COS221
L01 - Overview to Databases
(Chapter 1 - Editions 6 and 7)

Linda Marshall

22 February 2023

Databases and Database Users

Why are databases needed?

Database types

- Traditional
- Multimedia
- Geographic
- Data warehouse and Online Analytical Processing (OLAP)
- Real-time and Active
- ...

Definitions

1. A *database* is a collection of data.
2. *Data* is recorded known facts with implicit meaning.
3. *Database Management System (DBMS)* is a collection of programs used to create and maintain a database.

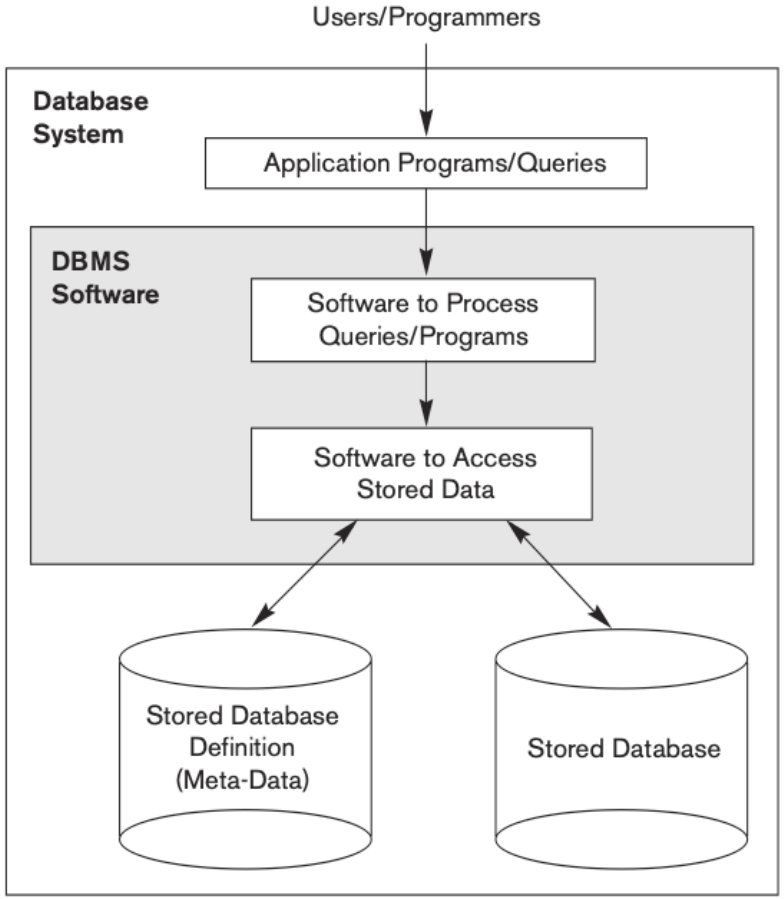
Definitions - Database

- A database comprises of *files* which stores *data records* comprising of *data elements*.
- implicit properties of a data base:
  - representation of an aspect of the real world (mini-world)
  - logically coherent data with implicit meaning
  - designed, built and populated with related data

Definitions - Database Management Systems (DBMS)

- Defines datatypes, structures and constraints of the data in the database
- Provides a construction process to store data on storage media
  - An *application program* sends queries or requests to the DBMS
  - A *query* is used to retrieve data
  - A*transaction* reads and writes data to the DBMS
- Querying and updating the database to reflect changes in the mini world
- Generating report results in the *manipulation* of the database
- By *sharing* the database, multiple users can access and manipulate the database simultaneously
- *Protects* against malfunction (hardware and software) and malicious access
- *Maintains* the system by allowing it to evolve over years

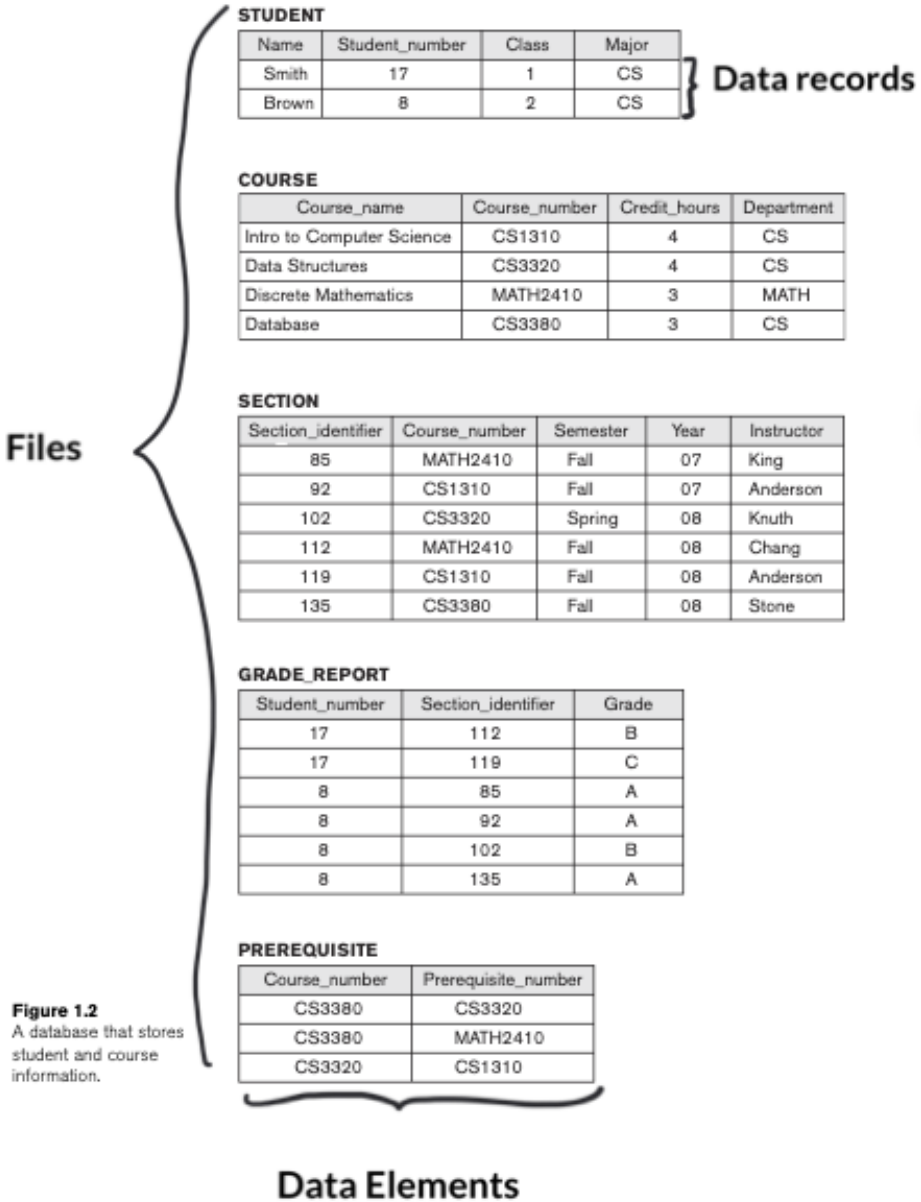
Definitions - Database Management Systems (DBMS)



Database + DBMS = Database System

Figure 1.1
A simplified database system environment.

An Example



STUDENT

Name	Student_number	Class	Major
Smith	17	1	CS
Brown	8	2	CS

COURSE

Course_name	Course_number	Credit_hours	Department
Intro to Computer Science	CS1310	4	CS
Data Structures	CS3320	4	CS
Discrete Mathematics	MATH2410	3	MATH
Database	CS3380	3	CS

SECTION

Section_identifier	Course_number	Semester	Year	Instructor
85	MATH2410	Fall	07	King
92	CS1310	Fall	07	Anderson
102	CS3320	Spring	08	Knuth
112	MATH2410	Fall	08	Chang
119	CS1310	Fall	08	Anderson
135	CS3380	Fall	08	Stone

GRADE_REPORT

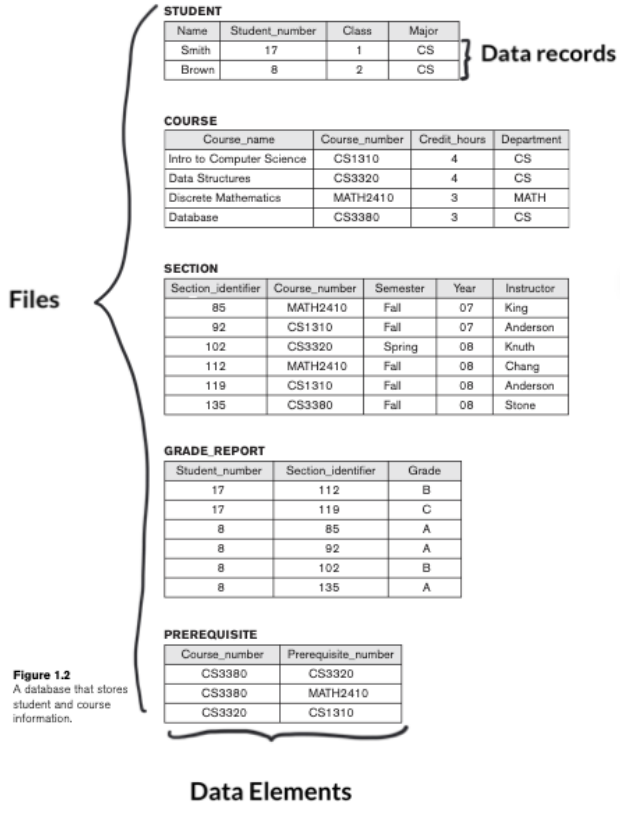
Student_number	Section_identifier	Grade
17	112	B
17	119	C
8	85	A
8	92	A
8	102	B
8	135	A

PREREQUISITE

Course_number	Prerequisite_number
CS3380	CS3320
CS3380	MATH2410
CS3320	CS1310

Figure 1.2
A database that stores student and course information.

An Example



STUDENT

Name	Student_number	Class	Major
Smith	17	1	CS
Brown	8	2	CS

Data records

COURSE

Course_name	Course_number	Credit_hours	Department
Intro to Computer Science	CS1310	4	CS
Data Structures	CS3320	4	CS
Discrete Mathematics	MATH2410	3	MATH
Database	CS3380	3	CS

Files

SECTION

Section_identifier	Course_number	Semester	Year	Instructor
85	MATH2410	Fall	07	King
92	CS1310	Fall	07	Anderson
102	CS3320	Spring	08	Knuth
112	MATH2410	Fall	08	Chang
119	CS1310	Fall	08	Anderson
135	CS3380	Fall	08	Stone

GRADE_REPORT

Student_number	Section_identifier	Grade
17	112	B
17	119	C
8	85	A
8	92	A
8	102	B
8	135	A

PREREQUISITE

Course_number	Prerequisite_number
CS3380	CS3320
CS3380	MATH2410
CS3320	CS1310

Data Elements

Figure 1.2
A database that stores student and course information.

- Each *data element* has a *data type* associated with it
- *Relationships* exist between *data records*

An Example - Meta-data

RELATIONS

Relation_name	No_of_columns
STUDENT	4
COURSE	4
SECTION	5
GRADE_REPORT	3
PREREQUISITE	2

COLUMNS

Column_name	Data_type	Belongs_to_relation
Name	Character (30)	STUDENT
Student_number	Character (4)	STUDENT
Class	Integer (1)	STUDENT
Major	Major_type	STUDENT
Course_name	Character (10)	COURSE
Course_number	XXXXNNNN	COURSE
....	....	.....
....	....	.....
....	....	.....
Prerequisite_number	XXXXNNNN	PREREQUISITE

Note: Major_type is defined as an enumerated type with all known majors.
XXXXNNNN is used to define a type with four alpha characters followed by four digits.

Figure 1.3
An example of a database catalog for the database in Figure 1.2.

An Example - Meta-data

RELATIONS

Relation_name	No_of_columns
STUDENT	4
COURSE	4
SECTION	5
GRADE_REPORT	3
PREREQUISITE	2

COLUMNS

Column_name	Data_type	Belongs_to_relation
Name	Character (30)	STUDENT
Student_number	Character (4)	STUDENT
Class	Integer (1)	STUDENT
Major	Major_type	STUDENT
Course_name	Character (10)	COURSE
Course_number	XXXXNNNN	COURSE
....	....	.....
....	....	.....
....	....	.....
Prerequisite_number	XXXXNNNN	PREREQUISITE

Note: Major_type is defined as an enumerated type with all known majors.
XXXXNNNN is used to define a type with four alpha characters followed by four digits.

Figure 1.3
An example of a database catalog for the database in Figure 1.2.

