETWEEN THE SAME ENTITIES

Instances of the M:N relationship Completes with an additional requirement



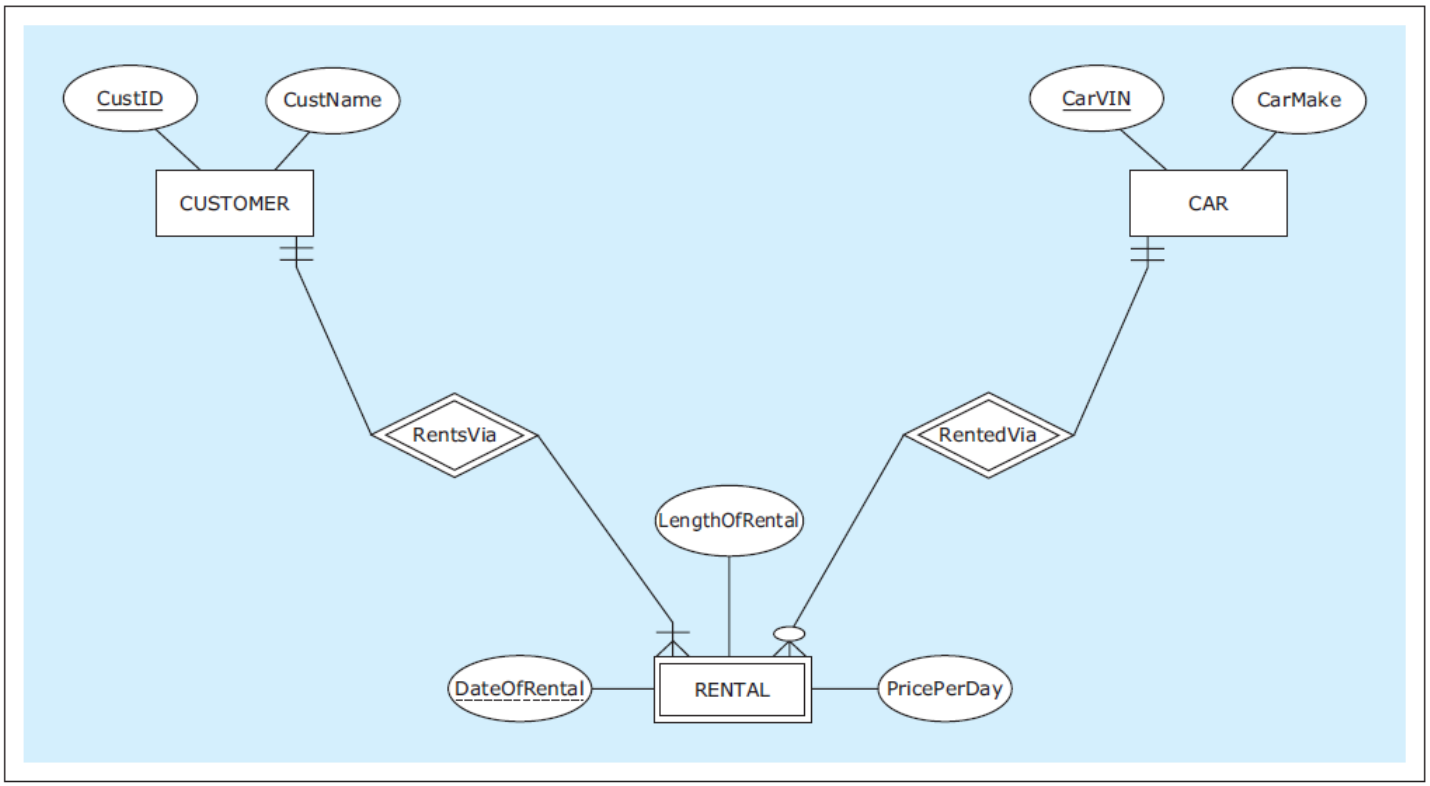
M:N RELATIONSHIPS WITH MULTIPLE INSTANCES BETWEEN THE SAME ENTITIES

An ER diagram for an M:N relationship represented as a weak entity



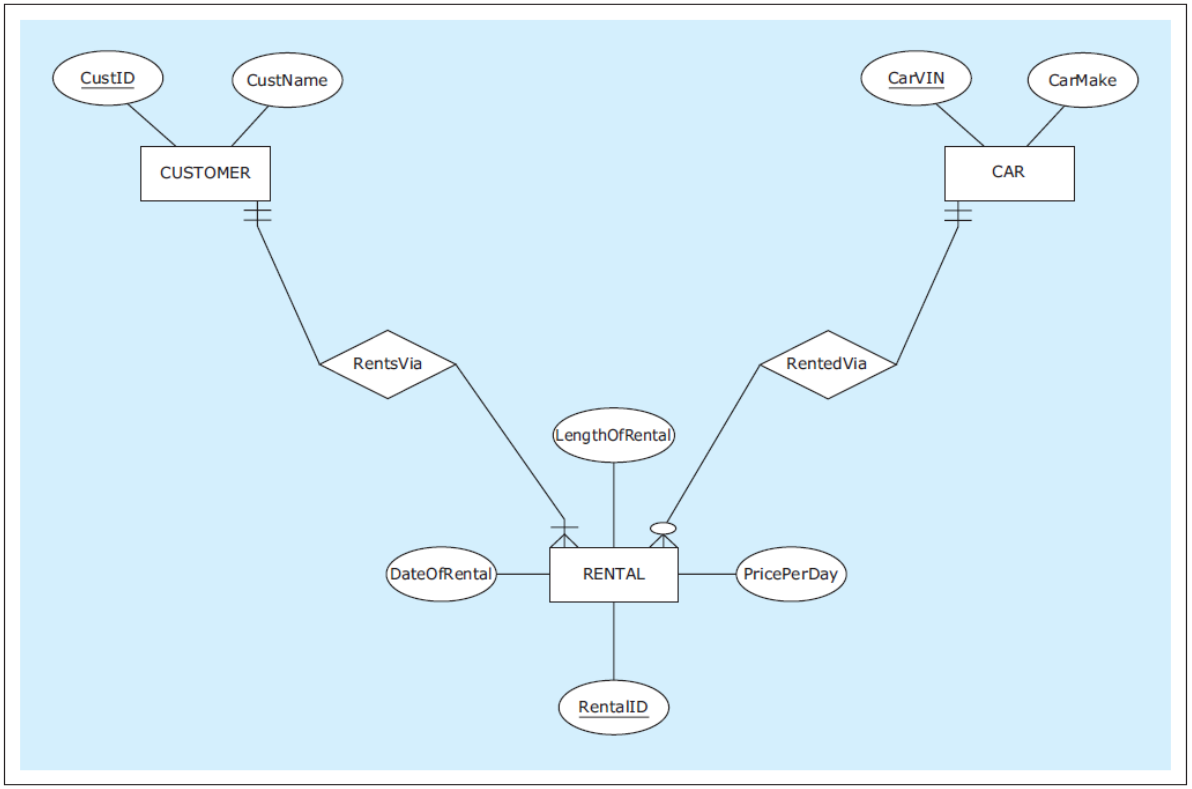
M:N RELATIONSHIPS WITH MULTIPLE INSTANCES BETWEEN THE SAME ENTITIES

Another M:N relationship represented as a weak entity



M:N RELATIONSHIPS WITH MULTIPLE INSTANCES BETWEEN THE SAME ENTITIES

A regular entity, instead of an M:N relationship represented as a weak entity

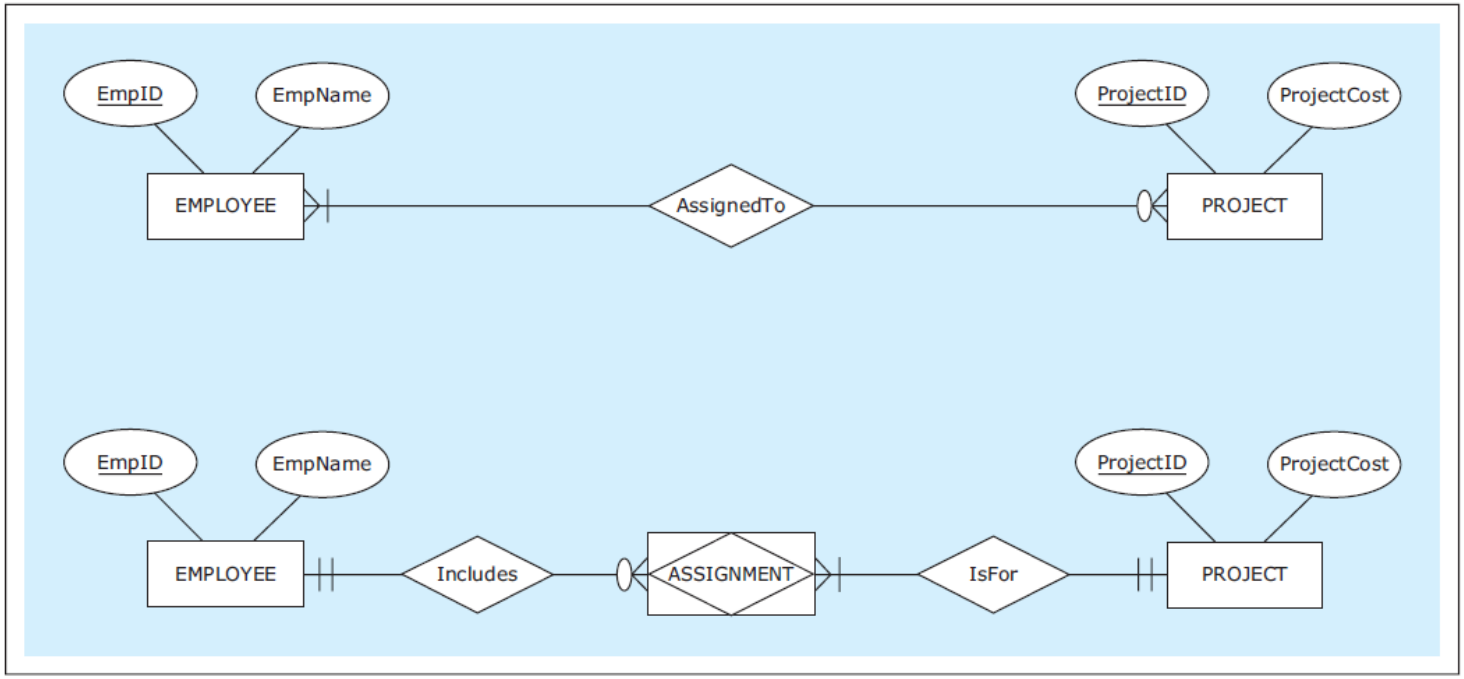


ASSOCIATIVE ENTITY

- **Associative entity** - construct used as an alternative way of depicting M:N relationships
  - Associative entities do not have unique or partially unique attributes, and often do not have any attributes at all

