Johns Hopkins Engineering

Principles of Database Systems

Module 1 / Lecture 1

Introduction to Databases

JOHNS HOPKINS
WHITING SCHOOL
of ENGINEERING

Course Introduction

- Instructor: Dr. Dar-Ning Kung
  - W: 301-827-3688; C: 240-888-4442;
  - Email: darningkung@aol.com; dnkung@yahoo.com

- Course Syllabus
  - Course Structure: Modules, Lectures, Readings, Discussion (10%), Assignments (10%), Database Project (40%) and Exams (40%)
  - Textbook
  - Office Hours via Adobe Connect and emails

Common Uses of Database Systems

- Companies and schools
- Supermarket purchases
- Bank credit card transactions
- Library borrowing activities
- Airline and hotel reservations
- PII in state and federal systems
- Online or traditional in-store shopping

What Is A Database System?

- A computerized record-keeping system essentially
- A kind of electronic filing cabinet
- A repository for a collection of computerized data files

Basic Database Operations by Its Users

- Add new, empty files to databases
- Insert new data into existing files
- Retrieve data from existing files
- Update data in existing files
- Delete data from existing files
- Remove existing files, empty or otherwise, from databases

Why Databases?

- Compactness: Reduce potentially voluminous paper files
- Speed: Perform database operations faster using computers
- Less labor: Save manpower for maintaining files
- Currency: Accurate, up-to-date information is available any time a user requests it
- Data Sharing: Access the data at the same time for multiple users (e.g., airline reservations)
- Security: Maintain access control to keep data from unauthorized access

Benefits of the Database Approach of A Multi-user System

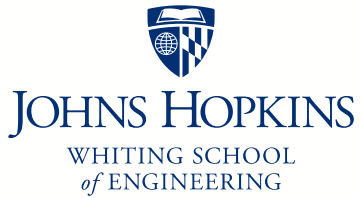
- Redundancy can be reduced
- Inconsistency can be avoided
- Data can be shared
- Standards can be enforced
- Security restrictions can be applied
- Integrity can be maintained
- Conflicting requirements can be balanced

Johns Hopkins Engineering

Principles of Database Systems

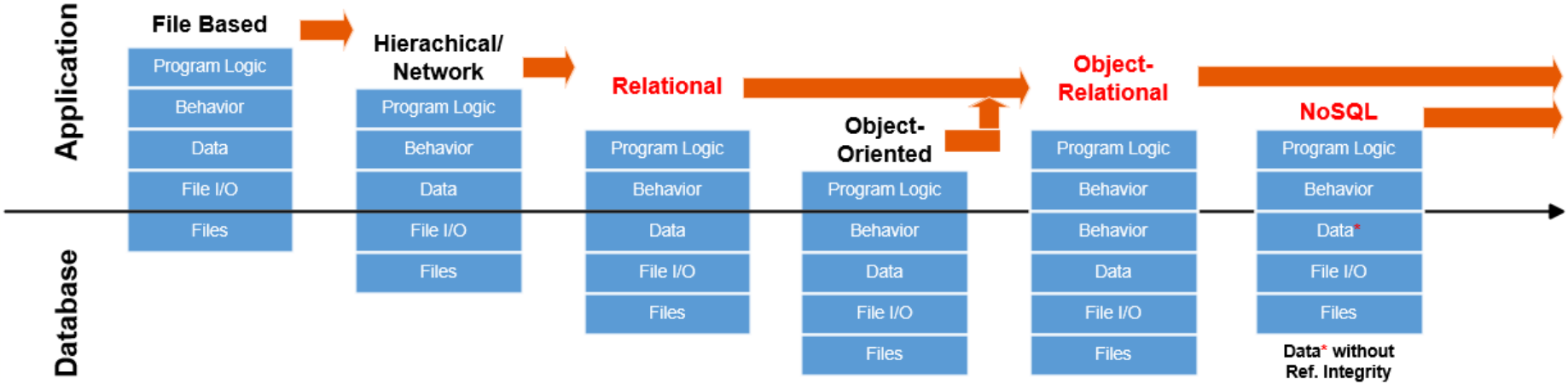
Module 1 / Lecture 2

Introduction to Databases



Database Management System (DBMS) Evolution

Database Systems
Database Applications and Databases



File Based DBMS

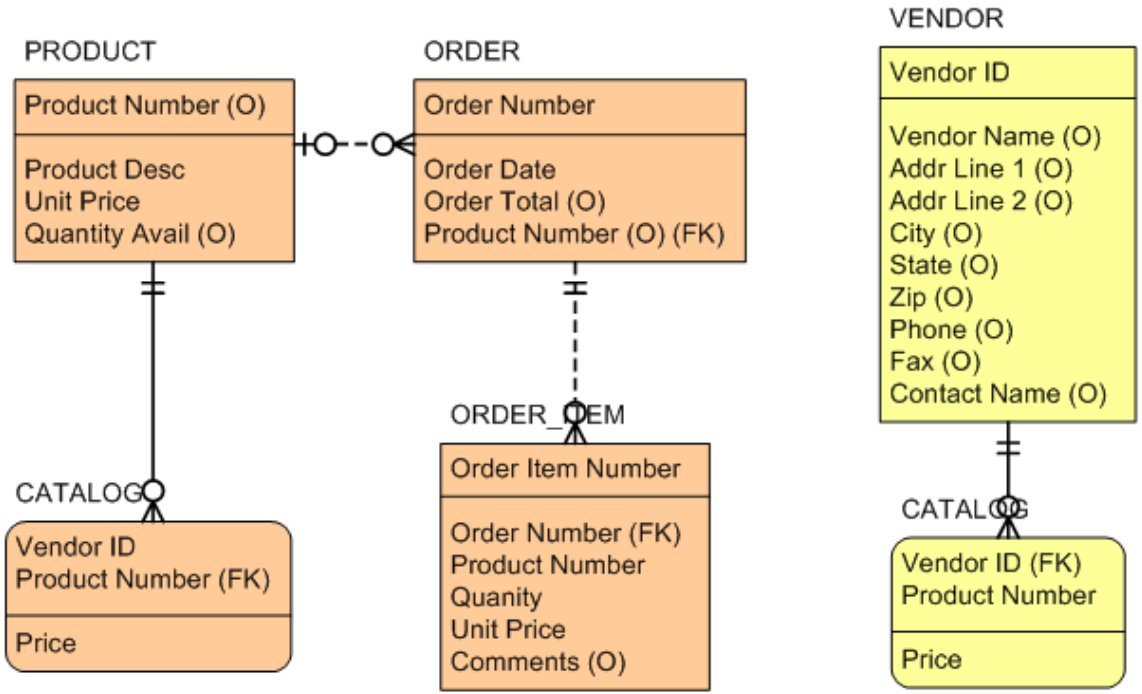
- File Based DBMS Characteristics
  - Access Method: Navigational
    - The data integrity problems due to duplication of data
    - The inability to represent logical data relationships easily
    - Program strongly depends data format (program-data dependence.)
  - Data is stored in text files or binary files

Hierarchical DBMS

- Hierarchical DBMS Characteristics
  - Access Method: Navigational (data access is only through the predefined relationships, and through the entity at the top of the hierarchy, the root, and must proceed in hierarchical order)
    - Fast access by following the predefined relationship
    - Do not provide multiple parents (or many-to-many relationships) for an entity that causes duplicated data and access restrictions
    - Do not have the ability to direct data access
    - Is not suitable for ad hoc query type applications
  - Commercial Product: IBM Information Management System (IMS), MUMPS

Hierarchical DBMS Implementation

Implementation Example

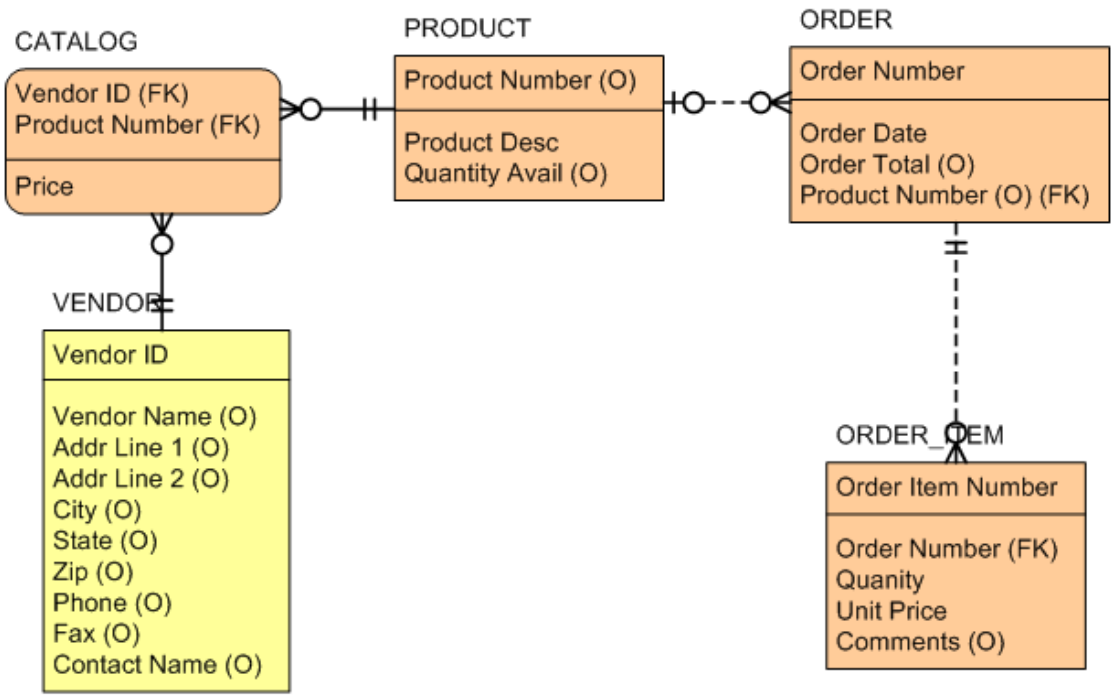


Network DBMS

- Network DBMS Characteristics
  - Access Method: Navigational
    - Design to eliminate the hierarchical database restrictions
    - Allow multiple parents for an entity
    - Fast access by following the predefined relationship
    - Allow direct access through hashing algorithm
    - Is not suitable for ad hoc query type applications
  - Commercial Product: CA-IDMS
    - Advantage CA-IDMS database is navigational and non-relational legacy sources
    - SQL is integrated in the database engine to enable SQL access to existing Advantage CA-IDMS

Network DBMS implementation

Implementation Example



Johns Hopkins Engineering

Principles of Database Systems

Module 1 / Lecture 3

Introduction to Databases

JOHNS HOPKINS
WHITING SCHOOL
of ENGINEERING

Relational DBMS

- Relational DBMS Characteristics
  - Access Method: Use primary key and foreign key for the logical relationships
    - Allow one-to-one or one-to-many relationships between entities
    - Use proper logical design with indexes, and careful query formulation to improve performance
    - Benefit from a sound theory and the SQL standard (Non-procedural, e.g., no IF-THEN-ELSE logic)
    - Vendor and platform independent
    - Provide superior ad hoc query support
    - Has been widely deployed and dominates the DBMS market

Relational DBMS (cont.)

- Relational DBMS Characteristics
  - Non-navigational: No need to follow chains or pointers. User specifies only "what", not "how". DBMS "optimizer" determines access path.
  - Properties: Atomicity, Consistency, Isolation, Durability (ACID)
  - Commercial/Open Source Products: IBM DB2 UDB, Microsoft SQL Server, Oracle, Sybase, MySQL, Postgre SQL, MariaDB, ...under mainframe, midrange, LAN, workstation, PC, …

  Note: Most vendors also support user-defined data types and other object-oriented functions, and their products become object-relational DBMS.

Object-Oriented DBMS

- Object-Oriented DBMS (ODBMS or OODBMS) Characteristics
  - Access Method: Navigational using object ids
    - Allow data and procedures to be stored together
    - Reuse the self-contained entities
    - Access data through relationships stored within the data themselves
    - Support complex objects, classes, methods, inheritance, and encapsulation
    - Is not suited for ad hoc querying as relational databases are
  - Commercial Products: ObjectDB, ObjectStore, Versant

Object-Relational DBMS

- Object-Relational DBMS (ORDBMS) Characteristics
  - Access Method: Using both key index and object ids (navigational)
    - Inherit all strengths from the RDBMS
    - Allow data and procedures to be stored together
    - Reuse the self-contained entities
    - Support complex objects, classes, methods, inheritance, and encapsulation
    - Support SQL (Structural Query Language) standards
  - Commercial Products: IBM DB2 UDB, Microsoft Server, and Oracle

NoSQL

- NoSQL Characteristics
  - Access Method: Navigational and non-navigational
  - Do not use fixed schema structures, referential integrity, defined joins, or a common storage model
  - Is good for scaling out horizontally, and is well suited for Big Data; while RDMBS is good for scaling up vertically
  - Follow BASE: Basically Available, Soft state, Eventual Consistency; do not follow the strict ACID rules of relational databases

NoSQL (cont.)

- NoSQL Characteristics
  - Four types NoSQL databases based on storage category:
    - Column-based Store: Cassandra, Hbase, Google - Big Table
    - Key Value-based Store: Amazon DynamoDB, Riak, Berkeley DB
    - Document-based Store: MongoDB, Couchbase, CouchDB, MarkLogic
    - Graph-based Store: Neo4j, OrientDB
  - Usage:
    - Big Data, schema-less design, better horizontal scalability, not mission critical applications
  - NoSQL does not replace SQL databases; it is a complementary addition to RDMBS and SQL to support business needs

Johns Hopkins Engineering

Principles of Database Systems

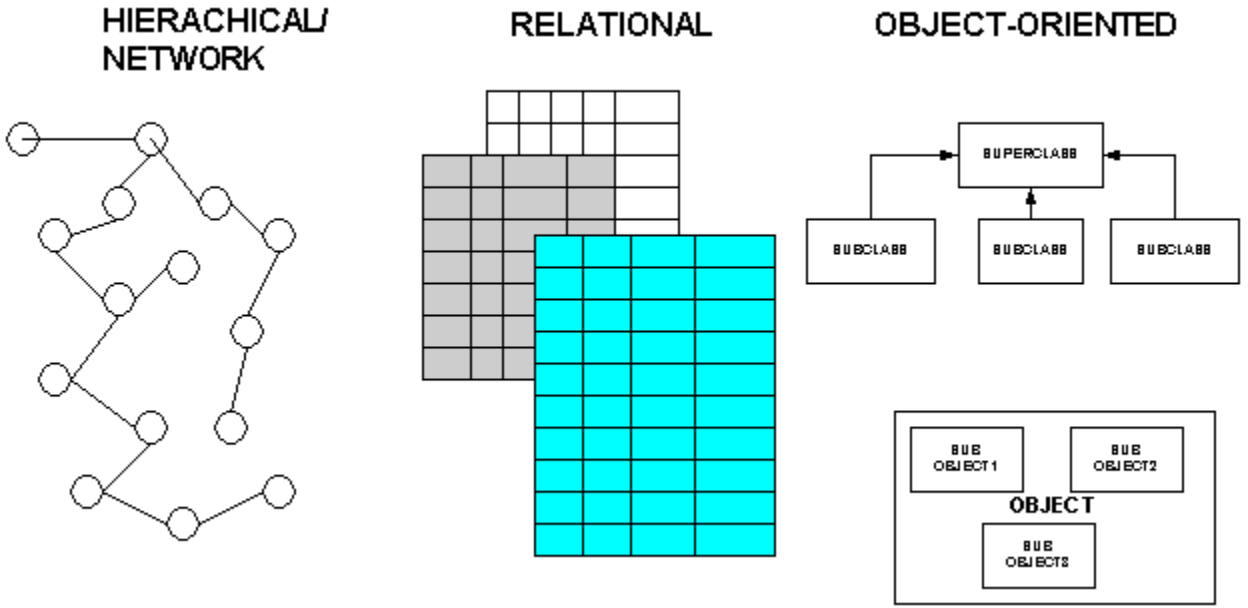
Module 1 / Lecture 4

Introduction to Databases

JOHNS HOPKINS
WHITING SCHOOL
of ENGINEERING

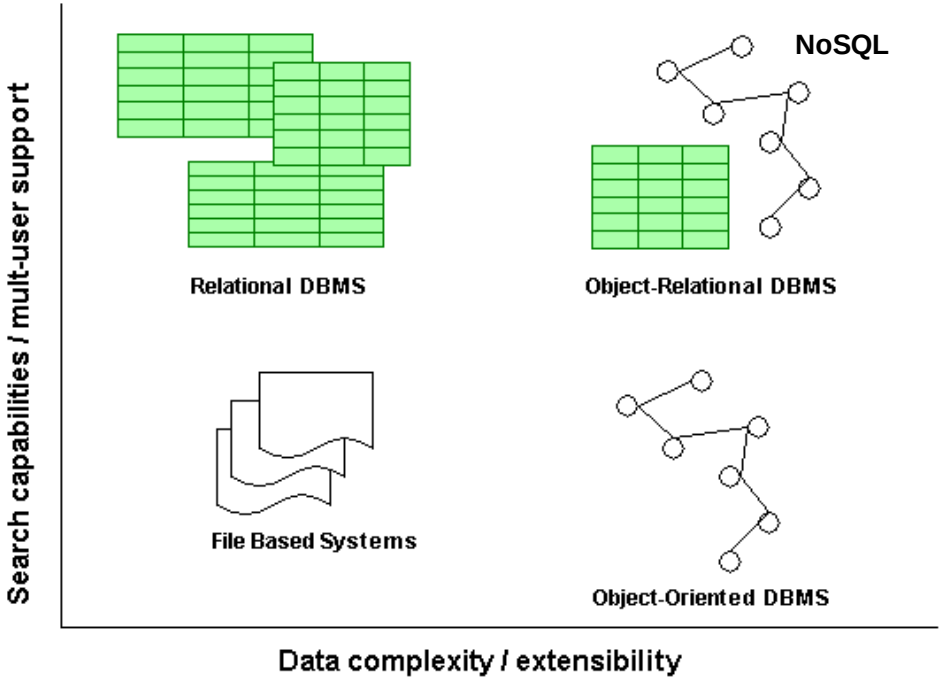
Graphical Representations for Various Data Models

Implementation Example



Data Complexity / Search Capabilities for Various Data Models

Implementation Example



Support of Multiple Views of Data

- Provide a means to present a different representation of data that resides within the base tables
- Are powerful because they allow the database designer or application developer to tailor the presentation of data to different types of users

Sharing of Data and Multi-user

- Transaction process
  - A logical unit of work on the database
  - An action or actions that perform database record reads or updates by a user or program

- Concurrency control
  - Managing simultaneous operations on the database without having them interface with one another
  - Without proper control, the updated data may be lost, cause inconsistent analysis, or cause uncommitted dependency problems

Types of Database Users

- Database Administrators (DBA)
  - Install and upgrade the DBMS on database server and application tools
  - Allocate system storage and plan for future storage requirements for the database system
  - Create primary database storage structures (tablespaces) once application developers have designed an application
  - Create primary objects (tables, views, indexes) once application developers have designed an application

Types of Database Users (cont.)

- Database Administrators (DBA)
  - Modify the database structure, as necessary, from information given by application developers
  - Enroll users and maintain system security
  - Control and monitor user access to the database
  - Monitor and optimize the performance of the database
  - Plan for back-up and recovery of database information
  - Back-up and restore the database
  - Create redundancy or replicate database instances

Types of Database Users (cont.)

- System Analysts / Database Designers / Application Developers
  - Design the database structure for an application
  - Design and develop the database application
  - Estimate storage requirements for an application
  - Specify modifications of the database structure for an application

Types of Database Users (cont.)

- System Analysts / Database Designers / Application Developers
  - Relay the above information to a database administrator
  - Tune the application during development
  - Establish an application's security measures during development
  - Remediate application vulnerabilities reported by static or dynamic scans

Types of Database Users (cont.)

- Database Users (ad-hoc, regular, or power users)
  - Enter, modify, and delete data, where permitted
  - Generate reports of data
  - Provide input of new features and enhancements for an application
  - Test new release of a database application and report issues

A DBMS Simplified Environment

- A simplified database system environment
  - Application, DBMS, Database Definition (Meta Data), Database
- Program-Data Independence
  - Programs to form an application do not depend on the data storage structure or access technique.

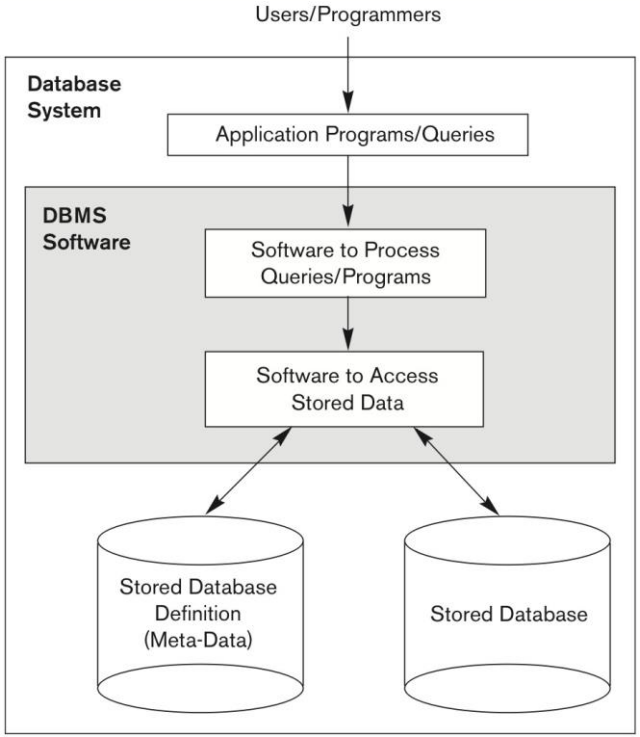


Figure 1.1 A simplified database system environment.

Data Abstraction

- Program-data independence and program-operation (function) independence.
- DBMS allows a conceptual representation of data.
- Database can grow without considering the applications.

Johns Hopkins Engineering

Principles of Database Systems

Module 1 / Lecture 5

Introduction to Databases

JOHNS HOPKINS
WHITING SCHOOL
of ENGINEERING

Data Models, Schemas, and Instances

- A data model is a set of concepts that can be used to describe the structure of a database including data types, relationships, and constraints of the data.

- The description of a database is called the database schema (meta-data). It is a collection of schema objects, such as tables, views, clusters, procedures, and packages used by a database.

Data Models, Schemas, and Instances (cont.)

- The data in the database at a particular moment is called database state (or set of occurrences or instances.) For example, STUDENT or ORDER is an instance construct. They can have multiple occurrences or instances.

- However, an Oracle instance, as a database engine, represents a set of background processes and allocated memory structure that are shared by all users to access the data in the database.

Data Models, Schemas, and Instances (cont.)

- An Oracle *database* is a collection of data that is treated as a unit. The purpose of a database is to store and retrieve related information.

- The database has *logical structures* and *physical structures*. Because the physical and logical structures are separate, the physical storage of data can be managed without affecting the access to logical storage structures.

Database Architecture - Three-schema Architecture

- External level
  - Include a number of external schemas or user views
- Conceptual level
  - Has a conceptual schema, which describes the structure of the entire database for a community of users
- Internal level
  - Has an internal schema, which describes the physical storage structure (data storage, and access path) of the database

Database Architecture - Three-schema Architecture (cont.)

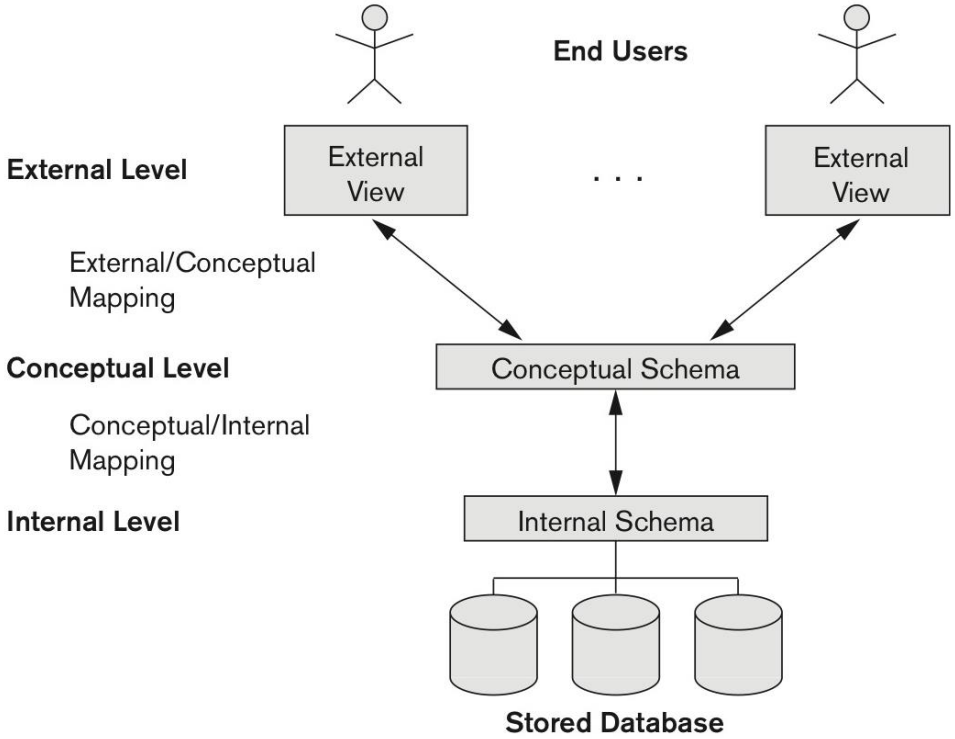


Figure 2.2 The three-schema architecture.

A DBMS Components And Their Interactions

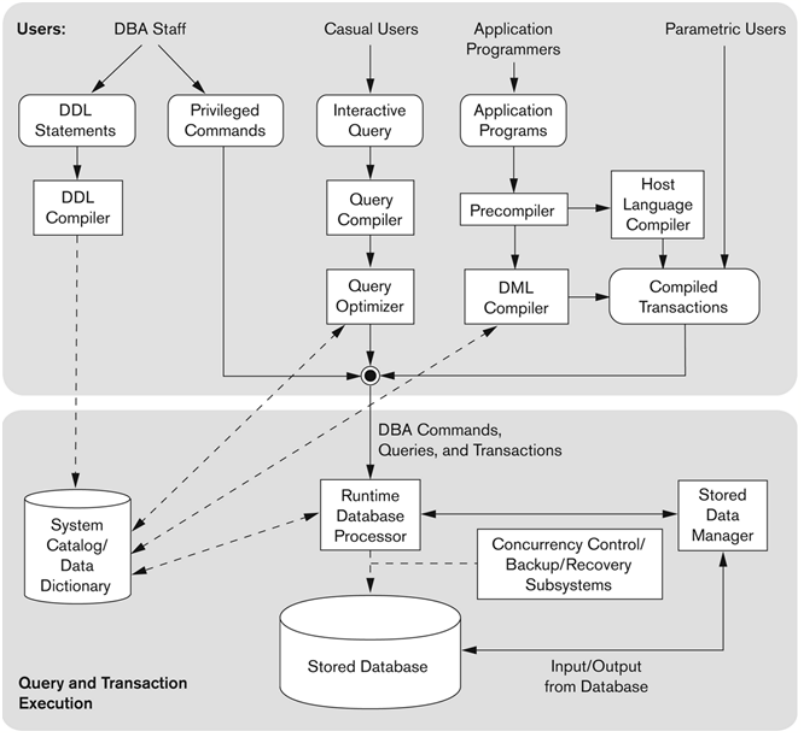
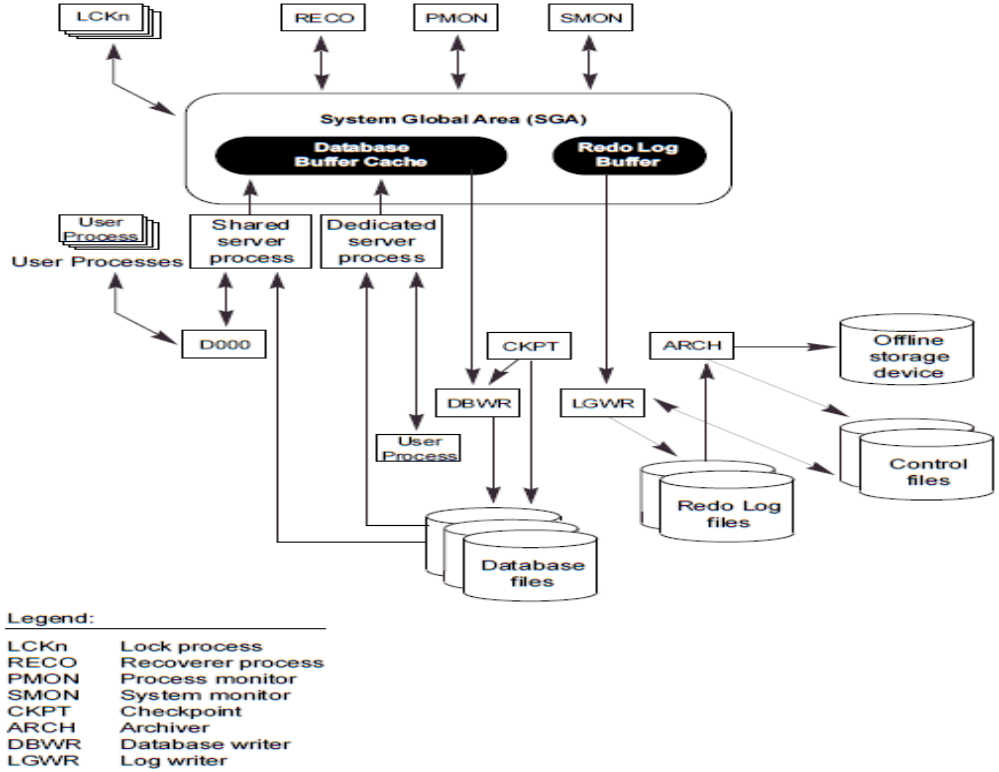


Figure 2.3 Component modules of a DBMS and their interactions.