- *Meta-data* is stored in the *catalog*
- Using a *catalog* promotes *program-data independence*
- *Program-data independence* is as a result of *data abstraction*
- A DBMS provides users with a *conceptual representation*

An Example - Meta-data

Data Item Name	Starting Position in Record	Length in Characters (bytes)
Name	1	30
Student_number	31	4
Class	35	1
Major	36	4

Figure 1.4
Internal storage format for a STUDENT record, based on the database catalog in Figure 1.3.

- A data model is a data abstraction that is used to provide the conceptual representation and hides the internal representation.

An Example - Views

(a)

TRANSCRIPT

Student_name	Student_transcript				
	Course_number	Grade	Semester	Year	Section_id
Smith	CS1310	C	Fall	08	119
	MATH2410	B	Fall	08	112
Brown	MATH2410	A	Fall	07	85
	CS1310	A	Fall	07	92
	CS3320	B	Spring	08	102
	CS3380	A	Fall	08	135

(b)

COURSE_PREREQUISITES

Course_name	Course_number	Prerequisites
Database	CS3380	CS3320
		MATH2410
Data Structures	CS3320	CS1310

Figure 1.5
Two views derived from the database in Figure 1.2. (a) The TRANSCRIPT view. (b) The COURSE_PREREQUISITES view.

- Multiple views can be derived from the database.
- A view may contain data derived from the database but not explicitly stored. This is referred to as *virtual data*.

Multiuser support

- DBMS must include concurrency control, especially for OnLine Transaction Processing (OLTP) type applications, e.g. aircraft ticket reservations
- The DBMS enforces the following properties on transactions
  - *isolation* - all transactions appear to execute in isolation
  - *atomicity* - all transactions execute or none do

Actors on the Scene

- Database Administrator
- Database Designers
  - Requirements specification and analysis
  - Conceptual design (e.g. ER diagrams)
  - Logical design (e.g. Relational model)
  - Physical design (implementation of the design in a DBMS)
- End Users
  - casual
  - naive/parametric
  - sophisticated
  - standalone
- Systems Analysts and application programmers - use the database

Workers behind the Scene

- DBMS designers and implementers
- Tool developers
- Operators and maintenance - typically do not use the database

Advantages of using a DBMS

- Controlling redundancy - Normalisation, denormalisation and controlled redundancy.
- Restricting unauthorised access - Security and authorisation subsystem. Access to privileged software.
- Persistent storage of widgets
- Providing storage structures and search techniques for query processing - Indexing, buffering, caching ... optimisation of query processing.
- Backup and Recovery
- Multiple user interfaces
- Representation of complex relationships
- Enforcement of integrity constraints - Referential integrity, key/uniqueness, business rules and semantics.

Advantages of using a DBMS, *cont.*

- Permitting inferencing and actions using rules - Deductive and Active databases make use of declarative/complex rules to extract information. Triggers are used to activate rules
- Enforcing standards
- Reduction in application development time
- Flexibility
- Availability of up-to-date information
- Economies of scale

COS221
L02 - Database System Concepts and Architecture
(Chapter 2 - Editions 6 and 7)

Linda Marshall

23 February 2023

Evolution of the DBMS Architecture

There was a transition from monolithic systems to *client/server* architecture.
Client/server architecture comprises of two modules, the *client* and the *server*.
The client runs on the user workstations and provides the interface between the user and the database.
The server handles storage, access, searching etc. of the database.

Concepts Relating to the Study of Database Systems

- **Data abstraction** does not consider data organisation or storage. It provides a better understanding of the data. Users perceive data at a level of abstraction relevant to them.
- A **Data model** is a collection of concepts that can be used to describe the structure of the database.

Categories of Concepts used to describe the database (Data Models)

- High-level or **conceptual data model**s - relates to how the user perceives that data. Concepts include entities, attributes and relationships. e.g. Entity-relationship model
- Representational or **implementation data model**s - understood by the end-users, but not to far removed from how the data is organised. e.g. *Relational data model*, network model, hierarchical model, object data model, XML model
- Low-level or **physical data model**s - provides details on how data is stored. e.g. record formats, record ordering, access paths (indexes)
- **Self-describing data model**s (Schema and data values combined). e.g Key-value stores and NOSQL systems

Schemas, Instances, and Database State

The *description of a database* is called the **database schema,** which is specified during database design and is not expected to change frequently. Example of a **schema diagram:**

STUDENT

Name	Student_number	Class	Major

COURSE

Course_name	Course_number	Credit_hours	Department

PREREQUISITE

Course_number	Prerequisite_number

SECTION

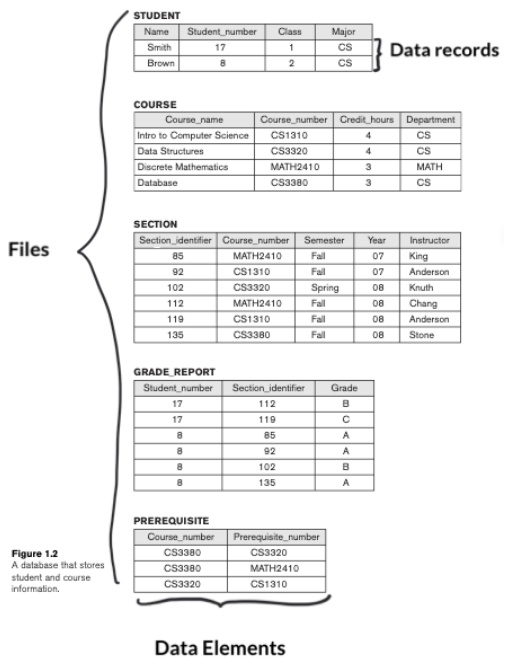
Section_identifier	Course_number	Semester	Year	Instructor

GRADE_REPORT

Student_number	Section_identifier	Grade

Schemas, Instances, and Database State Cont...

- We call each object in the schema - such as STUDENT or COURSE - a **schema construct.**
- The data in the database at a particular moment in time is called a **database state** or **snapshot.**
- It is also called the *current* set of **occurrences** or **instances** in the database.
- The schema is sometimes called the **intension,** and a database state is called an **extension** of the schema.



STUDENT

Name	Student_number	Class	Major
Smith	17	1	CS
Brown	8	2	CS

Data records

COURSE

Course_name	Course_number	Credit_hours	Department
Intro to Computer Science	CS1310	4	CS
Data Structures	CS3320	4	CS
Discrete Mathematics	MATH2410	3	MATH
Database	CS3380	3	CS

SECTION

Section_identifier	Course_number	Semester	Year	Instructor
85	MATH2410	Fall	07	King
92	CS1310	Fall	07	Anderson
102	CS3320	Spring	08	Knuth
112	MATH2410	Fall	08	Chang
119	CS1310	Fall	08	Anderson
135	CS3380	Fall	08	Stone

GRADE_REPORT

Student_number	Section_identifier	Grade
17	112	B
17	119	C
8	85	A
8	92	A
8	102	B
8	135	A

PREREQUISITE

Course_number	Prerequisite_number
CS3380	CS3320
CS3380	MATH2410
CS3320	CS1310

Files

Data Elements

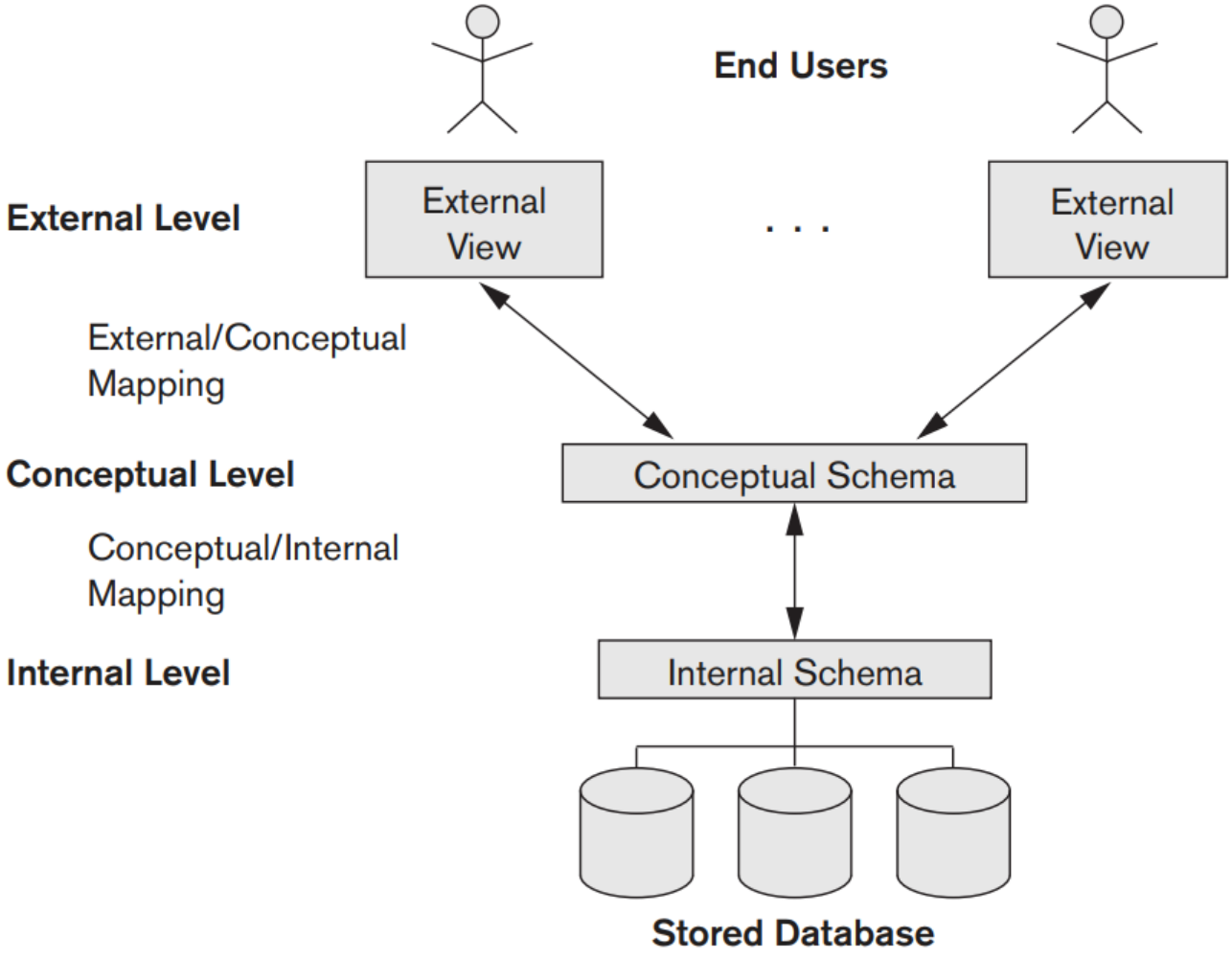
Figure 1.2
A database that stores student and course information.

The Three-Schema Architecture (ANSI/SPARC)

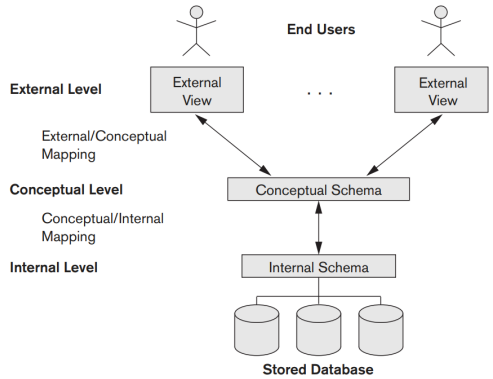
The three-schema architecture is used to:

- achieve multiple views
- separate programs and data, provides data abstraction
- provide a catalog to store the schema
- promote data independence

The Three-Schema Architecture (ANSI/SPARC)



The Three-Schema Architecture (ANSI/SPARC)



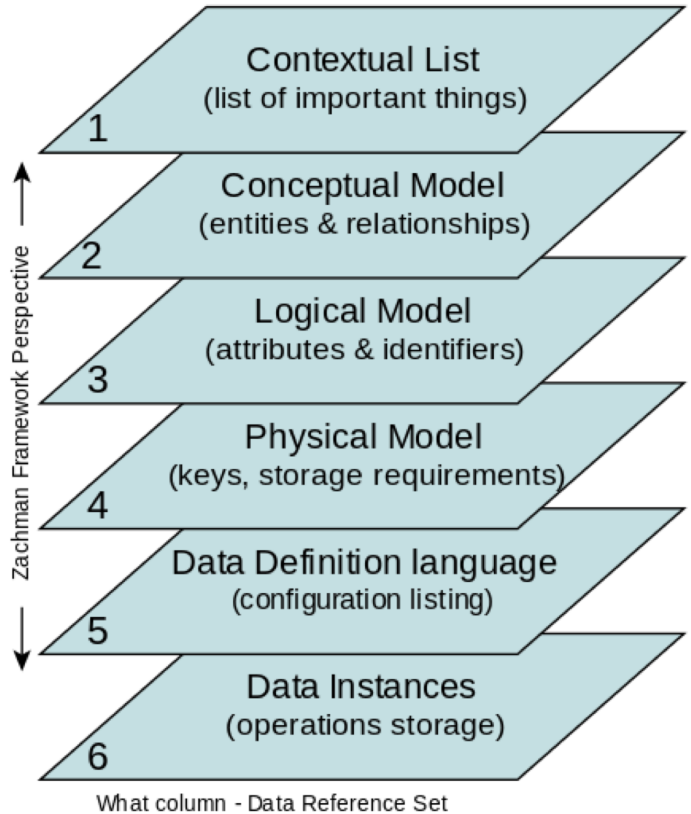
- the mappings transform requests and results between the levels
- the schemas
  - External - includes multiple user views in a high-level data model
  - Conceptual - describes entities, data types, relationships, user operations and constraints
  - Internal - describes the physical storage structure of the database and access paths in a physical data model

Data Independence

The capacity to change the schema at one level of a database system without having to change the schema at the next higher level.