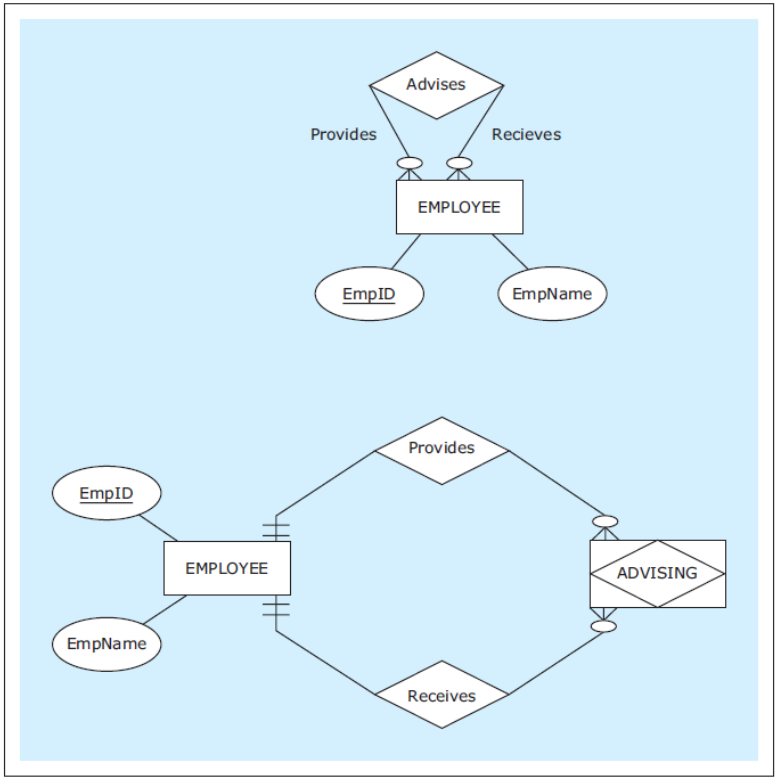
ASSOCIATIVE ENTITY

An identical relationship represented as a M:N relationship and as an associative entity



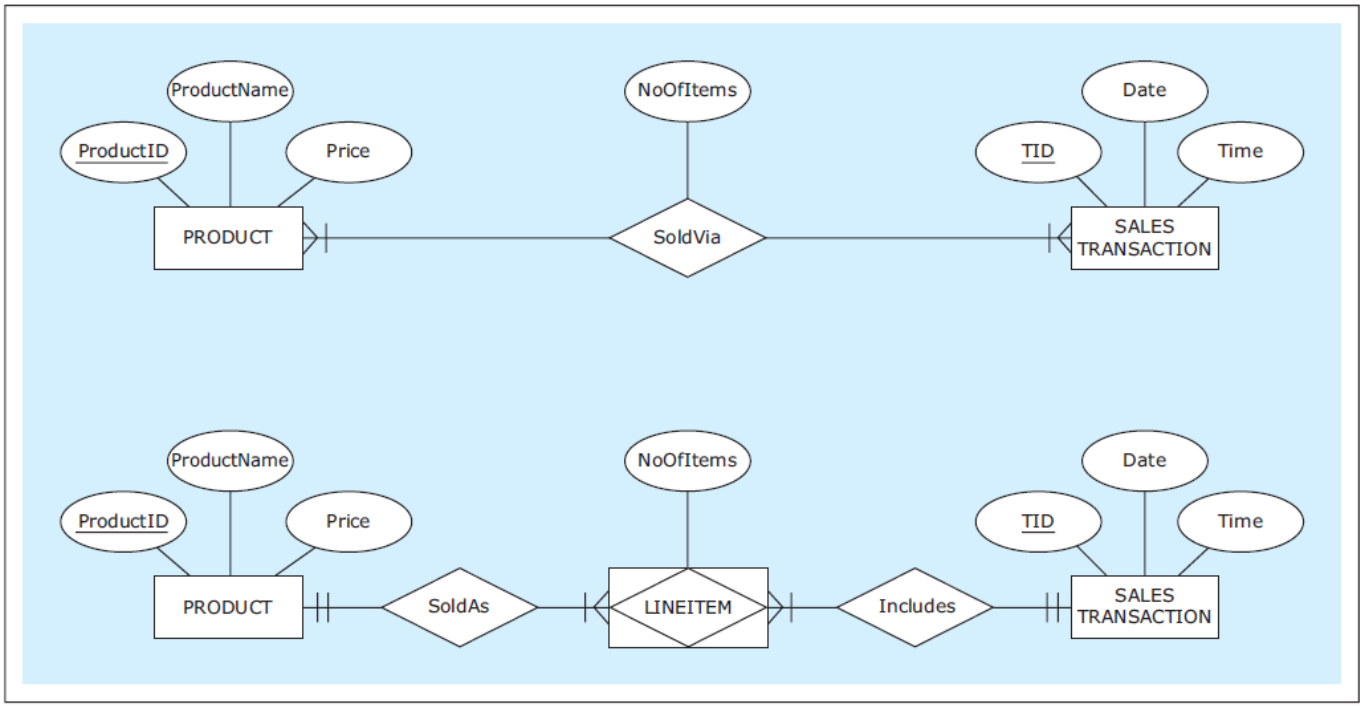
ASSOCIATIVE ENTITY

An identical relationship represented as a unary M:N relationship and as an associative entity



ASSOCIATIVE ENTITY

An identical relationship represented as an M:N relationship with an attribute and as an associative entity with an attribute



ASSOCIATIVE ENTITY

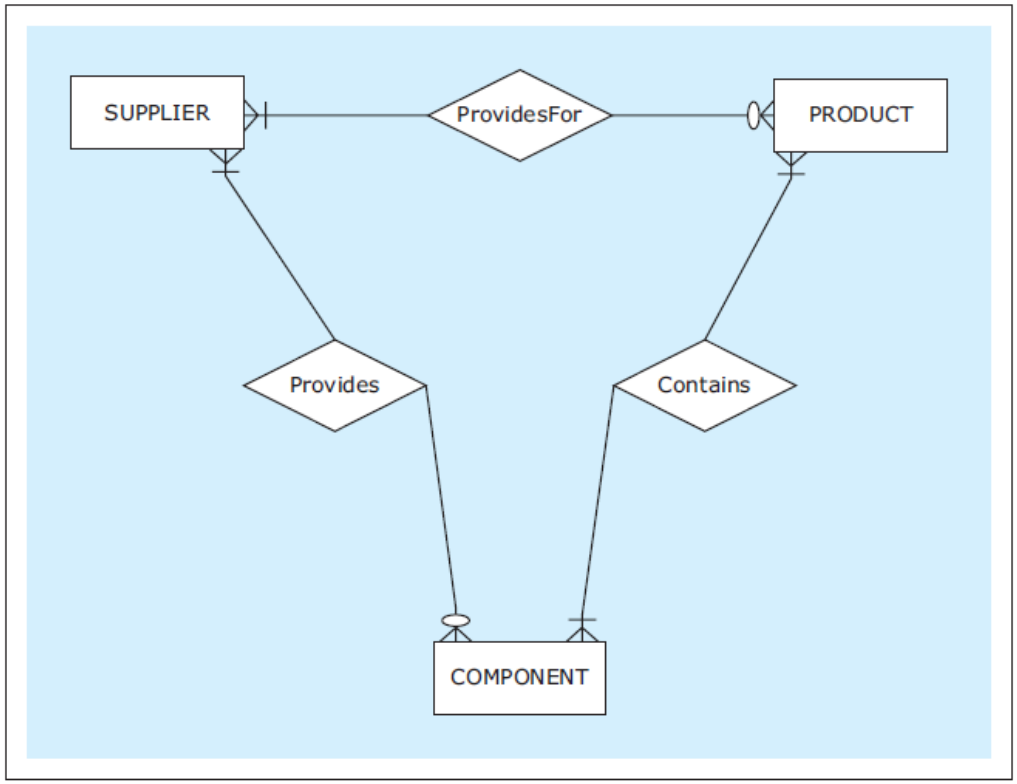
- For relationships with a degree higher than 2, such as ternary relationships, associative entities provide a way to eliminate potential ambiguities in the ER diagrams

TERNARY RELATIONSHIP

- **Ternary relationship -** relationship involving three entities *(degree 3 relationship)*

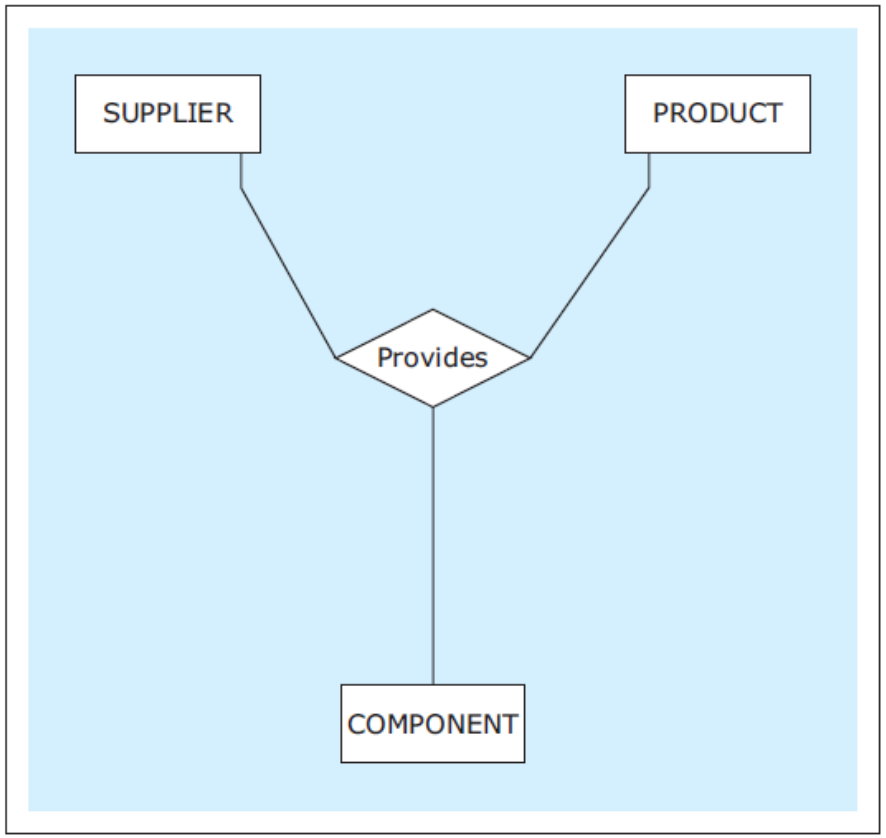
TERNARY RELATIONSHIP

Three binary relationships that are insufficient for depicting given requirements