Oracle RDBMS Architecture - An Instance



Source : Oracle

Database Languages

- Data definition language (DDL)
  - After a DBMS is chosen and the design of a database is completed, DDL is used to create a database. For RDBMS, basic commands include CREATE, ALTER, DROP, and RENAME.

- Data manipulation language (DML)
  - DML is used to perform basic operations on a database such as retrieve, insert, update, delete data; commit or rollback changes of data. These commands can be embedded as data sublanguage (DSL) into a host language.

Database Languages (cont.)

- Date query language (DQL)
  - A query language only involves data retrieval. Users may need complex queries with set operations or aggregate functions for reporting.
  - The database state remains the same.

- Data control language (DCL)
  - DCL provides user access control to a database. Basic commands are GRANT, REVOKE, AUDIT, and LOCK.

Database Interfaces

- Create forms-based applications that provide users access to information stored in a database
- Create applications with graphical user interface (GUI) environments
- Increase productivity for users by creating menu driven and mouse driven applications that do not require users to remember function keys, or understand commands and programming

Johns Hopkins Engineering

Principles of Database Systems

Module 1 / Lecture 6

Introduction to Databases

JOHNS HOPKINS
WHITING SCHOOL
of ENGINEERING

Centralized and Client/Server Architectures for DBMSs

- Centralized
  - Terminals and a DBMS server
- Basic two-tier client/server architecture
  - Clients and servers
  - Clients running programs interact with DBMS
- Three-tier client/server architecture
  - Client – GUI, Web Interfaces
  - Application server and/or Web server
  - Database server with DBMS

Classification of DBMSs

- Based on data model or DBMS implementation:
  - Hierarchical, network, relational, object-oriented and object-relational databases, NoSQL databases (based on column-based, key-value, document-based, and graph-based)
- Based on number of users:
  - Single-user, multi-user
- Based on number of sites:
  - Centralized, distributed

Classification of DBMSs (cont.)

- Based on purpose:
  - General – On-Line Transaction Processing (OLTP)
  - Data warehouse – On-Line Analytical processing (OLAP) for decision support or data mining
  - Special – XML DB, GIS, NoSQL, and others

DBMS Market

- Operational DBMSs – Oracle, Microsoft, IBM, and SAP as leaders
  - Dominate RDBMS and ORDBMS market shares
- Open-source relational DBMSs have matured significantly and can be used to replace commercial RDBMSs
  - Can be chosen for a substantial savings in Total Cost of Ownership (TCO)
  - MySQL, MariaDB, PostgreSQL and others

Continue Growth in the DBMS Market

- Maintain or increase levels of DBMS spending for most organizations

- Look for cost-savings for better business competitiveness
  - Increase the use of “open-source” RDBMSs
  - May use for mission-critical and internal applications
  - May have few functions, DB admin tools, DBA resources – can serve the needed features
  - Linux has become a leading DBMS platform
  - Oracle MySQL and Postgres SQL

Big Data

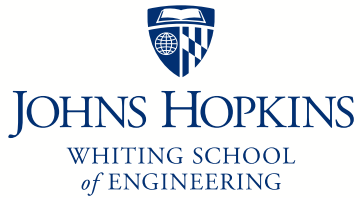
- Is a large and complex collection of data sets
  - Data characteristics/dimensions: volume, velocity, variety, and complexity
  - Data sources
    - Sensor/machine/device data (Internet of Things)
    - Unstructured content from email, office documents, etc.
    - Large audio/video/photographic data
    - Scientific/genomic data
  - NoSQL Databases and Big Data (See Ch 24 and Ch 25)

Johns Hopkins Engineering

Principles of Database Systems

Module #2

Conceptual Database Design I

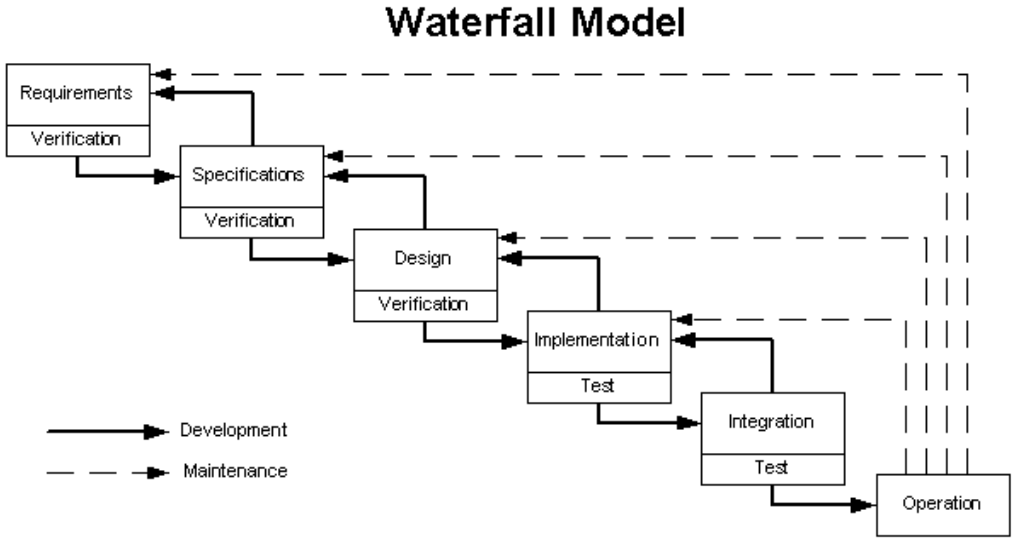


Software Engineering

- SW engineering is a discipline that focuses on production of quality software, delivered on time, within budget, that meets requirements, and satisfies clients' expectations.
- Software lifecycle includes requirements gathering, specification, design, implementation, integration, and maintenance phases.
- A database design and processing approach follows the phases of a software life cycle.

Software Lifecycle Model

- The series of steps through the product progresses
  - Waterfall model was widely used



The waterfall model with its feedback loops allows for revisions of all other stages of the the process.

Software Lifecycle Model (cont.)

- Waterfall works well when
  - Requirements are stable:
    - Customers know what they want and customers will commit
  - Technology/methodology is well understood
  - Problem may be complex but well-understood

- Other lifecycle models
  - Agile, Rapid Prototyping Model, Incremental Model, Spiral Model, Staged Delivery Model, Design-to-Schedule Model

Project Management

- Project process groups
  - Initiating processes – define and authorize the project
  - Planning processes – define objectives, refine and plan the actions required to attain them
  - Executing processes – integrate resources to carry out the plan
  - Monitoring and controlling processes – measure progress to identify variance, and take corrective action if necessary
  - Closing processes – formally and orderly close the project with formal acceptance

Project Management (cont.)

- There are 9 Knowledge Areas:
  - Project management integration, scope, time, cost, quality, human resource, communication, risk, and procurement.
- Each area has multiple processes
  - Scope planning, definition, creating WBS, verification, and control.
- Project scope defines requirements
- A process has input, tools and techniques, and output.

(See the Project Management Institute (PMI) – Project Management Body of Knowledge (PMBOK® Guide))

System Lifecycle Model

- Database development is integrated into a system life cycle.

System Operations Concept Definition	System Requirements Definition	System Design	System Implementation	System Integration and Test	System Installation and Test	System Operations and Maintenance
Data System Concept Definition	Data System Requirements Definition	Data System Design	Database Implementation	Database Integration and Test	Database Installation and Test	Database Operations and Maintenance

Subsystem Requirements Definition	Subsystem Design	Subsystem Implementation	Subsystem Integration and Test
Data System Requirements Definition	Data System Subsystem Design	Subsystem Database Implementation	Subsystem Database Integration and Test

Johns Hopkins Engineering

Principles of Database Systems

Module 2 / Lecture 2

Conceptual Database Design I

JOHNS HOPKINS
WHITING SCHOOL
of ENGINEERING

Phases of Database Design

- Requirement collection and analysis.
- Conceptual database design to create conceptual schema, which is DBMS independent, and in a high-level data model.
- Logical database design to implement the database into a DBMS. This step is DBMS dependent.
- Physical database design to address the internal storage structure and file organization. This step is also DBMS dependent.

Sources of Unplanned and Planned Downtime

- Understand downtime sources

SOURCES OF UNPLANNED DOWNTIME

Percentile	Source
40%	Application Failure
40%	Operator Error
20%	Environmental factors such as hardware, operating system, power, disasters

SOURCES OF PLANNED DOWNTIME

Percentile	Source
65%	Application and database
13%	Hardware, networks, operating systems, systems software
10%	Batch application processing
10%	Backup and recovery
2%	Physical plant / environmentals

Source: Gartner research

Cost of Fixing or Tuning A Database System

- Two-thirds of the total software costs are devoted to maintenance.
- Cost vs. Time (design, development, production)
- Benefit vs. Time (design, development, production)

Sound database design is the foundation for a successful database project

The Main Phases Of Database Design

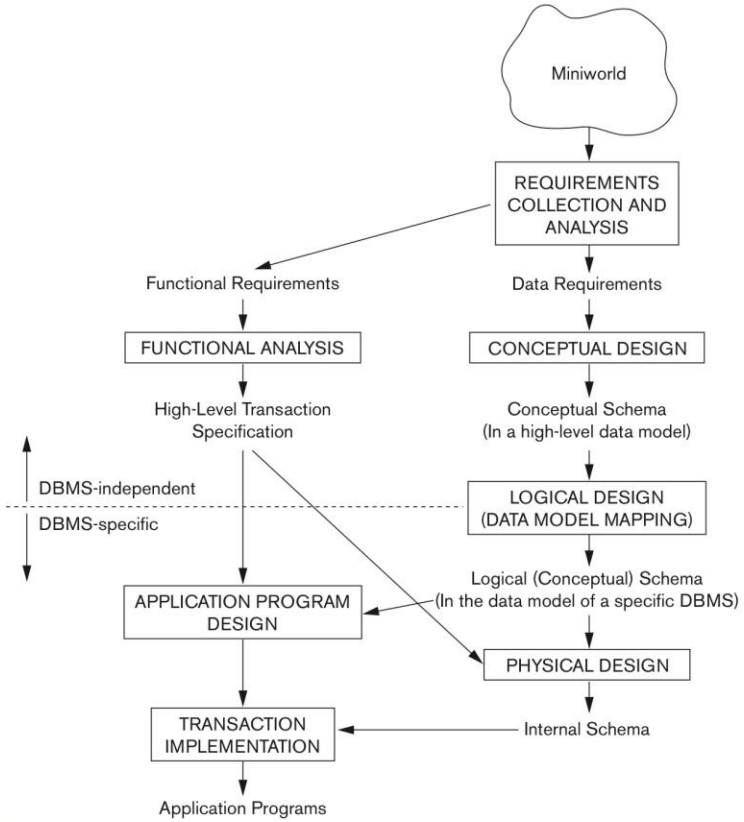


Figure 3.1 A simplified diagram to illustrate the main phases of database design.

Requirements

- Contains everything from business processes and functions
- Provides basis for creating database design and database application development
- Can be Word or Excel documents
- May include use cases:
  - Scenarios of WHO and What without concept of time

Requirements (cont.)

- May include data flow diagrams
  - Data elements are structured, but not normalized
  - May be used for data element or entity discovery
  - Lack of timing

- May include business process models (control flow)
  - Close to the business
  - Easy to understand by various stakeholders
  - Shows use of data with respect to timing
  - Describe what or who is working on a particular subset of a process

Johns Hopkins Engineering

Principles of Database Systems

Module 2 / Lecture 3

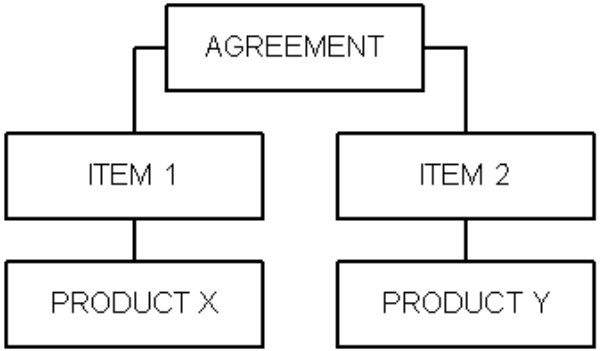
Conceptual Database Design I

JOHNS HOPKINS
WHITING SCHOOL
of ENGINEERING

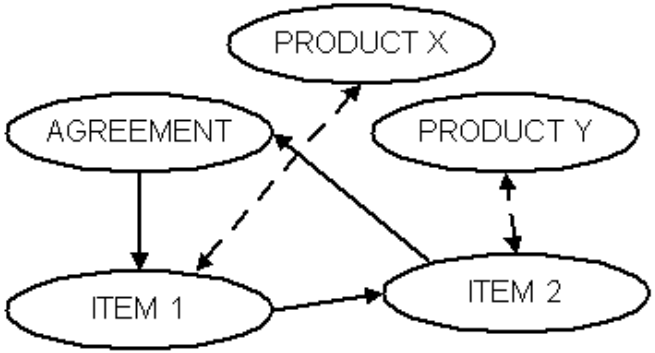
Entity-Relationship Model (ERM)

- Entity-Relationship Model or ER Model is a conceptual representation of data.
- Conceptual modeling is independent of the hardware or software to be used for implementation.
- An ER model can be mapped to a hierarchical, network, or relational database.
  - For Example: AGREEMENT to PRODUCT

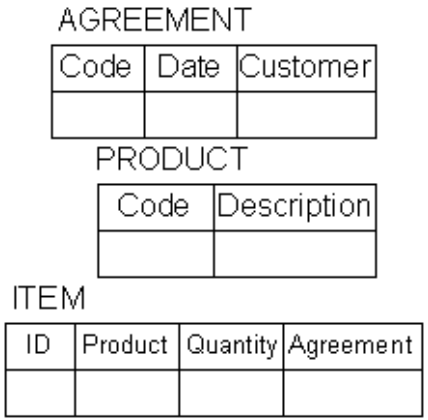
Entity-Relationship Model (ERM) (cont.)



Hierarchical database



Network database



Relational database

Conceptual Data Modeling

- A business process is a sequences of steps, tasks, or activities that converts inputs to outputs among business entities.
- Data modeling is to logically organize the governed data and its relationships, based on the business rules identified through analyzing the business process.
- The goal is to develop an entity-relationship (E-R) model that represents the information requirements of the business.

ER Model Components

- Entity
  - Important information needs to be held in the user environment within an organization.
  - It can be an object(s) of interest to the business.
  - It is a class or category of things such as a person, place, object, and event.
    - Person – EMPLOYEE, STUDENT
    - Place – STORE, STATE, HOUSE, DEPARTMENT
    - Object – VEHICLE, PROJECT, ORDER
  - Entities can be thought of as nouns. Each entity has attributes to describe its properties.

ER Model Components (cont.)

- Entity Type
  - A collection of entity instances that share common characteristics or properties
  - Name of an entity type is singular
  - An attribute, or set of attributes that uniquely identifies an entity is called a Unique Identifier (UID)

- An Entity Instance
  - Is a single occurrence of an entity type
  - Is a specific thing
  - Each instance must be uniquely identifiable from other instances of the same entity type.
    - For example: an employee John Smith with SSN or an employee ID, a vehicle with VIN as ‘1FALP52S6TA110981’

ER Model Components (cont.)

- Entity: An object or concept with attributes, that is important to the business
- Entity Type: A collection of entities that share common characteristics or properties
- Entity Instance: A single occurrence of an entity type

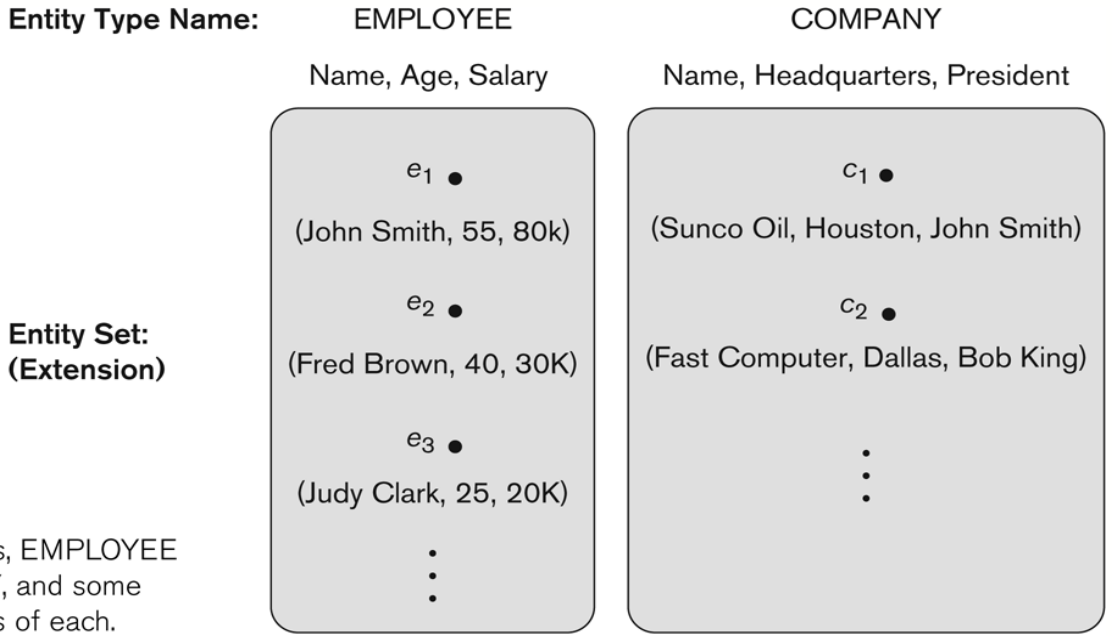


Figure 3.6
Two entity types, EMPLOYEE and COMPANY, and some member entities of each.

ER Model Components (cont.)

- Attribute
  - A property or characteristic of an entity type
  - The specific information that needs to be held. They describe an entity by qualifying, identifying, classifying, quantifying, or expressing the state of an entity instance.
    - For example, an employee's NAME, SSN, DOB, and SALARY.
  - An attribute represents a type of description or detail, not an instance.
    - For example, Mary is the first name of an EMPLOYEE instance, ZIP Code is a part of ADDRESS.

ER Model Components (cont.)

- Simple Attribute
  - An attribute that cannot be broken down into smaller components
    - For example, VIN, weight, or horsepower of a vehicle

- Composite Attribute
  - An attribute that can be broken down into component parts
    - For example, an address of a CUSTOMER entity type

Street Line 1 = street # + street name + suite/apartment # or PO box #
Street Line 2 = building # or name + other address qualifier
City, State, and Zip code

ER Model Components (cont.)

- Multivalued Attribute
  - An attribute that may have more than one value for a given entity instance
    - For example, skill of an employee, color of a vehicle

- Derived Attribute
  - An attribute whose value can be calculated from related attribute values
    - For example, a grand total of a transaction, GPA for a student

Johns Hopkins Engineering

Principles of Database Systems

Module 2 / Lecture 4

Conceptual Database Design I

JOHNS HOPKINS
WHITING SCHOOL
of ENGINEERING

Common Practices For Creating Attributes

- Always break attributes down into their lowest meaningful components:
  - For example, the name of PERSON can be broken down into last_name, first_name

- Break down aggregate attributes, composite attributes, and embedded code fields into simple attributes:
  - Address can be decomposed into multiple fields: street, apartment/suite, city, state, and zip code
  - In general, dates, times, SSN, and zip code are not decomposed further

Common Practices For Creating Attributes (cont.)

- Verify that an attribute is not derived or calculated from the existing values of other attributes, e.g. age, total

- Artificial attributes are used often for UIDs. An artificial code is defined when the business does not have a natural attribute that uniquely identifies an entity.
  - For example: first name, last name, address, and customer_id on a CUSTOMER entity type

Strong vs. Weak Entity Types

- Strong or Independent Entity Type:
  - An entity type that exists independently of other entity types.
  - An entity type whose instances *can* be uniquely identified without determining its *relationship* to another entity.

- Weak or Dependent Entity Type:
  - An entity type whose existence depends on other entity types.
  - An entity type whose instances *cannot* be uniquely identified without determining its *relationship* to another entity.

Relationship

- How the entities are related
- It is a two-directional, significant association between two entities, or between two entity types
- Relationship can be either - *must* be (mandatory) or *may be* (optional)
  - For example, the relationship between DEPARTMENT and EMPLOYEE: Explain a reasonable relationship(s)

Identifying vs. Non-identifying Relationship

- Identifying Relationship:
  - The relationship is between a weak entity type and its owner.
  - An instance of a weak (child) entity type fully depends on an instance of a strong (parent) entity type.

- Non-identifying Relationship:
  - The relationship may be between two strong entity types.
  - An instance of a strong entity type can exist without depending on other entity type.

Relationship Cardinality

- Relationships have a property called cardinality. Cardinality statements designate "how many" instances of the parent entity are connected to "how many" instances of the child.
- Most relationships are "one to something".
- Any many-to-many relationship must be broken down into a pair of one-to-many relationships.

Relationship Cardinality (cont.)

- Cardinality specifies that an entity instance can connect to how many other entity instances in a relationship.
  - A cardinality 0 represents a may be relationship.
  - Any many-to-many relationship must be broken down into a pair of one-to-many relationships.

- Relationship Types
  - One to Many (1:M);
  - Many to Many (M:M) or (M:N);
  - One to One (1:1)

- All relationships should represent the information requirements and rules of the business.

Chen's Notation for ER Modeling

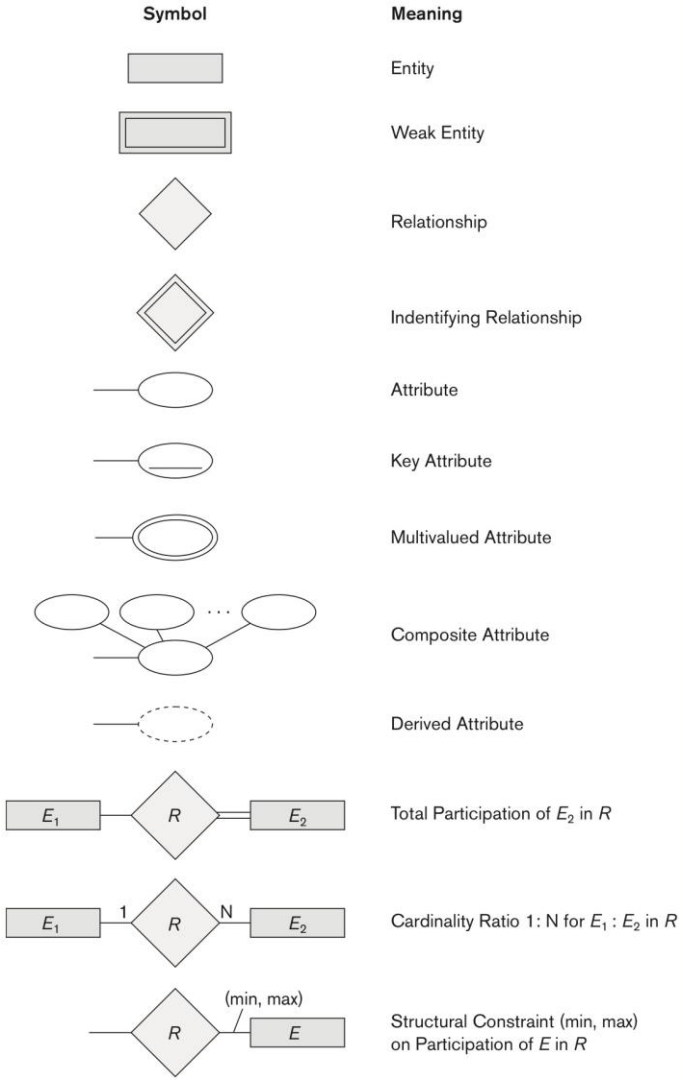


Figure 3.14 Summary of the notation for ER diagrams.

Johns Hopkins Engineering

Principles of Database Systems

Module 2 / Lecture 5

Conceptual Database Design I

JOHNS HOPKINS
WHITING SCHOOL
of ENGINEERING

Total Participation (Existence Dependence)

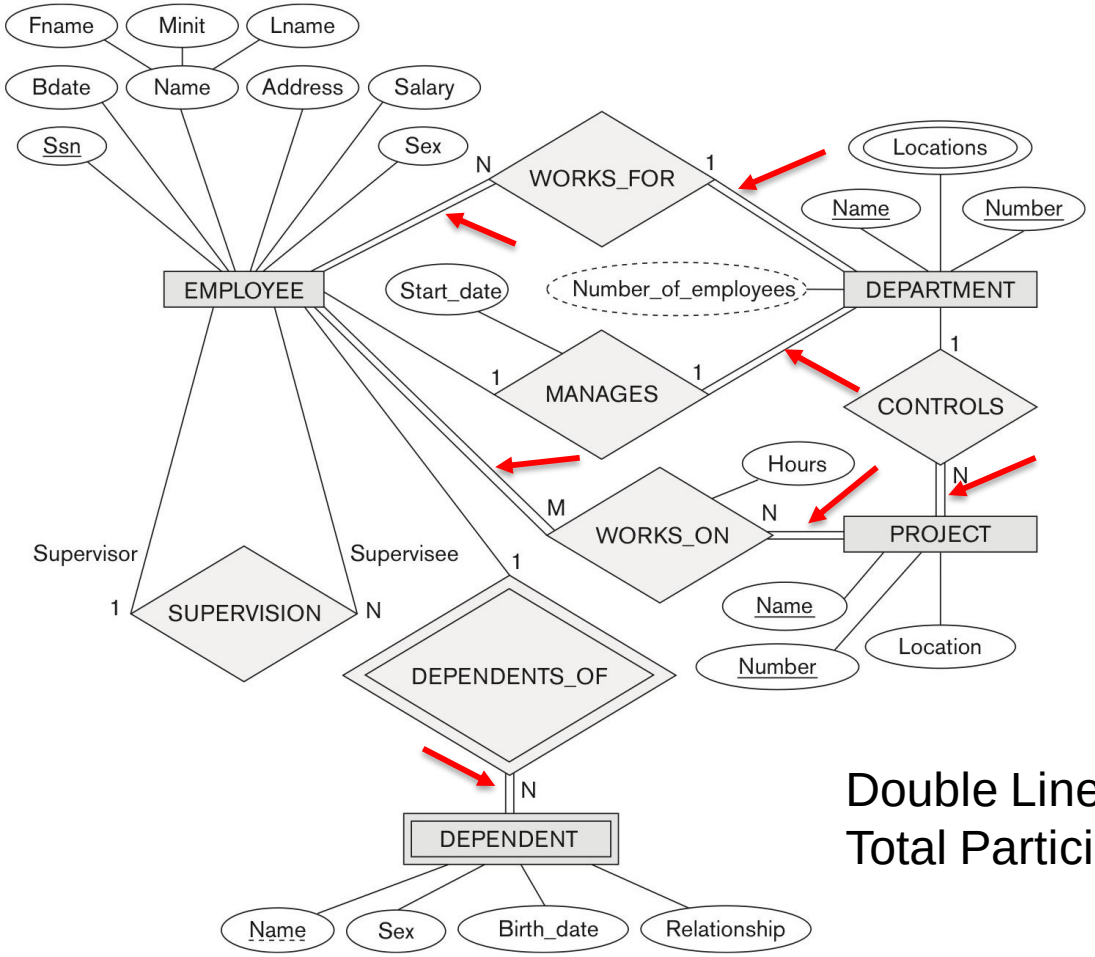
- Every entity in an entity type *must be related* to another entity type via a relationship.
- The relationship for total participation is displayed as *double lines* connecting the participating entity type to the relationship.

Figure 3.2 – An ER schema diagram for COMPANY database.

EMPLOYEE and WORK_FOR (An employee must work for a department)
DEPARTMENT and WORK_FOR (A department must have an employee(s))
DEPARTMENT and MANAGES (A department must have one manager)
EMPLOYEE and WORK_ON (An employee must work on a project)
PROJECT and WORK_ON (A project has to be assigned an employee to work)
DEPENDENT and DEPENDENTS_OF (A dependent must link to an employee)

Total Participation (cont.)

Figure 3.2 An ER schema diagram for the COMPANY database. The diagrammatic notation is introduced gradually throughout this chapter and is summarized in Figure 3.14.