1. **Logical data independence** is the capacity to change the conceptual schema without having to change external schemas or application programs.
2. **Physical data independence** is the capacity to change the internal schema without having to change the conceptual schema.

Interesting fact, the *Zachman framework*, used in Enterprise Architecture Frameworks, evolved from the three schema architectural model.



DBMS Languages

- **Data Definition Language (DDL),** is used by the DBA and by database designers to define the conceptual schema. The DDL can also be used to define the internal schema if there is no strict separation of the schema levels in the DBMS.
- **Storage Definition Language (SDL),** is used to specify the internal schema if there is a clear separation of the schema levels. The DDL is used to specify the conceptual schema and the mappings to the internal schemas.
- **View Definition Language (VDL),** is used to specify user views and their mappings to the conceptual schema, but in most DBMSs the DDL is used to define both conceptual and external schemas.
- **Data Manipulation Language (DML),** is used for retrieval, insertion, deletion, and modification of the data that is stored in the database.

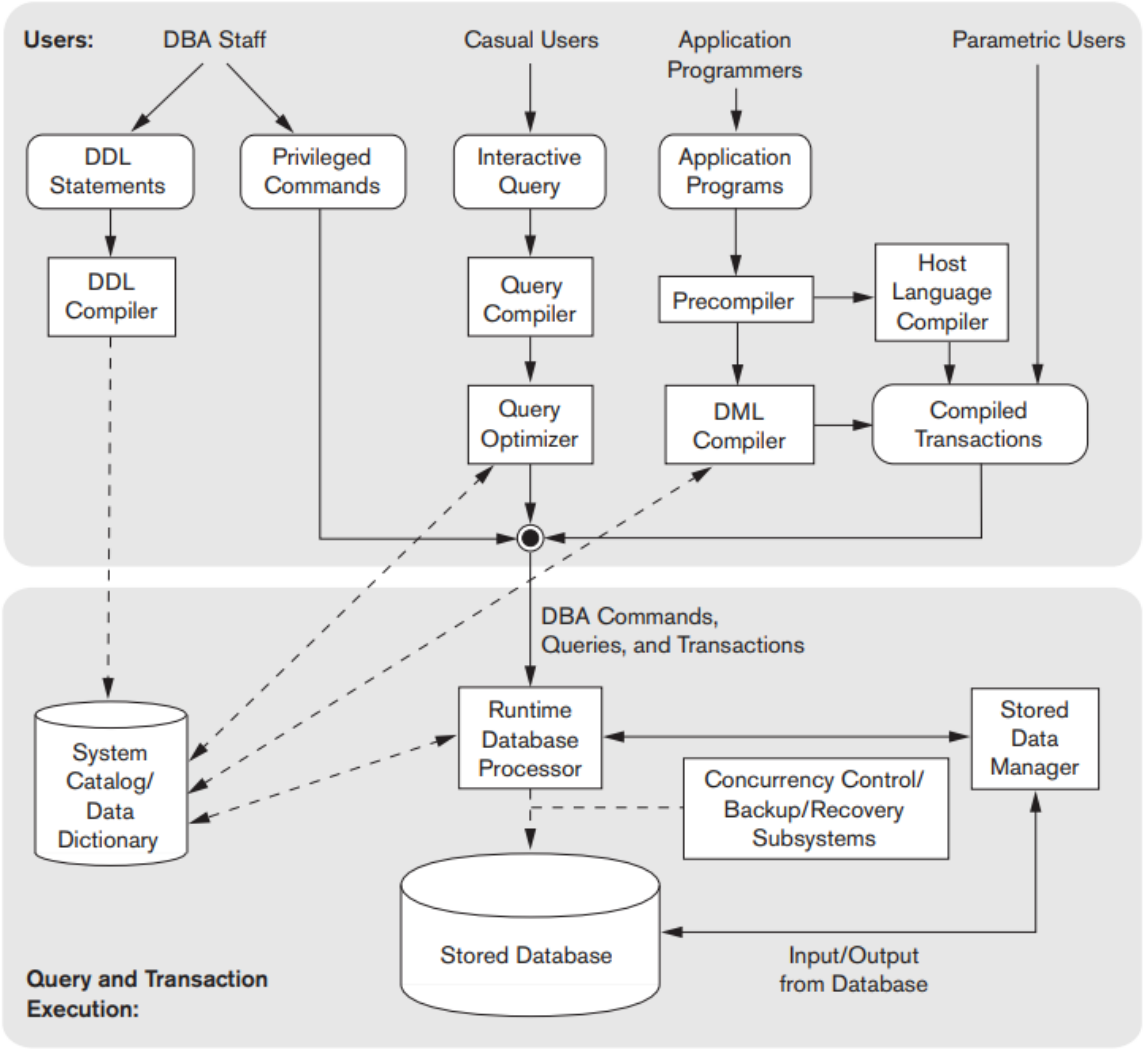
Two types of Data Manipulation Language (DML) exist:

- high-level (non-procedural, or set-at-a-time, also referred to as a query language e.g. SQL) DML's specify *what* is to be retrieved rather than *how*. These fall into the declarative paradigm of computer languages.
- low-level (procedural, record-at-a-time)

DBMS Interfaces

- Menu-based Interfaces for Web Clients or Browsing
- Apps for Mobile Devices
- Forms-based Interfaces
- Graphical User Interfaces
- Natural Language Interfaces
- Keyword-based Database Search
- Speech Input and Output
- Interfaces for Parametric Users
- Interfaces for the DBA

DBMS Component Modules



Database System Utilities

- **Loading.** A loading utility is used to load existing data files - such as text files or sequential files - into the database.
- **Backup.** A backup utility creates a backup copy of the database, usually by dumping the entire database onto tape or other mass storage medium.
- **Database storage reorganisation.** This utility can be used to reorgansze a set of database files into different file organisations and create new access paths to improve performance.
- **Performance monitoring.** Such a utility monitors database usage and provides statistics to the DBA.

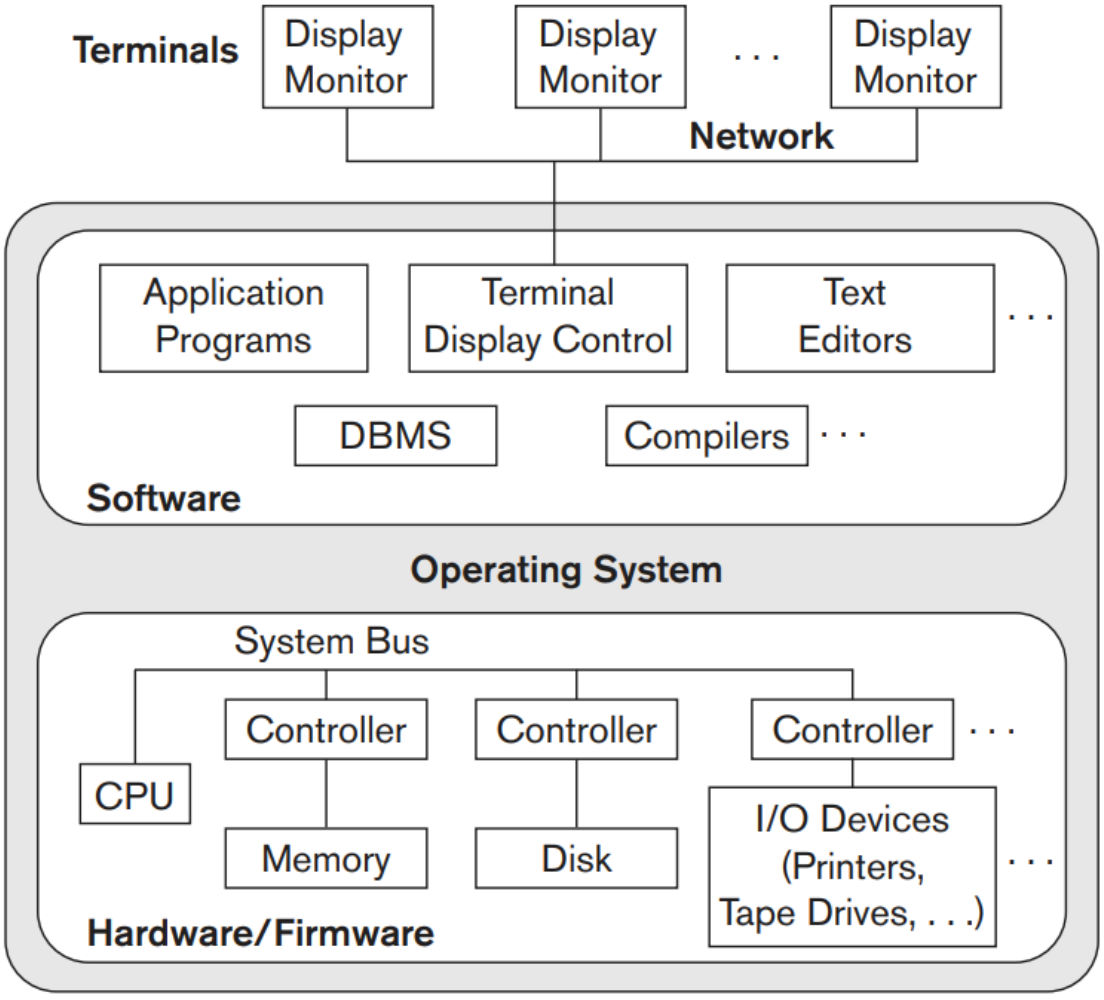
Classification of Database Management Systems

- Based on the data model: relational, object, object-relational, NOSQL, key-value, hierarchical, network, and others.
- Based on the number of users: single-user systems, multi-user systems.
- Based on the number of sites over which the database is distributed: centralised, distributed, etc.
- Based on cost: open source, etc.
- Based on types of access paths: inverted file structures, etc.
- Based on purpose: general purpose and special purpose.

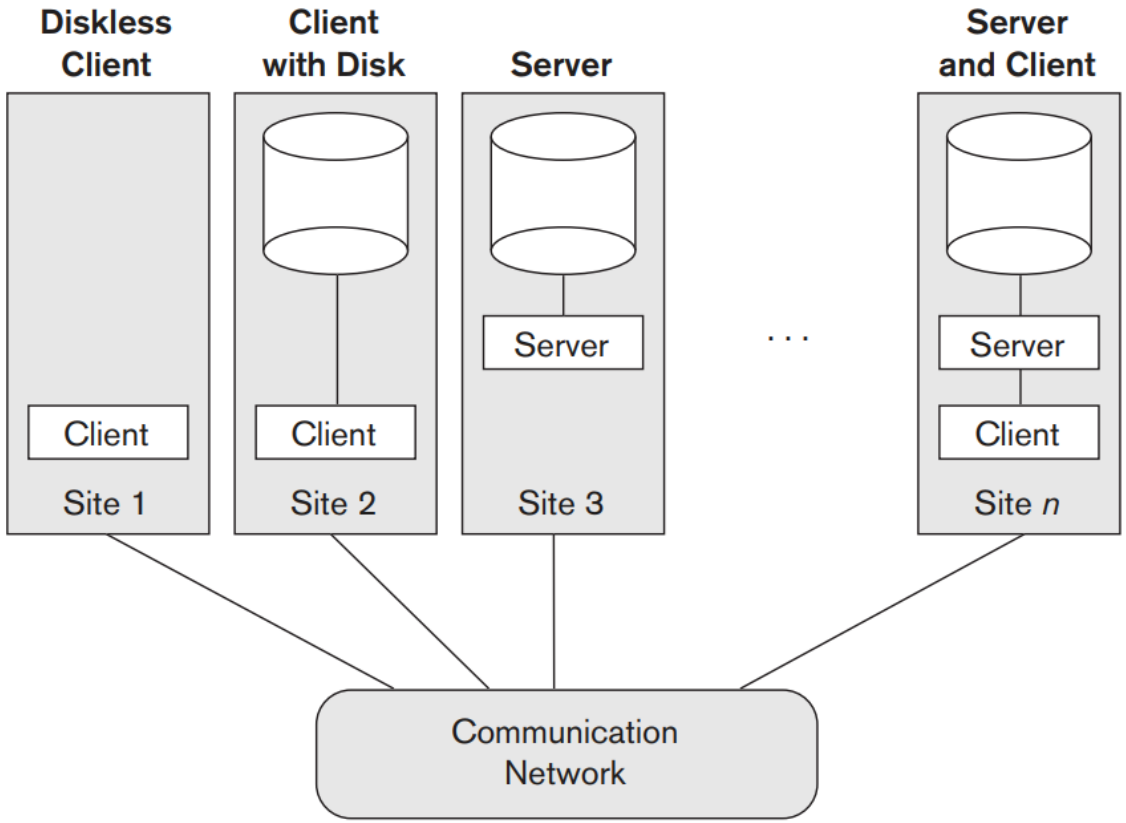
Architectures other than the three-tier C/S architecture