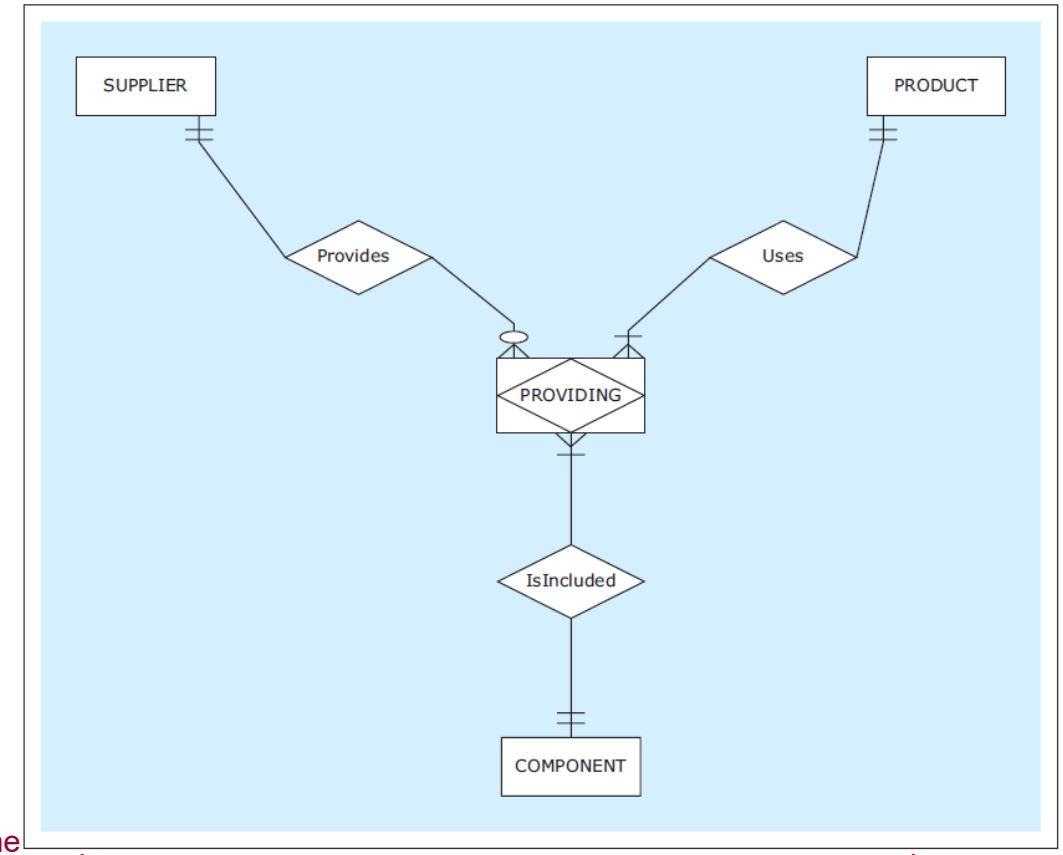
TERNARY RELATIONSHIP

A ternary relationship



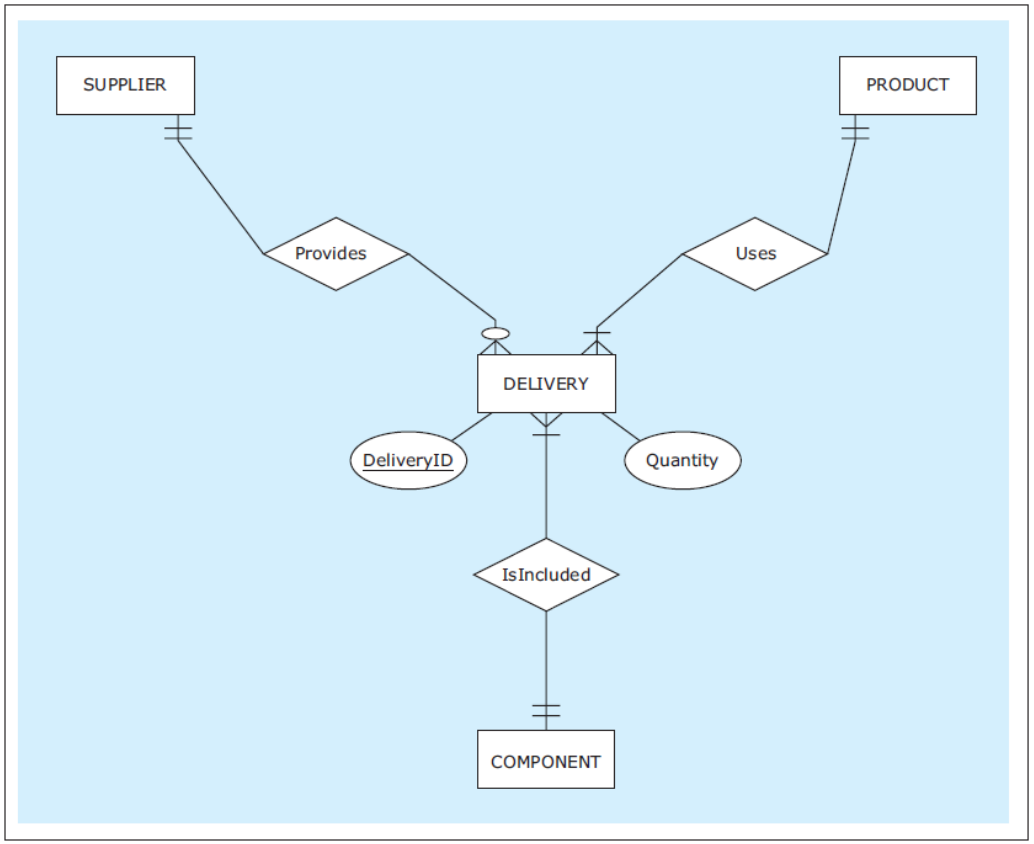
TERNARY RELATIONSHIP

A ternary relationship via associative entity



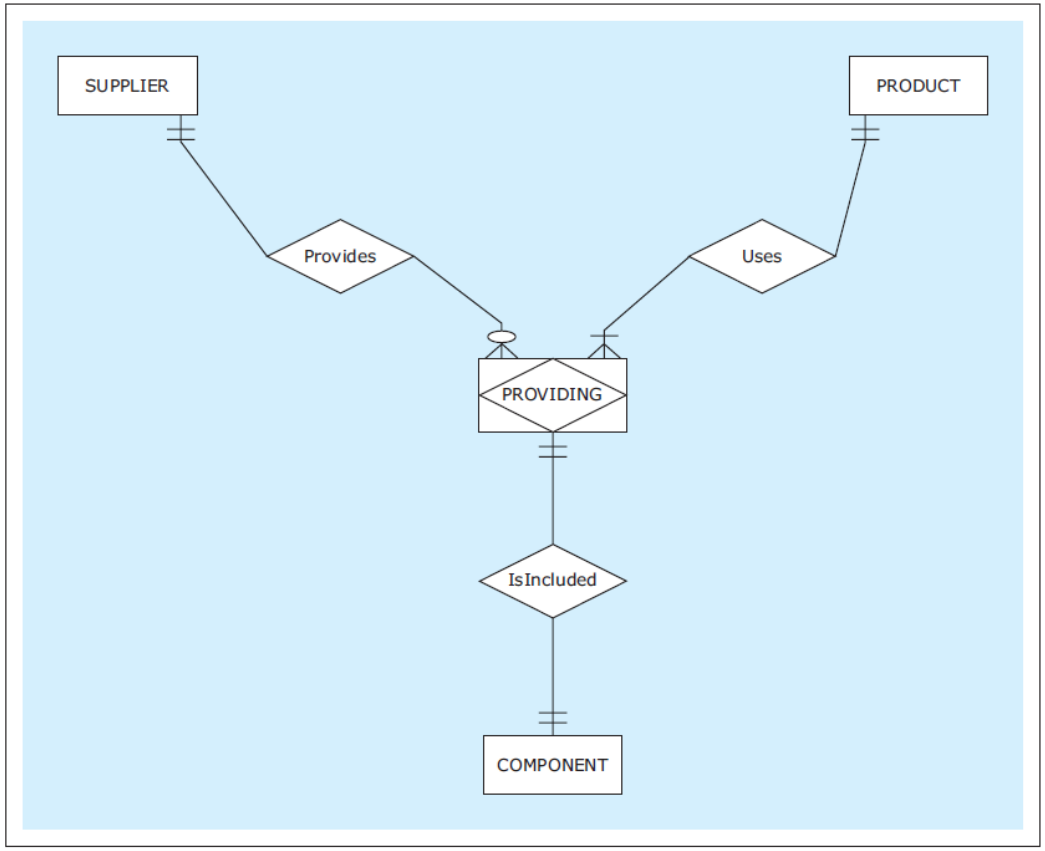
TERNARY RELATIONSHIP

A regular entity replacing a ternary relationship



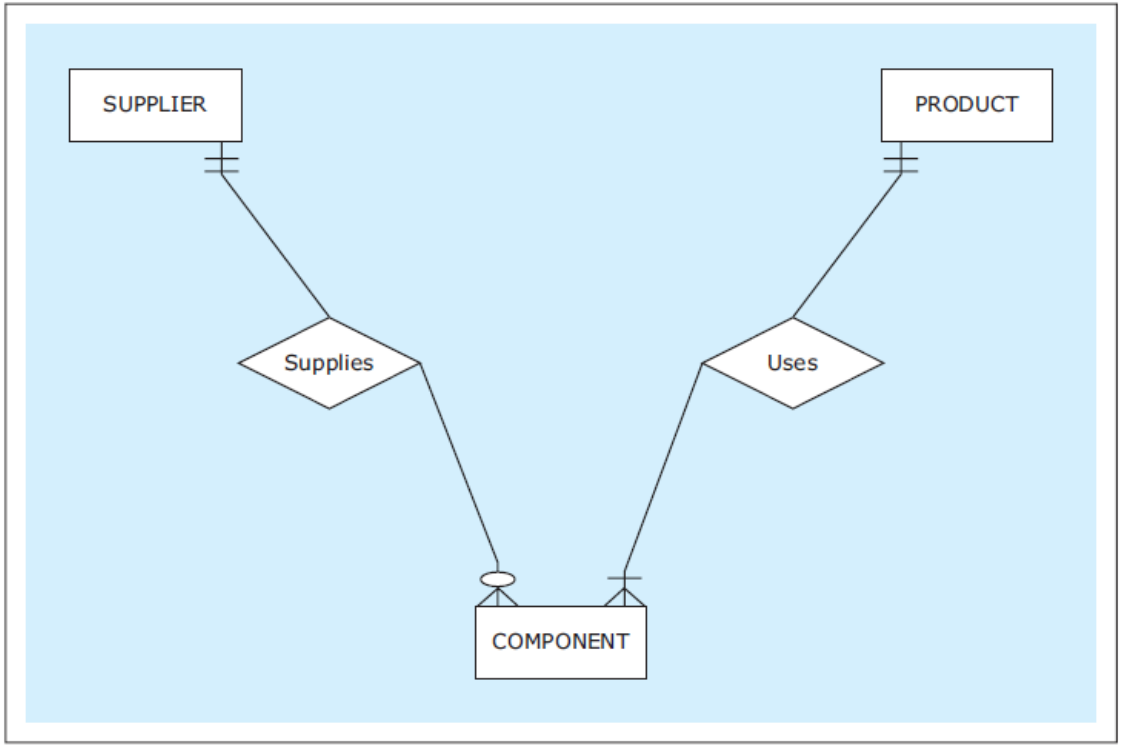
TERNARY RELATIONSHIP

A many-to-many-to-one ternary relationship



TERNARY RELATIONSHIP

A many-to-many-to-one ternary relationship



TERNARY (AND HIGHER DEGREE) RELATIONSHIPS

- In practice, ternary relationships are relatively rare, and relationships of degree higher than 3 are rarer still

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 Models (continued)

- **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)

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. 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).
- **Self-Describing Data Models:**
  - Combine the description of data with the data values. Examples include XML, key-value stores and some NOSQL systems.

Schemas versus Instances

- Database Schema:
  - The ***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.

Schemas versus Instances

- 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 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 state when it is initially loaded into the system.
- Valid State:
  - A state that satisfies the structure and constraints of the database.

Database Schema vs. Database State (continued)

- 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**.