Double Lines – Total Participation

Total Participation Interpretation

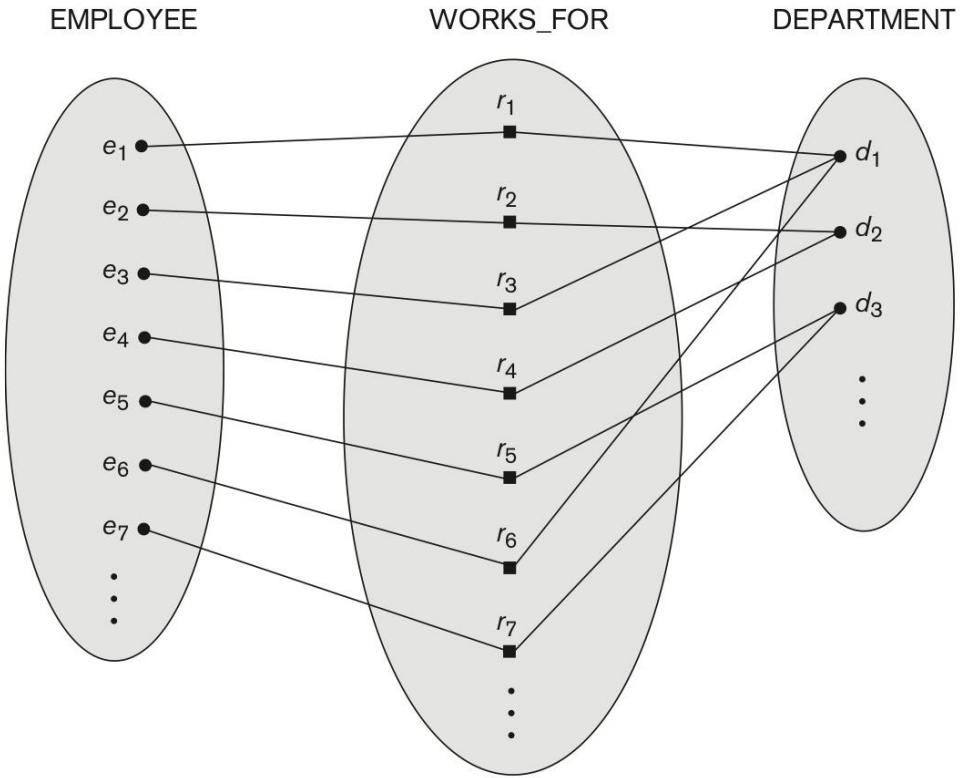


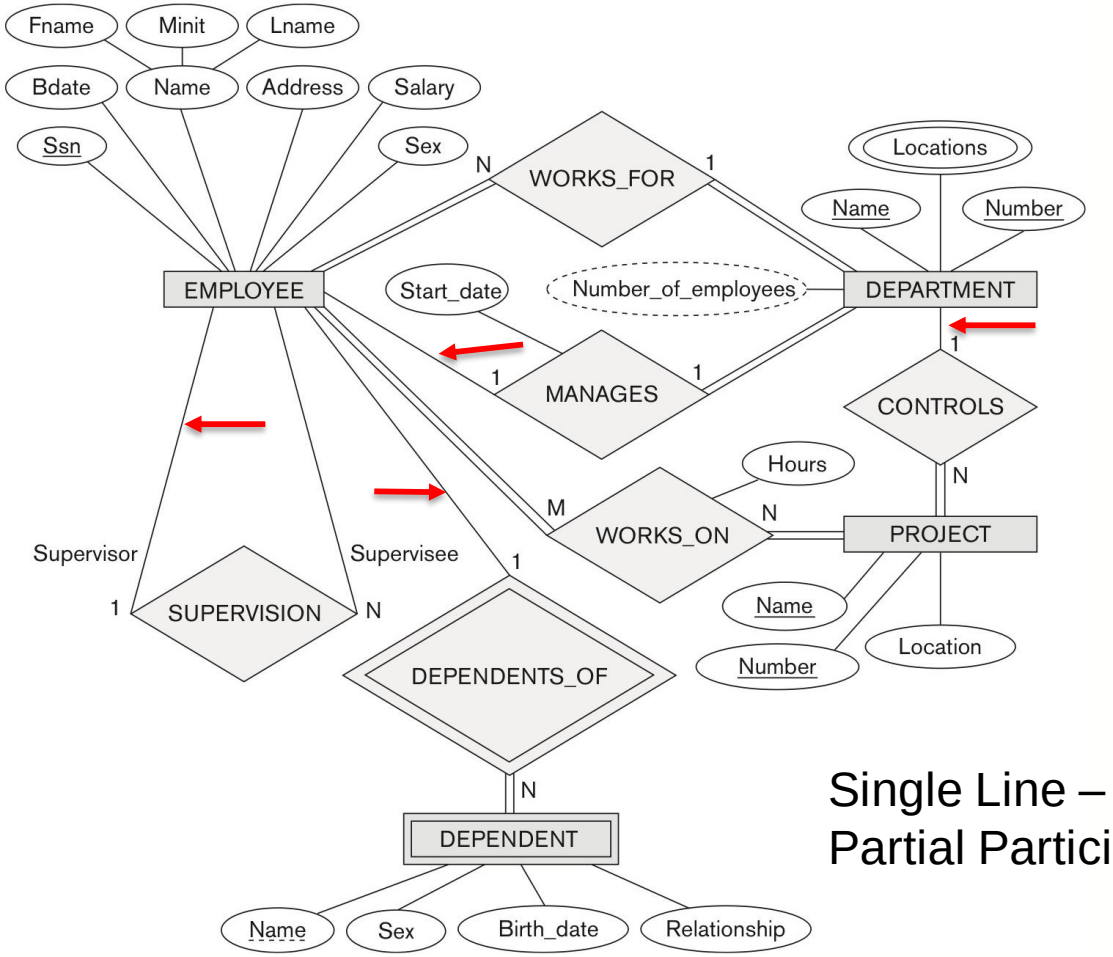
Figure 3.9 Some instances in the WORKS_FOR relationship set, which represents a relationship type WORKS_FOR between EMPLOYEE and DEPARTMENT.

Partial Participation

- If not a total participation, then relationship is a partial participation with *a single-line* connection.
  - Examples
    - EMPLOYEE and MANAGES (An employee may manage a department.)
    - EMPLOYEE and DEPENDENTS_OF (An employee may have a dependent).

Partial Participation (cont.)

Figure 3.2 An ER schema diagram for the COMPANY database. The diagrammatic notation is introduced gradually throughout this chapter and is summarized in Figure 3.14.



Single Line – Partial Participation

Partial Participation (cont.)

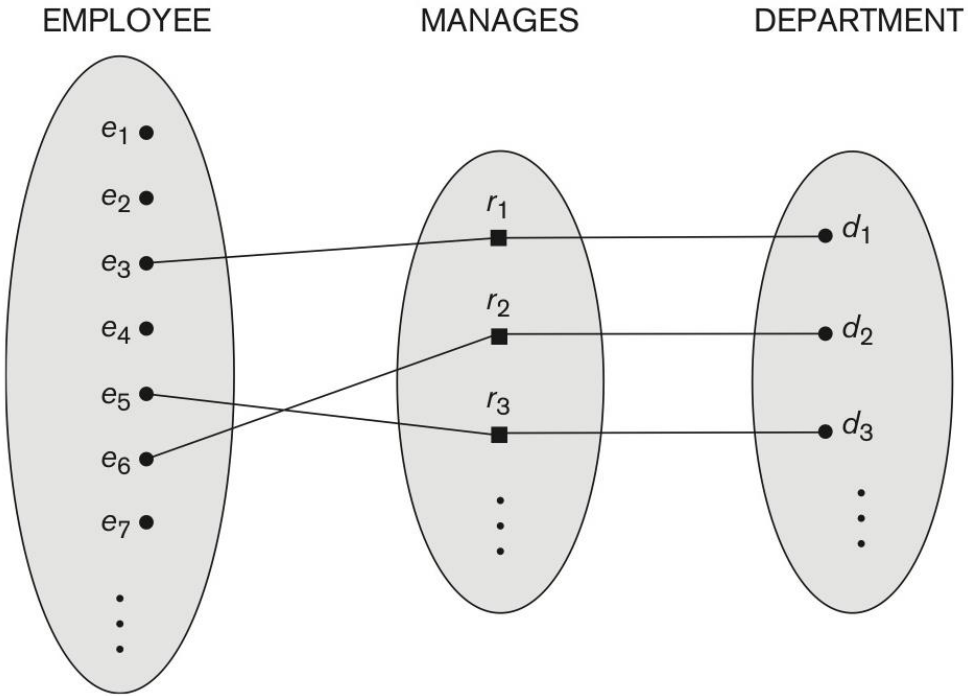
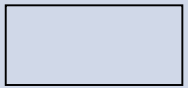
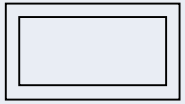
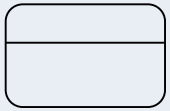
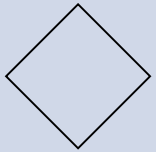
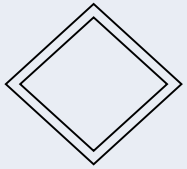
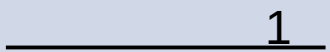
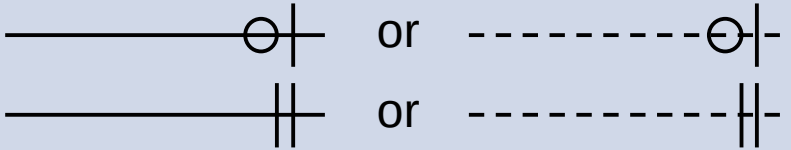
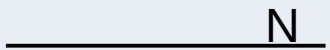
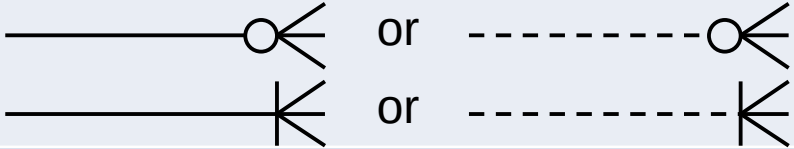
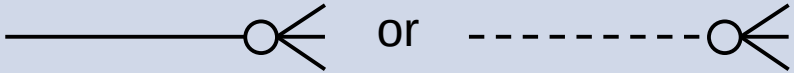
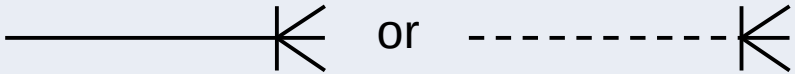


Figure 3.12 A 1:1 relationship, MANAGES, with partial participation of EMPLOYEE and total participation of DEPARTMENT.

Chen's and IE/IDEF1X Notations

	Chen	IE
Strong Entity		
Weak Entity		
Non-identifying Relationship		
Identifying Relationship		

Chen's and IE/IDEF1X Notations (cont.)

	Chen	IE
Single Cardinality		
Multiple Cardinality		
A Pair of Cardinality		
A Pair of Cardinality		

Understanding Cardinality Constraints

- Cardinality constraint:
  - Specifies the number of instances of one entity type that can (or must) be associated with each instance of another entity type.
- Minimum cardinality:
  - The minimum number of instances of one entity type that may be associated with each instance of another entity type.
- Maximum cardinality:
  - The maximum number of instances of one entity type that may be associated with each instance of another entity type.

Johns Hopkins Engineering

Principles of Database Systems

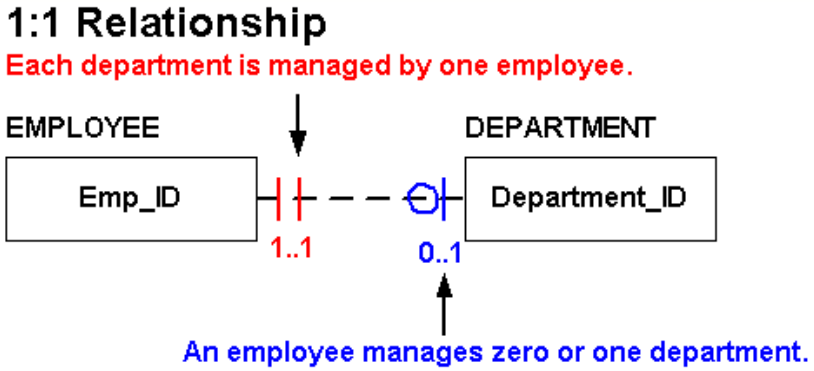
Module 2 / Lecture 6

Conceptual Database Design I

JOHNS HOPKINS
WHITING SCHOOL
of ENGINEERING

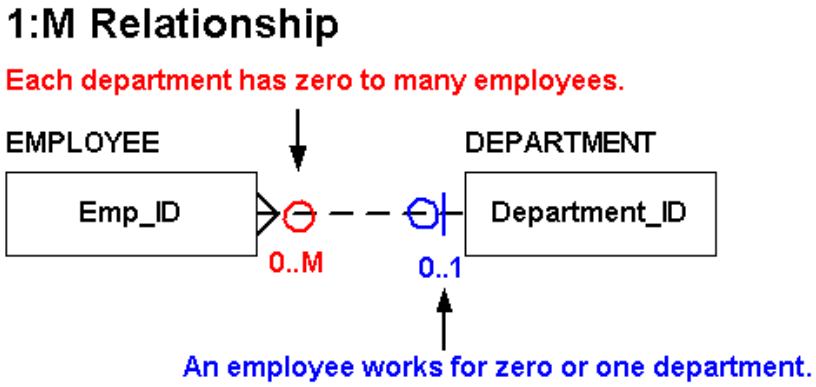
Interpreting Relationships in IE (Crow's Foot)

A pair of (1, 1) associated with a Parent EMPLOYEE



A pair of (0, 1) associated with a Child DEPARTMENT

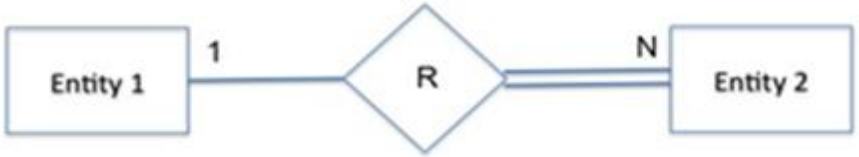
A pair of (1, M) associated with a Child EMPLOYEE



A pair of (0, 1) associated with a Parent DEPARTMENT

Interpreting Cardinality Ratios with Chen's Notation And IE Notation

- Typical parent-child entities as a 1:N relationship with "**Entity 1**" (a parent) and "**Entity 2**" (a child).



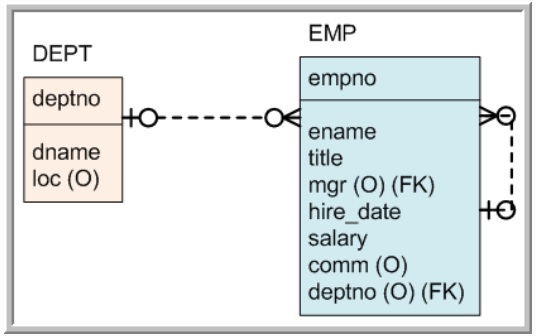
- Chen's and IE notations may not show a pair of (min, max).

- IE notation includes a pair of (min, max) cardinality ratios. How to identify/interpret (min, max) for both entities with total and partial participations:

An ERD using IE Notation - Example 1

- Entities: EMP, DEPT
- Relationship 1:
  - One department may have zero or many employees
    - For instance, a new department may not have any employees in the beginning
  - One employee may be assigned to zero or one department
    - For instance, a new employee may be not assigned to a department yet
- Relationship 2:
  - An employee may report to another employee

- A relationship connecting from DEPT (parent) to EMP (child) shows a primary key (PK) migration to a child entity as a foreign key (FK).
- A FK value in a record of EMP links to a parent record – an employee assigned to a department.



Tool: Visio

Chen's Notation with (Min, Max) Pairs of Cardinality Ratios

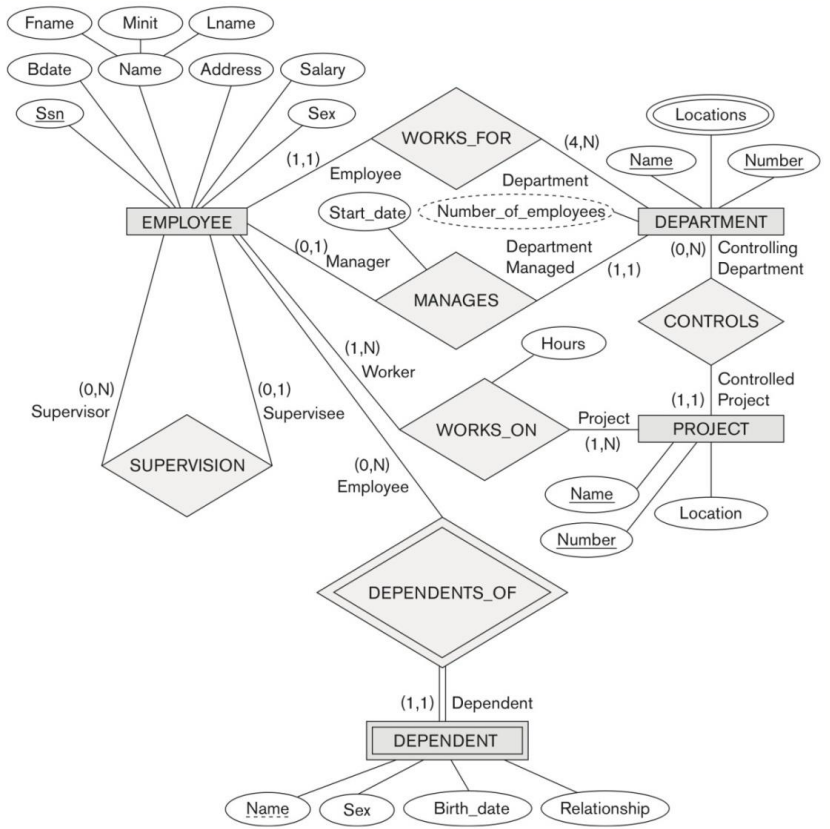
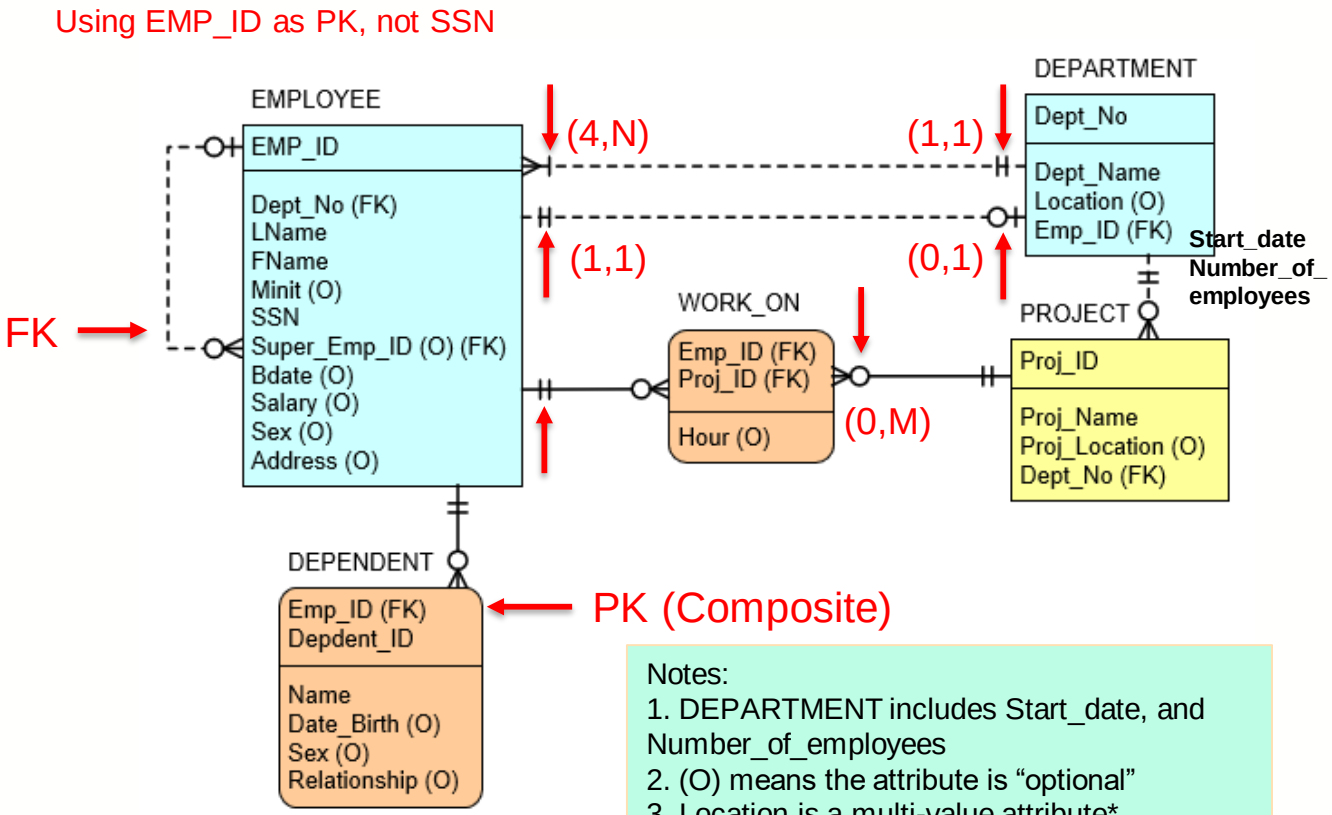


Figure 3.15 ER diagrams for the company schema, with structural constraints specified using (min, max) notation and role names.

Company ERD Using Visio with IE Notation



Please compare and contrast the Company ERD using IE notation and the Company ERD using Chen’s notation shown in Figure 3.

Chen's Notation vs. IE Notation

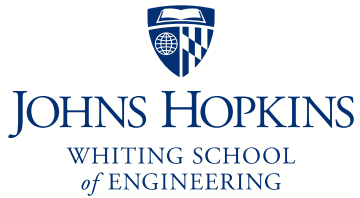
- Chen's Notation
  - May be easy to learn for a beginner.
  - May be easier to understand relationships among entities (removing attributes) at a high level.
  - Can show derived attribute and multivalued attributes.
  - An ERD begins to feel cluttered when the data model begins to scale.
  - Can draw an ERD with a graphical tool. As a result, an ERD only supports conceptual database design. Most database design tools don't support Chen's notation.
- IE Notation
  - May be is intuitive to a beginner.
  - Lead to a compact and precise ERDs that are closer to actual implementations.
  - Supports foreign key migration automatically that helps a designer and other stakeholders better understand the design and implementation.
  - Database design tools support IE. They can easily support a conceptual design to logical and physical designs and further database application development.

Johns Hopkins Engineering

Principles of Database Systems

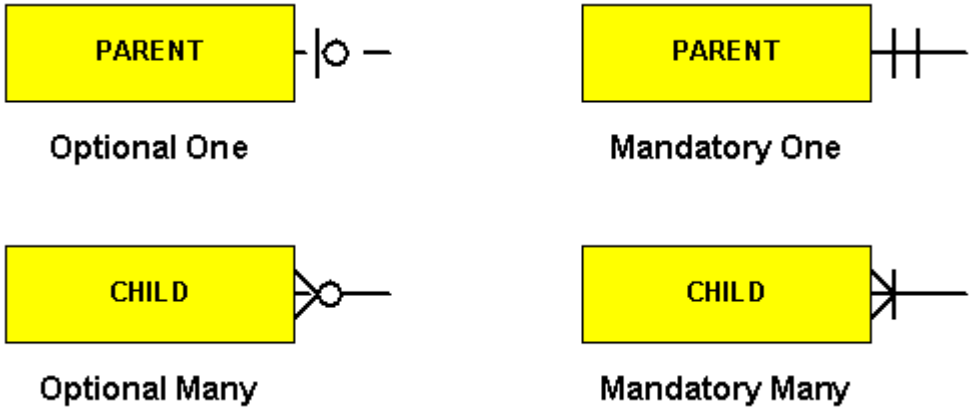
Module #3

Conceptual Database Design II



IE (Crow's Foot) Notation

- Relationship Cardinality
  - Two pairs of (min, max) associated with Parent and Child Entities may vary due to business rules.



Exercise Examples:

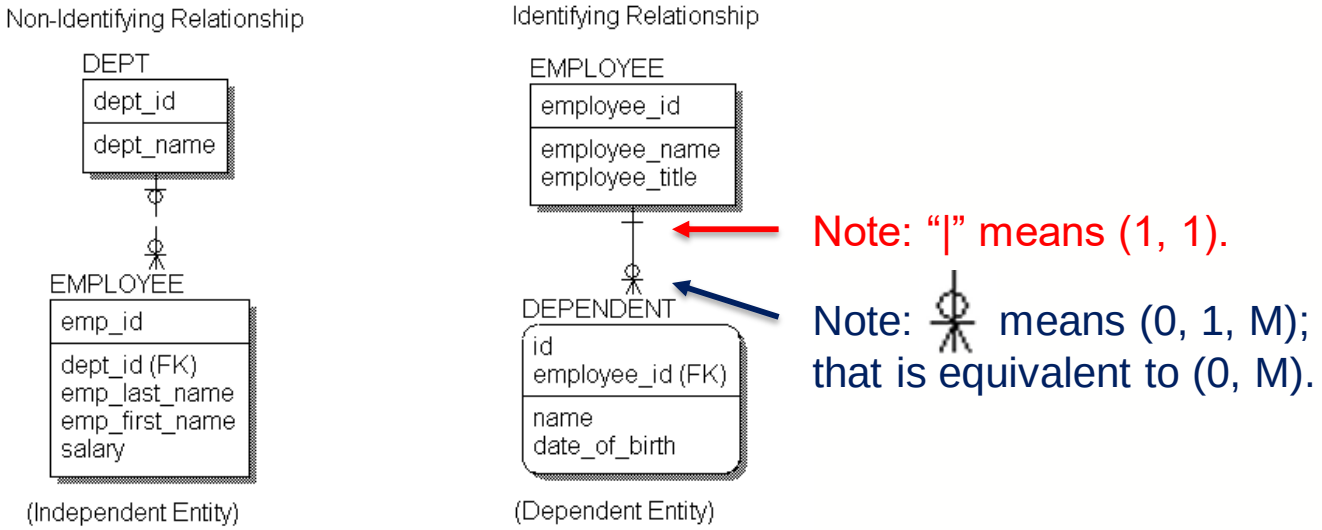
- DEPARTMENT and EMPLOYEE
- EMPLOYEE and DEPENDENT
- EMPLOYEE and PAYCHECK
- EMPLOYEE and OFFICE

Creating ERD with A Database Design Tool

- Using database design tools (e.g. MySQL Workbench, Visio)
- Conceptual database design and logical database design:
  - Create entities and attributes
    - Create an ERD with DEPARTMENT, EMPLOYEE, and DEPENDENT entities, PKs, and non-key attributes
  - Create relationships
    - Identifying and non-identifying with FK migration
  - Can do "forward engineering “ and/or "reverse engineering"
- Different tools may have different notations to draw ERDs. Read the tool documentation.

Creating 1:M with A Database Design Tool

- One to Many Relationships (IE using ERwin)

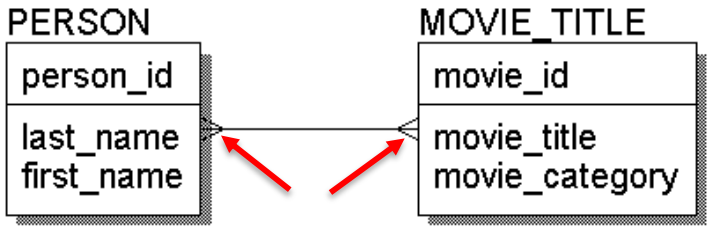


Two types of One to Many Relationships, identifying and non-identifying.

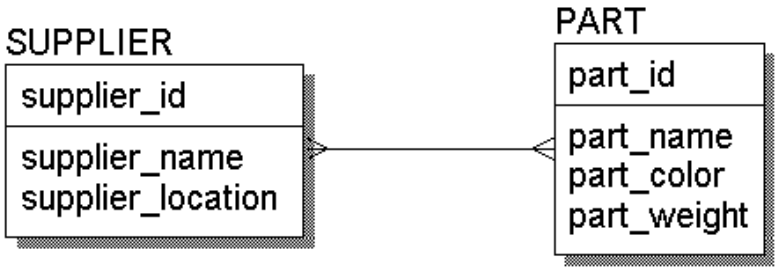
Parent to Child Rule: A DEPT contains many EMPLOYEEs.
Child to Parent Rule: An EMPLOYEE is associated with exactly one DEPT.

Creating M:N Relationship with A Database Design Tool

- Many to Many Relationships (IE using ERwin)



What's the date for "John Smith" to rent "Star Wars"?

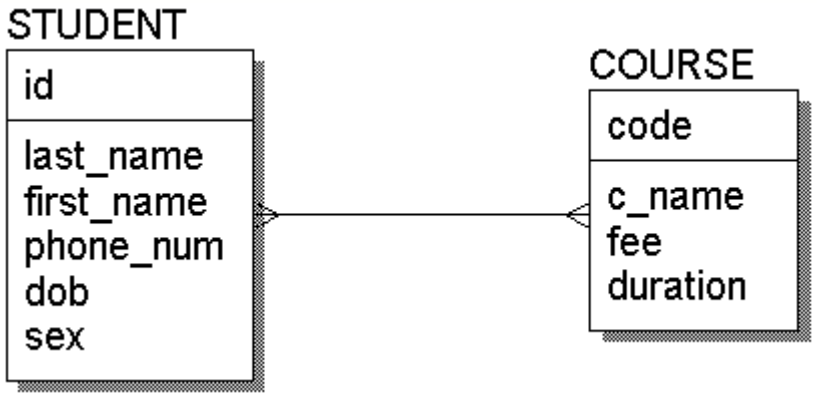


How many of "Part 3" are supplied by "Supplier 1"?

Note: A database design tool may not support many to many notation

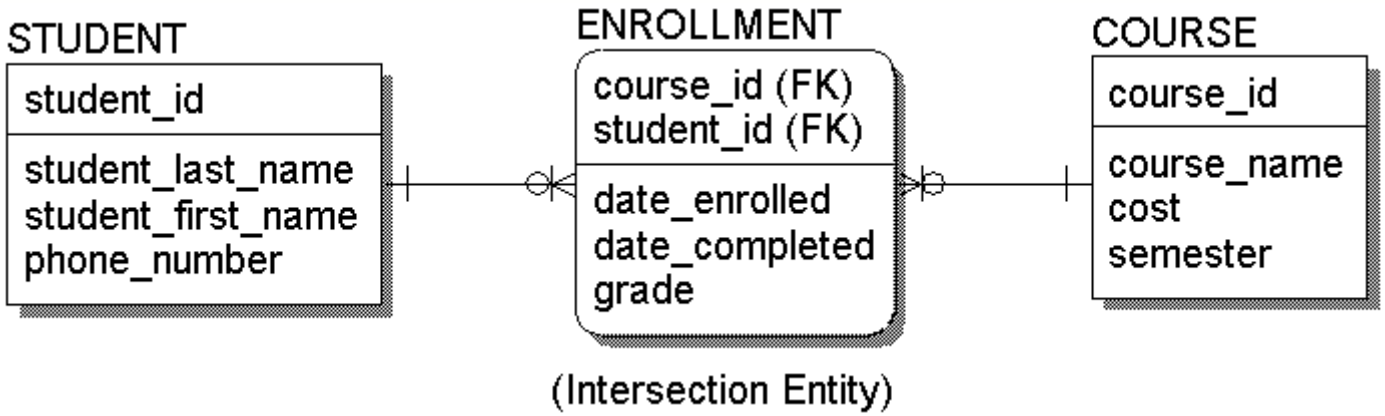
Creating M:N Relationship with A Database Design Tool (cont.)