- Centralised DBMSs Architecture
- Two-Tier Client/Server Architectures for DBMSs
- Three-Tier and n-Tier Architectures for Web Applications

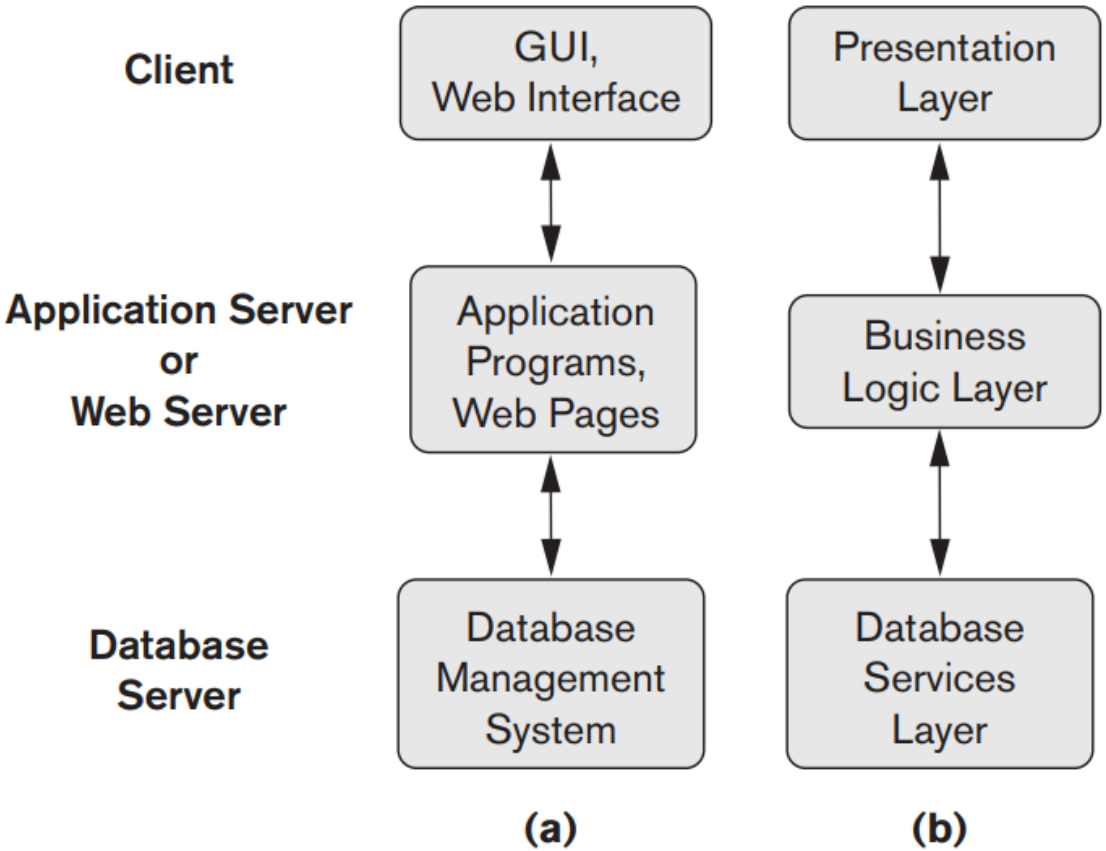
Centralised DBMSs Architecture



Two-Tier Client/Server Architectures for DBMSs



Three-Tier and n-Tier Architectures for Web Applications



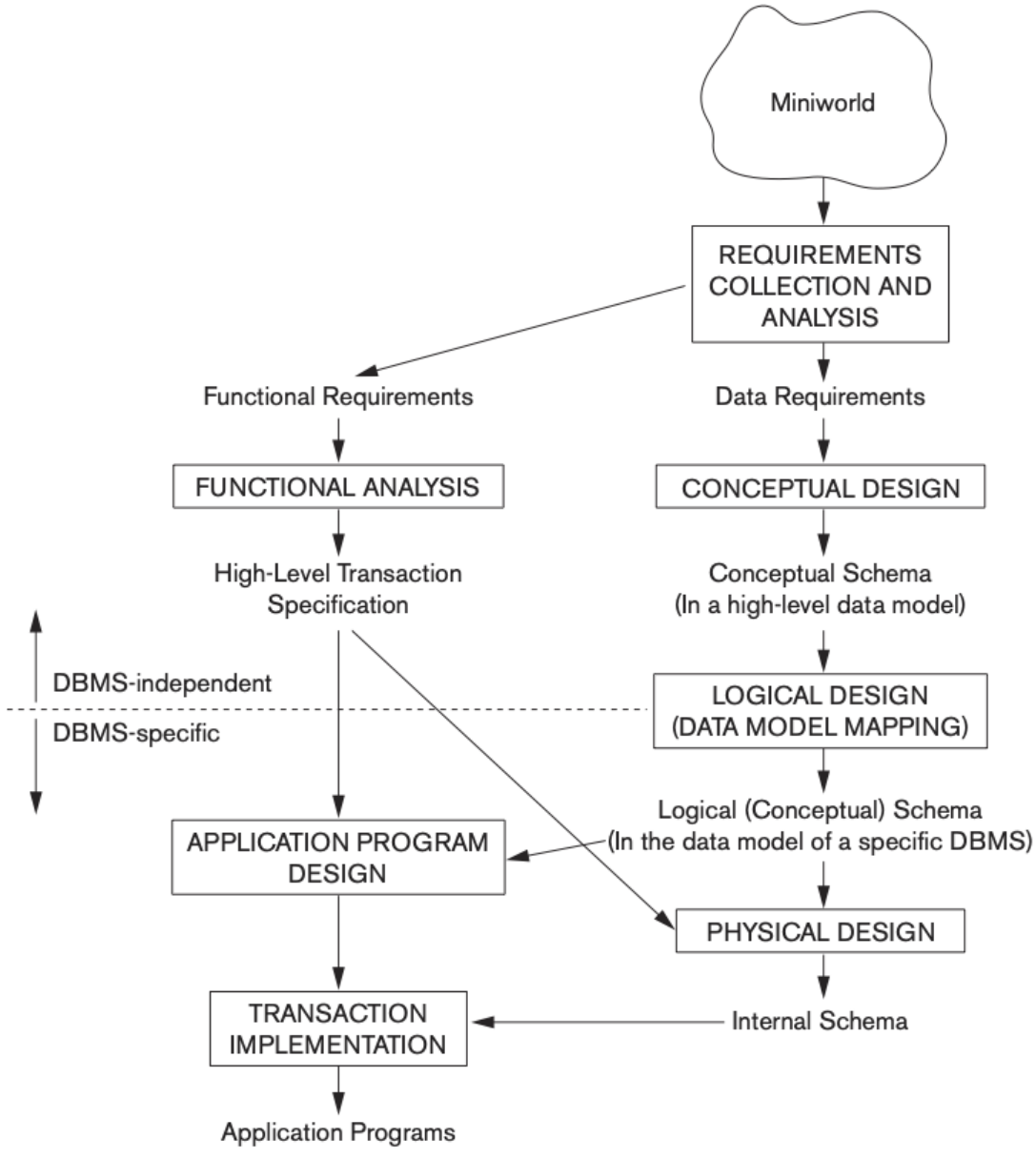
COS221
L03 - Entity-Relationship Conceptual Modelling (Part 1)

(Chapter 7 in Edition 6 and Chapter 3 in Edition 7)

Linda Marshall

24 February 2023

High-level Conceptual Data Models For Database Design - The Phases



High-level Conceptual Data Models For Database Design - The Phases

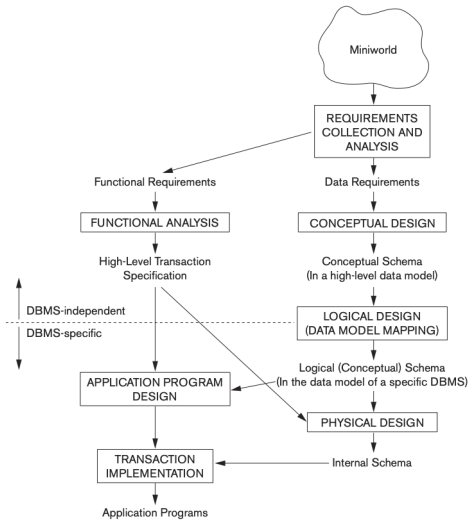


Figure 3.1
A simplified diagram to illustrate the main phases of database design.

Requirements Collection and Analysis

- Prospective database users are interviewed by designers.
- The functional and data requirements for the system are documented. These two requirements are refined to produce the the applications and physical database design which the applications will manipulate.

High-level Conceptual Data Models For Database Design - The Phases

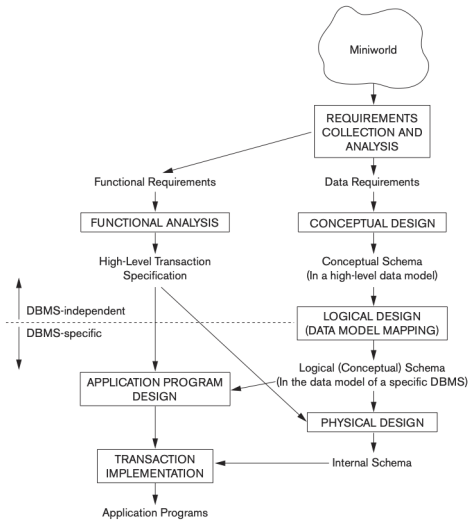


Figure 3.1
A simplified diagram to illustrate the main phases of database design.

Functional Requirements

- User-defined operations (or transactions) are used to retrieve and update data in the database
- In COS301, use cases, activity diagrams, data flow diagrams, sequence diagrams, scenarios etc. will be used to capture functional requirements

High-level Conceptual Data Models For Database Design - The Phases

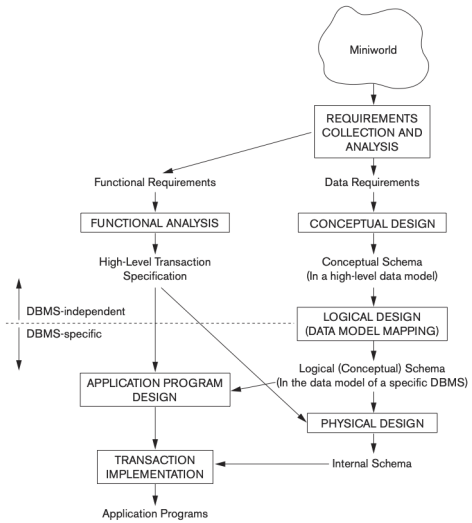


Figure 3.1
A simplified diagram to illustrate the main phases of database design.

Conceptual Design

- After collection and analysis of the data requirements, a conceptual schema for the database is created

High-level Conceptual Data Models For Database Design - The Phases

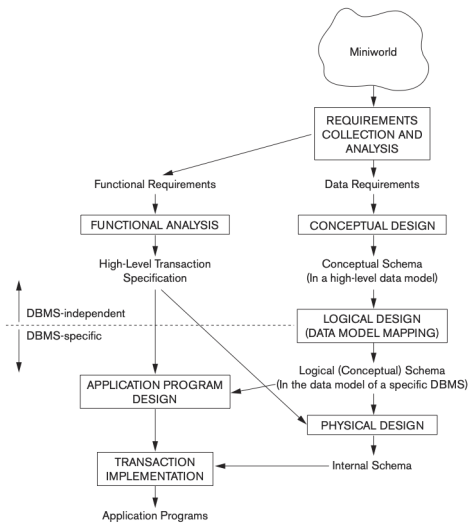


Figure 3.1
A simplified diagram to illustrate the main phases of database design.