Example of a Database Schema

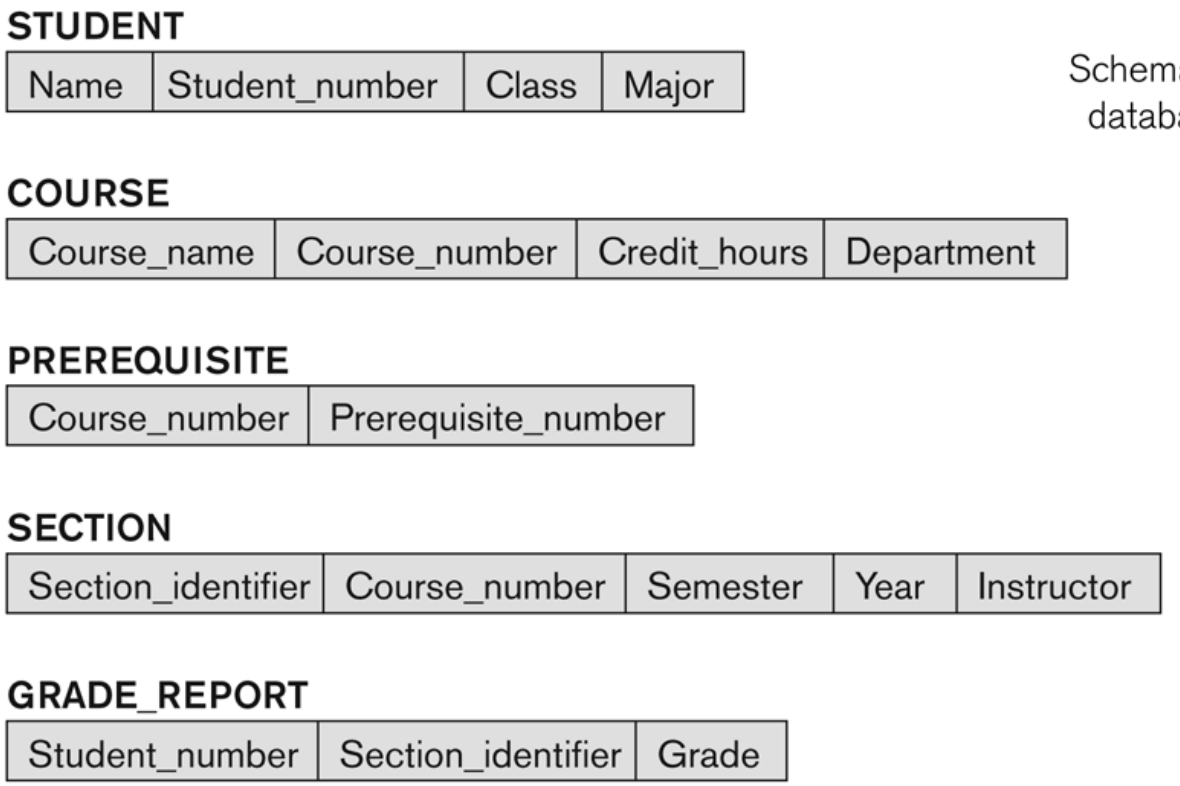


Figure 2.1
Schema diagram for the database in Figure 1.2.

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.

Three-Schema Architecture

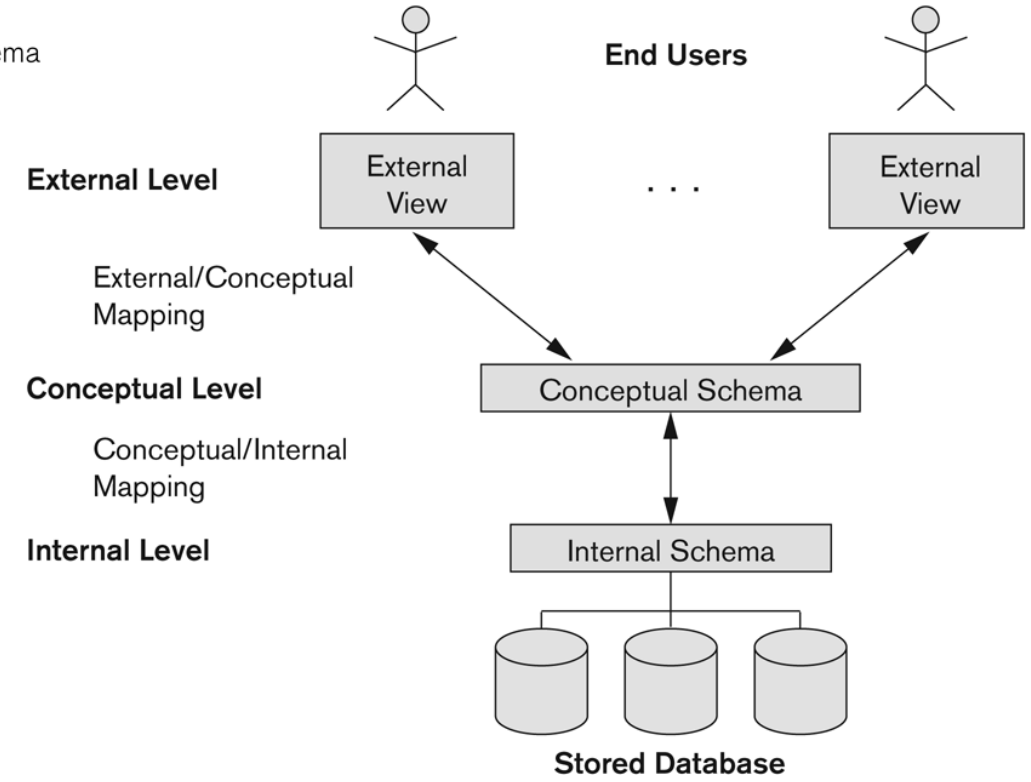
- Proposed to support DBMS characteristics of:
  - **Program-data independence.**
  - Support of **multiple views** of the data.
- Not explicitly used in commercial DBMS products, but has been useful in explaining database system organization

Three-Schema Architecture

- 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.

The three-schema architecture

Figure 2.2
The three-schema architecture.



Three-Schema Architecture

- 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

Data Independence (continued)

- 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)
- 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

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.
    - SDL is typically realized via DBMS commands provided to the DBA and database designers

DBMS Languages

- **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.
- Mobile Interfaces:interfaces allowing users to perform transactions using mobile apps

DBMS Programming Language Interfaces

- Programmer interfaces for embedding DML in a programming languages:
  - **Embedded Approach**: e.g embedded SQL (for C, C++, etc.), SQLJ (for Java)
  - **Procedure Call Approach**: e.g. JDBC for Java, ODBC (Open Databse Connectivity) for other programming languages as API's (application programming interfaces)
  - **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
  - **Scripting Languages:** PHP (client-side scripting) and Python (server-side scripting) are used to write database programs.

User-Friendly DBMS Interfaces

- Menu-based (Web-based), popular for browsing on the web
- Forms-based, designed for naïve users used to filling in entries on a form
- Graphics-based
  - Point and Click, Drag and Drop, etc.
  - Specifying a query on a schema diagram
- Natural language: requests in written English
- Combinations of the above:
  - For example, both menus and forms used extensively in Web database interfaces

Other DBMS Interfaces

- Natural language: free text as a query
- Speech : Input query and Output response
- Web Browser with keyword search
- 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

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.
  - Performance monitoring utilities.
  - Report generation utilities.
  - Other functions, such as sorting, user monitoring, data compression, etc.

Other Tools

- Data dictionary / repository:
  - Used to store schema descriptions and other information such as design decisions, application program descriptions, user information, usage standards, etc.
  - **Active data dictionary** is accessed by DBMS software and users/DBA.
  - **Passive data dictionary** is accessed by users/DBA only.

Other Tools

- Application Development Environments and CASE (computer-aided software engineering) tools:
- Examples:
  - PowerBuilder (Sybase)
  - JBuilder (Borland)
  - JDeveloper 10G (Oracle)
  - Open-Source Platforms

Typical DBMS Component Modules

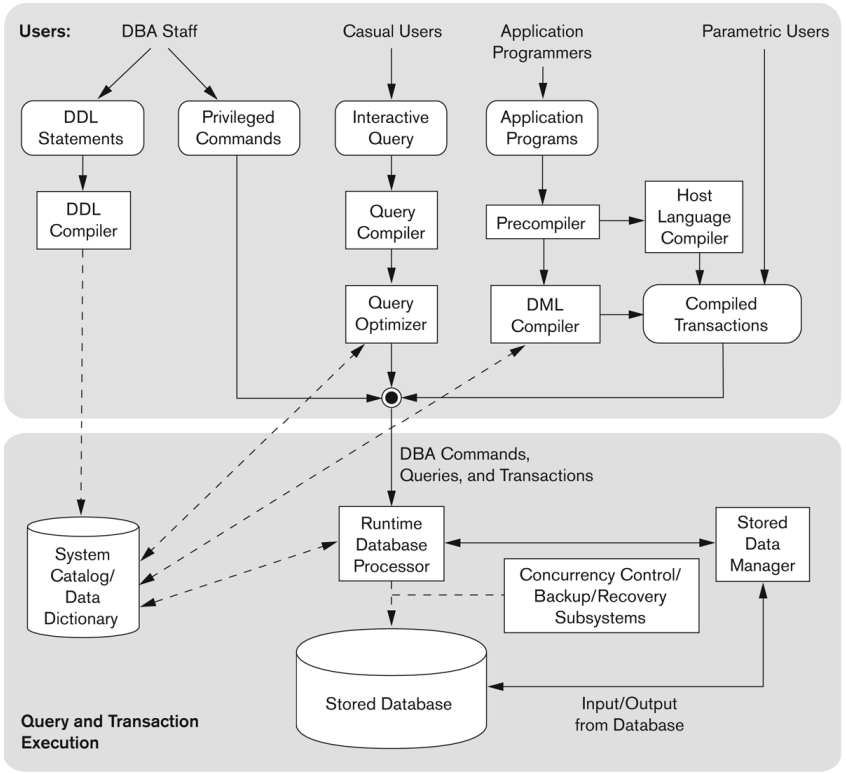


Figure 2.3
Component modules of a DBMS and their interactions.

Centralized and Client-Server DBMS Architectures

- Centralized DBMS:
  - Combines everything into single system including- DBMS software, hardware, application programs, and user interface processing software.
  - User can still connect through a remote terminal – however, all processing is done at centralized site.