- Attributes only describe entities
- Additional attributes are required to describe a relationship



Procedure to resolve M:M relationship

- Resolve the relationship by adding an intersection entity with two identifying relationships and attributes.



Procedure to resolve M:M relationship (cont.)

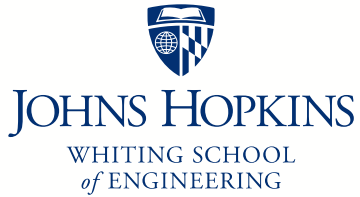
- In general, the UID of an Intersection Entity is composed of its relationships to the two originating entities.
- An Intersection Entity with no attributes is just a two-way cross-reference list between occurrences of the entities, e.g. supplier-sp-part relationship.
- The relationships from the Intersection Entity are normally mandatory.
- The relationships can be optional due to business rules, convenience, or performance.

Johns Hopkins Engineering

Principles of Database Systems

Module 3 / Lecture 2

Conceptual Database Design II



Common Approach to Develop ERD

- Identify the major entity types:
  - Identify the attributes for each entity type
  - Determine the unique identifier for each entity type

- Determine the relationships between the entity types:
  - Identifying or non-identifying (strong or week entity types)
  - Mandatory or optional
  - Cardinality

- Resolve M:N relationships (not necessary to conceptual database design)

- Refine ERD as needed

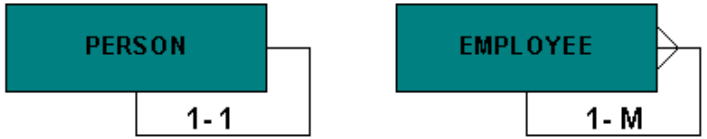
Database Design Exercise

- Identify, and analyze the entities, the relationships, and attributes in the following set of information requirements:

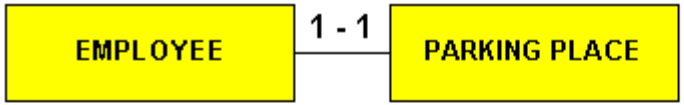
  "I'm the manager of a training company that provides instructor-led courses in management techniques. We teach many courses, each of which has a code, a name, and a fee. Popular courses are Introduction to Network Security, Introduction to UNIX, Foundations of Algorithms, Software Project Management, XML: Technology and Applications, Foundations of Computer Architecture, and Introduction to TCP/IP. Courses vary in length from one day to five days. An instructor can teach several courses. New courses may be added based on students' requests or instructors' suggestions. Each course is taught by only one instructor. We create a course and then line up an instructor. We track each instructor's name and phone number. The students can take several courses over time, and many of them do this. We track each student's name and phone number. Some of our students and instructors do not give us their phone numbers."

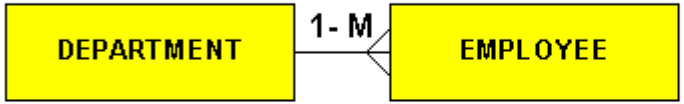
Degree of a Relationship

- The number of entity types that participate in a relationship



The unary relationships

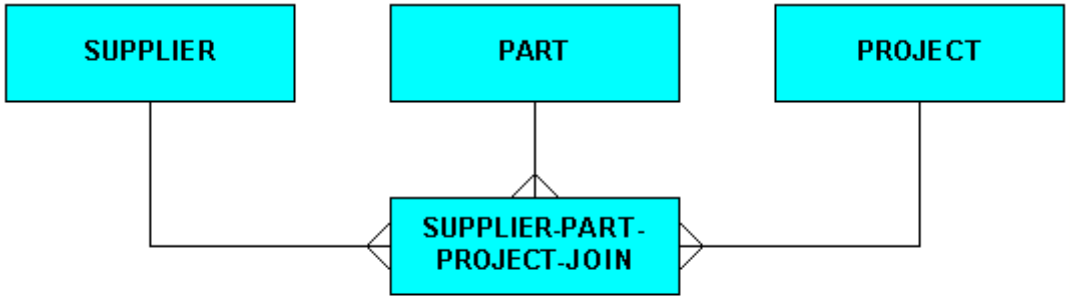




The binary relationship

Degree of a Relationship (cont.)

- The number of entity types that participate in a relationship



The ternary relationship SUPPLY

Recursive Relationship

- A Recursive Relationship is a relationship between an entity and itself. In other words, a recursive relationship is one where the parent entity and the child entity for the relationship are the same entity.
  - Persons are parents of persons; employees manage employees; companies own companies.
  - This is an 1:M recursive relationship

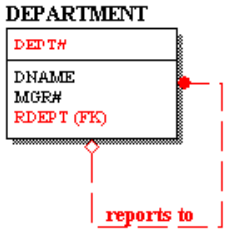
Recursive Relationship (cont.)

- When a Primary Key (PK) attribute migrates from a parent entity to a child entity, it becomes a Foreign Key (FK) attribute in the child entity. Since a FK may have a different role of the related PK, a rolename is assigned to the attribute, e.g. emp_id, supervisor_id from EMP; place of birth and address with a reference from STATE entity.

- Recursive non-identifying relationships may be either mandatory (No Nulls) or optional (Nulls Allowed). However, in practice it is very unusual for a recursive relationship to be set to "No Nulls."

Recursive Relationship (cont.)

- Examples (ERwin and MS Visio)

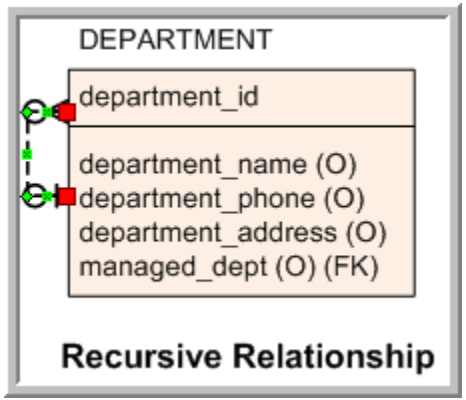


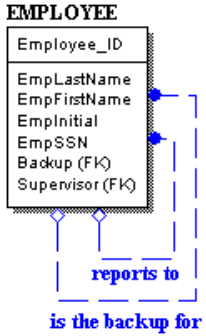
Entity-Attribute Editor Strings

Primary Key: DEPT#

Non-Key Attributes:
DNAME
MGR#
RDEPT.DEPT#

Note: RDEPT is a rolename.



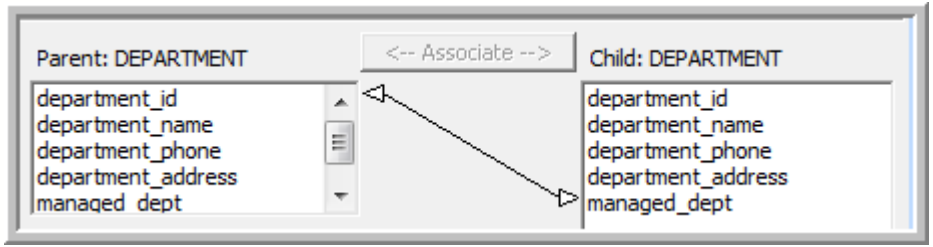


Entity-Attribute Editor Strings

Primary Key: Employee_ID

Non-Key Attributes:
EmpLastName
EmpFirstName
EmpInitial
EmpSSN

Note: Backup and Supervisor attributes are references to Employee_ID



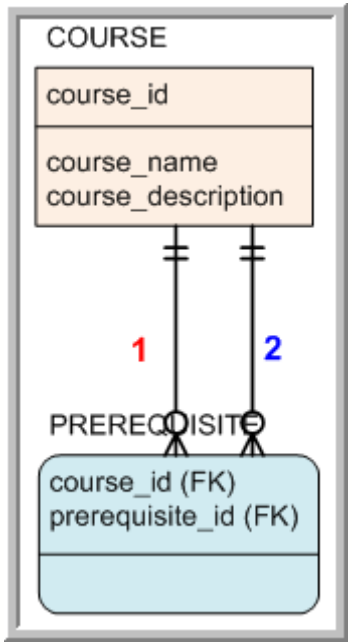
M:N Recursive Relationship

- An M:N recursive relationship:
  - One course has zero, one, or many prerequisites. The course may be other courses' prerequisites.
  - One part contains many other parts, and the part can be a component of many other parts.

- Like an M:N relationship, this can be resolved into an intersection entity with two 1:M relationships.

M:N Recursive Relationship (cont.)

- Example: Course and it's prerequisites
  - **Relationship 1**: A course has zero to many prerequisites. COURSE.course_id equals PREREQUISITE.course_id as a FK
  - **Relationship 2**: A course may be other courses' prerequisites. COURSE.course_id equals PREREQUISITE.prerequisite_id as a FK
  - Additional constraint: PREREQUISITE.prerequisite_id is not equal to PREREQUISITE.course_id

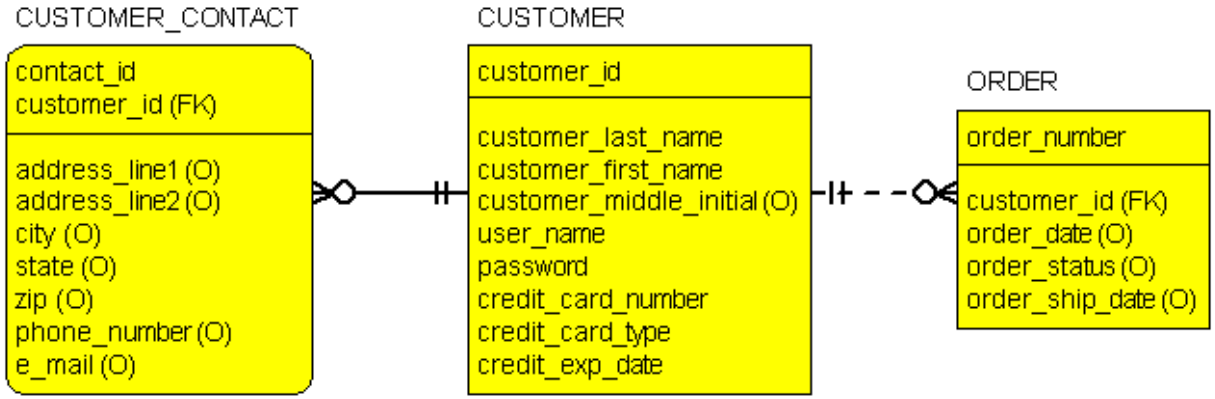


1:1, 1:M and M:N Relationships

- The cardinality ratios and participation constraints together are the structural constraints of a relationship type.
- A relationship can be identifying or non-identifying.
- How to determine identifying or non-identifying relationships:
  - From Child Entity Type to Parent Entity Type
  - Does the Child Entity Type have a key attribute?
  - Child is DEPENDENT upon the Parent for its identification and cannot exist without the parent

1:1, 1:M and M:N Relationships (cont.)

- Examples:
  - Each CUSTOMER places zero to many ORDERs
  - Each ORDER is received from one and only one CUSTOMER
  - An ORDER can be identified without information about a CUSTOMER, but requires a value for customer_id (mandatory and not null)
  - An CUSTOMER_CONTACT requires customer_id or customer's info (e.g., name) in order to get the proper CUSTOMER's contact information



Johns Hopkins Engineering

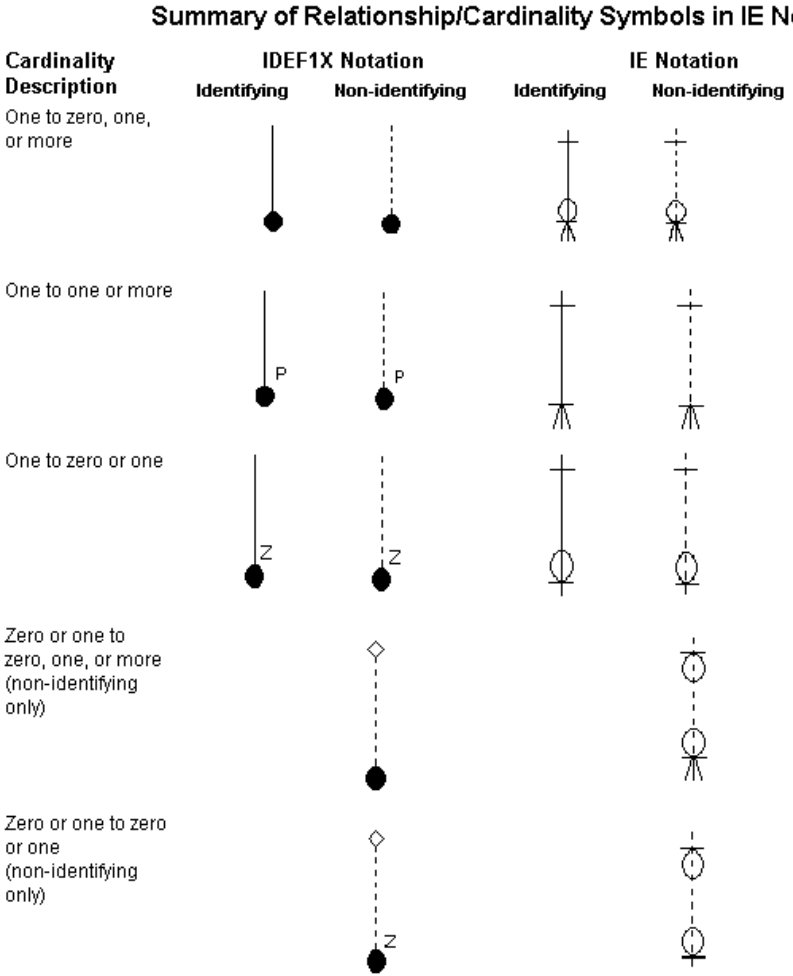
Principles of Database Systems

Module 3 / Lecture 3

Conceptual Database Design II

JOHNS HOPKINS
WHITING SCHOOL
of ENGINEERING

Summary of Relationship / Cardinality Symbols in IE Notation: Know your database design tool notation



Know your database design tool notation

Converting an ERD from Chen's Notation to IE Notation

- **Strategy: Divide and Conquer**
- **Strong Entity vs. Weak Entity**
- **Identifying vs. Non-identifying Relationships;**
- **Total Participation (Existence Dependence) vs. Partial Participation;**
- **Cardinality Ratios;**
- **Foreign Key Migration;**
- **M:N Relationship Resolution**

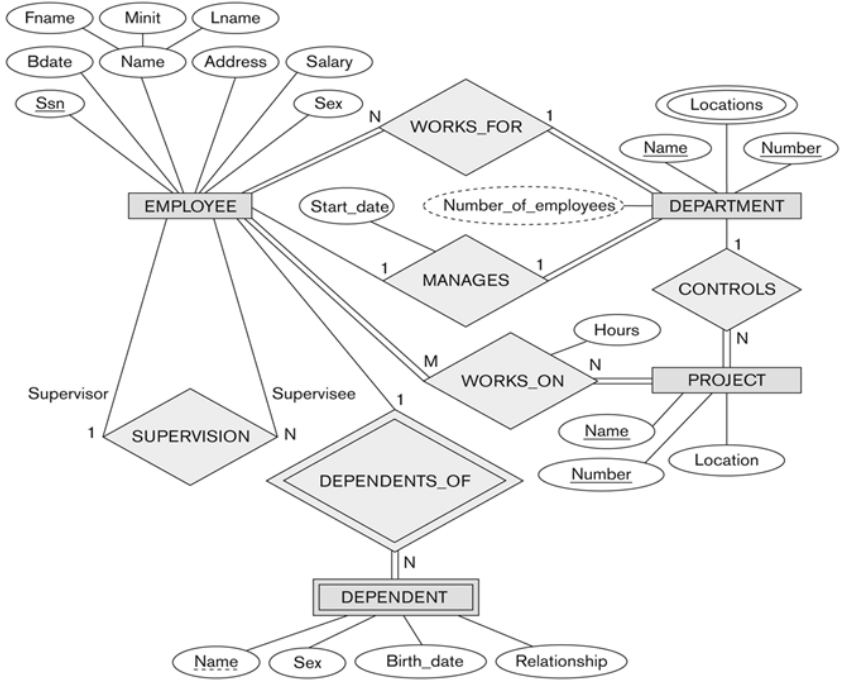
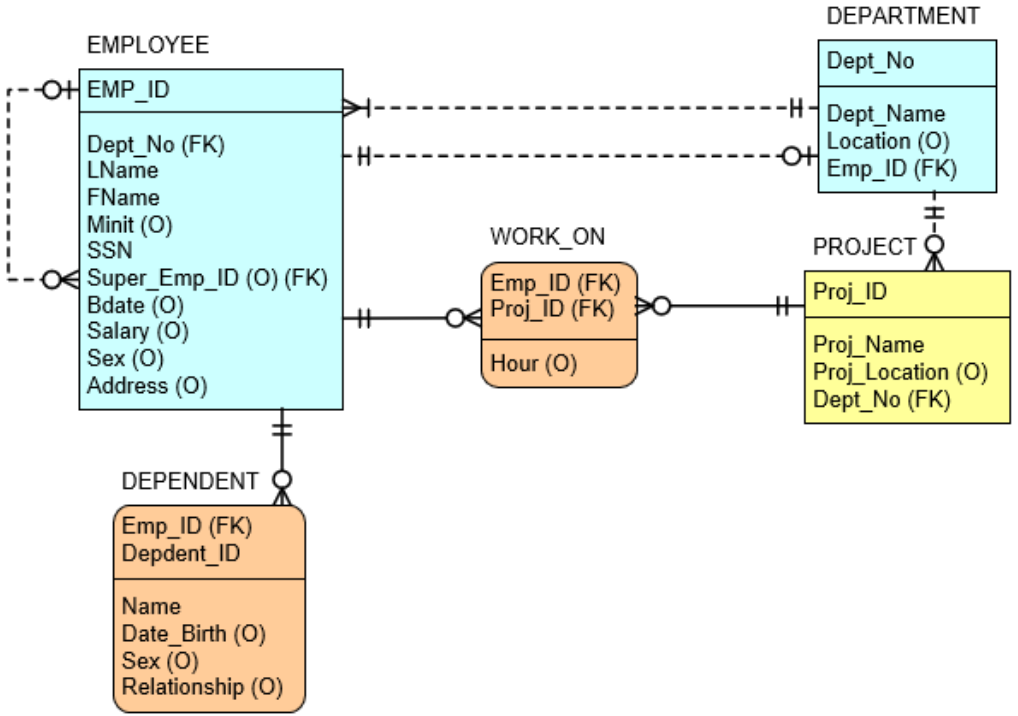


Figure 3.2
An ER schema diagram for the COMPANY database. The diagrammatic notation is introduced gradually throughout this chapter.

Converting an ERD from Chen's Notation to IE Notation (cont.)

- **Strategy: Divide and Conquer**
- **Strong Entity vs. Weak Entity**
- **Identifying vs. Non-identifying Relationships;**
- **Total Participation (Existence Dependence) vs. Partial Participation;**
- **Cardinality Ratios;**
- **Foreign Key Migration;**
- **M:N Relationship Resolution**



Company ERD Review

- Design questions on Company ERD:
  - Can an employee exist without assigning a department who they can work for?
  - Can a department exist without assigning an (or four) employee?
  - Can a department exist without assigning a department head?
  - Can a project exist without assigning a controlling department?
  - Can a dependent exist without associating an employee?
  - Business rules for each company may be different. What are reasonable business rules – a general consensus?

Supertype and Subtype Relationship

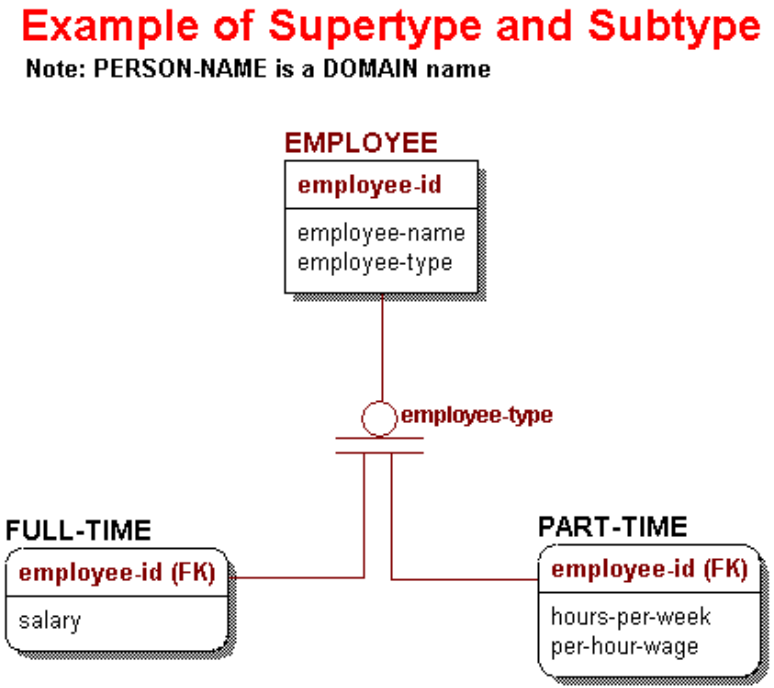
- A subtype relationship (also known as a categorization relationship) is a relationship between a subtype entity and its generic parent. A subtype relationship always relates to one instance of a generic parent with zero or one instance of the subtype.

- A COMPANY should be either a SUPPLIER or a BUYER. Both SUPPLIER and BUYER must be only one COMPANY and are examples of subtype entities while the COMPANY is supertype.

- In general, this is a mandatory one-to-one relationship.

Supertype and Subtype Relationship (cont.)

- The relationship can be complete or incomplete.



Category: complete/incomplete
Parent to category
Category to child

Category discriminator
employee type: full/part time
employee class: secretary…

Arc Relationship

- Oracle designer uses Arc relationship for this exclusive OR case while ERwin uses supertype and subtype relationships.

- For example, a MEMBERSHIP must be held by either a CUSTOMER or a COMPANY. A CUSTOMER or a COMPANY may each hold more than one MEMBERSHIP.

Relationship Reading Syntax

- Each source entity (may be / must be) relationship name (one and only one / one or more) destination entity.
- Each direction of a relationship has a name, e.g. taught by or assigned to; an optionality (must be / may be), a degree (one and only one / one or more).

Naming Convention

- Based on requirements, a noun will be either an entity type, or attribute (or an instance). The verbs usually indicate names of relationship types.
- Use upper case for an entity type name and lower case for an attribute. Name should be descriptive. Entity type name is singular.
- Organize the E/R diagram with top-down and left-to-right, grouping, and coloring to increase readability, and to support the business operations.

Modeling Business Rules using ER Diagram

- Entity types with attributes and relationships
- Categories of sets of data within data
- Mandatory versus optional relationship
- Multiple relationships

Data Modeling and Database Design

- Create a business narrative based on interview notes, requirements, and specifications.
- Create an ER diagram. ER diagrams are derived from business narratives.
- Create a database design. In a relational database, the ER diagram maps entity types to tables, attributes to columns, relationships to foreign keys, and business rules to constraints.
- Create a table definition. The basis for the SQL CREATE TABLE statement creates tables, columns, and constraints.

Advantages for using ER Model

- Easy presentation for the design
- Easy to communicate with users
- Easy to be developed, and refined

The ER diagram is a graphical representation of business information needs and rules.

Any change made during later stages of the development life-cycle can be very expensive. Therefore, the conceptual modeling stage is very important for properly implementing the system requirements.

Johns Hopkins Engineering

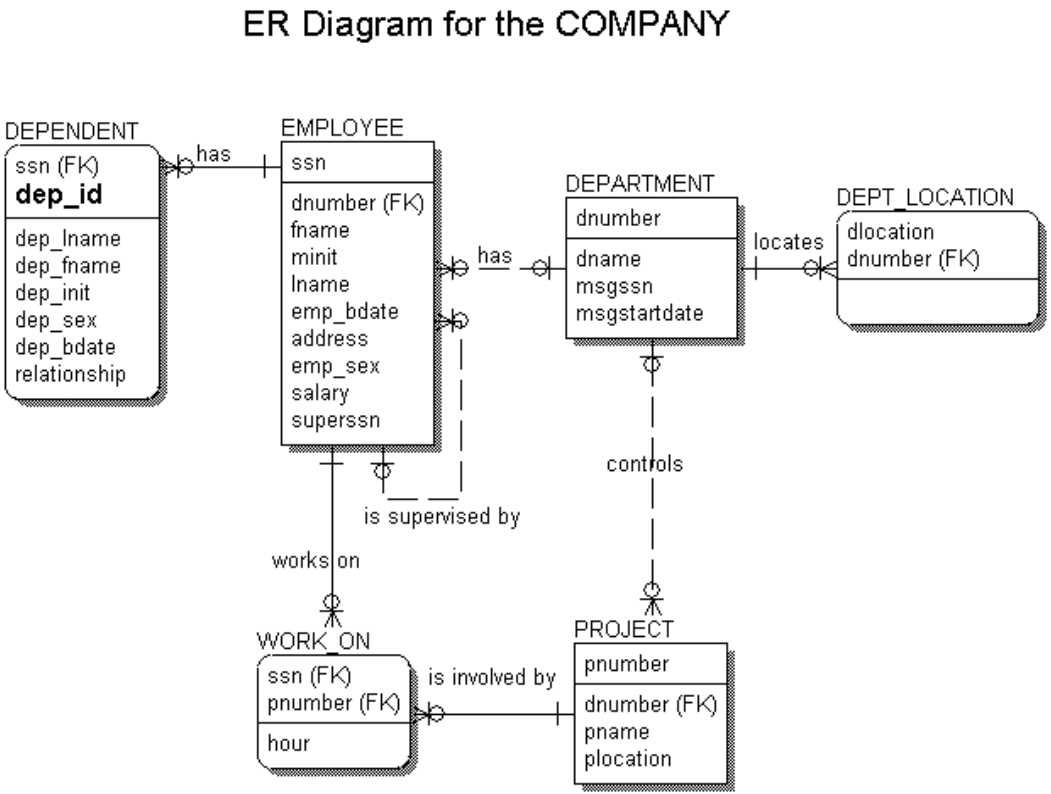
Principles of Database Systems

Module 3 / Lecture 4

Conceptual Database Design II

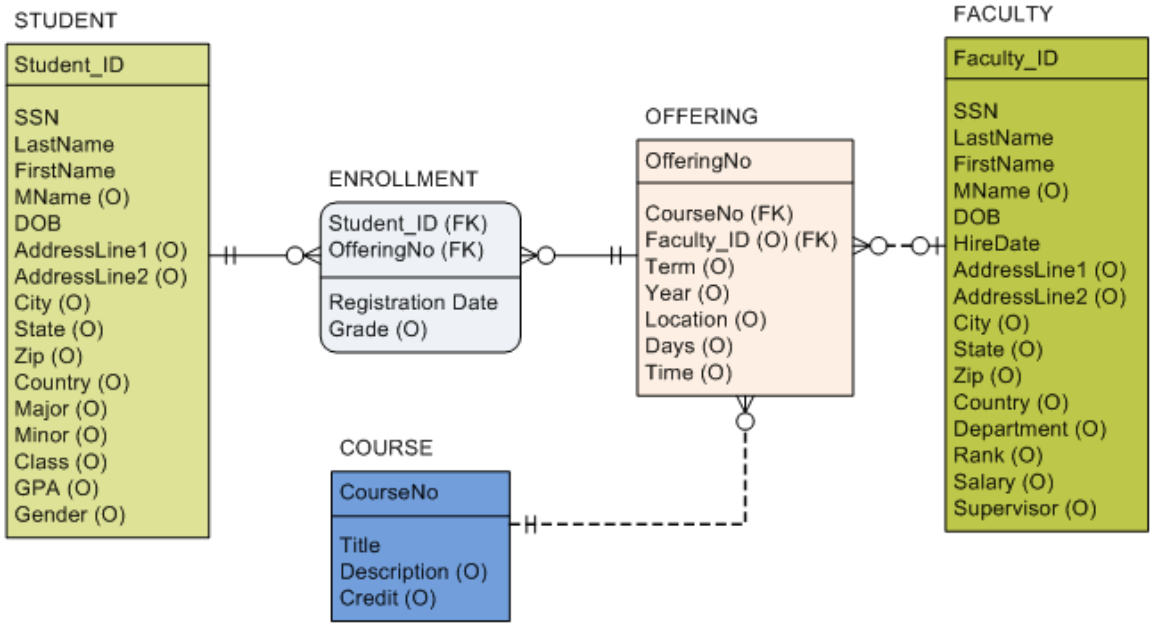
JOHNS HOPKINS
WHITING SCHOOL
of ENGINEERING

Company ERD Using ERwin



Different Database Design tools may have slightly different notations to create ERDs.

Case Study – University ERD



Questions:

- Explain how to model Term in the OFFERING table
- Explain how to model faculty rank in the FACULTY table
- Explain how to model Country attribute in the STUDENT table and FACULTY table
- Explain how to model department in the FACULTY table
- Explain how to model Student Majors and Minors
- Explain how to model that a student is also a faculty member
- Discuss the pro and con if a PERSON table to cover both STUDENT and FACULTY

Form Analysis

- Requirements may come from a form, a written document, or both.

- Steps in the form analysis process -
  1. Define and Analyze Form Structure:
     - Construct a hierarchy from the form structure. Most forms consist of a simple hierarchy where the main form is the parent and the subform is the child. Complex forms (subforms inside subforms) are not as common because they can be difficult for users to understand.