Logical Design (Data Model Mapping)

- Translation of the conceptual schema to the specific DBMS model takes place (in our case, the relational model)
- High-level data transformed to the implementation data model

High-level Conceptual Data Models For Database Design - The Phases

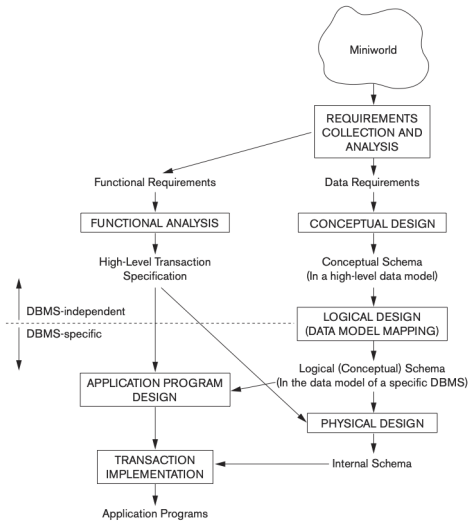


Figure 3.1
A simplified diagram to illustrate the main phases of database design.

Physical Design

- The internal storage structures, file organisations, indexes, access paths etc. are specified

Sample Database Application - The COMPANY Database

Verbal description

- The company is organized into departments. Each department has a unique name, a unique number, and a particular employee who manages the department. We keep track of the start date when that employee began managing the department. A department may have several locations.
- A department controls a number of projects, each of which has a unique name, a unique number, and a single location.
- The database will store each employee's name, Social Security number,[2] address, salary, sex (gender), and birth date. An employee is assigned to one department, but may work on several projects, which are not necessarily controlled by the same department. It is required to keep track of the current number of hours per week that an employee works on each project, as well as the direct supervisor of each employee (who is another employee).
- The database will keep track of the dependents of each employee for insurance purposes, including each dependent's first name, sex, birth date, and relationship to the employee.

Sample Database Application - The COMPANY Database

So what can we make out from the description?

Sample Database Application - The COMPANY Database

ER Representation

ER diagram describes data as:

- *entities*
- entities have *attributes*
- entities have *relationships* with other entities

In this lecture we will concentrate on the entities and their attributes.

Entities and their Attributes

- An *entity* is the basic object that is represented. It may be an object with:
  - *physical existence* - something particular, such as a person, car etc.
  - *conceptual existence* - for example a University, Company etc.
- Each entity has *attributes*
  - Attributes are properties that describe the entity. Values will be assigned to the attributes to define/identify a particular entity

Entities and their Attributes

Attribute types

- simple (atomic) vs. composite
- single-valued vs. multivalued
- stored vs. derived
- NULL values
- Complex
- Key

Entities and their Attributes

Attribute types

- simple (atomic) vs. composite
  - Simple attributes are not divisible.
  - Composite attributes can be divided into smaller parts. They can be referred to as a unit or their constituent parts can be referenced.

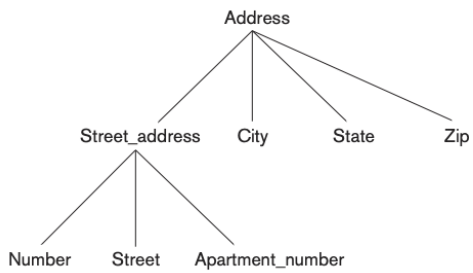


Figure 3.4
A hierarchy of composite attributes.

- single-valued vs. multivalued
  - Most attributes have a single value for a particular entity.
  - Where one value does not suffice, an attribute may be represented by a set of values. For example, a person may have more than one university degree. These attributes may be bounded to constrain the number of values.

Entities and their Attributes

Attribute types, cont.

- stored vs. derived
  - Derived attributes make use of stored attributes to calculate (derive) a value. Examples of derived attributes include *age* which can be derived from the data of birth, the number of students in a class which can be derived from counting the number of students registered in a course, ...
- NULL values
  - An entity without an applicable value is assigned a NULL value
- Complex
  - Consist of composite and/or multivalued attributes. Composite attributes are represented using round braces (...), while multivalued attributes are represented using curly braces { ... }

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

Figure 3.5
A complex attribute: Address_phone.

A person has multiple addresses. At each address there may be multiple phones. An address is a nested composite and the phones at the addresses are in turn composites.

Entities and their Attributes

Attribute types, cont.

- Key
  - Identifies each entity uniquely.
  - Keys may be composite.
  - If more than one attribute of an entity type is a key attribute, then they are so independently of each other.
  - An entity type with no keys is a *weak entity type*

Entities and their Attributes

Attribute value sets

- Attributes have *value sets* (or domains).
- That is a set of all values that may be assigned to that attribute.
- Value sets are typically not represented in ER-diagrams.

Entities and their Attributes

Attribute value sets - mathematically

A:E -> *P*(V), that is, an attribute A of entity set E with a value set of V is the function of E to the power set *P*(V) of V

A(e) is the attribute A of entity e

These definitions make provision for single- and multi-valued attributes and NULL.

NULL is represented by the empty set {}

Single valued attributes is a singleton set for each e in E

There is no restriction on multivalued attributes

For a composite attribute A, the value set V is the powerset of the cartesian product of $(P(V_1)), \ldots, (P(V_n))$, that is, $V = P(P(V_1) \times \ldots \times (P(V_n)))$

Entity types and Entity sets

- A database comprises of groups of similar entities
- An *entity type* defines a set/collection of entities with the same attributes. It is described by an entity name and its attributes.
- An *entity set* is a set of all entities belonging to an entity type.

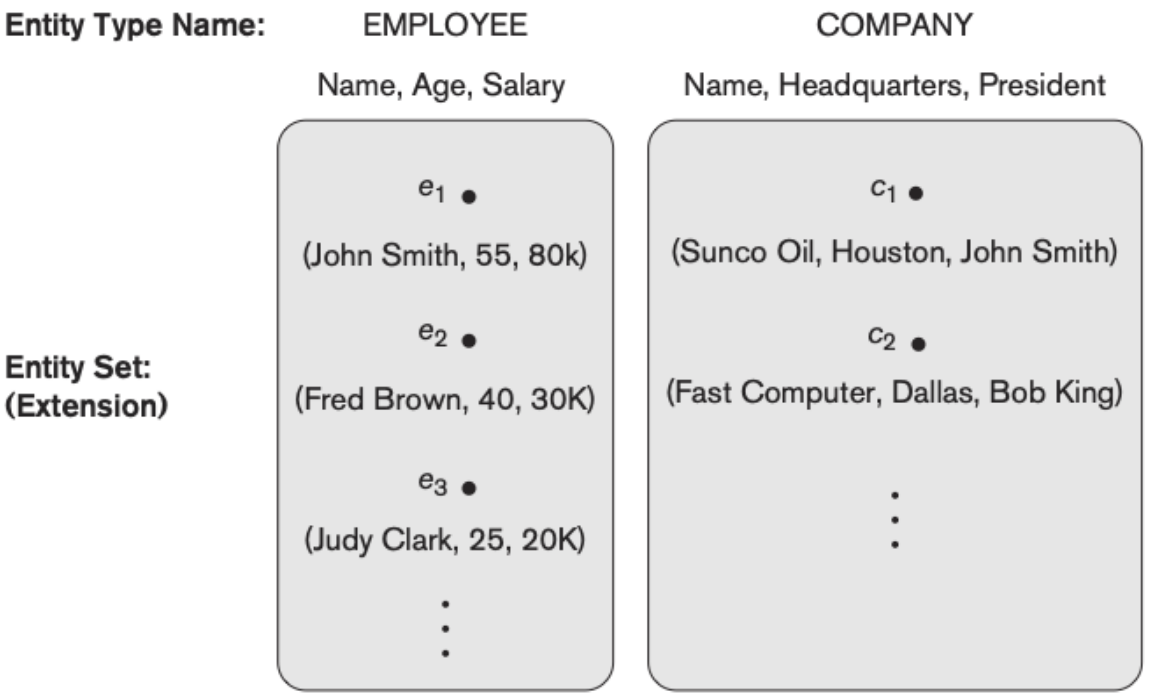


Figure 3.6
Two entity types, EMPLOYEE and COMPANY, and some member entities of each.

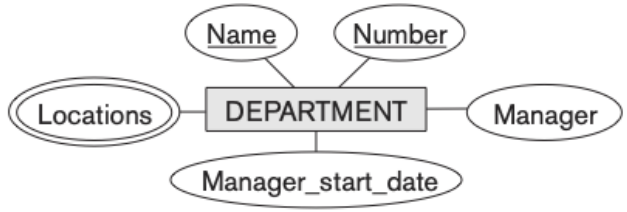
Sample Database Application - The COMPANY Database

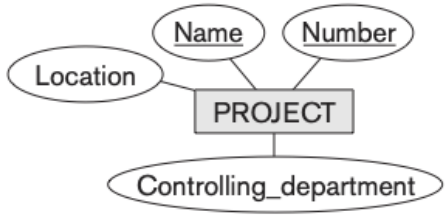
Verbal description revisited

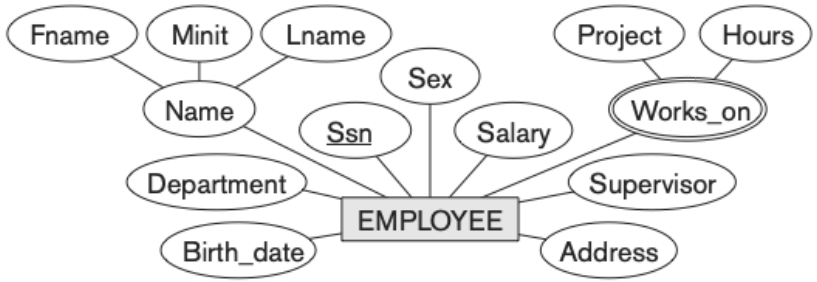
- The company is organized into departments. Each department has a unique name, a unique number, and a particular employee who manages the department. We keep track of the start date when that employee began managing the department. A department may have several locations.
- A department controls a number of projects, each of which has a unique name, a unique number, and a single location.
- The database will store each employee's name, Social Security number,[2] address, salary, sex (gender), and birth date. An employee is assigned to one department, but may work on several projects, which are not necessarily controlled by the same department. It is required to keep track of the current number of hours per week that an employee works on each project, as well as the direct supervisor of each employee (who is another employee).
- The database will keep track of the dependents of each employee for insurance purposes, including each dependent's first name, sex, birth date, and relationship to the employee.

Sample Database Application - The COMPANY Database

Preliminary Entities and attributes







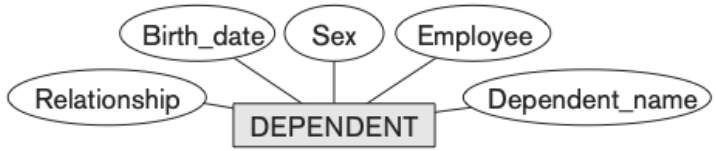


Figure 3.8
Preliminary design of entity types for the COMPANY database. Some of the shown attributes will be refined into relationships.

In the next lecture we will consider the relationships between entities and model those.

COS221
L04 - Entity-Relationship Conceptual Modelling (Part 2)

(Chapter 7 in Edition 6 and Chapter 3 in Edition 7)

Linda Marshall

27 February 2023

Quick recap - From L03

- The company is organized into departments. Each department has a unique name, a unique number, and a particular employee who manages the department. We keep track of the start date when that employee began managing the department. A department may have several locations.
- A department controls a number of projects, each of which has a unique name, a unique number, and a single location.
- The database will store each employee's name, Social Security number,[2] address, salary, sex (gender), and birth date. An employee is assigned to one department, but may work on several projects, which are not necessarily controlled by the same department. It is required to keep track of the current number of hours per week that an employee works on each project, as well as the direct supervisor of each employee (who is another employee).
- The database will keep track of the dependents of each employee for insurance purposes, including each dependent's first name, sex, birth date, and relationship to the employee.

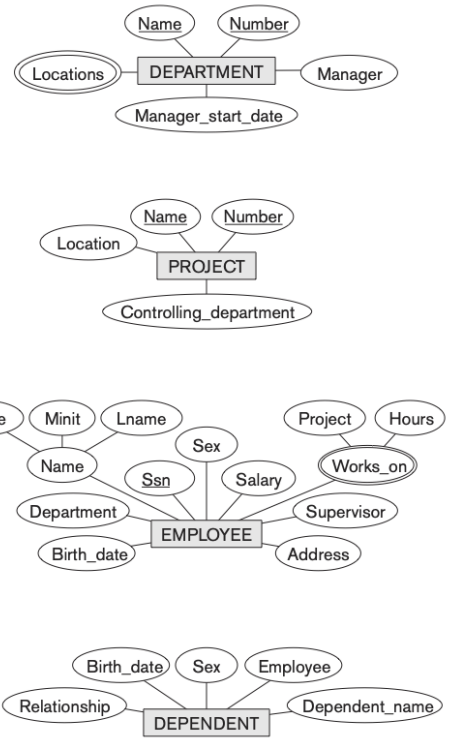


Figure 3.8
Preliminary design of entity types for the COMPANY database. Some of the shown attributes will be refined into relationships.