A Physical Centralized Architecture

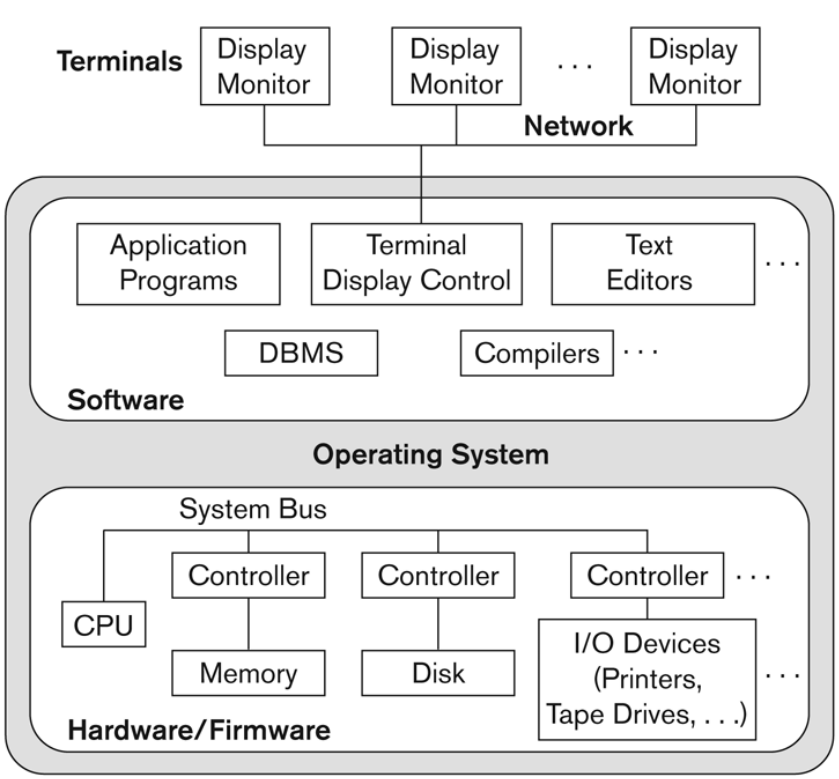


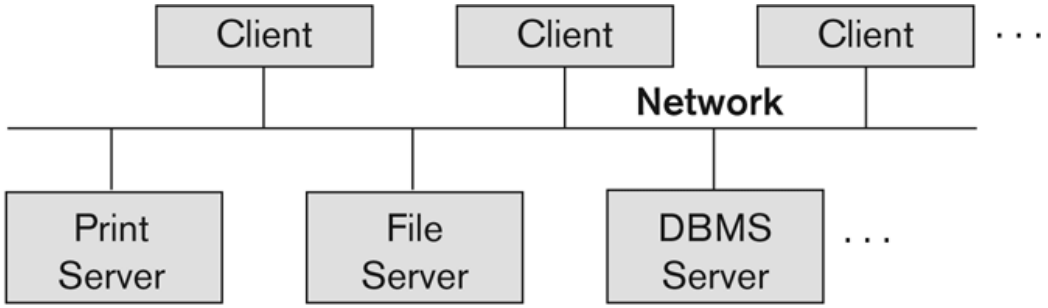
Figure 2.4
A physical centralized architecture.

Basic 2-tier Client-Server Architectures

- Specialized Servers with Specialized functions
  - Print server
  - File server
  - DBMS server
  - Web server
  - Email server
- Clients can access the specialized servers as needed

Logical two-tier client server architecture

Figure 2.5
Logical two-tier client/server architecture.



Clients

- Provide appropriate interfaces through a client software module to access and utilize the various server resources.
- Clients may be diskless machines or PCs or Workstations with disks with only the client software installed.
- Connected to the servers via some form of a network.
  - (LAN: local area network, wireless network, etc.)

DBMS Server

- Provides database query and transaction services to the clients
- Relational DBMS servers are often called SQL servers, query servers, or transaction servers
- Applications running on clients utilize an Application Program Interface (**API**) to access server databases via standard interface such as:
  - ODBC: Open Database Connectivity standard
  - JDBC: for Java programming access

Two Tier Client-Server Architecture

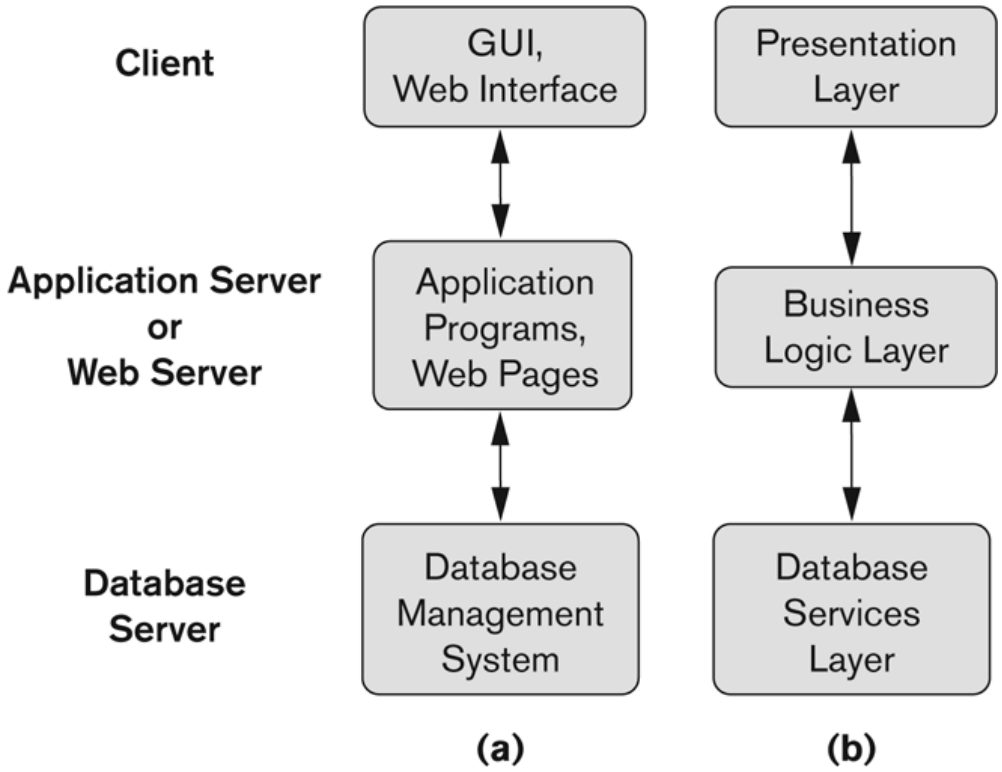
- Client and server must install appropriate client module and server module software for ODBC or JDBC
- A client program may connect to several DBMSs, sometimes called the data sources.
- In general, data sources can be files or other non-DBMS software that manages data.
- See Chapter 10 for details on Database Programming

Three Tier Client-Server Architecture

- Common for Web applications
- Intermediate Layer called Application Server or Web Server:
  - Stores the web connectivity software and the business logic part of the application used to access the corresponding data from the database server
  - Acts like a conduit for sending partially processed data between the database server and the client.
- Three-tier Architecture Can Enhance Security:
  - Database server only accessible via middle tier
  - Clients cannot directly access database server
  - Clients contain user interfaces and Web browsers
  - The client is typically a PC or a mobile device connected to the Web

Three-tier client-server architecture

Figure 2.7
Logical three-tier client/server architecture, with a couple of commonly used nomenclatures.



Classification of DBMSs

- Based on the data model used
  - Legacy: Network, Hierarchical.
  - Currently Used: Relational, Object-oriented, Object-relational
  - Recent Technologies: Key-value storage systems, NOSQL systems: document based, column-based, graph-based and key-value based. Native XML DBMSs.
- Other classifications
  - Single-user (typically used with personal computers) vs. multi-user (most DBMSs).
  - Centralized (uses a single computer with one database) vs. distributed (multiple computers, multiple DBs)

Variations of Distributed DBMSs (DDBMSs)

- Homogeneous DDBMS
- Heterogeneous DDBMS
- Federated or Multidatabase Systems
  - Participating Databases are loosely coupled with high degree of autonomy.
- 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.

Cost considerations for DBMSs

- Cost Range: from free open-source systems to configurations costing millions of dollars
- Examples of free relational DBMSs: MySQL, PostgreSQL, others
- Commercial DBMS offer additional specialized modules, e.g. time-series module, spatial data module, document module, XML module
  - These offer additional specialized functionality when purchased separately
  - Sometimes called cartridges (e.g., in Oracle) or blades
- Different licensing options: site license, maximum number of concurrent users (seat license), single user, etc.

Other Considerations

- Type of access paths within database system
  - E.g.- inverted indexing based (ADABAS is one such system). Fully indexed databases provide access by any keyword (used in search engines)
- General Purpose vs. Special Purpose
  - E.g.- Airline Reservation systems or many others-reservation systems for hotel/car etc. Are special purpose OLTP (Online Transaction Processing Systems)

Relational Data Model and Relational Database Constraints

- Relational Model Concepts
- Relational Model Constraints and Relational Database Schemas
- Update Operations and Dealing with Constraint Violations

Relational Model Concepts

- The relational Model of Data is based on the concept of a *Relation*
  - The strength of the relational approach to data management comes from the formal foundation provided by the theory of relations
- We review the essentials of the *formal relational model* in this chapter
- In *practice*, there is a *standard model* based on SQL – this is described in Chapters 6 and 7 as a language
- Note: There are several important differences between the *formal* model and the *practical* model, as we shall see

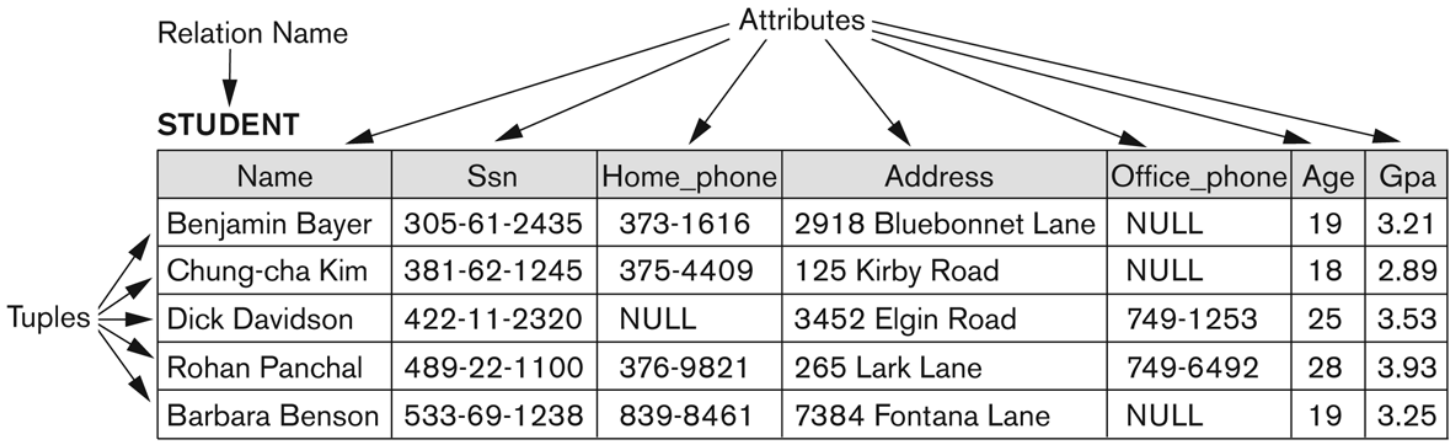
Relational Model Concepts

- A Relation is a mathematical concept based on the ideas of sets
- The model was first proposed by Dr. E.F. Codd of IBM Research 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 Dr. Codd the coveted ACM Turing Award

Informal Definitions

- Informally, a **relation** looks like a **table** of values.
- A relation typically contains a **set of rows**.
- The data elements in each **row** represent certain facts that correspond to a real-world **entity** or **relationship**
  - In the formal model, rows are called **tuples**
- Each **column** has a column header that gives an indication of the meaning of the data items in that column
  - In the formal model, the column header is called an **attribute name** (or just **attribute**)

Example of a Relation



Name	Ssn	Home_phone	Address	Office_phone	Age	Gpa
Benjamin Bayer	305-61-2435	373-1616	2918 Bluebonnet Lane	NULL	19	3.21
Chung-cha Kim	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
Rohan Panchal	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

Figure 5.1
The attributes and tuples of a relation STUDENT.