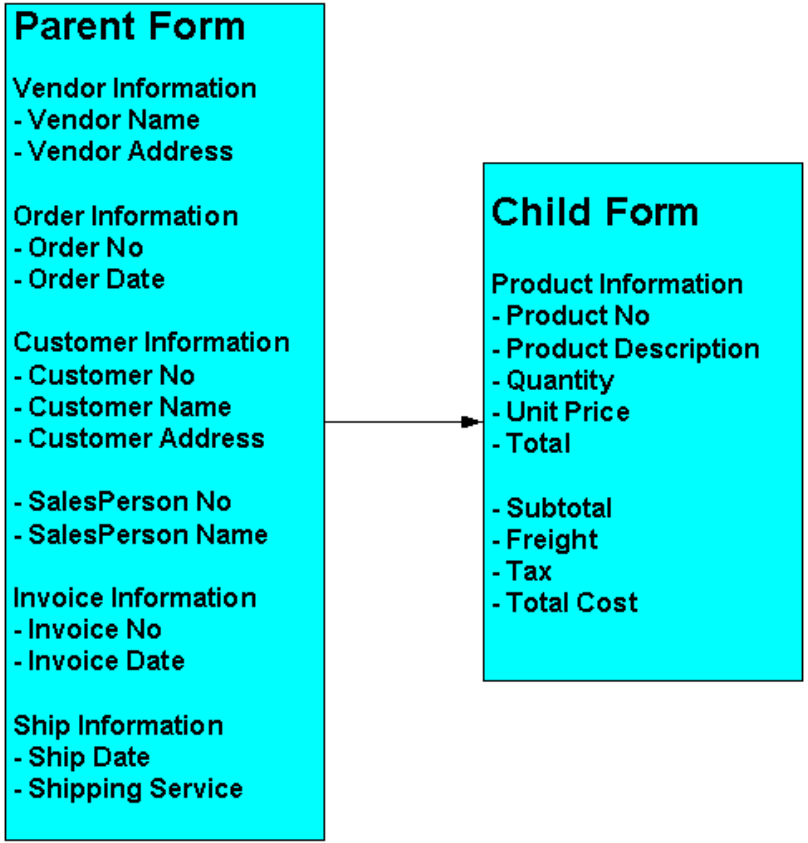
Form Analysis (cont.)

- Steps in the form analysis process -
  2. Identify Entity Types:
     - Map and split the hierarchical structure into one or more entity types.
     - Look for form fields that can be a primary key(s) of an entity type.
     - Group form fields into entity types using functional dependencies.
  3. Attach Attributes:
     - Attach attributes to the entity types identified in the previous step.
     - Be aware of attributes belonging to a Many-to-Many relationship that require an additional entity type.

Form Analysis (cont.)

- Steps in the form analysis process -
  4. Add Relationships:
     - Connect entity types with (identifying or non-identifying) relationships and specify cardinalities.
  5. Check Completeness and Consistency:
     - Check the ERD for consistency and completeness with the form structure
     - Verify minimum and maximum cardinalities for all relationships, a primary key for all entity types, and a name for all relationships.

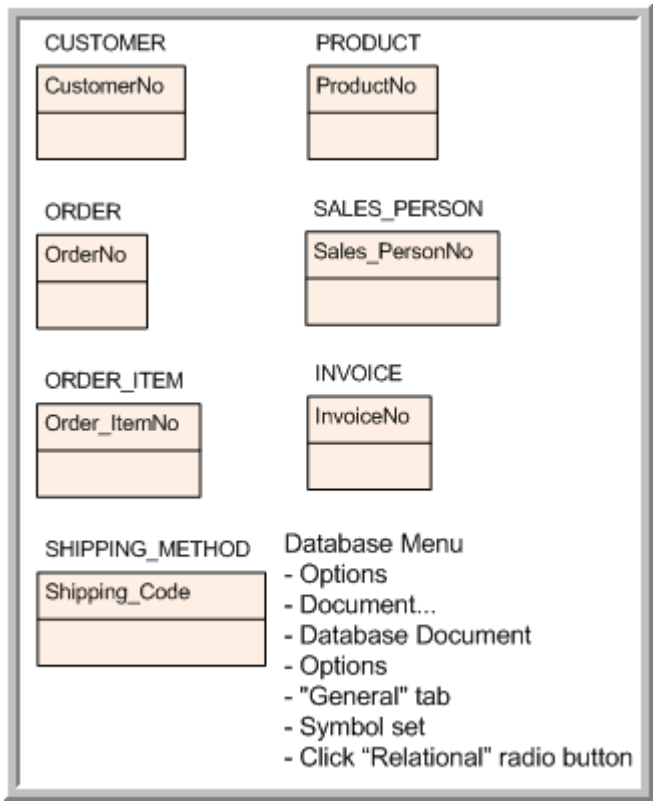
Form Analysis: Invoice Example

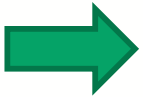


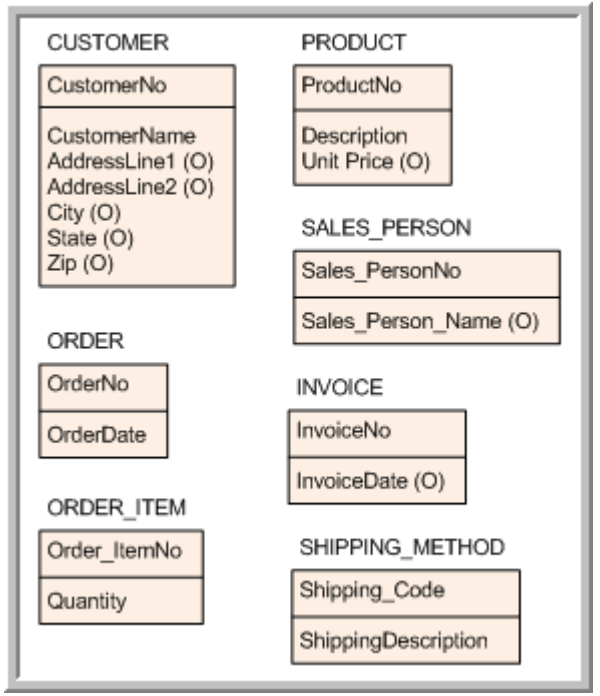
Parent Form includes entities with one occurrence on a form.

Child Form may include multiple occurrences on a form.

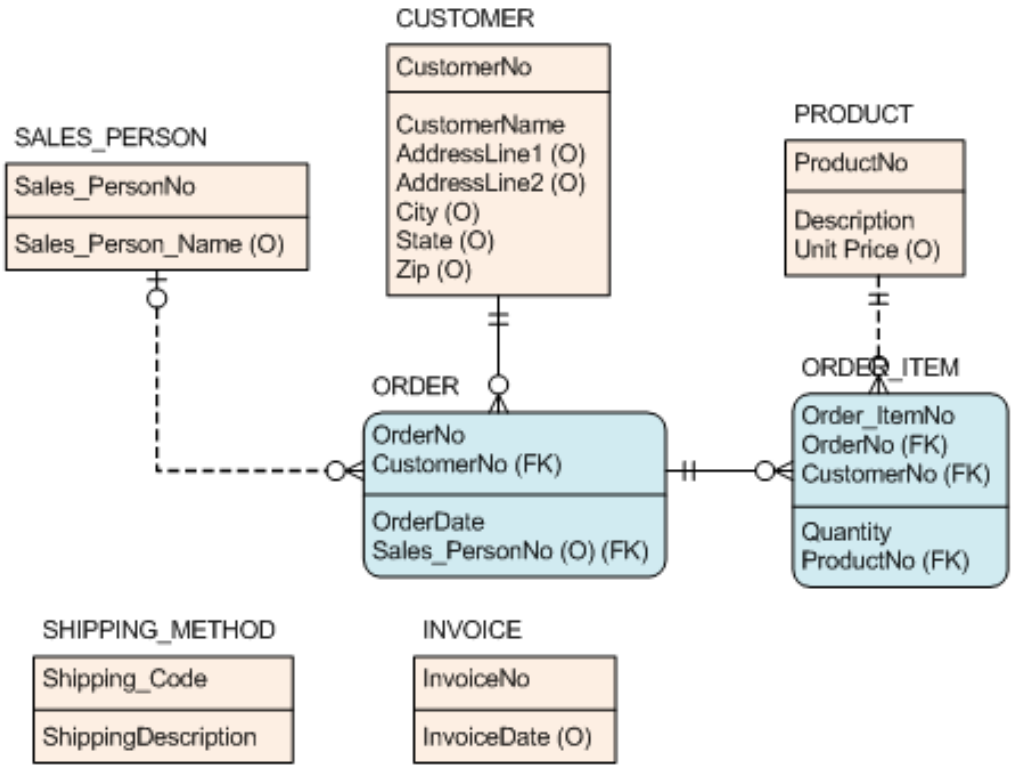
Form Analysis: Invoice Example (cont.)







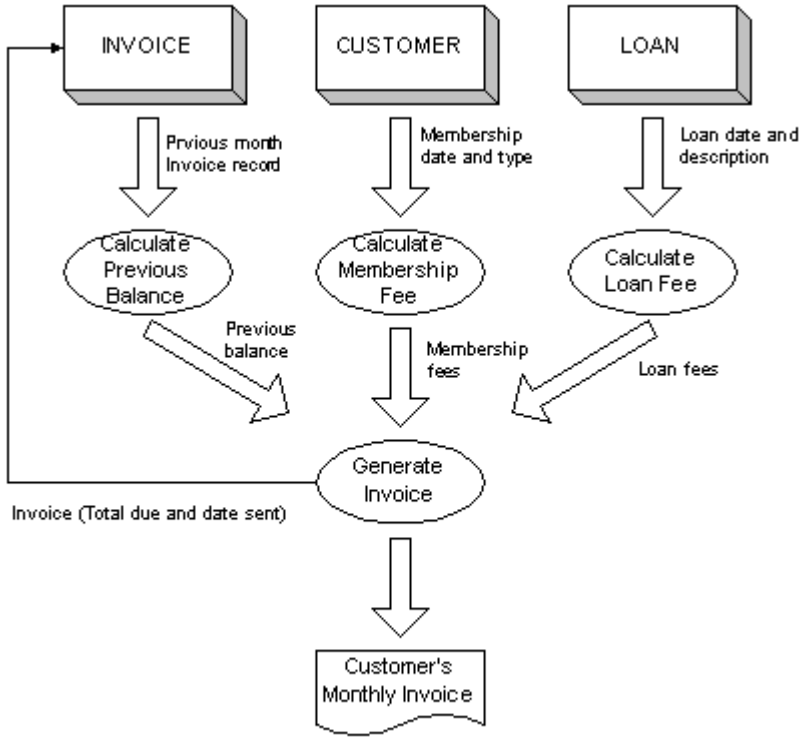
Form Analysis: Invoice Example (cont.)



Questions:

- Do you need a COMPANY for vendor information?
- What is the relationship between ORDER and INVOICE entities?
- What is the role for SHIPPING METHOD entity?
- How do you handle BILL TO and SHIP TO information?
- How to enforce the STATE reference occurred at various entities?

Form Analysis: Invoice Example (cont.)



Data flow diagram for the customer invoicing operation

Invoice Processing:
The invoice process may involve multiple entities and calculations in order to generate a Customer's Monthly Invoice.

Johns Hopkins Engineering

Principles of Database Systems

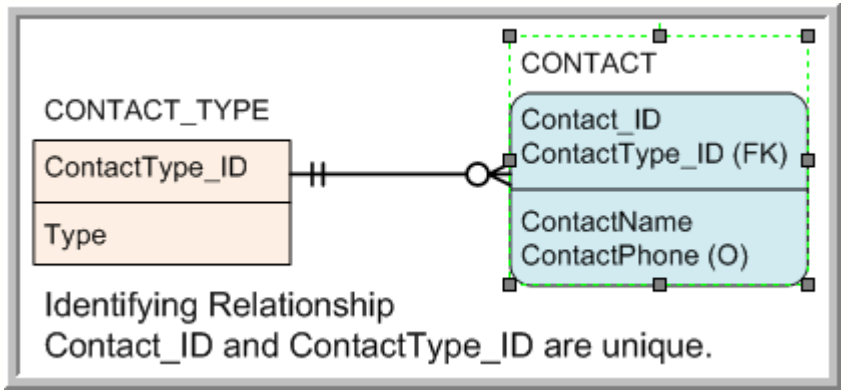
Module 3 / Lecture 5

Conceptual Database Design II

JOHNS HOPKINS
WHITING SCHOOL
of ENGINEERING

An Alternative Design for An Identifying Relationship

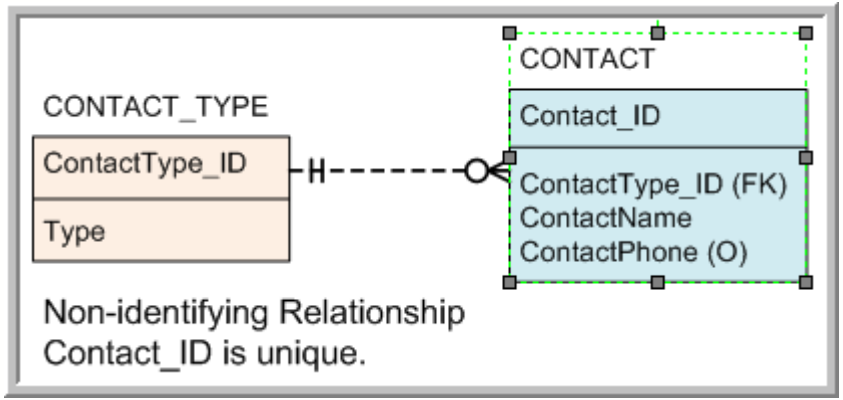
- Using an identifying relationship, a PK of a parent entity will migrate to a child entity as a FK as well as a part of PK.
- The child entity may have a composite PK with many attributes that can be an overhead on implementation.



Physical Name	Data Type	Req'd	PK
Contact_ID	INTEGER	☑	☑
ContactType_ID	INTEGER	☑	☑
ContactName	VARCHAR(30)	☑	☐
ContactPhone	CHAR(10)	☐	☐

An Alternative Design for An Identifying Relationship (cont.)

- The alternative design with an identifying relationship has the following properties:
  - The child entity becomes a strong entity.
  - The PK of the child entity remains unique and NOT NULL with an artificial sequence number.
  - The FK remains FK with a NOT NULL constraint.



Physical Name	Data Type	Req'd	PK
Contact_ID	INTEGER	☑	☑
ContactType_ID	INTEGER	☑	☐
ContactName	VARCHAR(30)	☑	☐
ContactPhone	CHAR(10)	☐	☐

Checking the Data Model

- The check process is to ensure that the information required by the functions can actually be derived from the model.
  - Make sure entity types are connected properly so that the required data can be effectively retrieved from the database.
  - Look into the functional usage of the data.
  - Check 1:M identifying relationships and non-identifying relationships.
  - Check M:N relationships and resolve with intersection entity types.
  - Check recursive relationships (normally non-identifying relationship).

Relevant Database-related Tools

- Comparison of RDBMSs - Compare general and technical information OS support, Limits, Partitions, DB capabilities: http://en.wikipedia.org/wiki/Comparison_of_relational_database_management_systems#Fundamental_features

- Comparison of Database Administrator Tools - Compare general and technical information: http://en.wikipedia.org/wiki/Comparison_of_database_tools

- Web Application Development Tools - the process and practice of developing web applications: https://en.wikipedia.org/wiki/Web_application_development

Making A Decision on Technology Solution

- Best technology or most popular technology (e.g. Borland Delphi)
- Cost
- Future vendor perspectives:
  - Company size and financial condition
  - Company merge or acquisition
    - Oracle bought BEA, Sun Microsystems, PeopleSoft, J.D Edward, Siebel and others
    - SAP acquired Sybase
    - Microsoft
    - IBM brought Informix

Conceptual Database Design Conclusions

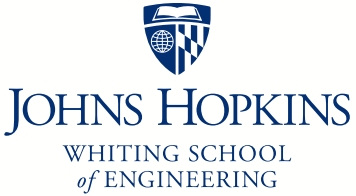
- Good data modeling:
  - Lead a model/structure to be smaller, faster, simpler, more understandable, and easier to maintain.
  - Decrease development time.
  - Increase database system performance and quality.
- A good model comes from skilled staff with intelligence on abstract and analytical thinking, experience, and communication skills.
- If possible, use commercial database design tools to create your model.

Johns Hopkins Engineering

Principles of Database Systems

Module #4

Enhanced ER (EER) Modeling



Enhanced ER Modeling Introduction

- Additional semantic data modeling with additional symbols in order to describe more complex business requirements.
- There is no standard for EERD.
- Commercial database design tools do not support EERD.

Enhanced ER Modeling Introduction (cont.)

- Most EERDs can be represented and implemented with regular ERDs with relational mappings.

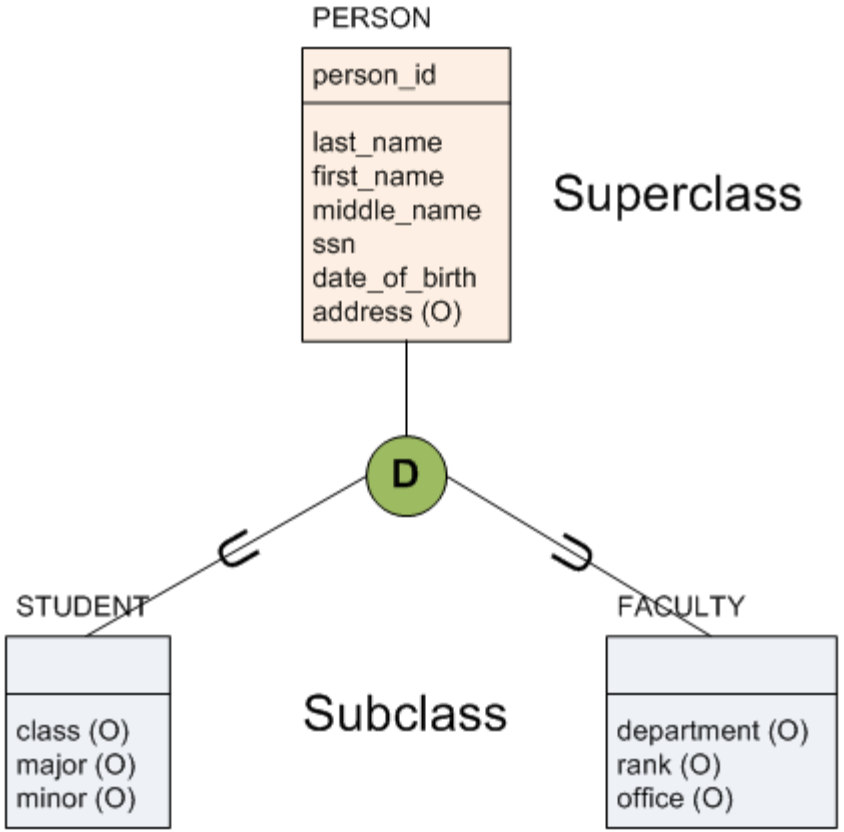
- There are challenges to implementing EERD.
  - Database design tools do not support EERD.
  - There are multiple ways to implement EERD.
    - Efficiency for implementation
    - Performance
    - Business rules

Superclasses and Subclasses

- Superclass
  - An entity type requires representation in a data model.
  - It includes one or more distinct subgroupings of its occurrences, which require representation in a different data model.

Example: The University ERD
PERSON entity contains common attributes for a person as a superclass.
FACULTY entity contains special attributes for a faculty instance as a subclass.
STUDENT entity contains special attributes for a student instance and a subclass.

Superclasses and Subclasses (cont.)



Superclass
- PERSON

Subclass
- STUDENT
- FACULTY

1:1 relationship between superclass and subclass

Superclasses and Subclasses (cont.)

- Superclass
  - Distinct subgroupings of an entity type require representation in a data model.
  - An entity instance cannot exist in the subclass without a corresponding entity instance existing in its superclass.
  - The subclass has additional attributes of interest.
  - Potential subclasses represent variations of a similar concept.
  - All potential subclasses have the same attribute that can be factored out or expressed in the superclass.

Superclasses and Subclasses (cont.)

- Subclass
  - Each member of a subclass is also a member of the superclass. In other words, the entity in the subclass is the same entity in the superclass, but with a distinct role.
  - The relationship between the superclass and the subclass is **1:1** and is called superclass/subclass relationship.
  - Not every entity (instance or member) of a superclass needs be an entity of a subclass.

Superclasses and Subclasses (cont.)

- Examples

  Example 1: An OWNER entity type may be classified as ORGANIZATION or PERSON. Each entity type has distinct attributes.

  Example 2: An EMPLOYEE entity type may be classified as MANAGER, ENGINEER, SALEPERSON, TECHNICIAN, and SECRETARY. Each entity type has distinct attributes.

Superclasses and Subclasses (cont.)

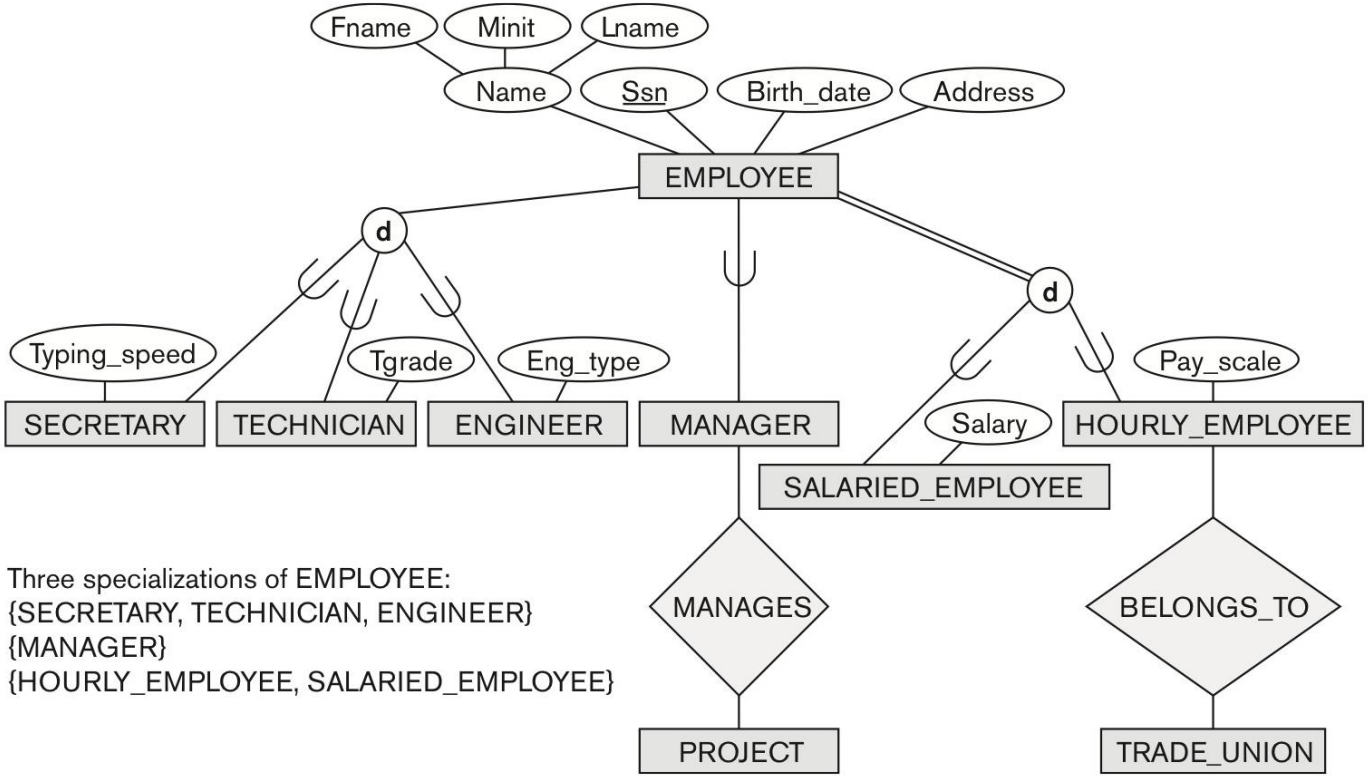


Figure 4.1 EER diagram notation to represent subclasses and specialization.

Superclasses and Subclasses (cont.)

- A superclass may include more than one subclass (e.g., an employee is an engineer and has full-time status.)

- In Object-Oriented Concept:
  - A subclass inherits all the attributes of its superclass (type inheritance).
  - A superclass/subclass relationship is often called an **IS-A** relationship or hierarchy. For example, a Manager IS-A member of Staff or Employee.

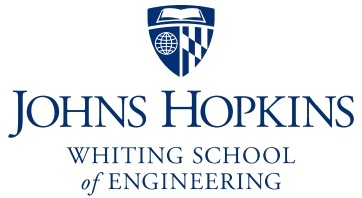
- Comparison between EER and Object-Oriented (OO) designs.

Johns Hopkins Engineering

Principles of Database Systems

Module 4 / Lecture 2

Enhanced ER (EER) Modeling



Specialization and Generalization

- Specialization: The process of maximizing the differences between members of an entity by identifying their unique characteristics.

- Specialization is a top-down approach to defining a set of superclasses and their related subclasses. Like a process of defining a supertype and (a one or more) subtypes with proper relationships.

Specialization and Generalization (cont.)

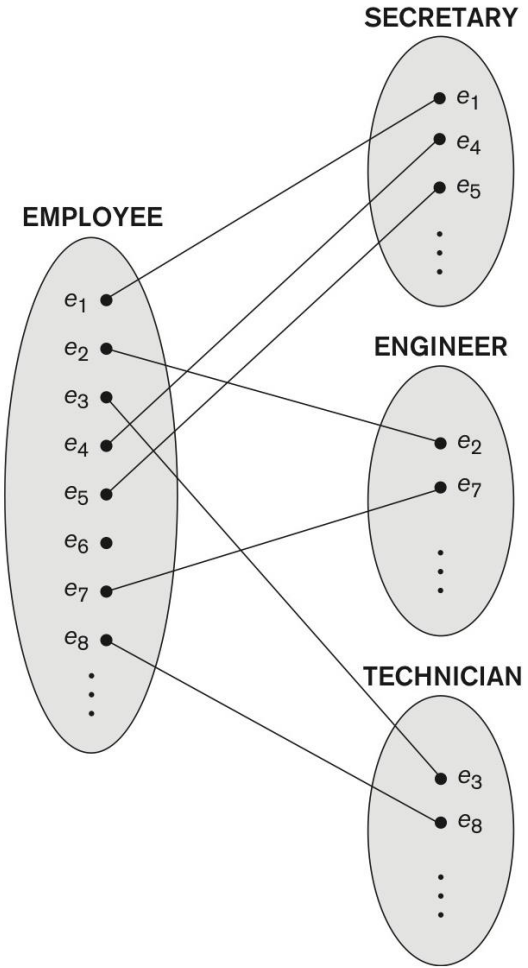


Figure 4.2
Instances of a specialization.

Specialization, and Generalization (cont.)

- Generalization: The process of minimizing the differences between entities by identifying their common characteristics.
- Generalization is a bottom-up approach, which results in the identification of a generalized superclass from the original entity types. The same process can be used to define supertype and subtypes and their relationships.

Specialization and Generalization (cont.)

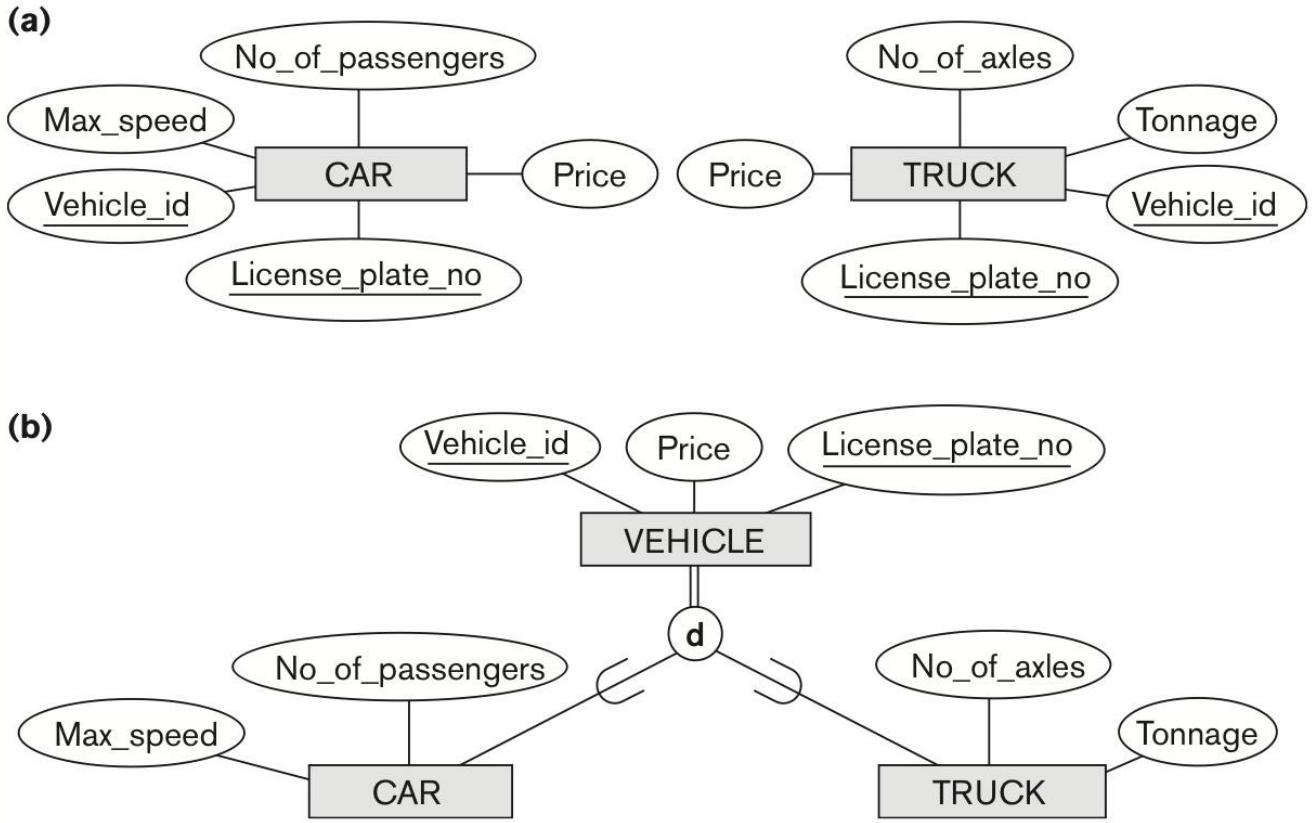


Figure 4.3
Generalization.
(a) Two entity types, CAR and TRUCK.
(b) Generalizing CAR and TRUCK into the superclass VEHICLE.