Identify the following: Key attribute, Composite attribute, Derived attribute, Multivalued attribute, Atomic attribute, Entity type and a Weak entity type

Relationship Types, Sets and Instance

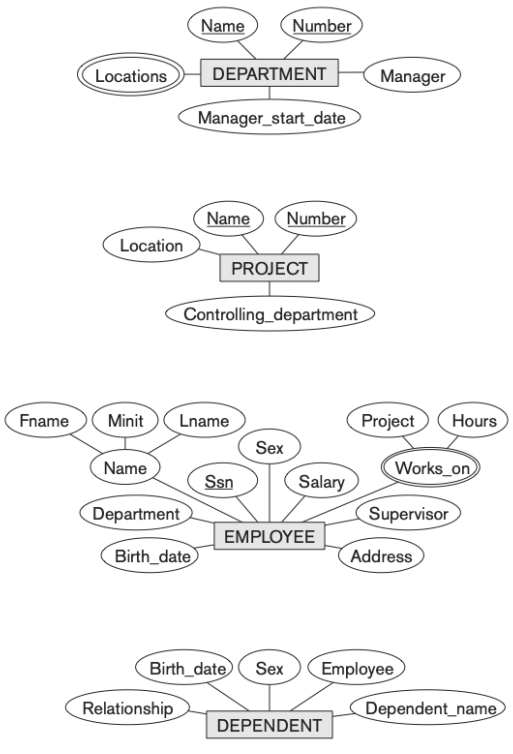


Figure 3.8
Preliminary design of entity types for the COMPANY database. Some of the shown attributes will be refined into relationships.

A *relationship* is when one *entity type* refers to another *entity type*. For example `DEPARTMENT` refers to an `EMPLOYEE` through the `Manager` attribute.

This kind of association should not be modelled as attributes, but using the *relationship* notation.

Relationship Types, Sets and Instance - Formalised

- A *relationship type*, R, among n entity types $E_1, E_2, E_3, ..., E_n$ defines a set of associations between these entity types – also referred to as *relationship sets*.
- The *relationship set*, R, is a set of *relationship instances*, r_i, where each r_i has n individual entities associated with it, $(e_1, e_2, e_3, ..., e_n)$. Each e_j in r_i is a member of E_j.
- A relationship is a subset of the Cartesian product of $E_1 \times E_2 \times E_3 \times ... \times E_n$. Each E_k participates in the relationship type, R, and each e_k participates in a relationship instance $r_i = (e_1, e_2, e_3, ..., e_n)$.

Note: As with entities, a relationship type and a relationship set are referred to by the same name.

Relationship Types, Sets and Instance - Informally

- Each r_i in R is an association of entities.
- The association includes one entity from each participating entity type.
- Each entity type in a relationship plays a role in the relationship and has a role associated with it.

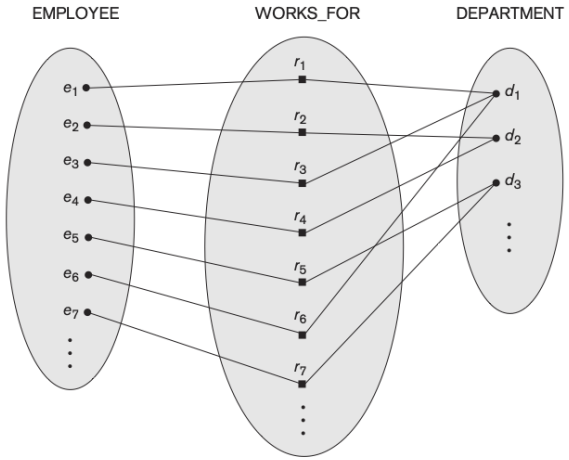


Figure 3.9
Some instances in the WORKS_FOR relationship set, which represents a relationship type WORKS_FOR between EMPLOYEE and DEPARTMENT.

WORKS_FOR associates each employee with a department In the WORKS_FOR relationship, the employee plays a *employee/worker* role and the department a *department/employer* role.

Relationship Types, Sets and Instance - Recursive

- Role names are not always necessary.
- In some instances when it is not clear what the role is, such as in a *recursive relationships*, the role name may be significant.

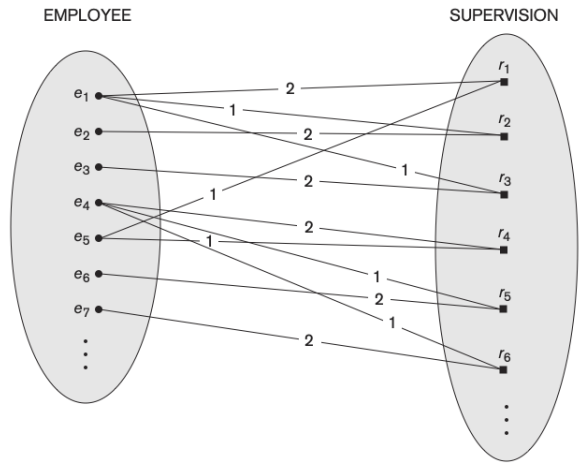


Figure 3.11
A recursive relationship SUPERVISION between EMPLOYEE in the *supervisor* role (1) and EMPLOYEE in the *subordinate* role (2).

Relationship Types, Sets and Instance - Degree

- The *degree* of a relationship is the number of participating entity types in the relationship.
- The WORKS_FOR relationship is an example of a *binary* relationship.
- Higher dimension relationships are possible, for example *ternary*, etc.

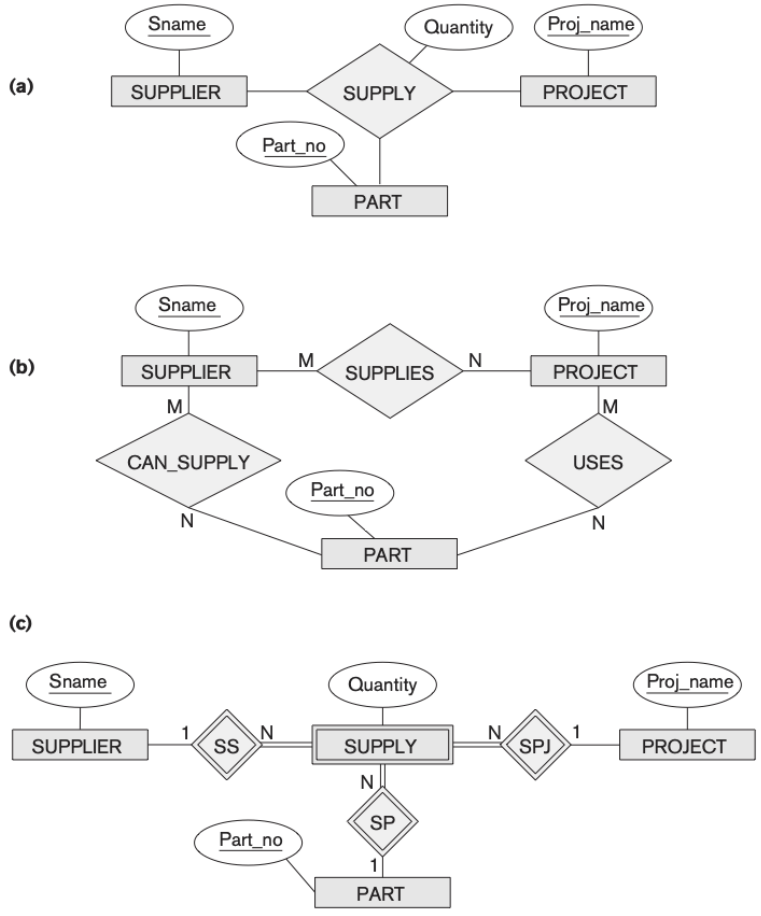


Figure 3.17
Ternary relationship types. (a) The SUPPLY relationship. (b) Three binary relationships not equivalent to SUPPLY. (c) SUPPLY represented as a weak entity type.

Relationship Types, Sets and Instance - Constraints

- Relationships may have constraints, referred to as *structural constraints*
- The two (2) main types of structural constraints are:
  - Cardinality ratios
  - Participation

Relationship Types, Sets and Instance - Constraints

Cardinality ratios

- The maximum number of relationship instances that an entity may participate in.
- Possible cardinality ratios are: 1:1, 1:N, N:1, M:N

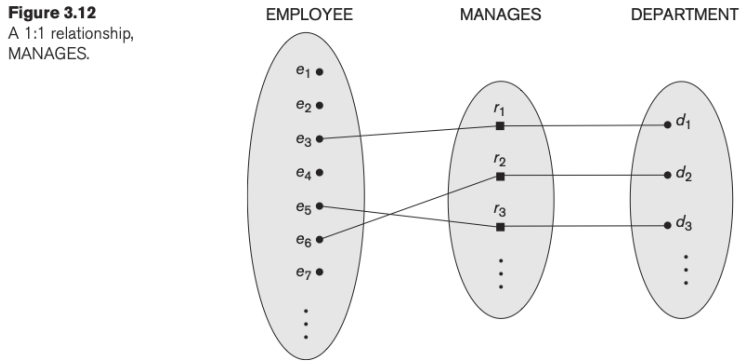


Figure 3.12
A 1:1 relationship, MANAGES.

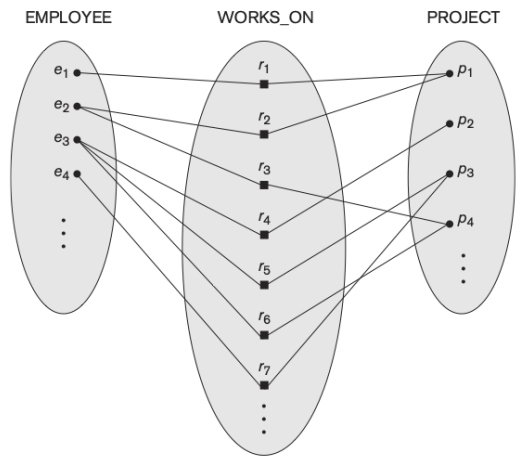
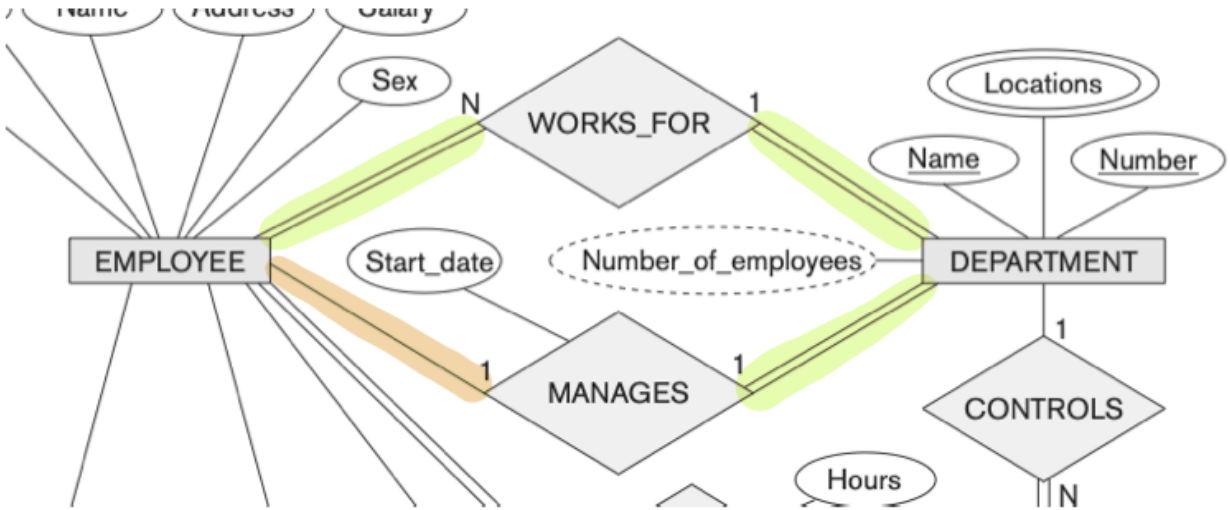


Figure 3.13
An M:N relationship, WORKS_ON.