Informal Definitions

- Key of a Relation:
  - Each row has a value of a data item (or set of items) that uniquely identifies that row in the table
    - Called the *key*
  - In the STUDENT table, SSN is the key

  - Sometimes row-ids or sequential numbers are assigned as keys to identify the rows in a table
    - Called *artificial key* or *surrogate key*

Formal Definitions - Schema

- The **Schema** (or description) of a Relation:
  - Denoted by R(A1, A2, ...An)
  - R is the **name** of the relation
  - The **attributes** of the relation are A1, A2, ..., An
- Example:

  CUSTOMER (Cust-id, Cust-name, Address, Phone#)
  - CUSTOMER is the relation name
  - Defined over the four attributes: Cust-id, Cust-name, Address, Phone#
- Each attribute has a **domain** or a set of valid values.
  - For example, the domain of Cust-id is 6 digit numbers.

Formal Definitions - Tuple

- A **tuple** is an ordered set of values (enclosed in angled brackets '< … >')
- Each value is derived from an appropriate *domain*.
- A row in the CUSTOMER relation is a 4-tuple and would consist of four values, for example:
  - <632895, "John Smith", "101 Main St. Atlanta, GA 30332", "(404) 894-2000">
  - This is called a 4-tuple as it has 4 values
  - A tuple (row) in the CUSTOMER relation.
- A relation is a **set** of such tuples (rows)

Formal Definitions - Domain

- A **domain** has a logical definition:
  - Example: “USA_phone_numbers” are the set of 10 digit phone numbers valid in the U.S.
- A domain also has 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.
  - Dates have various formats such as year, month, date formatted as yyyy-mm-dd, or as dd mm,yyyy etc.

- The attribute name designates the role played by a domain in a relation:
  - Used to interpret the meaning of the data elements corresponding to that attribute
  - Example: The domain Date may be used to define two attributes named “Invoice-date” and “Payment-date” with different meanings

Formal Definitions - State

- The **relation state** is a subset of the Cartesian product of the domains of its attributes
  - each domain contains the set of all possible values the attribute can take.
- Example: attribute Cust-name is defined over the domain of character strings of maximum length 25
  - dom(Cust-name) is varchar(25)
- The role these strings play in the CUSTOMER relation is that of the *name of a customer*.

Formal Definitions - Summary

- Formally,
  - Given R(A1, A2, ..., An)
  - r(R) $\subset$ dom (A1) X dom (A2) X ....X dom(An)
- R(A1, A2, …, An) is the **schema** of the relation
- R is the **name** of the relation
- A1, A2, …, An are the **attributes** of the relation
- r(R): a specific **state** (or "value" or “population”) of relation R – this is a *set of tuples* (rows)
  - r(R) = {t1, t2, …, tn} where each ti is an n-tuple
  - ti = <v1, v2, …, vn> where each vj *element-of* dom(Aj)

Formal Definitions - Example

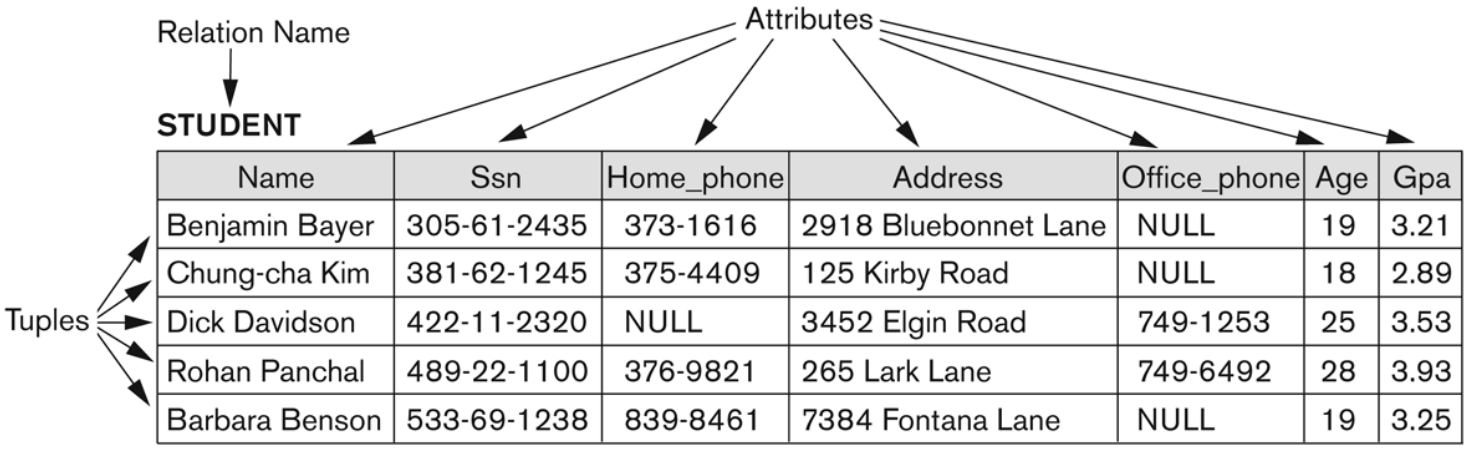
- Let R(A1, A2) be a relation schema:
  - Let dom(A1) = {0,1}
  - Let dom(A2) = {a,b,c}
- Then: dom(A1) X dom(A2) is all possible combinations:
  {<0,a> , <0,b> , <0,c>, <1,a>, <1,b>, <1,c> }

- The relation state r(R) $\subset$ dom(A1) X dom(A2)
- For example: r(R) could be {<0,a> , <0,b> , <1,c> }
  - this is one possible state (or "population" or "extension") r of the relation R, defined over A1 and A2.
  - It has three 2-tuples: <0,a> , <0,b> , <1,c>

Definition Summary

Informal Terms		Formal Terms
Table		Relation
Column Header		Attribute
All possible Column Values		Domain
Row		Tuple
Table Definition		Schema of a Relation
Populated Table		State of the Relation

Example – A relation STUDENT



Name	Ssn	Home_phone	Address	Office_phone	Age	Gpa
Benjamin Bayer	305-61-2435	373-1616	2918 Bluebonnet Lane	NULL	19	3.21
Chung-cha Kim	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
Rohan Panchal	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

Figure 5.1
The attributes and tuples of a relation STUDENT.

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(A1, A2, ..., An) and the values in t=<v1, v2, ..., vn> to be ordered .
    - (However, a more general alternative definition of relation does not require this ordering. It includes both the name and the value for each of the attributes).
    - Example: t= { <name, “John” >, <SSN, 123456789> }
    - This representation may be called as “self-describing”.

Same state as previous Figure (but with different order of tuples)

Figure 5.2
The relation STUDENT from Figure 5.1 with a different order of tuples.

STUDENT

Name	Ssn	Home_phone	Address	Office_phone	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
Rohan Panchal	489-22-1100	376-9821	265 Lark Lane	749-6492	28	3.93
Chung-cha Kim	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

Characteristics Of Relations

- Values in a tuple:
  - All values are considered atomic (indivisible).
  - Each value in a tuple must be from the domain of the attribute for that column
    - If tuple t = <v1, v2, …, vn> is a tuple (row) in the relation state r of R(A1, A2, …, An)
    - Then each *vi* must be a value from *dom(Ai)*
  - A special **null** value is used to represent values that are unknown or not available or inapplicable in certain tuples.

Characteristics Of Relations

- Notation:
  - We refer to **component values** of a tuple t by:
    - t[Ai] or t.Ai
    - This is the value vi of attribute Ai for tuple t
  - Similarly, t[Au, Av, ..., Aw] refers to the subtuple of t containing the values of attributes Au, Av, ..., Aw, respectively in t

CONSTRAINTS

Constraints determine which values are permissible and which are not in the database.

They are of three main types:

1. **Inherent or Implicit Constraints**: These are based on the data model itself. (E.g., relational model does not allow a list as a value for any attribute)
2. **Schema-based or Explicit Constraints**: They are expressed in the schema by using the facilities provided by the model. (E.g., max. cardinality ratio constraint in the ER model)
3. **Application based or semantic constraints**: These are beyond the expressive power of the model and must be specified and enforced by the application programs.

Relational Integrity Constraints

- Constraints are **conditions** that must hold on **all** valid relation states.
- There are three *main types* of (explicit schema-based) constraints that can be expressed in the relational model:
  - **Key** constraints
  - **Entity integrity** constraints
  - **Referential integrity** constraints
- Another schema-based constraint is the **domain** constraint
  - Every value in a tuple must be from the *domain of its attribute* (or it could be **null**, if allowed for that attribute)

Key Constraints

- **Superkey** of R:
  - Is a set of attributes SK of R with the following condition:
    - No two tuples in any valid relation state r(R) will have the same value for SK
    - That is, for any distinct tuples t1 and t2 in r(R), t1[SK] $\neq$ t2[SK]
    - This condition must hold in *any valid state* r(R)
- **Key** of R:
  - A "minimal" superkey
  - That is, a key is a superkey K such that removal of any attribute from K results in a set of attributes that is not a superkey (does not possess the superkey uniqueness property)
- A Key is a Superkey but not vice versa

Key Constraints (continued)

- Example: Consider the CAR relation schema:
  - CAR(State, Reg#, SerialNo, Make, Model, Year)
  - CAR has two keys:
    - Key1 = {State, Reg#}
    - Key2 = {SerialNo}
  - Both are also superkeys of CAR
  - {SerialNo, Make} is a superkey but *not* a key.
- In general:
  - Any *key* is a *superkey* (but not vice versa)
  - Any set of attributes that *includes a key* is a *superkey*
  - A *minimal* superkey is also a key

Key Constraints (continued)

- If a relation has several **candidate keys**, one is chosen arbitrarily to be the **primary key**.
  - The primary key attributes are underlined.
- Example: Consider the CAR relation schema:
  - CAR(State, Reg#, SerialNo, Make, Model, Year)
  - We chose SerialNo as the primary key
- The primary key value is used to *uniquely identify* each tuple in a relation
  - Provides the tuple identity
- Also used to *reference* the tuple from another tuple
  - General rule: Choose as primary key the smallest of the candidate keys (in terms of size)
  - Not always applicable – choice is sometimes subjective

CAR table with two candidate keys – LicenseNumber chosen as Primary Key

Figure 5.4
The CAR relation, with two candidate keys: License_number and Engine_serial_number.

CAR

License_number	Engine_serial_number	Make	Model	Year
Texas ABC-739	A69352	Ford	Mustang	02
Florida TVP-347	B43696	Oldsmobile	Cutlass	05
New York MPO-22	X83554	Oldsmobile	Delta	01
California 432-TFY	C43742	Mercedes	190-D	99
California RSK-629	Y82935	Toyota	Camry	04
Texas RSK-629	U028365	Jaguar	XJS	04

Relational Database Schema

- **Relational Database Schema:**
  - A set S of relation schemas that belong to the same database.
  - S is the name of the whole **database schema**
  - S = {R1, R2, ..., Rn} and a set IC of integrity constraints.
  - R1, R2, …, Rn are the names of the individual **relation schemas** within the database S
- Following slide shows a COMPANY database schema with 6 relation schemas

COMPANY Database Schema

EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno

DEPARTMENT

Dname	Dnumber	Mgr_ssn	Mgr_start_date

DEPT_LOCATIONS

Dnumber	Dlocation

PROJECT

Pname	Pnumber	Plocation	Dnum

WORKS_ON

Essn	Pno	Hours

DEPENDENT

Essn	Dependent_name	Sex	Bdate	Relationship

Figure 5.5
Schema diagram for the COMPANY relational database schema.