Constraints and Characteristics of Specialization and Generalization

- Determine a condition on the value of an attribute of the superclass to identify the participations on its subclasses.

- Disjointness constraint: the subclasses of specialization must disjoint. An entity can be a member of, at most, one of the subclasses of the specialization. The **d** in a circle stands for disjoint.

Constraints and Characteristics of Specialization and Generalization (cont.)

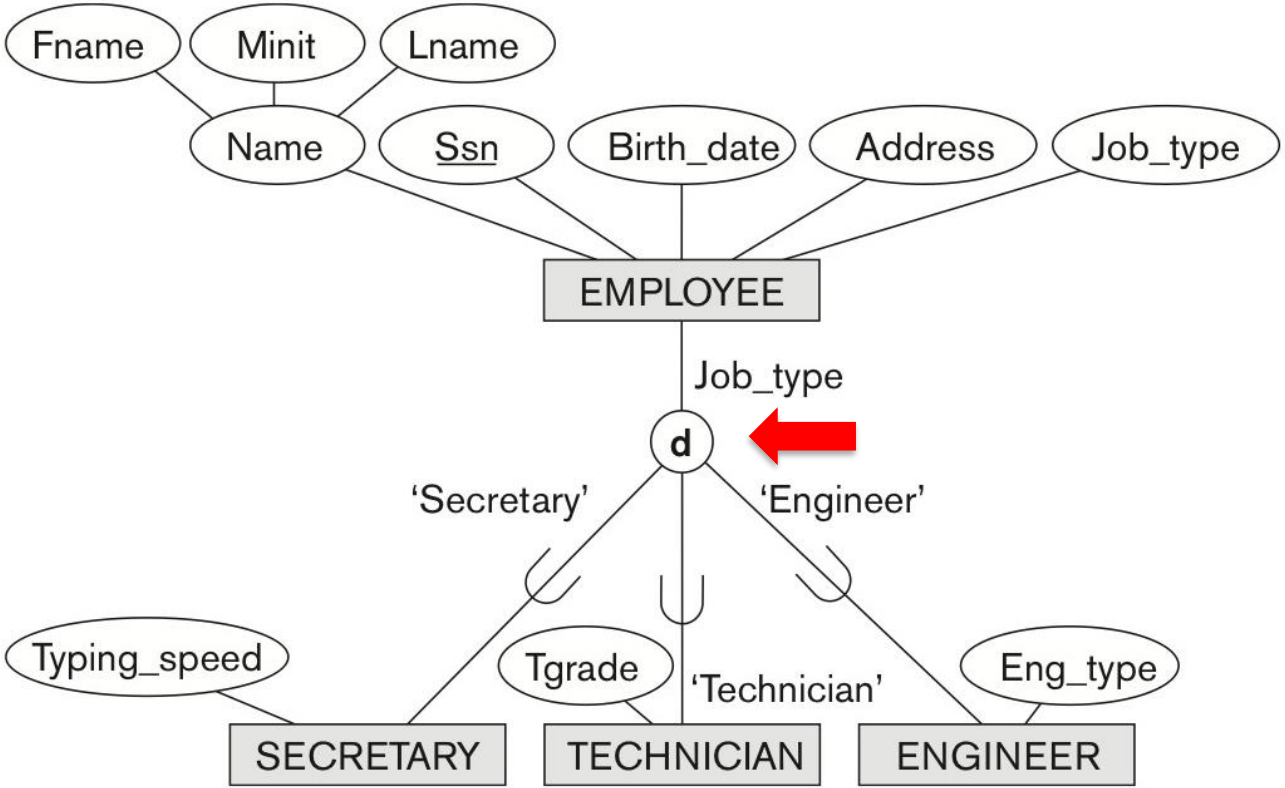


Figure 4.4 EER diagram notation for an attribute-defined specialization on Job_type.

Constraints and Characteristics of Specialization and Generalization (cont.)

- Overlap: If the subclasses are not constrained to be disjoint, their sets of entities may overlap. The entity in a superclass may have a member in more than one subclass of specialization. An **o** in a circle stands for overlap.

Constraints and Characteristics of Specialization and Generalization (cont.)

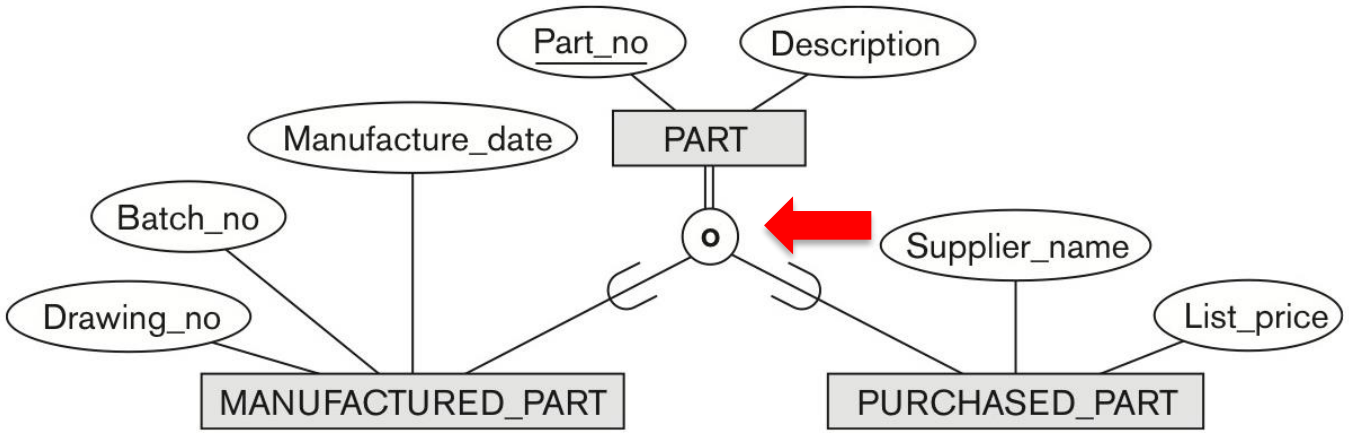


Figure 4.5 EER diagram notation for an overlapping (nondisjoint) specialization.

Constraints and Characteristics of Specialization and Generalization (cont.)

- Completeness constraint: A total specialization constraint specifies every entity in the superclass must be a member of at least one subclass in the specialization.

  Example: An EMPLOYEE entity has to be either an HOURLY_EMPLOYEE or SALARY_EMPLOYEE entity types. An entity can be a member of, at most, one of the subclasses of the specialization. A *double line connects the superclass to the circle*.

Constraints and Characteristics of Specialization and Generalization (cont.)

- A partial specialization allows an entity not to belong to any of the subclasses.

  Example: An employee entity may not be a secretary, technician, or engineer.

- A specialization hierarchy, has the constraint that every subclass participates as a subclass in only one class/subclass relationship.

Constraints and Characteristics of Specialization and Generalization (cont.)

- A specialization lattice allows a subclass to have more than one class/subclass relationship.

  Example: an Engineering_Manager is manager, engineer, and full-time employee, a union of three superclasses

Constraints and Characteristics of Specialization and Generalization (cont.)

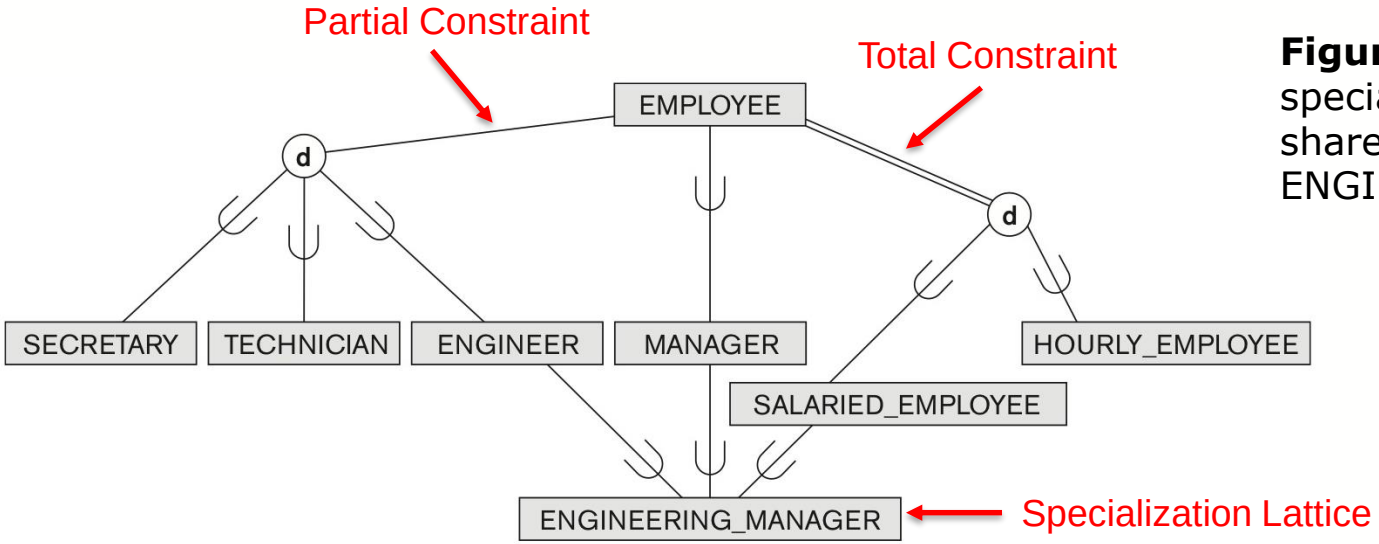


Figure 4.6 A specialization lattice with shared subclass ENGINEERING_MANAGER.

Constraints and Characteristics of Specialization and Generalization (cont.)

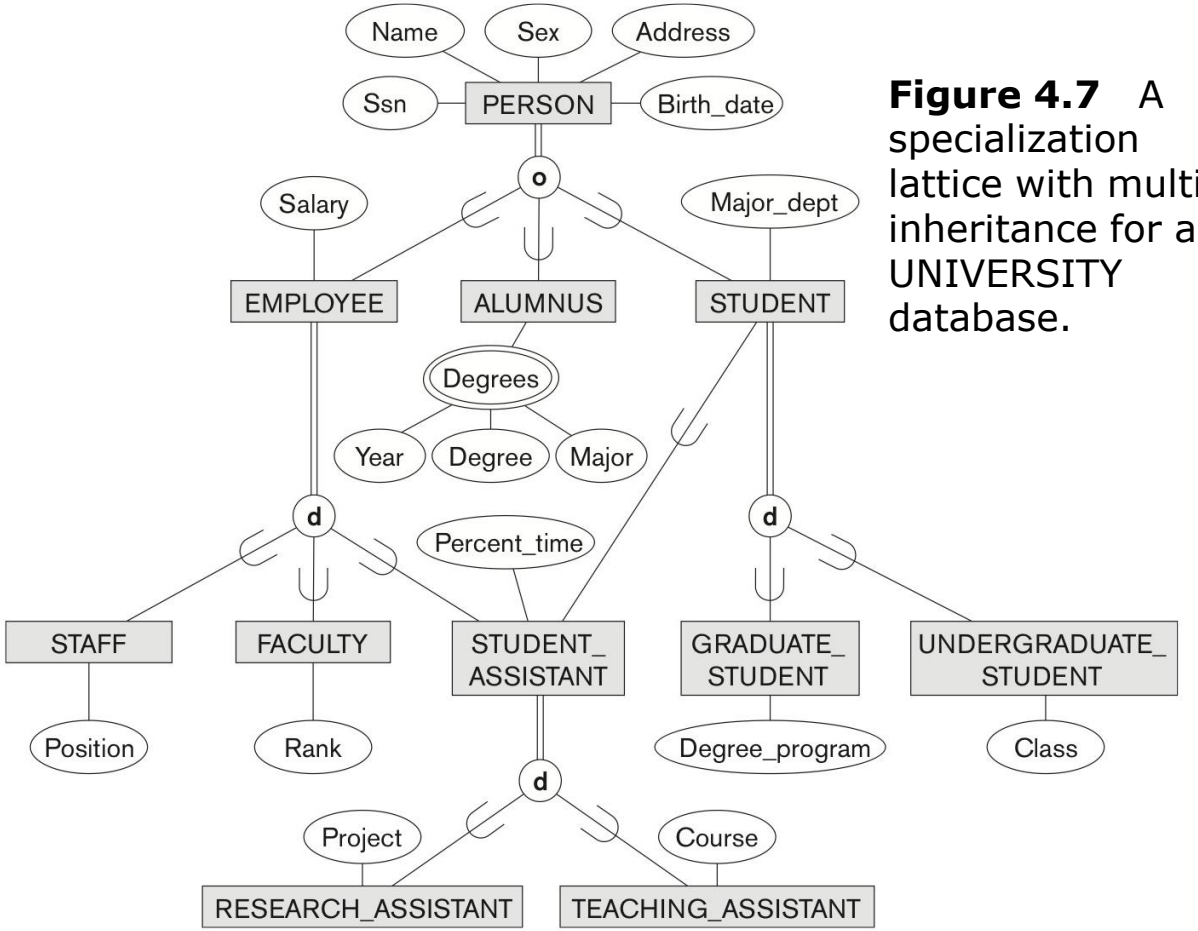


Figure 4.7 A specialization lattice with multiple inheritance for a UNIVERSITY database.

ohns Hopkins Engineering

Principles of Database Systems

Module 4 / Lecture 3

Enhanced ER (EER) Modeling

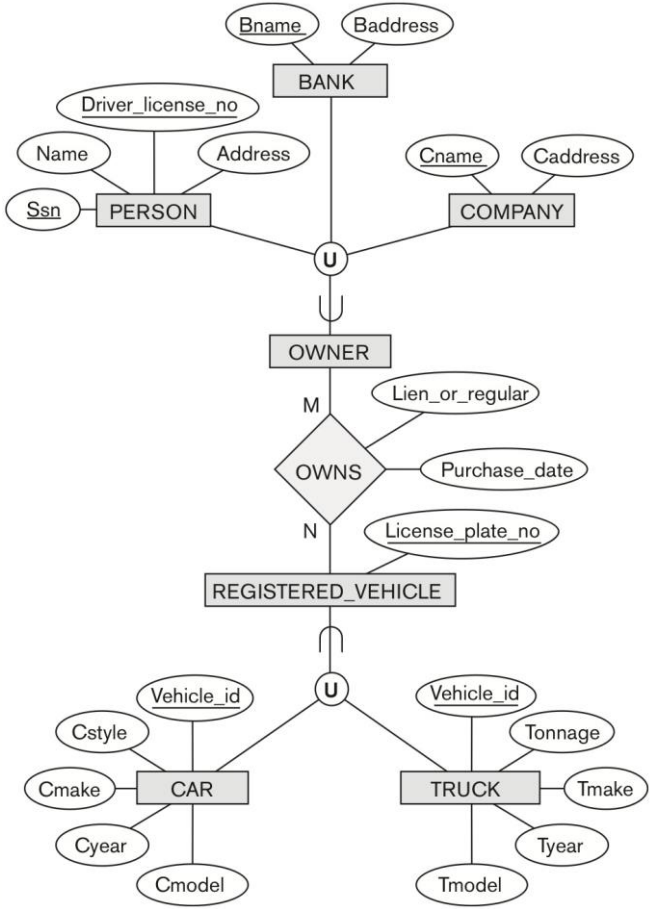
JOHNS HOPKINS
WHITING SCHOOL
of ENGINEERING

Modeling with Union Types Using Categories

- A union type subclass is a collection of objects that is a union of a distinct entity types.
- An instance of a subclass links to only one instance of one entity type.

Example:
An owner of a registered vehicle can be a PERSON, a BANK, or a COMPANY.
A registered vehicle can be a CAR or a TRUCK.

Modeling with Union Types Using Categories (cont.)



Superclass:
PERSON, BANK, COMPANY
Subclass:
OWNER

Figure 4.8 Two categories (union types): OWNER and REGISTERED_VEHICLE

Superclass:
CAR, TRUCK
Subclass:
REGISTERED_VEHICLE

An Example: UNIVERISTY EERD Using Chen's Notation

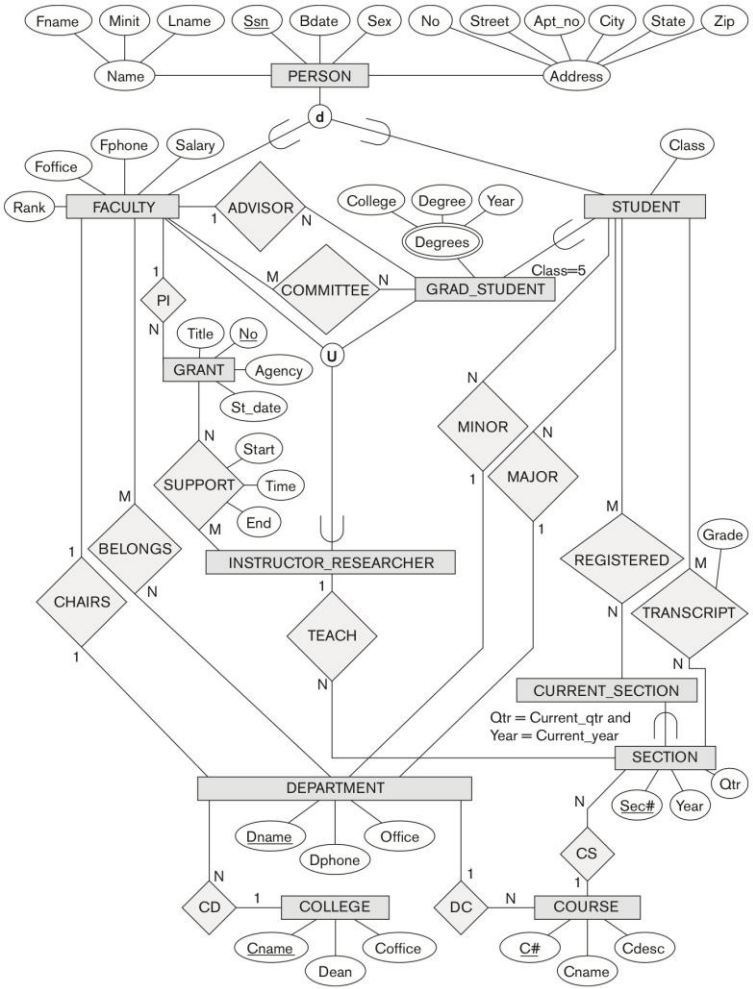
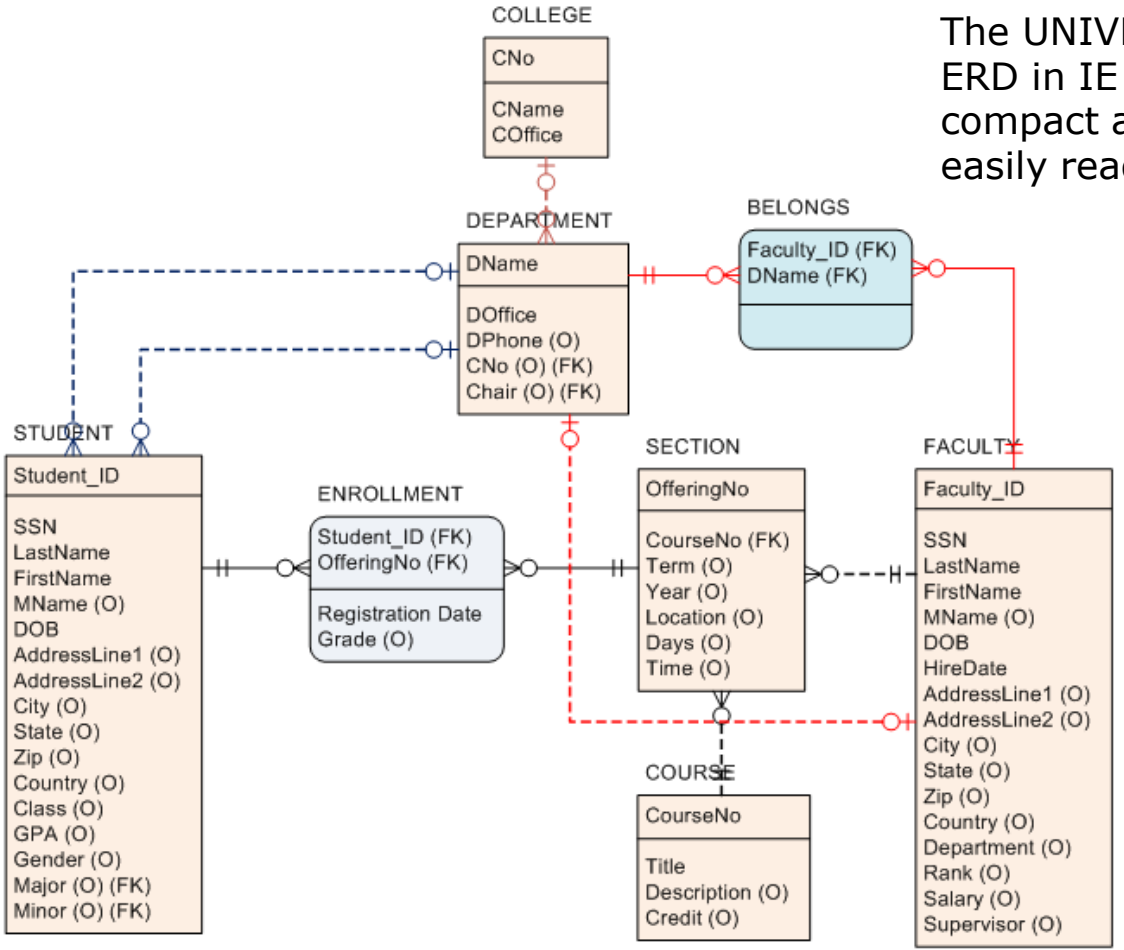


Figure 4.9 An EER conceptual schema for a different UNIVERSITY database.

Chen's notation vs. IE notation

EERD and ERD compare and contrast

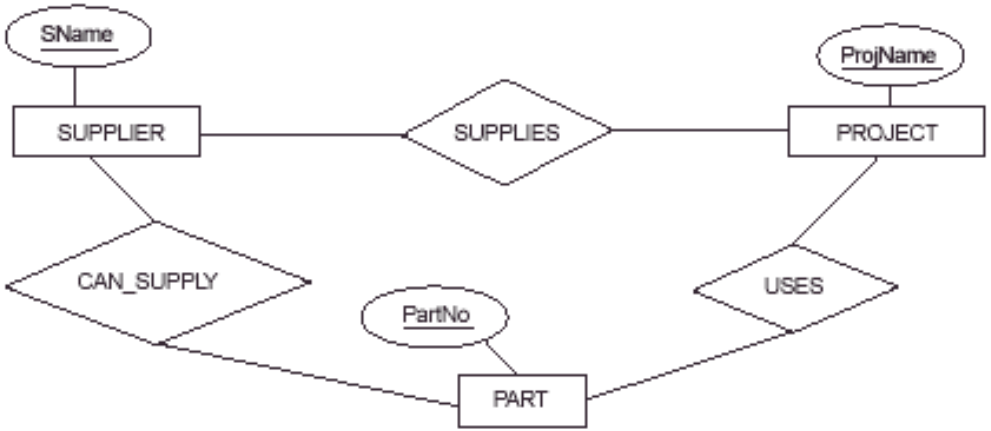
An Example: UNIVERISTY ERD Using IE Notation



The UNIVERISTY ERD in IE is compact and easily read.

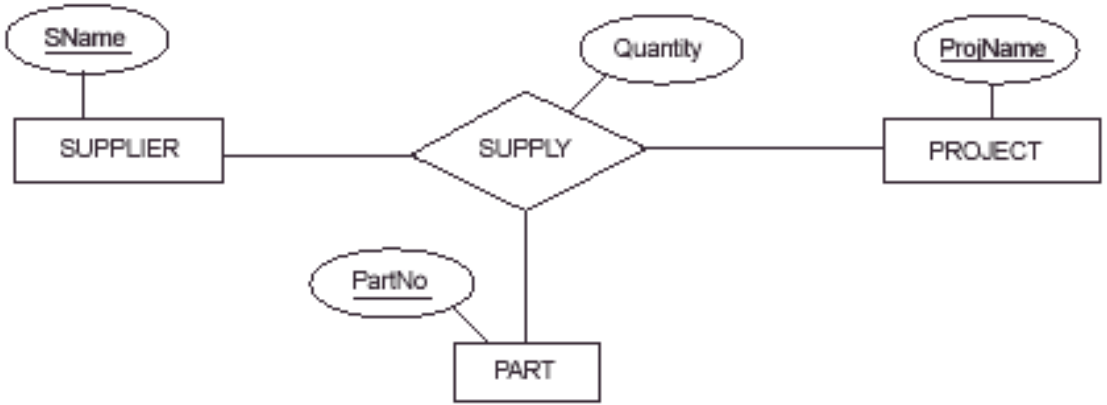
Relationship Types with Higher Degrees

- The relationships among SUPPLIER, PROJECT, and PART:
  - Three binary M:N relationships



Relationship Types with Higher Degrees (cont.)

- The relationships among SUPPLIER, PROJECT, and PART:
  - One ternary relationship

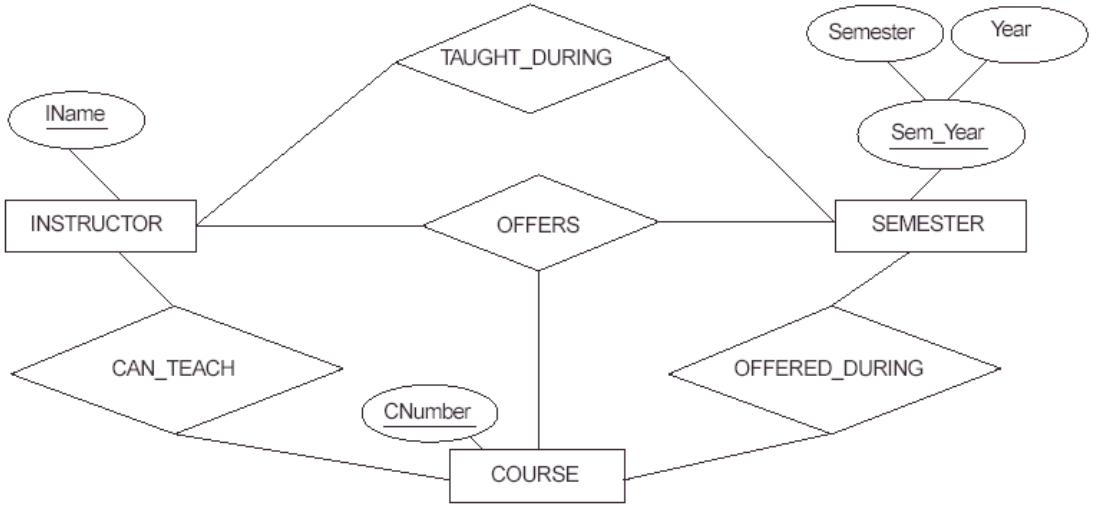


Relationship Types with Higher Degrees (cont.)

- Three binary M:N relationships vs. a ternary relationship:
  - Which one you use depends on the business requirements.
- Different relationships represent different information.
  - Do not create all relationships if business requirements don’t specify them.

Relationship Types with Higher Degrees (cont.)

- The relationships among INSTRUCTOR, COURSE, and SEMESTER:
  - Three binary relationships or a ternary relationship.



Johns Hopkins Engineering

Principles of Database Systems

Module 4 / Lecture 4

Enhanced ER (EER) Modeling

JOHNS HOPKINS
WHITING SCHOOL
of ENGINEERING

Unified Modeling Language Class Diagram

- Unified Modeling Language (UML)
  - A set of graphical notations are used for business modeling and software design.
  - Class diagrams include classes, object features (attributes and methods) and associations among classes.
    - Class: A class defines both the structural attributes and behavioral features (methods or operations).
    - Class Association: A relationship between classes.
      - Binary: Simple association between two classes
      - N-ary: An association between multiple classes

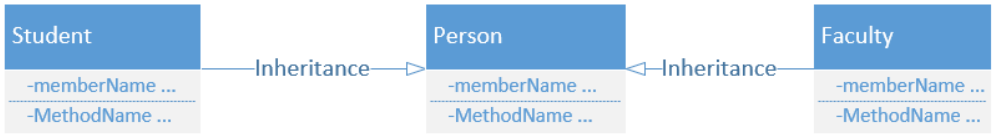
Unified Modeling Language Class Diagram (cont.)

- Class Association
  - A family of links like uni-directiononal, bi-directiononal, and aggregation
  - One line connects two classes depicting an association

  - Multiplicity: ***0*** (no instance); ***0..1*** (no instance or one instance);***1*** or ***1..1*** (exactly one instance); ***** or ***0..**** (zero or more instances); and ***1..**** (one or more instances.)