Relationship Types, Sets and Instance - Constraints

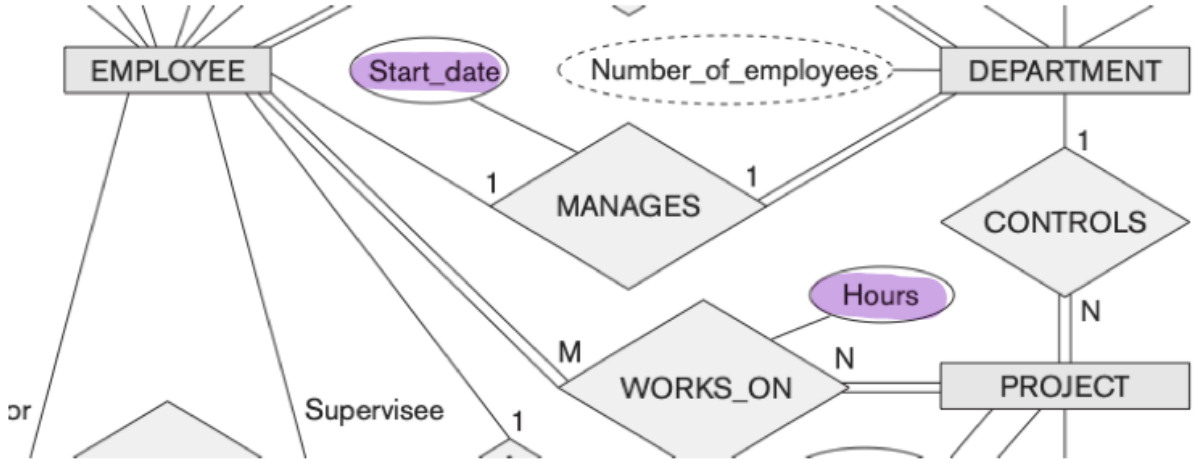
Participation

- Specifies whether an entity depends on being related.
- Participation constraints can be:
  - *Total* - every entity in one set must be related to another set via a relationship. For example, WORKS_FOR. Drawn as a double line on the ER-diagram.
  - *Partial* - some of the entities in one set are related to the entities in the other set. For example, MANAGES. Drawn as a single line on the ER-diagram.



Attributes of relationship types

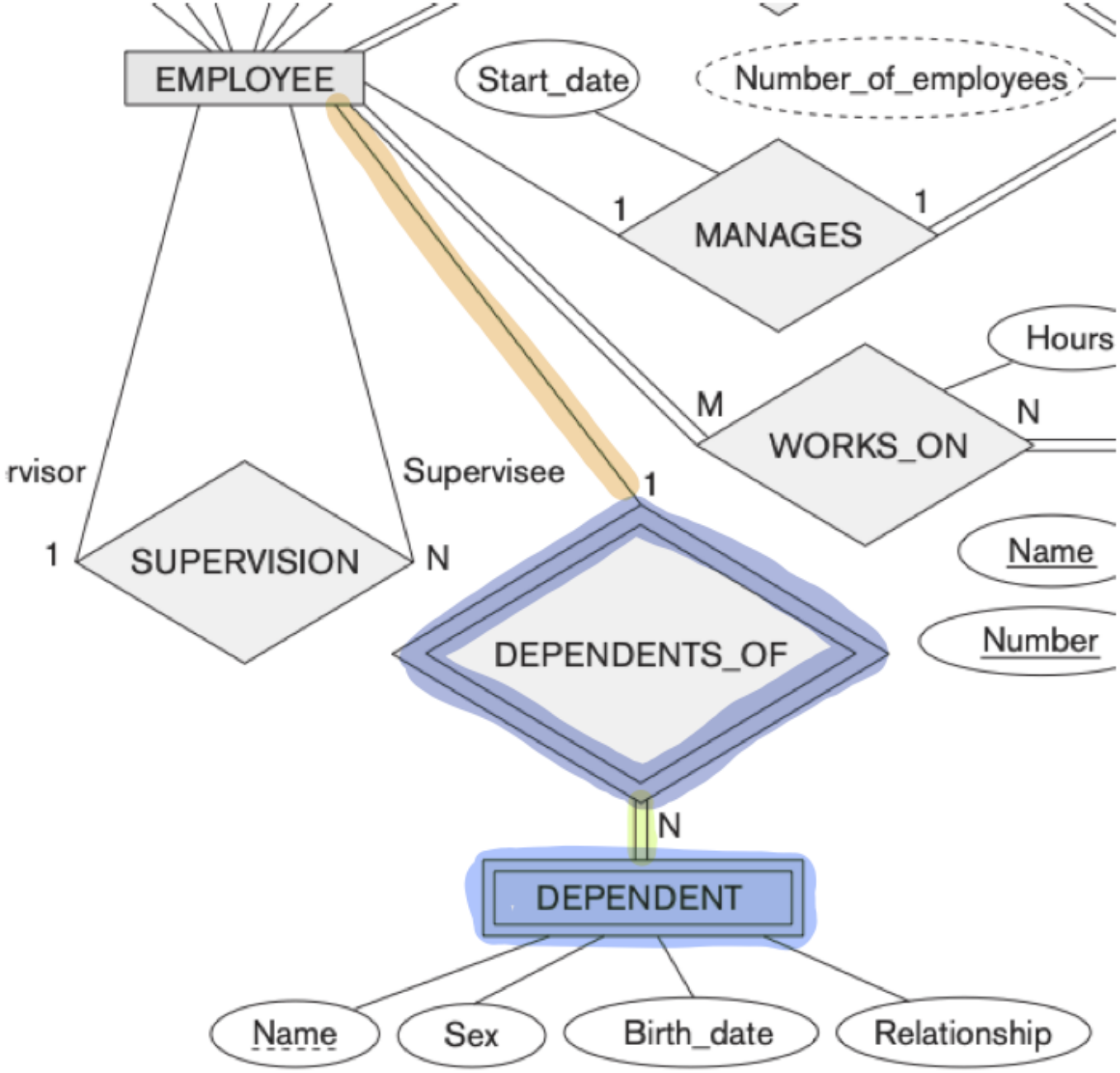
- Similar to entity types having relationships, relationship type may also have attributes.
- For example:
  - In the WORKS_ON relationship, the hours an employee works is modelled on the relationship.
  - Rather than storing the start date that an employee began managing a department in the DEPARTMENT or EMPLOYEE entity types, the attribute is captured as part of the MANAGES relationship type.



Weak entity types

- Entity types that do not have keys of their own are referred to as *weak entity types*. *Strong entity types* can therefore be seen as having keys.
- Entities of a weak entity type are related to specific entities of another entity type - referred to as the *identifying or owner entity type*.
- The *identifying relationship* (drawn with double lines) is the relationship between the owner of the weak entity type and the weak entity type. The owner of a weak entity type may itself be a weak entity type.
- A weak entity type has a *total participation constraint* between it and the owner entity. The weak entity type cannot exist without the owner.
- A weak entity type has a *partial key*. It can uniquely identify the weak entity related to the same owner entity.
- Weak entity types and their relationships are identified in the ER-diagrams as rectangles and diamonds with double boarders.

Weak entity types



Representation in an ER-diagram - Know your model

Chen's conceptual notation

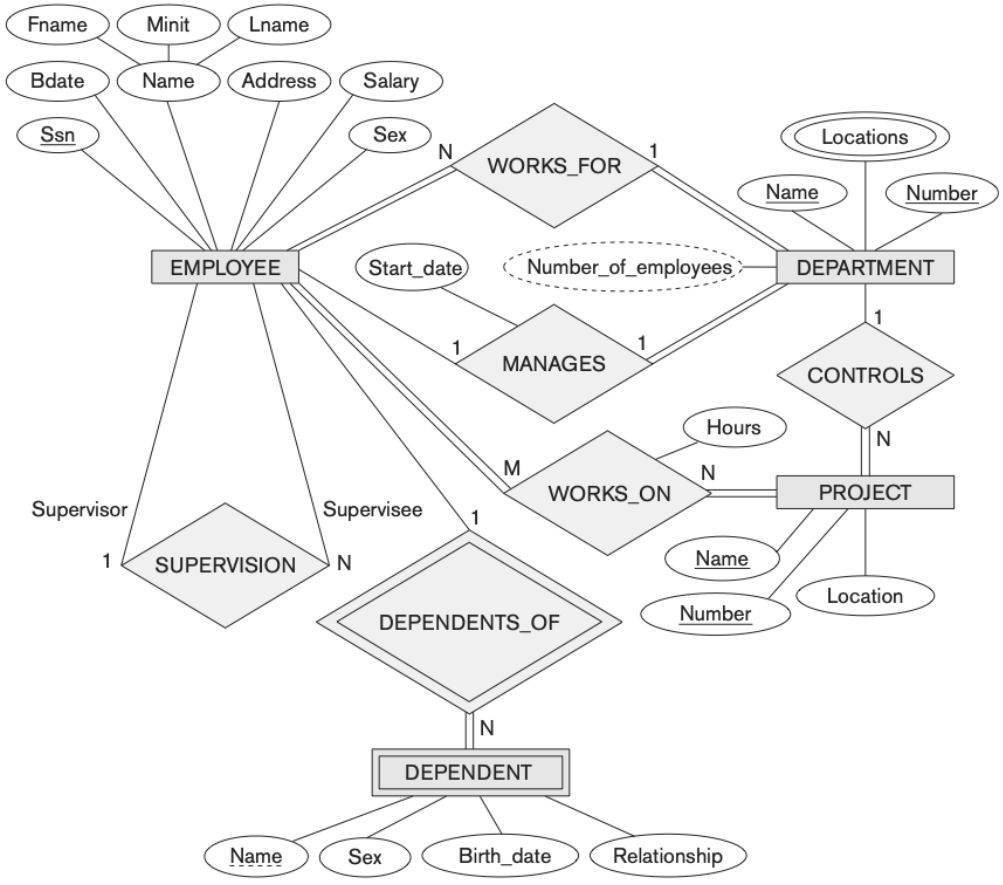


Figure 3.2
An ER schema diagram for the COMPANY database. The diagrammatic notation is introduced gradually throughout this chapter and is summarized in Figure 3.14.

ER-diagram - Notation summary

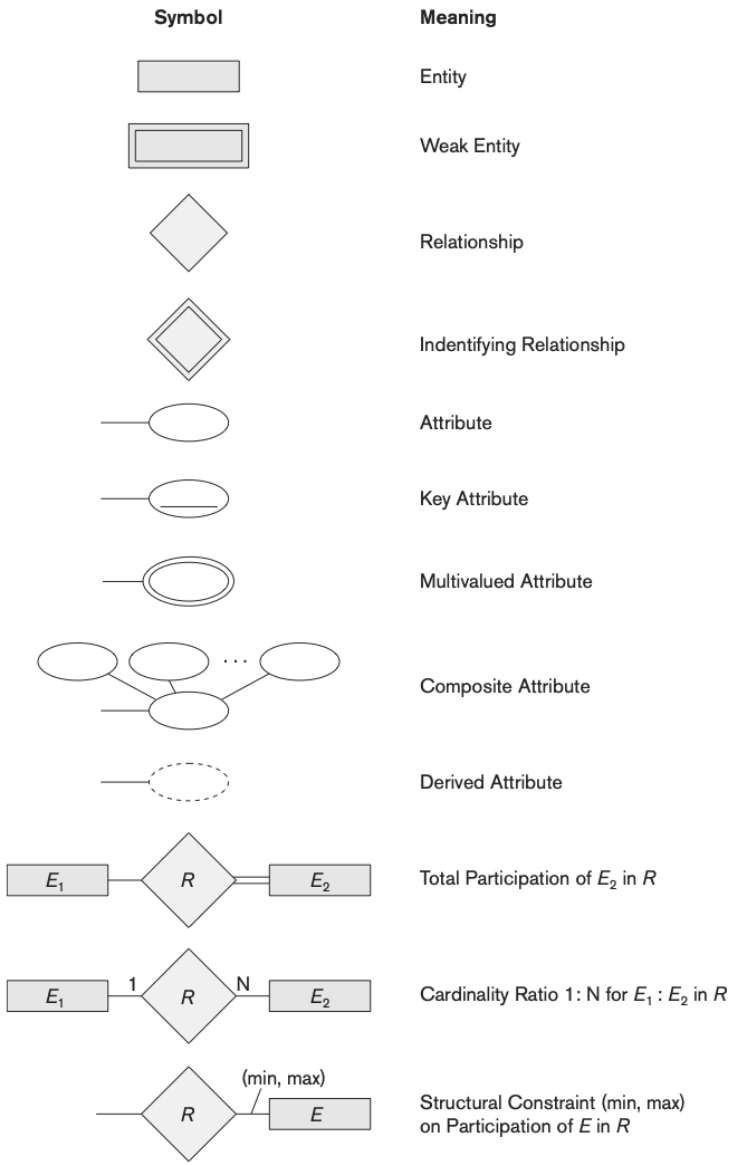


Figure 3.14
Summary of the notation for ER diagrams.