Relational Database State

- A **relational database state** DB of S is a set of relation states DB = $\{r_1, r_2, ..., r_m\}$ such that each r_i is a state of R_i and such that the r_i relation states satisfy the integrity constraints specified in IC.
- A relational database *state* is sometimes called a relational database *snapshot* or *instance*.
- We will not use the term *instance* since it also applies to single tuples.
- A database state that does not meet the constraints is an invalid state

Populated database state

- Each *relation* will have many tuples in its current relation state
- The *relational database state* is a union of all the individual relation states
- Whenever the database is changed, a new state arises
- Basic operations for changing the database:
  - INSERT a new tuple in a relation
  - DELETE an existing tuple from a relation
  - MODIFY an attribute of an existing tuple
- Next slide (Fig. 5.6) shows an example state for the COMPANY database schema shown in Fig. 5.5.

Populated database state for COMPANY

Figure 5.6
One possible database state for the COMPANY relational database schema.

EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
John	B	Smith	123456789	1965-01-09	731 Fondren, Houston, TX	M	30000	333445555	5
Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5
Alicia	J	Zelaya	999887777	1968-01-19	3321 Castle, Spring, TX	F	25000	987654321	4
Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	43000	888665555	4
Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble, TX	M	38000	333445555	5
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5
Ahmad	V	Jabbar	987987987	1969-03-29	980 Dallas, Houston, TX	M	25000	987654321	4
James	E	Borg	888665555	1937-11-10	450 Stone, Houston, TX	M	55000	NULL	1

DEPARTMENT

Dname	Dnumber	Mgr_ssn	Mgr_start_date
Research	5	333445555	1988-05-22
Administration	4	987654321	1995-01-01
Headquarters	1	888665555	1981-06-19

DEPT_LOCATIONS

Dnumber	Dlocation
1	Houston
4	Stafford
5	Bellaire
5	Sugarland
5	Houston

WORKS_ON

Essn	Pno	Hours
123456789	1	32.5
123456789	2	7.5
666884444	3	40.0
453453453	1	20.0
453453453	2	20.0
333445555	2	10.0
333445555	3	10.0
333445555	10	10.0
333445555	20	10.0
999887777	30	30.0
999887777	10	10.0
987987987	10	35.0
987987987	30	5.0
987654321	30	20.0
987654321	20	15.0
888665555	20	NULL

PROJECT

Pname	Pnumber	Plocation	Dnum
ProductX	1	Bellaire	5
ProductY	2	Sugarland	5
ProductZ	3	Houston	5
Computerization	10	Stafford	4
Reorganization	20	Houston	1
Newbenefits	30	Stafford	4

DEPENDENT

Essn	Dependent_name	Sex	Bdate	Relationship
333445555	Alice	F	1986-04-05	Daughter
333445555	Theodore	M	1983-10-25	Son
333445555	Joy	F	1958-05-03	Spouse
987654321	Abner	M	1942-02-28	Spouse
123456789	Michael	M	1988-01-04	Son
123456789	Alice	F	1988-12-30	Daughter
123456789	Elizabeth	F	1967-05-05	Spouse

Entity Integrity

- **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)
    - If PK has several attributes, null is not allowed in any of these attributes
  - Note: Other attributes of R may be 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**.

Referential Integrity

- Tuples in the **referencing relation** R1 have attributes FK (called **foreign key** attributes) that reference the primary key attributes PK of the **referenced relation** R2.
  - A tuple t1 in R1 is said to **reference** a tuple t2 in R2 if t1[FK] = t2[PK].
- A referential integrity constraint can be displayed in a relational database schema as a directed arc from R1.FK to R2.

Referential Integrity (or foreign key) Constraint

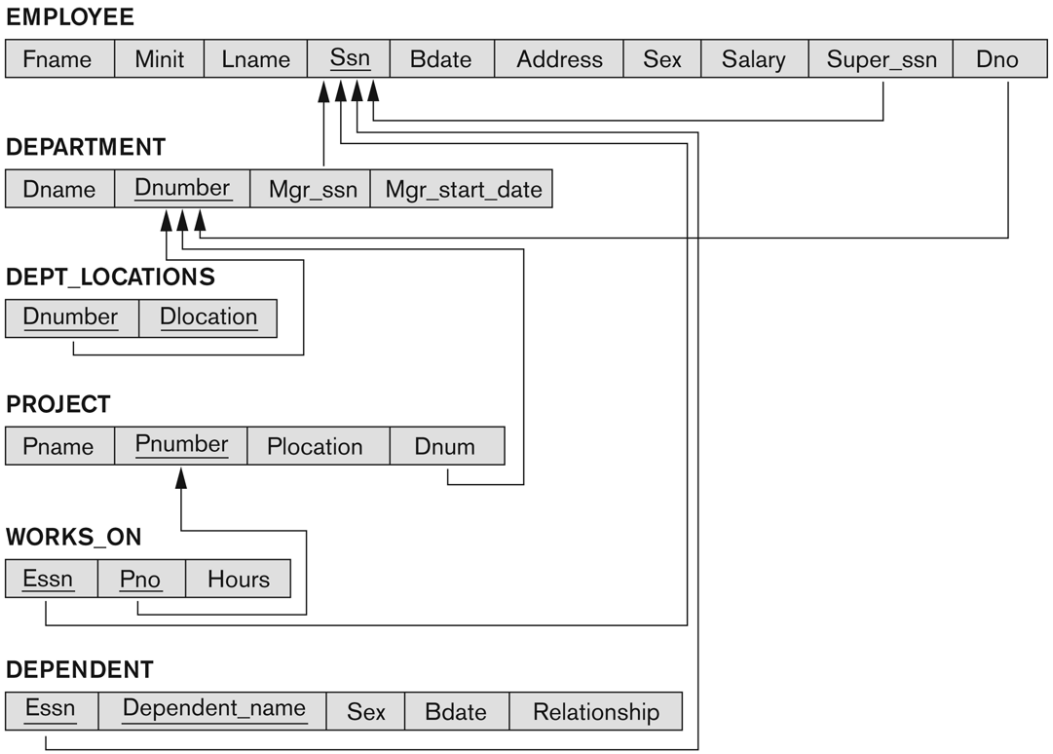
- Statement of the constraint
  - The value in the foreign key column (or columns) FK of the the **referencing relation** R1 can be **either**:
    - (1) a value of an existing primary key value of a corresponding primary key PK in the **referenced relation** R2, or
    - (2) a **null**.
- In case (2), the FK in R1 should **not** be a part of its own primary key.

Displaying a relational database schema and its constraints

- Each relation schema can be displayed as a row of attribute names
- The name of the relation is written above the attribute names
- The primary key attribute (or attributes) will be underlined
- A foreign key (referential integrity) constraints is displayed as a directed arc (arrow) from the foreign key attributes to the referenced table
  - Can also point the the primary key of the referenced relation for clarity
- Next slide shows the COMPANY **relational schema diagram with referential integrity constraints**

Referential Integrity Constraints for COMPANY database

Figure 5.7
Referential integrity constraints displayed on the COMPANY relational database schema.



Other Types of Constraints

- Semantic Integrity Constraints:
  - based on application semantics and cannot be expressed by the model per se
  - Example: “the max. no. of hours per employee for all projects he or she works on is 56 hrs per week”
- A **constraint specification** language may have to be used to express these
- SQL-99 allows **CREATE TRIGGER** and **CREATE ASSERTION** to express some of these semantic constraints
- Keys, Permissibility of Null values, Candidate Keys (Unique in SQL), Foreign Keys, Referential Integrity etc. are expressed by the **CREATE TABLE** statement in SQL.

Update Operations on Relations

- INSERT a tuple.
- DELETE a tuple.
- MODIFY a tuple.
- Integrity constraints should not be violated by the update operations.
- Several update operations may have to be grouped together.
- Updates may **propagate** to cause other updates automatically. This may be necessary to maintain integrity constraints.

Update Operations on Relations

- In case of integrity violation, several actions can be taken:
  - Cancel the operation that causes the violation (RESTRICT or REJECT option)
  - Perform the operation but inform the user of the violation
  - Trigger additional updates so the violation is corrected (CASCADE option, SET NULL option)
  - Execute a user-specified error-correction routine

Possible violations for each operation

- INSERT may violate any of the constraints:
  - Domain constraint:
    - if one of the attribute values provided for the new tuple is not of the specified attribute domain
  - Key constraint:
    - if the value of a key attribute in the new tuple already exists in another tuple in the relation
  - Referential integrity:
    - if a foreign key value in the new tuple references a primary key value that does not exist in the referenced relation
  - Entity integrity:
    - if the primary key value is null in the new tuple

Possible violations for each operation

- DELETE may violate only referential integrity:
  - If the primary key value of the tuple being deleted is referenced from other tuples in the database
    - Can be remedied by several actions: RESTRICT, CASCADE, SET NULL (see Chapter 6 for more details)
      - RESTRICT option: reject the deletion
      - CASCADE option: propagate the new primary key value into the foreign keys of the referencing tuples
      - SET NULL option: set the foreign keys of the referencing tuples to NULL
  - One of the above options must be specified during database design for each foreign key constraint

Possible violations for each operation

- UPDATE may violate domain constraint and NOT NULL constraint on an attribute being modified
- Any of the other constraints may also be violated, depending on the attribute being updated:
  - Updating the primary key (PK):
    - Similar to a DELETE followed by an INSERT
    - Need to specify similar options to DELETE
  - Updating a foreign key (FK):
    - May violate referential integrity
  - Updating an ordinary attribute (neither PK nor FK):
    - Can only violate domain constraints

Summary

- Presented Relational Model Concepts
  - Definitions
  - Characteristics of relations
- Discussed Relational Model Constraints and Relational Database Schemas
  - Domain constraints
  - Key constraints
  - Entity integrity
  - Referential integrity
- Described the Relational Update Operations and Dealing with Constraint Violations

Structured Query Language

- Structured English QUEry Language (SQL)
- pronounced es′-queue-elle
  - also pronounced see′-quell, like the word sequel
  - ANSI SQL standards
  - SQL:86–1st version, not very complete
  - SQL:89–earliest comprehensive standard
  - SQL:92
    - most widely used
    - most vendors more/less adhere to this with minor variations
  - SQL:99, SQL:03, SQL:06
    - includes OO features, spatial features, remote query features
    - modest vendor support
    - beyond scope of this course –
  - SQL:08, SQL:11
    - Legalizes ORDER BY outside cursor definitions
    - Adds INSTEAD OF triggers
    - Adds TRUNCATE statement
    - Enhances temporal databases (time-based)