Unified Modeling Language Class Diagram (cont.)

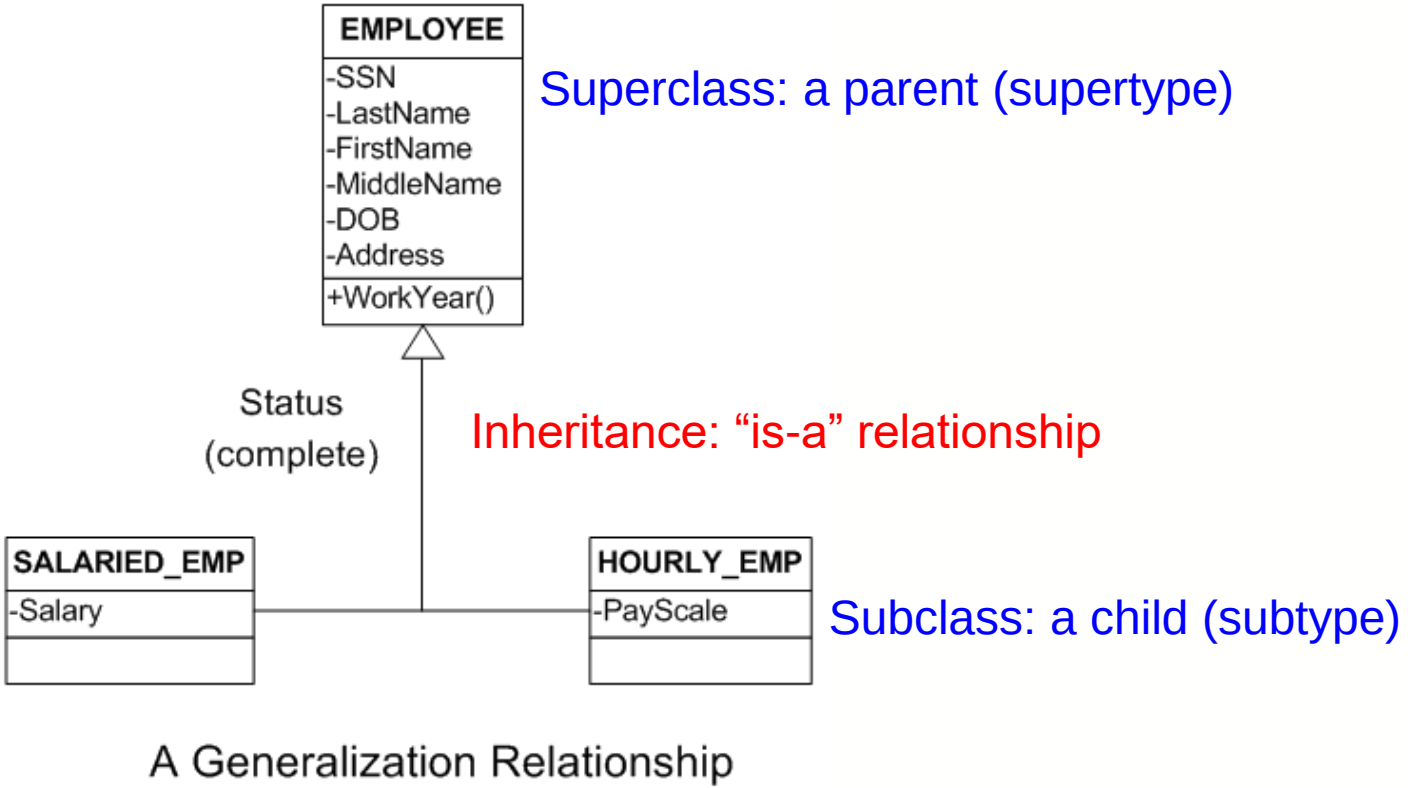
- Inheritance: “is-a” relationship



- Aggregation: “has-a” relationship

- Composition: “is-composed-of” relationship

Unified Modeling Language Class Diagram (cont.)



A Generalization Relationship

Unified Modeling Language Class Diagram (cont.)

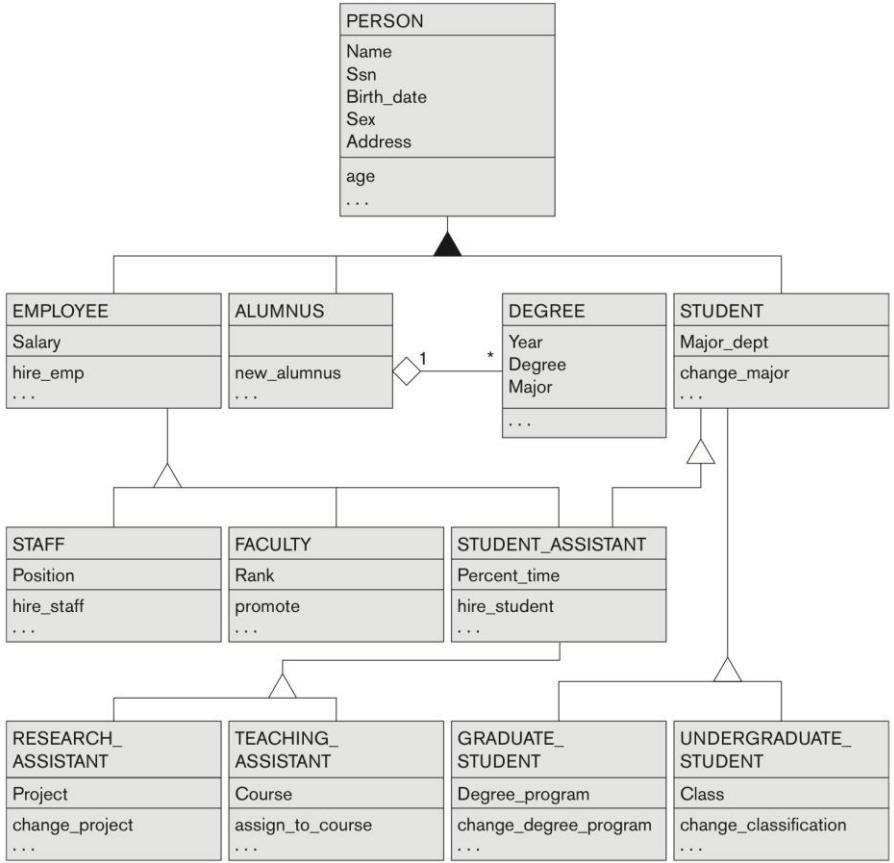


Figure 4.10 A UML class diagram corresponding to the EER diagram in Figure 4.7, illustrating UML notation for specialization / generalization.

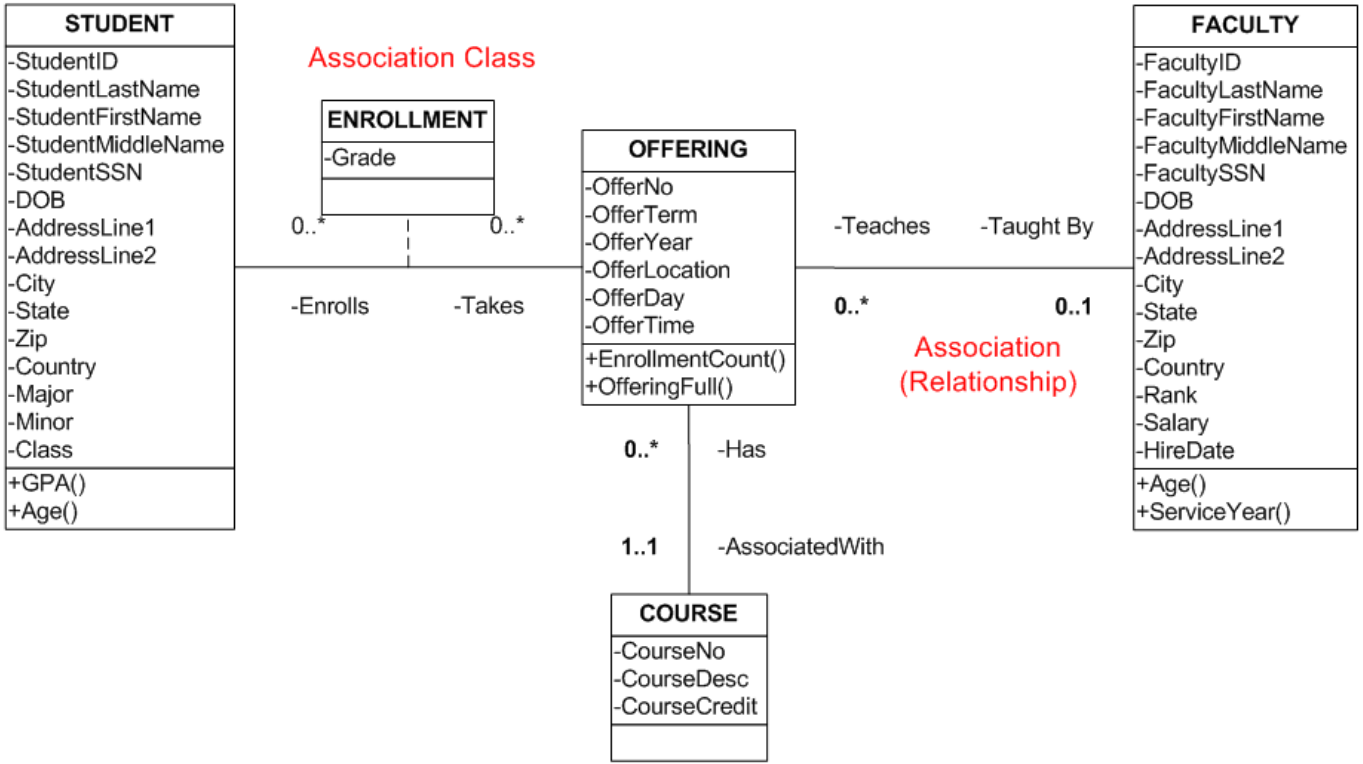
Aggregation:
ALUMNUS and DEGREE

Inheritance with overlapping constraint:
PERSON vs. EMPLOYEE, ALUMUS, STUDENT

Inheritance with disjoint constraints:
EMPLOYEE vs. STAFF, FACULTY, STUDENT_ASSISTANT

STUDENT vs. GRADUATE_STUDENT, UNDERGRADUATE_STUDENT

Unified Modeling Language Class Diagram (cont.)



More Information for OO Analysis and Design

- Most object-oriented applications involve storing the data associated with objects through data retrieval and data manipulation in databases (which will most likely be ‘relational’ or ‘object-relational’.)

- CS 605.704 Object-Oriented Analysis & Design covers the principals of OO approaches to modeling software requirements and design.

EERD and ERD Conclusions

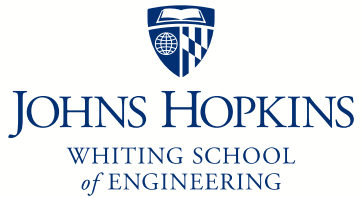
- EERD improves conceptual database design with rich notations and constraints.
  - Superclass and subclass
  - Total or partial
  - Disjoint or overlapping
- Traditional data modeling for ERDs still dominates database design.
- Database design starts/completes first before the application development.

Johns Hopkins Engineering

Principles of Database Systems

Module 5 / Lecture 1

The Relational Data Model and Relational Algebra



Relational Model Concept

- 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 Relational Model of Data is based on the concept of a Relation.
- A database is represented as a collection of relations by the relational model.

Relational Model Concept (cont.)

- A relation is a table of values (a set of tuples).
- A relation is a mathematical concept based on the ideas of sets.
- Each row (a tuple) in the table represents a collection of related data values that can be interpreted for either an entity or a relationship.
- The strength of the relational approach to data management comes from the formal foundation provided by the theory of relations.

The Formal and Informal Relational Terms

Formal Relational Term	Informal Equivalents
Relation	Table
Schema of Relation	Table Definition
Tuple	Row or Record
Cardinality	Number of Rows
Attribute	Column or Field Header
Degree	Number of Columns
Primary Key (PK)	Unique Identifier (UID)
Domain	Pool of Legal Value
Extension	Populated Table

The Formal and Informal Relational Terms (cont.)

- Textbooks present the model and operations on the relational database using the formal terms.
- Programming languages (e.g., SQL) and commercial DBMS world use the informal terms of TABLE, COLUMN, and ROW.

The Formal and Informal Relational Terms (cont.)

- A relation schema *R* consists of a relational name and a list of attributes.
- Each attribute has its own domain. A domain defines the data type and valid values of an attribute. However, several attributes may have the same domain.

The Formal and Informal Relational Terms (cont.)

- A relation intension represents the ingredient of the relation (a set of attributes.) It does not change very often.
- A relation extension reflects the state of the relation. It may change very often.

Relational Example

- **A relation SUPPLIER (a relation name)**

s_id	s_name	status	location	phone
1	Queen	Active	North	301-798-2000
2	King	Active	East	410-531-7777
3	Duke	Active	South	202-567-1234
4	Knight	Inactive	West	240-820-7600

Where:

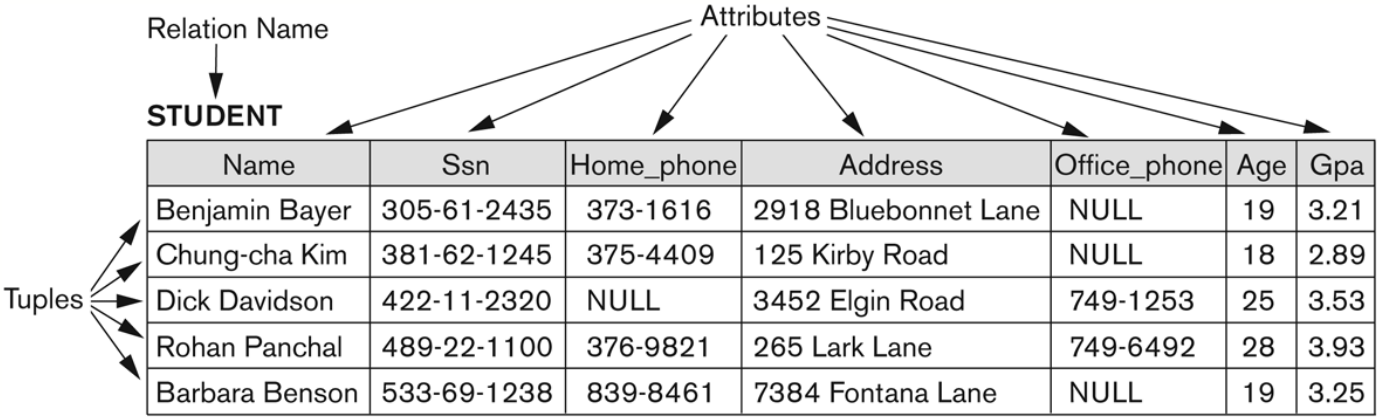
- s_id is primary key (PK);
- the set of headers of all columns: i.e., s_id, s_name, status, location, and phone are attributes;
- the set of values, i.e. (1, Queen, Active, East, 301-798-2000) is a tuple;
- Number of rows (4) is called cardinality;
- legal values for each column are called domain.

The Schema of A Relation

- The Schema of A Relation:
  - R (A_1, A_2 , ……, A_n)
  - Relation R is defined over attributes A_1 , A_2 , ……, A_n.
  - SUPPLIER is a relation defined over the five attributes s_id, s_name, status, location, phone.
  - Each value of an attribute is derived from an appropriate domain. For example, the domain of status includes ‘active’ or ‘inactive’ and s_name is defined over the domain of variable length of string with a maximum of 30 characters.

The Schema of A Relation (cont.)

- The **Schema** of A Relation:
  - Attributes in a relation are also called as columns of the table.
  - A **tuple** (row) is an ordered set of values.
  - An attribute of a relation is defined as “RelationName.AttributeName” (e.g., student.ssn) that is unique in the 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.

The Schema of A Relation (cont.)

- A relation may include a set of tuples.
- Tuples do not have order, even though they (e.g., employee_id) may have special order in a file at a physical storage or they may be presented in a logical level.
- At a logical level, the order of attributes and their values are not important.
- For performance, the order of attributes may be important regarding the physical storage.

Relational Example (cont.)

- All values in a tuple are considered *atomic* (indivisible).
  - Composite attributes
  - Multivalued attributes
- A value of NULL for an attribute in a tuple represents:
  - No value specified, an unknown value, or may not apply.

Johns Hopkins Engineering

Principles of Database Systems

Module 5 / Lecture 2

The Relational Data Model and Relational Algebra

JOHNS HOPKINS
WHITING SCHOOL
of ENGINEERING

The Relational Constraints

- Constraints are conditions that must hold on all valid relation instances

- Types of constraints:
  - Domain Constraints
  - Key Constraints and Constraints on Null
  - Entity Integrity Constraints
  - Referential Integrity Constraints

Domain Constraints

- A constraint includes data types and ranges associated with a domain.
  - The most fundamental integrity constraints applied to data in a database.
  - Data quality starts with proper data type.

Examples:

Month: 1 to 12
Year: Four digits
DOB: Date or MM/DD/YYYY
Vehicle Identification Number (VIN): 17 characters (digits and capital letters)
Last Name: Char(30) or Varchar(30)

Key Constraints

- A key constraint applies to the field with a unique identifier for each tuple. This field is called the key field. No two tuples can have the same value(s) of the key field(s).

  Example: s_id (Supplier ID) in the SUPPLIER relation

- Superkey of a relation is a set of attributes SK such that no two tuples in any valid relation instance will have the same value for SK.

- A key constraint is a minimal set of superkey by removing all non-key attributes.

Key Constraints and Constraints on Null

- Key Constraints
  - If more than one key field in a relation, each of the keys is called candidate key, i.e. ssn, emp_id in the EMP relation.
  - If a relation has several **candidate keys**, one can be chosen arbitrarily to be the **primary key**.

- Constraints on Null
  - Constraint on an attribute that may or may not accept a Null value.
    (e.g., NOT NULL for required key attributes, FKs with total constraints)

Entity Integrity Constraint

- No primary key value can be null.
- A tuple cannot be identified if the primary key is null.

  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 Constraint

- A constraint is specified between two relations and is used to maintain the consistency among tuples of the two relations.
- The primary key of a parent relation migrates to a child relation as a foreign key to maintain the referential integrity between two relations.
- Tuples in the *referencing relation* have attributes FK (called **foreign key** attributes) that reference the primary key attributes PK of the *referenced relation*.

Constraints in Practice

- All constraints are defined in a relational database schema.
- In a relational database, the data definition language (DDL) specifies all constraints.
- DBMSs maintain and enforce these constraints for all relations and all tuples of constraint attributes.

Other Ways to Enforce Constraints or Business Rules

- Triggers
  - Code stored in the database and invoked by events that occur in the application
  - Before/after insert/update/delete per row/table
- Stored Procedures or Functions
  - Code stored in the database and externally invoked by application code
- Packages
  - A collection of procedures, functions, variables, and SQL statements that are grouped together and stored as a single program unit

Relational Database Schema Example: COMPANY

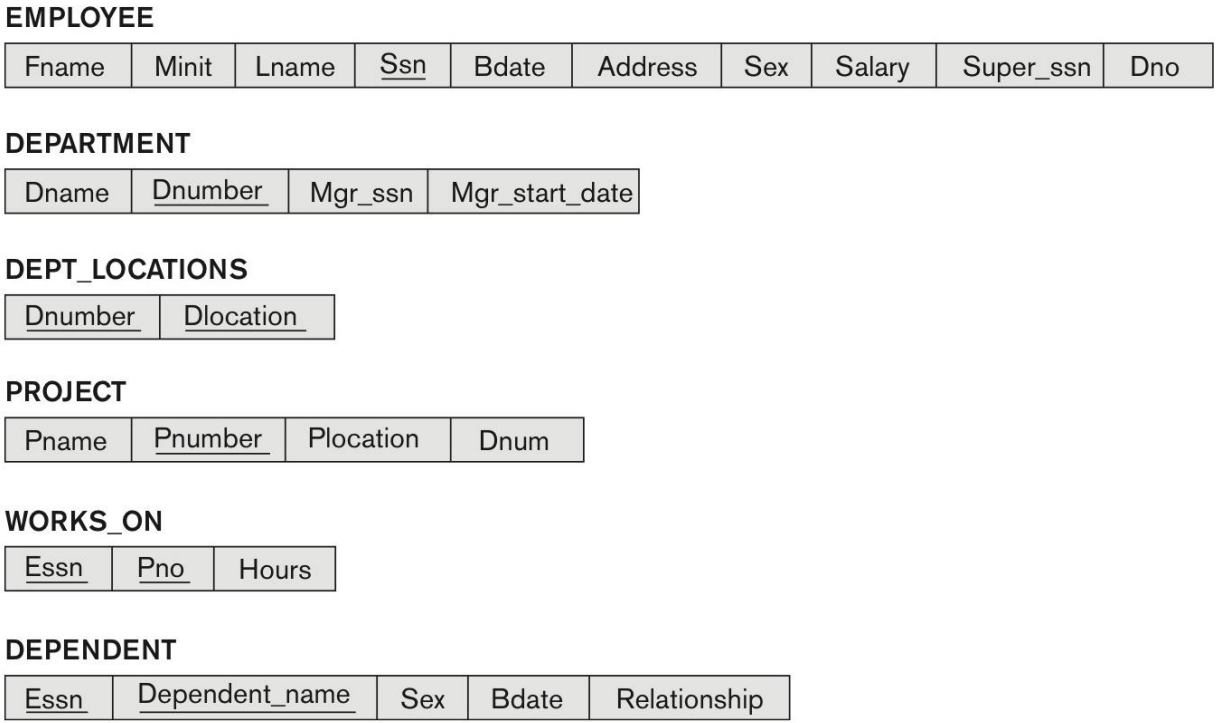


Figure 5.5 Schema diagram for the COMPANY relational database schema.

One Possible Database State for 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

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

Relational Database Schema Example: COMPANY (cont.)

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

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

Displayed Referential Integrity Constraints

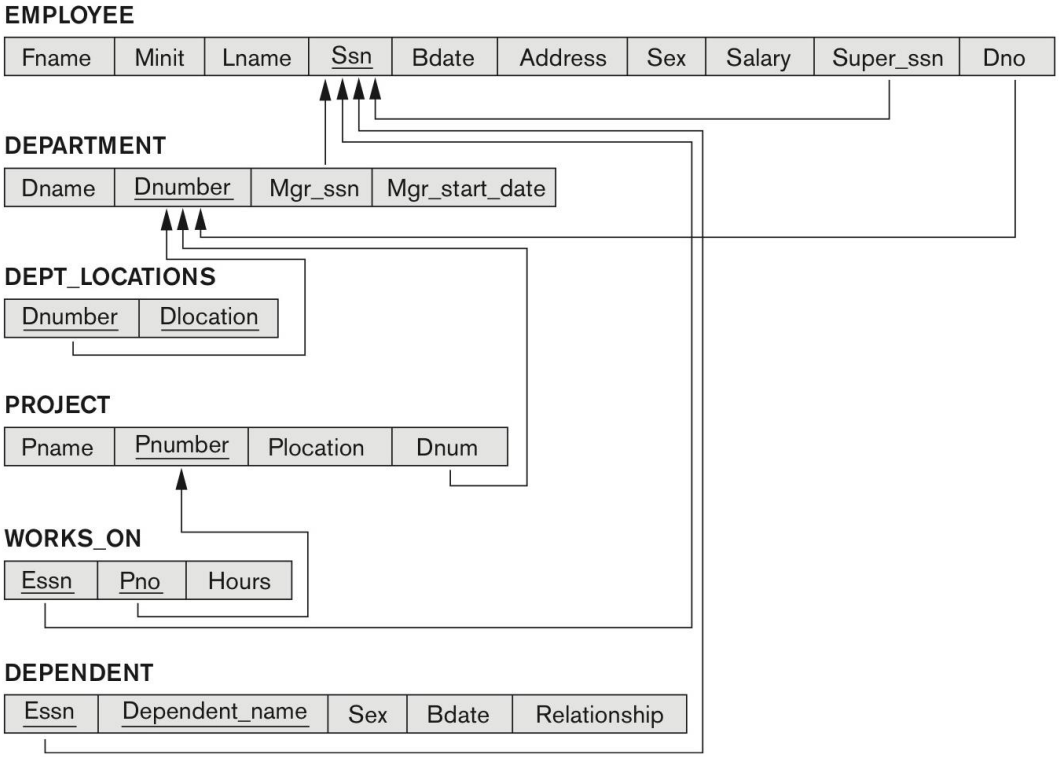


Figure 5.7 Referential integrity constraints displayed on the COMPANY relational database schema.

Operations on Relations

- Basic operations on relations include a data retrieval statement such as SELECT; and data manipulation statements such as INSERT, UPDATE, and DELETE.
- Data retrieval statements do not change the state of a database, data manipulation statements change the state of a database – always keep in a consistent state.

Operations on Relations (cont.)

- All data manipulation statements will be rejected if they violate the integrity constraints specified in the database schema implemented by data definition language.
- RDBMS enforces all constraints specified in the database schema.

Operations on Relations (cont.)

- Several update operations may have to be grouped together as a transaction and any integrity violation may result in all operations being rejected.
  - Possible actions for deleting a PK value while it has a FK referential integrity constraints:
    - Reject the operation(s) / Not allowed or RESTRICT
    - Cascade
    - Set Null
    - Set Default
  - How about updating a PK value?

Johns Hopkins Engineering

Principles of Database Systems

Module 5 / Lecture 3

The Relational Data Model and Relational Algebra

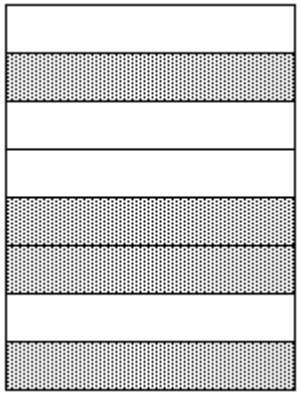
JOHNS HOPKINS
WHITING SCHOOL
of ENGINEERING

Relational Algebra

- The special relational operations are *select* (or *restrict*), *project*, *join*, and *divide*.
- The result of each operation on a relation(s) returns a new relation for further data manipulation if necessary.
- What is the algebra for?
  - In general, the expression serves as a high-level, symbolic representation of the user's intent to retrieve information from relations.
  - The algebra serves as a convenient basis for optimization based on commutative and associative properties for operators and operands.

Relational Operator SELECT

- SELECT (σ)
  - Returns a relation consisting of all tuples from a specified relation that satisfy a specified condition.



Relational Operator SELECT (cont.)

- SELECT
  - A select operation is used to choose a subset of the tuples in a relation that satisfy a selection condition.

    $\sigma_{\text{<selection condition>}}$ (<relation name>)

  - The symbol σ is used to denote the SELECT operator, and selection condition is a Boolean expression specified on the relation attributes.

Relational Operator SELECT (cont.)

- SELECT
  - The selection condition is made of a number of clauses of the form.

    <attribute name> <*comparison op*> <constant value>, or
    <attribute name> <*comparison op*> <attribute name>

  - The *comparison operator* can be (>, >=, =, <, <=). Clauses can be arbitrarily connected by the logical operators *AND*, *OR*, and *NOT* for form compound conditions.

    Oracle provide additional {*BETWEEN ...AND ...; NOT BETWEEN ... AND ...; IN; NOT IN; LIKE; IS NULL; IS NOT NULL*}.

Relational Operator SELECT (cont.)

- SELECT
  - The SELECT operation is commutative.

    $\sigma_{\text{<cond1>}}(\sigma_{\text{<cond2>}}(R)) = \sigma_{\text{<cond2>}}(\sigma_{\text{<cond1>}}(R))$

    $\sigma_{\text{<cond1>}}(\sigma_{\text{<cond2>}}(R)) = \sigma_{\text{<cond1> AND <cond2>}}(R))$

    Example:

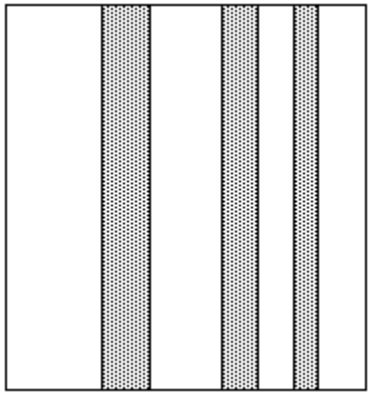
    $\sigma_{\text{DNO=4}}(\text{EMPLOYEE})$

    $\sigma_{\text{SALARY>30000}}(\text{EMPLOYEE})$

    $\sigma_{\text{(DNO=4 AND SALARY>25000) OR DNO=5}}(\text{EMPLOYEE})$

Relational Operator PROJECT

- PROJECT (π)
  - Returns a relation consisting of specified attributes and eliminating duplicates as a vertical subset of the original relation.



Relational Operator Interpretation (cont.)

- PROJECT
  - A project operation is used to choose columns in a relation or keeps only certain columns.

    $\pi_{<\textit{attribute list}>}$ (<relation name>)

  - The resulting relation has only the attributes specified in <*attribute list*> and in the same order as they appear in the list.

Relational Operator Interpretation (cont.)

- PROJECT
  - The PROJECT operation *eliminates duplicate tuples* in the resulting relation so that it remains a mathematical set.

  - The PROJECT is not commutative.
    Example:

    $\pi_{FNAME,LNAME,SALARY}$ (EMPLOYEE)

    $\pi_{ADDRESS}$ (EMPLOYEE)

Relational Operator σ, π Interpretations

(a)

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5
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

(b)

Lname	Fname	Salary
Smith	John	30000
Wong	Franklin	40000
Zelaya	Alicia	25000
Wallace	Jennifer	43000
Narayan	Ramesh	38000
English	Joyce	25000
Jabbar	Ahmad	25000
Borg	James	55000

(c)

Sex	Salary
M	30000
M	40000
F	25000
F	43000
M	38000
M	25000
M	55000

Figure 8.1 Results of SELECT and PROJECT operations. (a) $\sigma_{(Dno=4\ \textbf{AND}\ Salary>25000)\ OR\ (Dno=5\ \textbf{AND}\ Salary>30000)}(EMPLOYEE)$. (b) $\pi_{Lname,\ Fname,\ Salary}(EMPLOYEE)$. (c) $\pi_{Sex,\ Salary}(EMPLOYEE)$.

Relational Operator σ, π Interpretations (cont.)

- A *relational algebra expression* (query) may consist of several operations.

  Example: Retrieve the first names, last names and salaries of employees who work in department 4:

  $\pi_{FNAME,LNAME,SALARY}(\sigma_{DNO=4}(EMPLOYEE))$

  Alternatively, explicit intermediate relations for each step (e.g., $\sigma_{DNO=4}(EMPLOYEE)$)

Relational Operator σ, π Interpretations (cont.)

(a)

Fname	Lname	Salary
John	Smith	30000
Franklin	Wong	40000
Ramesh	Narayan	38000
Joyce	English	25000

(b)

TEMP

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
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

R

First_name	Last_name	Salary
John	Smith	30000
Franklin	Wong	40000
Ramesh	Narayan	38000
Joyce	English	25000

Figure 8.2 Results of a sequence of operations. (a) $\pi_{Fname,\ Lname,\ Salary}(\sigma_{Dno=5}(EMPLOYEE))$. (b) Using intermediate relations and renaming of attributes.

Johns Hopkins Engineering

Principles of Database Systems

Module 5 / Lecture 4

The Relational Data Model and Relational Algebra

JOHNS HOPKINS
WHITING SCHOOL
of ENGINEERING

Relational Operator Set Operations

- Set Operations

  UNION (∪),

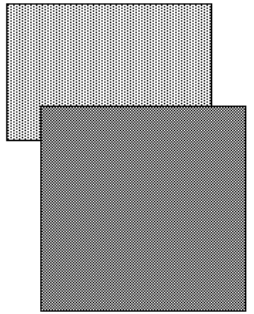
  INTERSECTION (∩),

  DIFFERENCE (–), and

  CARTESIAN PRODUCT (X)

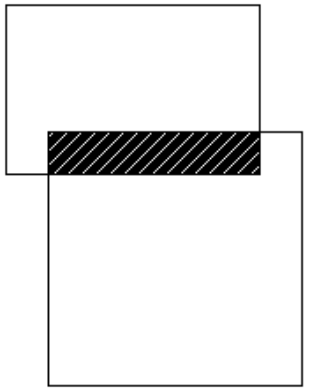
Relational Operator UNION

- UNION (∪)
  - Returns a relation consisting of all tuples appearing in either or both of two specified relations.
  - Both relations are union compatible
    - have the same number of attributes
    - have same domains of each attributes
  - UNION is a commutative operation.
  - The UNION operation "∪" eliminates the duplicate tuples.



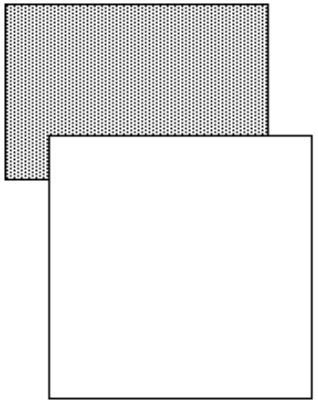
Relational Operator INTERSECT

- INTERSECT (∩)
  - Returns a relation consisting of all tuples appearing in both of two specified relations.
  - The INTERSECTION operation "∩" include all tuples that in both relations.
  - INTERSECTION is a commutative operation.

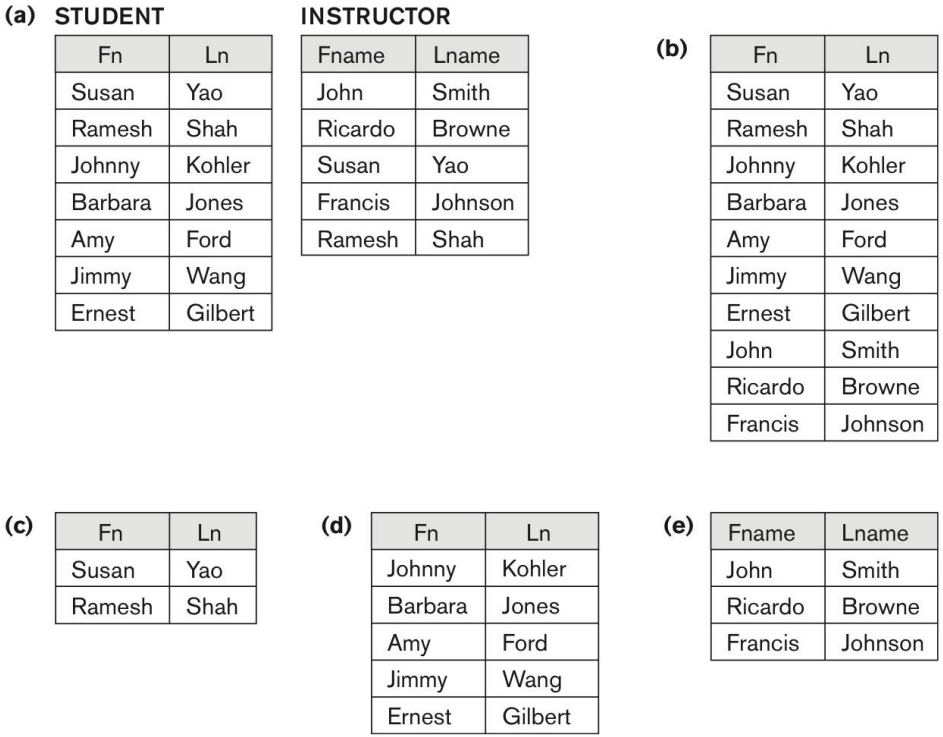


Relational Operator DIFFERENCE

- DIFFERENCE or MINUS (–)
  - Returns a relation consisting of all tuples appearing in the first and not the second of two specified relations.
  - DIFFERENCE operation is not a commutative operation.



Relational SET Operators Interpretation



(a) STUDENT

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

INSTRUCTOR

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

(b)

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

(c)

Fn	Ln
Susan	Yao
Ramesh	Shah

(d)

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

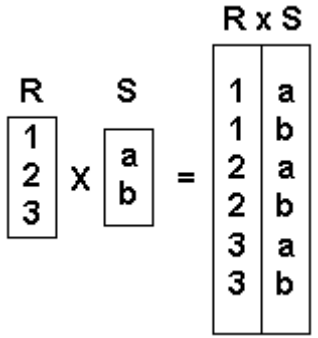
(e)

Fname	Lname
John	Smith
Ricardo	Browne
Francis	Johnson

Figure 8.4 The set operations UNION, INTERSECTION, and MINUS. (a) Two union-compatible relations. (b) STUDENT ∪ INSTRUCTOR. (c) STUDENT ∩ INSTRUCTOR. (d) STUDENT – INSTRUCTOR. (e) INSTRUCTOR – STUDENT.

Relational Operator CARTESIAN PRODUCT

- CARTESIAN PRODUCT (X)
  - Produces a relation that has all possible combinations of tuples of attributes of participated relations.
  - This operation “X” usually generates a big relation without much meaning. However, this operation works with the SELECT operation that can create a meaningful and desired relation.



Relational Operator CARTESIAN PRODUCT (cont.)

FEMALE_EMPS

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
Alicia	J	Zelaya	999887777	1968-07-19	3321Castle, Spring, TX	F	25000	987654321	4
Jennifer	S	Wallace	987654321	1941-06-20	291Berry, Bellaire, TX	F	43000	888665555	4
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5

EMPNAMES

Fname	Lname	Ssn
Alicia	Zelaya	999887777
Jennifer	Wallace	987654321
Joyce	English	453453453

Example: Retrieve a list of names of each female employee's dependents:

Steps 1 and 2:

FEMALE_EMPS $\leftarrow \sigma_{Sex='F'}$ (EMPLOYEE)

EMPNAMES $\leftarrow \pi_{Fname,Lname,Ssn}$ (FEMALE_EMPS)

Figure 8.5 The CARTESIAN PRODUCT (CROSS PRODUCT) operation.

Relational Operator CARTESIAN PRODUCT (cont.)

EMP_DEPENDENTS

Fname	Lname	Ssn	Essn	Dependent_name	Sex	Bdate	...
Alicia	Zelaya	999887777	333445555	Alice	F	1986-04-05	...
Alicia	Zelaya	999887777	333445555	Theodore	M	1983-10-25	...
Alicia	Zelaya	999887777	333445555	Joy	F	1958-05-03	...
Alicia	Zelaya	999887777	987654321	Abner	M	1942-02-28	...
Alicia	Zelaya	999887777	123456789	Michael	M	1988-01-04	...
Alicia	Zelaya	999887777	123456789	Alice	F	1988-12-30	...
Alicia	Zelaya	999887777	123456789	Elizabeth	F	1967-05-05	...
Jennifer	Wallace	987654321	333445555	Alice	F	1986-04-05	...
Jennifer	Wallace	987654321	333445555	Theodore	M	1983-10-25	...
Jennifer	Wallace	987654321	333445555	Joy	F	1958-05-03	...
Jennifer	Wallace	987654321	987654321	Abner	M	1942-02-28	...
Jennifer	Wallace	987654321	123456789	Michael	M	1988-01-04	...
Jennifer	Wallace	987654321	123456789	Alice	F	1988-12-30	...
Jennifer	Wallace	987654321	123456789	Elizabeth	F	1967-05-05	...
Joyce	English	453453453	333445555	Alice	F	1986-04-05	...
Joyce	English	453453453	333445555	Theodore	M	1983-10-25	...
Joyce	English	453453453	333445555	Joy	F	1958-05-03	...
Joyce	English	453453453	987654321	Abner	M	1942-02-28	...
Joyce	English	453453453	123456789	Michael	M	1988-01-04	...
Joyce	English	453453453	123456789	Alice	F	1988-12-30	...
Joyce	English	453453453	123456789	Elizabeth	F	1967-05-05	...

ACTUAL_DEPENDENTS

Fname	Lname	Ssn	Essn	Dependent_name	Sex	Bdate	...
Jennifer	Wallace	987654321	987654321	Abner	M	1942-02-28	...

RESULT

Fname	Lname	Dependent_name
Jennifer	Wallace	Abner

Example: Retrieve a list of names of each female employee's dependents: Steps 3, 4 and 5:

EMP_DEPENDENTS ← EMPNAMES **X** DEPENDENT

ACTUAL_DEPENDENTS ← $\sigma_{Ssn=Essn}$ (EMP_DEPENDENTS)

RESULT ← $\pi_{Fname,Lname,Dependent\_name}$ (ACTUAL_DEPENDENTS)

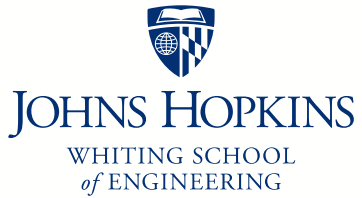
Figure 8.5 (continued) The CARTESIAN PRODUCT (CROSS PRODUCT) operation.

Johns Hopkins Engineering

Principles of Database Systems

Module 5 / Lecture 5

The Relational Data Model and Relational Algebra



Relational Operator JOIN

- JOIN
  - Returns a relation consisting of all possible tuples that are a combination of two tuples, one from each of two specified relations, such that the two tuples contributing to any given combination have a common value for the common attribute(s) of the two relations.

Relational Operator JOIN (cont.)

- JOIN
  - This operation merges two or more relations into one relation.
  - The JOIN operation combined with other operations for relations can process entities and relationships in a relational database.

- JOIN Types
  - THETA JOIN, EQUIJOIN, NATURAL JOIN

Relational Operator THETA JOIN

- THETA JOIN (⋈)
  - Similar to a CARTESIAN PRODUCT followed by a SELECT. It has a *join condition*.

    $R(A_1, A_2, \ldots, A_m, B_1, B_2, \ldots, B_n)$
    $\leftarrow R_1(A_1, A_2, \ldots, A_m) \bowtie_{\text{<join condition>}} R_2 (B_1, B_2, \ldots, B_n)$

    Join condition is one of the comparison operators {=, <, <=, >, >=, <>}

Relational Operator EQUIJOIN

- EQUIJOIN (e.g., R ← R1 $\bowtie_{<join\ condition>}$ R2)
  - The most common join operation is called EQUIJOIN "⋈" that is to generate a relation having one or more pairs of attributes that have identical values in every tuple.
  - The *join condition* includes one or more equality comparisons involving attributes from R1 and R2. (A special case of THETA JOIN.)
  - In an EQUIJOIN, the join contains a redundant attribute(s) in the result relation R.

Relational Operator EQUIJOIN (cont.)

- EQUIJOIN
  - Example:

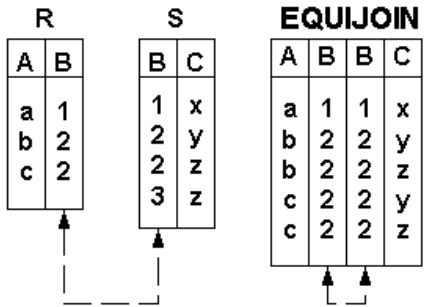
    Retrieve all EMPLOYEEs and their DEPENDENT information:

    $ACTUAL\_DEPENDENT \leftarrow EMP \bowtie_{SSN=ESSN} DEPENDENT$

    Retrieve each DEPARTMENT's name and its manager's name:

    $TEMP \leftarrow DEPARTMENT \bowtie_{MGRSSN=SSN} EMPLOYEE$

    $RESULT \leftarrow \pi_{DNAME,FNAME,LNAME} (TEMP)$



Relational Operator EQUIJOIN (cont.)

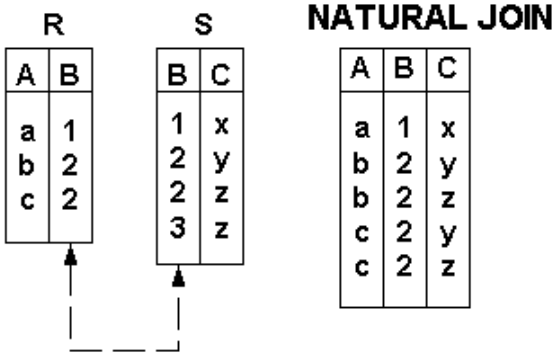
- EQUIJOIN
  - Example: Retrieve each EMPLOYEE and his/her DEPARTMENT's name and its location:

EMPLOYEE |X| DEPTNO=DEPTNO DEPARTMENT

EMPNO	ENAME	MGR	DEPTNO	DEPTNO	DNAME	LOC
7369	SMITH	7902	20	20	RESEARCH	DALLAS
7499	ALLEN	7698	30	30	SALES	CHICAGO
7521	WARD	7698	30	30	SALES	CHICAGO
7566	JONES	7839	20	20	RESEARCH	DALLAS
7654	MARTIN	7698	30	30	SALES	CHICAGO
7698	BLAKE	7839	30	30	SALES	CHICAGO
7782	CLARK	7839	10	10	ACCOUNTING	NEW YORK
7788	SCOTT	7566	20	20	RESEARCH	DALLAS
7839	KING		10	10	ACCOUNTING	NEW YORK
7844	TURNER	7698	30	30	SALES	CHICAGO
7876	ADAMS	7788	20	20	RESEARCH	DALLAS
7900	JAMES	7698	30	30	SALES	CHICAGO
7902	FORD	7566	20	20	RESEARCH	DALLAS
7934	MILLER	7782	10	10	ACCOUNTING	NEW YORK

Relational Operator NATURAL JOIN

- NATURAL JOIN
  - The same operation as EQUIJOIN except that the redundant attribute(s) of the second relation is excluded in the resulting relation. In fact, this is the most preferable/desirable relation for a join operation.



Relational Operator NATURAL JOIN (cont.)

- The degree of a NATURAL JOIN is the sum of the degrees of the two relations minus the number of attributes in the EQUIJOIN condition.
- Example: Retrieve each EMPLOYEE's name and the name of the DEPARTMENT he/she works for:

Using NATURAL JOIN

TEMP $\leftarrow$ EMPLOYEE $*_{\rho(\ldots,\ Dno)}$ DEPARTMENT (Note: ρ "rename" Dnumber)

RESULT $\leftarrow \pi_{FNAME,LNAME,DNAME}$ (TEMP)

Using EQUIJOIN

TEMP $\leftarrow$ EMPLOYEE $\bowtie_{(Dno)=(Dnumber)}$ DEPARTMENT

RESULT $\leftarrow \pi_{FNAME,LNAME,DNAME}$ (TEMP)

Relational Operator NATURAL JOIN (cont.)

(a)

PROJ_DEPT

Pname	Pnumber	Plocation	Dnum	Dname	Mgr_ssn	Mgr_start_date
ProductX	1	Bellaire	5	Research	333445555	1988-05-22
ProductY	2	Sugarland	5	Research	333445555	1988-05-22
ProductZ	3	Houston	5	Research	333445555	1988-05-22
Computerization	10	Stafford	4	Administration	987654321	1995-01-01
Reorganization	20	Houston	1	Headquarters	888665555	1981-06-19
Newbenefits	30	Stafford	4	Administration	987654321	1995-01-01

(b)

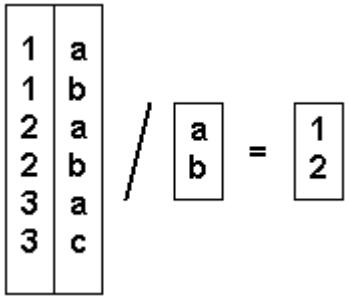
DEPT_LOCS

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

Figure 8.7 Results of two natural join operations. (a) proj_dept ← project * dept. (b) dept_locs ← department * dept_locations.

Relational Operator DIVISION

- DIVISION
  - Defines a relation over the attributes that takes two relations, and a first relation match the combination of every tuple in the second relation.

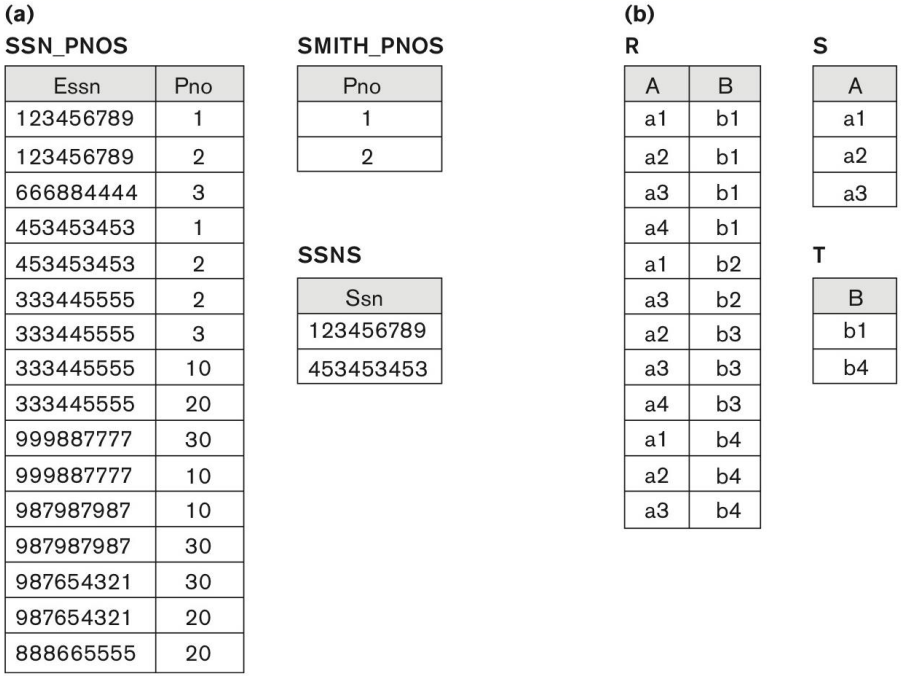


Relational Operator DIVISION (cont.)

- The complete set of relational algebra operations {**σ, π, ∪, –, ×**}. Any other relation algebra operations can be represented as a sequence of operations from this set.

- The DIVISION operator can be expressed as a sequence of **π, x,** and **–** operations as follows:
  - $\mathbf{T_1 \leftarrow \pi_Y(R)}$
  - $\mathbf{T_2 \leftarrow \pi_Y((S \times T_1) - R)}$
  - $\mathbf{T \leftarrow T_1 - T_2}$

Relational Operator DIVISION (cont.)



(a)

SSN_PNOS

Essn	Pno
123456789	1
123456789	2
666884444	3
453453453	1
453453453	2
333445555	2
333445555	3
333445555	10
333445555	20
999887777	30
999887777	10
987987987	10
987987987	30
987654321	30
987654321	20
888665555	20

SMITH_PNOS

Pno
1
2

SSNS

Ssn
123456789
453453453

(b)

R

A	B
a1	b1
a2	b1
a3	b1
a4	b1
a1	b2
a3	b2
a2	b3
a3	b3
a4	b3
a1	b4
a2	b4
a3	b4

S

A
a1
a2
a3

T

B
b1
b4

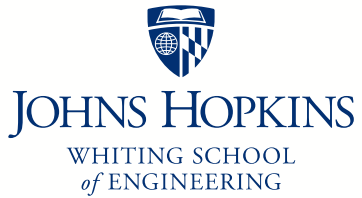
Figure 8.8 The DIVISION operation. (a) Dividing SSN_PNOS by SMITH_PNOS. (b) T ← R ÷ S.

Johns Hopkins Engineering

Principles of Database Systems

Module 5 / Lecture 6

The Relational Data Model and Relational Algebra



Operations of Relational Algebra

- The bottom line to form a complex relational algebra condition is to break the list into pieces and test each one in sequence.
- The relational algebra uses one declarative expression to specify a retrieval request while a sequence of operations may be needed for an equivalent relation algebra.
- The relational algebra expresses what to be retrieved instead of how to retrieve.

Operations of Relational Algebra (cont.)

Table 8.1 Operations of Relational Algebra

OPERATION	PURPOSE	NOTATION
SELECT	Selects all tuples that satisfy the selection condition from a relation R.	$\sigma_{\text{<selection condition>}}(R)$
PROJECT	Produces a new relation with only some of the attributes of R, and removes duplicate tuples.	$\pi_{\text{<attribute list>}}(R)$
THETA JOIN	Produces all combinations of tuples from R_1 and R_2 that satisfy the join condition.	$R_1 \bowtie_{\text{<join condition>}} R_2$
EQUIJOIN	Produces all the combinations of tuples from R_1 and R_2 that satisfy a join condition with only equality comparisons.	$R_1 \bowtie_{\text{<join condition>}} R_2$, OR $R_1 \bowtie_{\text{(<join attributes 1>), (<join attributes 2>)}} R_2$
NATURAL JOIN	Same as EQUIJOIN except that the join attributes of R_2 are not included in the resulting relation; if the join attributes have the same names, they do not have to be specified at all.	$R_1 *_{\text{<join condition>}} R_2$, OR $R_1 *_{\text{(<join attributes 1>), (<join attributes 2>)}} R_2$ OR $R_1 * R_2$

Operations of Relational Algebra (cont.)

Table 8.1 Operations of Relational Algebra

OPERATION	PURPOSE	NOTATION
UNION	Produces a relation that includes all the tuples in R_1 or R_2 or both R_1 and R_2; R_1 and R_2 must be union compatible.	$R_1 \cup R_2$
INTERSECTION	Produces a relation that includes all the tuples in both R_1 and R_2; R_1 and R_2 must be union compatible.	$R_1 \cap R_2$
DIFFERENCE	Produces a relation that includes all the tuples in R_1 that are not in R_2; R_1 and R_2 must be union compatible.	$R_1 - R_2$
CARTESIAN PRODUCT	Produces a relation that has the attributes of R_1 and R_2 and includes as tuples all possible combinations of tuples from R_1 and R_2.	$R_1 \times R_2$
DIVISION	Produces a relation $R(X)$ that includes all tuples $t[X]$ in $R_1(Z)$ that appear in R_1 in combination with every tuple from $R_2(Y)$, where $Z = X \cup Y$.	$R_1(Z) \div R_2(Y)$

Notation for Query Trees

- A query can be expressed in a tree structure (a query tree) with various steps of data retrieval using relation algebra.

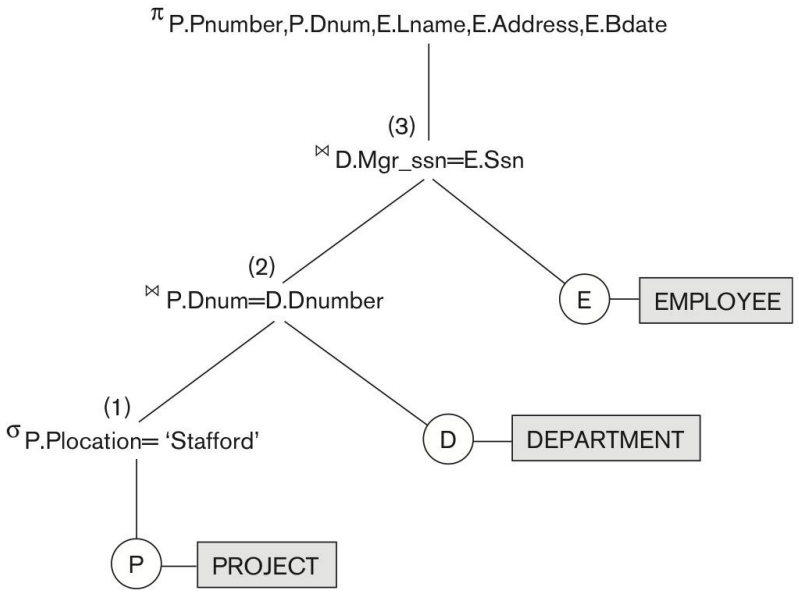


Figure 8.9 Query tree corresponding to the relational algebra expression for Q2.

Additional Relational Operations

- For developing database applications, additional operations are needed that were not part of the *original* relational algebra. These include:
  - Aggregate functions and grouping
  - Recursive Closure Operations
  - OUTER JOIN and OUTER UNION

Aggregate Functions

- Functions such as SUM, COUNT, AVERAGE, MINIMUM, MAXIMUN are often applied to sets of values or sets of tuples in database applications.

- The grouping attributes may be applied as needed.

  <grouping attributes> ***F*** *<function list>* (R)

Aggregate Functions (cont.)

- Aggregate Function (ℑ or F) Example:
  Retrieve the average salary of all employees
  R(AVGSAL) ← F *AVERAGE SALARY* (EMPLOYEE)

  For each department, retrieve the department number, the number of employees, and the average salary in the department. The grouping is needed and DNO is called the grouping attribute:
  R(DNO,NUMEMPS,AVGSAL) ←
  DNO F *COUNT SSN*, *AVERAGE SALARY* (EMPLOYEE)

Aggregate Functions (cont.)

(a) R

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

(b)

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

(c)

Count_ssn	Average_salary
8	35125

Figure 8.8 The aggregate function operation. a. $\rho_{R(\text{Dno, No\_of\_employees, Average\_sal})}(\text{Dno } \Im \text{ COUNT Ssn, AVERAGE Salary } (\text{EMPLOYEE}))$. b. $\text{Dno } \Im \text{ COUNT Ssn, AVERAGE Salary}(\text{EMPLOYEE})$. c. $\Im \text{ COUNT Ssn, AVERAGE Salary}(\text{EMPLOYEE})$.

Recursive Closure Operations

- Recursive closure operations apply to a recursive relationship (employee and supervisor).
  - Retrieve all employees (directly or indirectly) supervised by an employee
  - Can be implemented in SQL as hierarchical queries

OUTER JOIN

- In a regular EQUIJOIN or NATURAL JOIN operation, tuples in R_1 or R_2 that do not have matching tuples in the other relation do *NOT* appear in the result.
- Some queries require all tuples in R1 (or R2 or both) to appear in the result.
- When no matching tuples are found, **nulls** are placed for the missing attributes.

OUTER JOIN (cont.)

- LEFT OUTER JOIN:

  R1 ⟕ R2 lets every tuple in R1 appear in the result.

- RIGHT OUTER JOIN:

  R1 ⟖ R2 lets every tuple in R2 appear in the result.

- FULL OUTER JOIN:

  R1 ⟗ R2 lets every tuple in R1 and R2 appear in the result, padding tuples with nulls when no matching tuples are found.

OUTER JOIN (cont.)

- **Examples:**
  - LEFT (NATURAL) OUTER JOIN: R1 ⟕ R2
  - RIGHT (NATURAL) OUTER JOIN: R1 ⟖ R2
  - FULL (NATURAL) OUTER JOIN: R1 ⟗ R2

R

A	B
a	1
b	2
c	2
d	5

S

B	C
1	x
2	y
2	z
3	z
4	q

LEFT NATURAL OUTER JOIN

A	B	C
a	1	x
b	2	y
b	2	z
c	2	y
c	2	z
d	5	

RIGHT NATURAL OUTER JOIN

A	B	C
a	1	x
b	2	y
b	2	z
c	2	y
c	2	z
	3	z
	4	q

FULL NATURAL OUTER JOIN

A	B	C
a	1	x
b	2	y
b	2	z
c	2	y
c	2	z
d	5	
	3	z
	4	q

OUTER JOIN (cont.)

- **Example:**

RESULT $\leftarrow \pi_{\text{Fname, Minit, Lname, Dname}}$ (EMPLOYEE $\bowtie_{\text{Ssn=Mgr\_ssn}}$ DEPARTMENT)

RESULT

Fname	Minit	Lname	Dname
John	B	Smith	NULL
Franklin	T	Wong	Research
Alicia	J	Zelaya	NULL
Jennifer	S	Wallace	Administration
Ramesh	K	Narayan	NULL
Joyce	A	English	NULL
Ahmad	V	Jabbar	NULL
James	E	Borg	Headquarters

Figure 8.12 The result of a LEFT OUTER JOIN operation.