SQL: Relational Model DDL & DML

- CREATE TABLE defines table schemas
- INSERT, DELETE, UPDATE modify table contents
- SELECT statement
  - retrieves data
  - implements all RA & RC operations–relationally complete
- CREATE has other uses such as defining indexes
- Other commands perform some DBA functions

CREATE TABLE

```
CREATE TABLE    student
(   name        VARCHAR( 30 ),
    id          INTEGER UNIQUE NOT NULL,
    major       VARCHAR( 10 ),
    gpa         NUMERIC( 5, 2 ),
    advisor     VARCHAR( 30 )
  );

  INSERT INTO student
  VALUES ( 'joe college', 6160, 'cs', 2.6, 'prof keener' );
   …
```

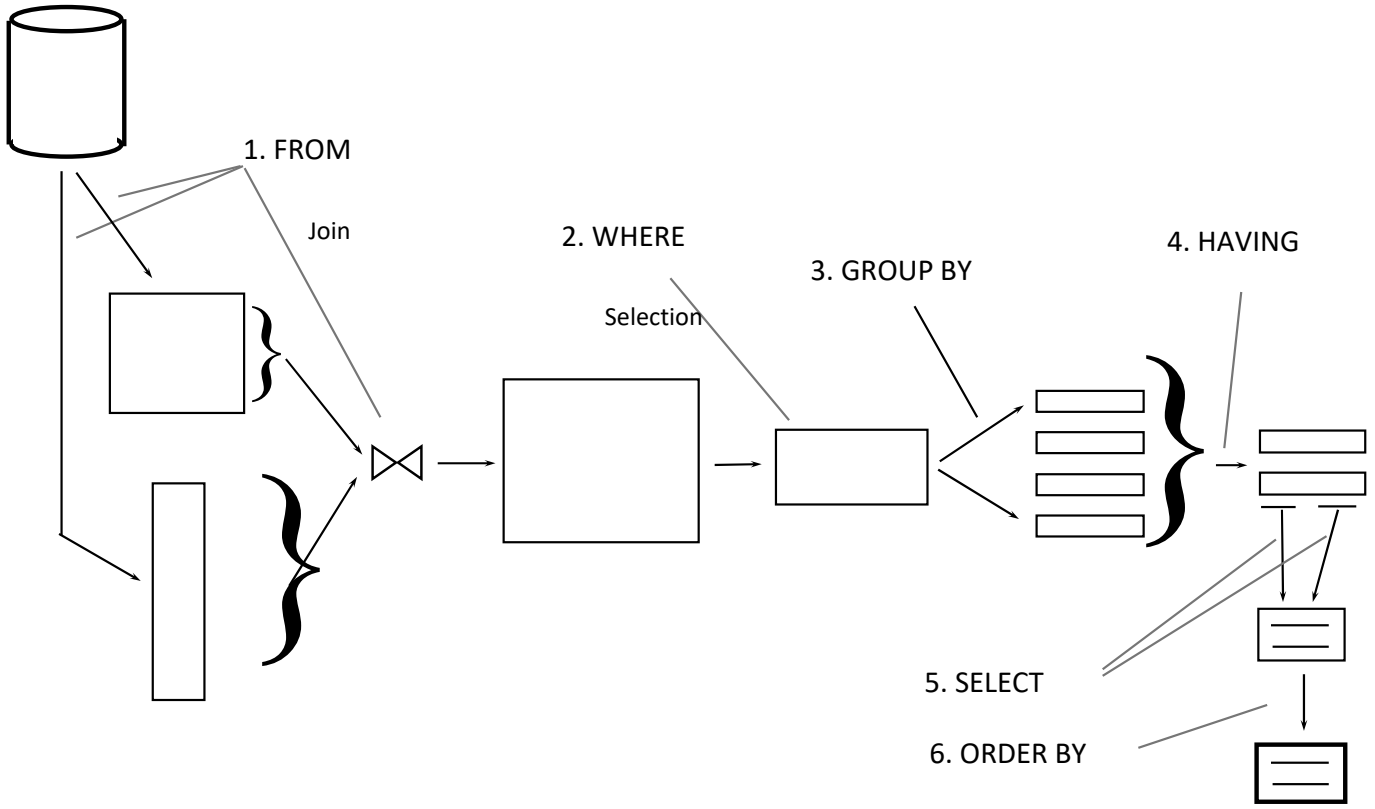
SELECT Syntax

SELECT <attribute-list>
FROM (<table-list> | <join-expression>)
[WHERE <conditional-expression>]
[GROUP BY <attribute-list>]
[HAVING <conditional-expression>]
[ORDER BY <attribute-list>]

SQL literals shown all caps but SQL is case insensitive
[ ] clauses are optional
< > sections are user specified

SELECT –– specifies attributes in the result (including calculated attr.)
FROM –– specifies the source relations for the data
WHERE –– specifies conditions retrieved data must meet
GROUP BY –– specifies grouping for statistical operators
HAVING –– specifies criteria for groups
ORDER BY –– specifies ordering of result

Select Processing –– SQL–92



Retrieval

- Retrieve entire table
  - absent or empty WHERE clause means all rows qualify

    SELECT name, id, major, gpa, advisor FROM student
  - asterisk is shorthand for all columns–equivalent to 1st

    SELECT * FROM student
- Retrieve some attributes

    SELECT major, gpa FROM student
  - duplicates can result when retrieval list excludes unique attribute
    - e.g., student.id
    - duplicates may arise in result (major, gpa)
  - most DBMS do not automatically eliminate duplicates
    - specify DISTINCT to force it

      SELECT DISTINCT name, major, gpa FROM student
  - duplicate removal takes time and is often unnecessary when
    - subquery result is an unseen intermediate result table
    - number of likely duplicates are too small to affect performance

WHERE Clause Returns Selected Rows

SELECT name, major, gpa FROM student WHERE advisor = 'wise';

- comparison expressions can include
  - attribute names from relation(s) in the FROM clause
  - and/or literals (numbers, strings, time, date)
  - separated by relational operators

    =, <> or !=, <, >, <=, >=

LIKE	implements pattern match
BETWEEN	tests upper & lower bounds in one operation
IN	tests set membership
IS NULL	tests for NULL value (more about NULL later)
IS NOT NULL	tests for absence of NULL
EXISTS	Boolean test (more later)

- logical expressions
  - comparison expressions
  - separated by Boolean operators AND, OR, NOT

Literals

- Numbers are written normally
  - periods, sign optional, no comma
  - compare numerically
- Strings in single quotes –– compare alphabetically
  - 'Joe College' = 'Joe College'
  - 'joe college' > 'Joe College' (strings are case sensitive)
  - 'Joe College' > 'Joe Collage'
  - 'Joe''s College' or 'Joe\'s College'
  - \n \b \t … also allowed in strings
- Dates and times compare chronologically

Sample SP Queries

- Comparison of Attribute & Literal

```
SELECT    *                 SELECT    *
 FROM     student            FROM     student
WHERE     gpa >= 3.0;       WHERE     major = 'cs';
```

- Comparison of Attributes

```
SELECT    *
 FROM     student
WHERE     name = advisor;
```

Compound Conditions

usual precedence NOT AND OR applies

parentheses override as usual

```
SELECT    *                 SELECT    name
 FROM     student            FROM     student
WHERE     major = 'cs'      WHERE     gpa >= 2.0
  AND     gpa > 3;            AND     gpa < 3.0;
```

```
SELECT    name
 FROM     student
WHERE     NOT ( gpa < 3.0 )
  AND     ( major = 'cs'
  OR
            major = 'math' );
```

LIKE–String Pattern Matching

- Special pattern characters
  - % represents arbitrary string of 0 or more characters
  - _ represents arbitrary single character
  - \ escape character
    - treats next character as a char literal
    - allows % and _ to be regular character literals

```
SELECT * FROM student WHERE name LIKE 'Joe %';

SELECT * FROM student WHERE major LIKE '%science %';

SELECT * FROM student WHERE name LIKE '%smythe\_jones%';

SELECT name FROM student WHERE name NOT LIKE 'Joe %';
```

BETWEEN -- performs an interval test

- Equivalent to a compound AND expression with <= & >= applied to the same attribute

```
SELECT    name
  FROM    student
WHERE     gpa BETWEEN 2.0 AND 3.0;
```

- NOT BETWEEN -- tests for values outside an interval equivalent to a compound OR expression with > | <

```
SELECT    name
  FROM    student
WHERE     gpa NOT BETWEEN 2.0 AND 3.0;
```

IN – tests for membership in a list of elements

- Equivalent to sequence of ORed tests

```
SELECT    name
  FROM    student
WHERE     major IN ( 'cs', 'math', 'ee' );
```

- NOT + IN -- excludes rows that satisfy membership test; equivalent to negated sequence of ANDed tests

```
SELECT    name
  FROM    student
WHERE     major NOT IN ( 'cs', 'math', 'ee' );
```

NULL is Not a Value

- NULL represents the absence of a value
  - NULLs behave non-intuitively in some queries
- Testing NULL with comparison operators always fails
- These two queries retrieve nothing

  SELECT name FROM student WHERE major = NULL
  - fails even for students with NULL major field

  SELECT name FROM student WHERE major <> NULL
  - the second fails even for students with non NULL major
- These queries together should retrieve all students, but they miss students with NULL gpas

  SELECT name FROM student WHERE gpa <= 3.0

  SELECT name FROM student WHERE gpa > 3.0

IS [NOT] NULL Predicates

- NULL is special, SQL has special test predicates for it
  - IS NULL retrieves rows with NULL in the tested attribute
  - IS NOT NULL excludes rows with NULL in the tested attribute
- Retrieve undeclared students

```
SELECT name FROM student WHERE major IS NULL;
```

- Retrieve students who have declared a major

```
SELECT name FROM student WHERE major IS NOT NULL;
```

- Retrieve all students

```
SELECT name
  FROM student
WHERE major IS NULL
    OR major IS NOT NULL;
```

Ordering Result –– ORDER BY

```
SELECT name, major, gpa FROM student
 ORDER BYgpa [DESC];
```

- *ascending* [ASC] is the default ordering
- Mixed ordering is possible

```
SELECT name, major, gpa FROM student
 ORDER BY major, gpa DESC;
```

- Specifies:
  - students listed by major in alphabetical order
  - within each major, students appear in descending gpa order
  - major is called the major sort field
  - gpa is called the minor sort field

In-Class Exercise

Consider the following relations for a database that keeps track of student enrollment in courses and the books adopted for each course:

STUDENT(SSN, Name, Major, Bdate)

COURSE(Course#, Cname, Dept)

ENROLL(SSN, Course#, Quarter, Grade)

BOOK_ADOPTION(Course#, Quarter, Book_ISBN)

TEXT(Book_ISBN, Book_Title, Publisher, Author)

Draw a relational schema diagram specifying the foreign keys for this schema.

What's Next

- Software Installation
  - MariaDB – part of XAMPP installation or independent
  - HeidiSQL or SequelPro or phpMyAdmin
  - Create a user account that has Global Privileges to use MariaDB (don't forget your root login password, though)
  - Chinook Database
- Homework Assignment 1
  - Due Wednesday, September 21 by 11:59pm ET
- Quiz 1
  - Due next Friday, September 24 by 11:59pm ET
- Homework Assignment 2
  - Due next Wednesday, September 28 by 11:59pm ET
- Read Chapters 2 and 3 from the textbook
  - Anytime this week
  - Consider working some problems in the back of the chapter