Alternative notations

(min,max) notation

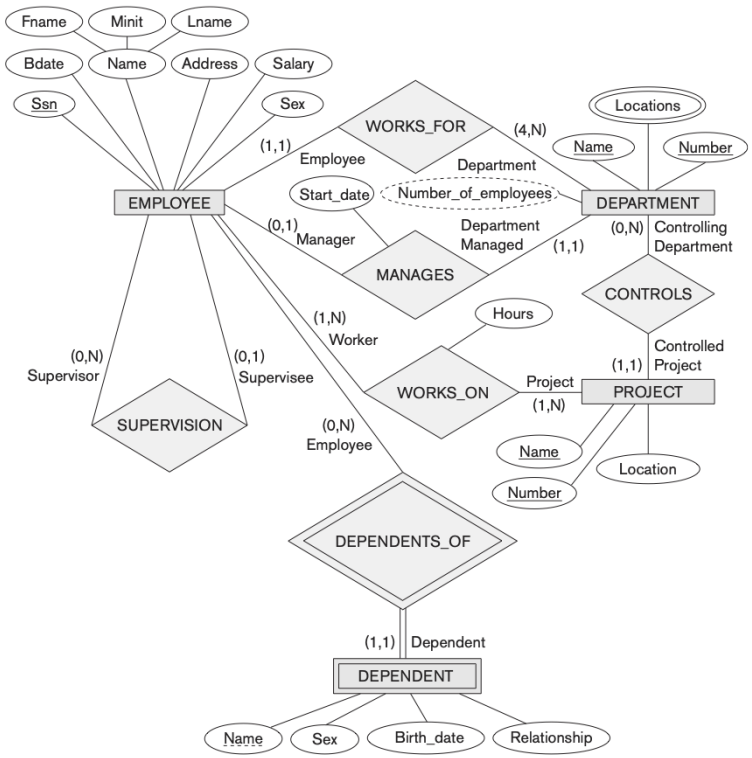


Figure 3.15
ER diagrams for the company schema, with structural constraints specified using (min, max) notation and role names.

UML notation

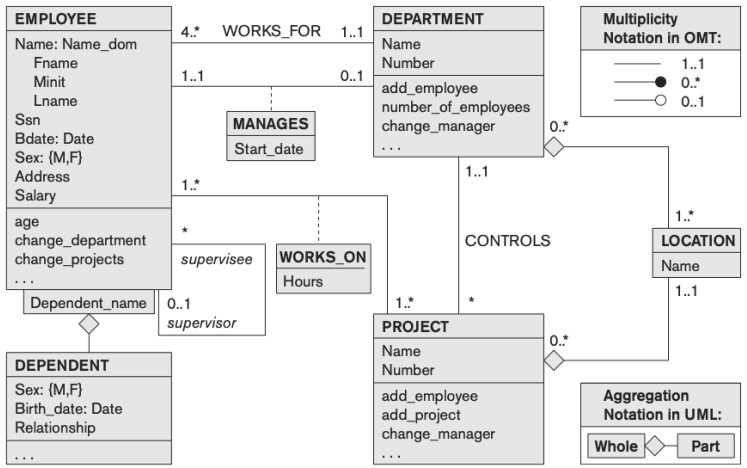


Figure 3.16
The COMPANY conceptual schema in UML class diagram notation.

Alternative notations

Crow's foot notation

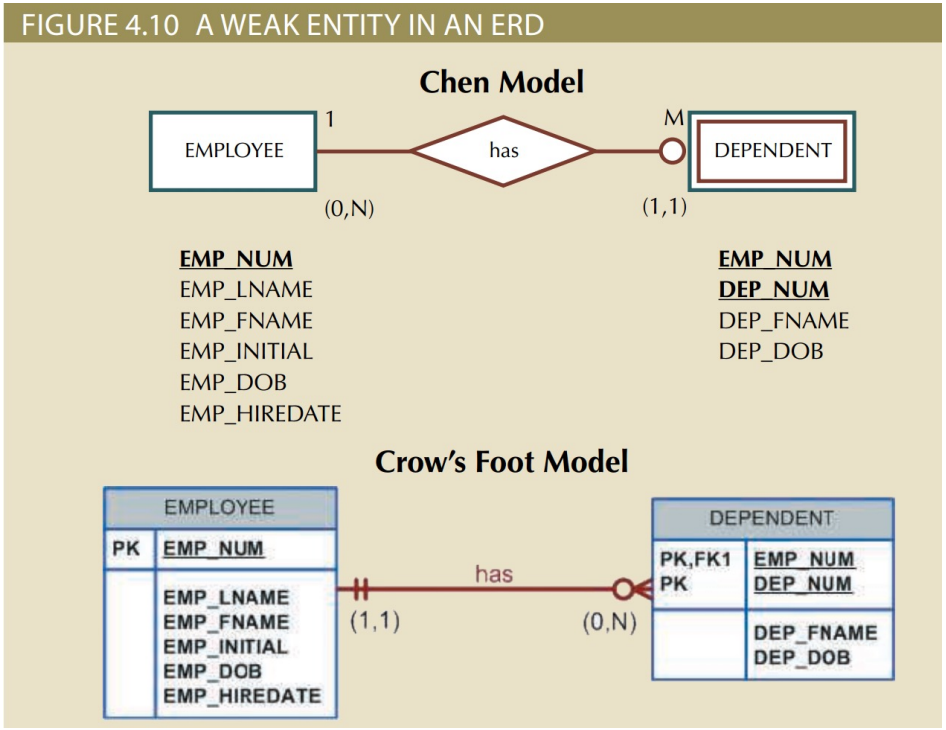


Image taken from: `https://frankfu.click/database/database-system-design-management/chapter4-entity-relationship-modeling/`

`https://vertabelo.com/blog/crow-s-foot-notation/`

`https://www.codeproject.com/Articles/878359/Data-Modelling-using-ERD-with-Crow-Foot-Notation`

Notations

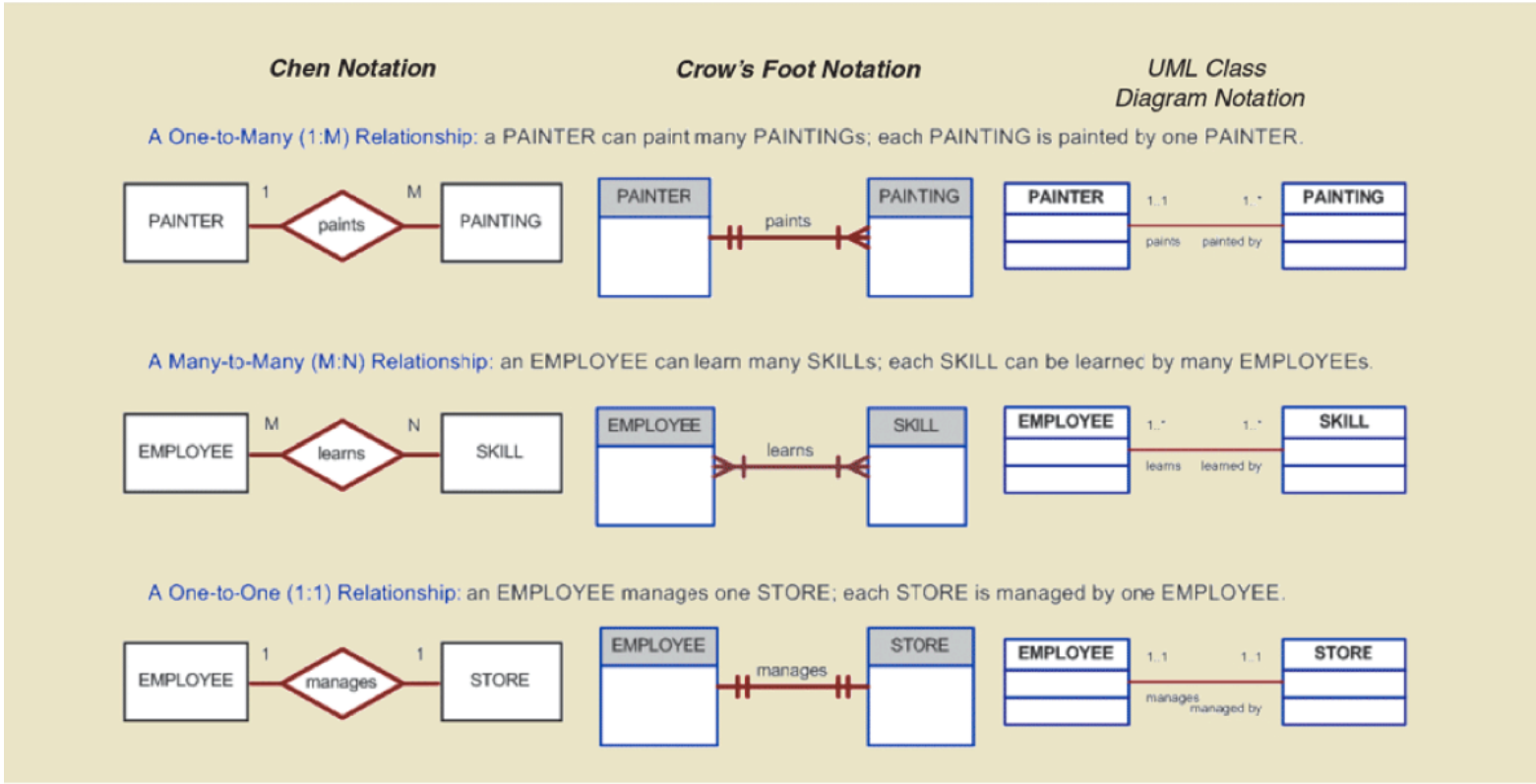


Image taken from: `http://www.csc.villanova.edu/~enwafor/database/lectures/lecture_3.pdf`

COS221
L05 - Enhanced Entity-Relationship Modelling (Part 1)

(Chapter 8 in Edition 6 and Chapter 4 in Edition 7)

Linda Marshall

2 March 2023

Semantic data modelling in Computer Science

Various semantic data models have been introduced into computer science

- Knowledge representation in AI
- Object modelling in SE
- Enhanced ER for Database systems.
  This includes concepts such as:
  - class/subclass relationships
  - type inheritance
  - specialisation/generalisation and constraints
  - modelling UNION contracts with categories
  - ...

Subtypes (or Subclasses)

In ER-modelling (L03 and L04),

- An entity type represented both a *type of entity* and an *entity set* (or *collection of entities of that type*) that exist in the database. For example, EMPLOYEE describes the type (attributes and relationships) and the current set of employee entities in the COMPANY database.

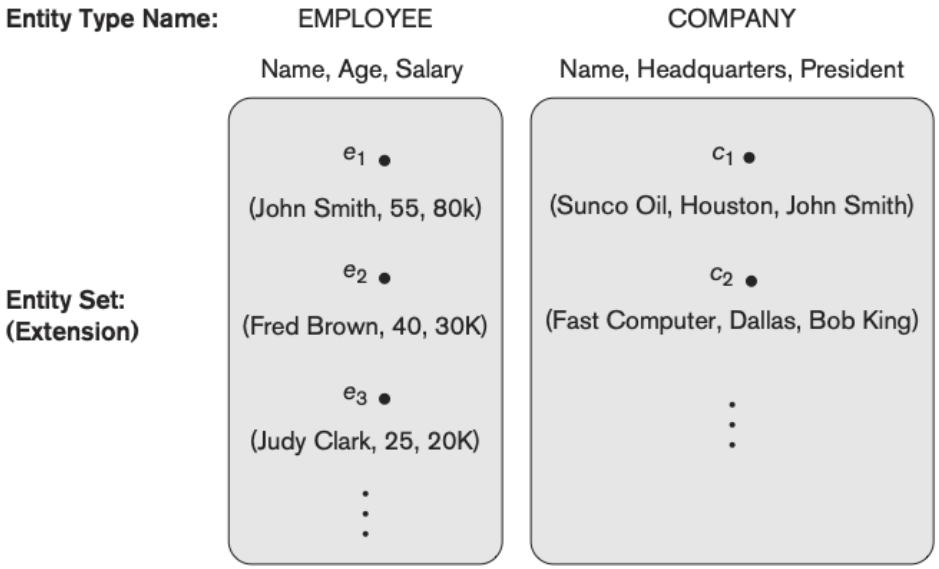


Figure 3.6
Two entity types, EMPLOYEE and COMPANY, and some member entities of each.

- There are relationships between entities.

Subtypes (or Subclasses)

- An entity type may have many *subgroupings* (or *subtypes or subclasses*) that are meaningful. For example, Employees can be differentiated by post-level (secretary, manager, ...), or employment contract (part-time, ...). The set of entities in the subgrouping is a subset of the entities belonging to the EMPLOYEE entity set.
  - A 'parent' type/class of a subtype is called *supertype* or *superclass*
  - A subtype/class has an 'is-a' relationship with it supertype/class.
- Because the entity of the subtype represents the same real-world entity in the superclass, it possesses (*inherits*) all the attributes associated with the superclass. This is referred to as *type inheritance*.

Specialisation and Generalisation

A *specialisation* defines a set of subclasses of an entity type - the superclass of the specialisation. In the figure below there are three (3) specialisations of EMPLOYEE shown.

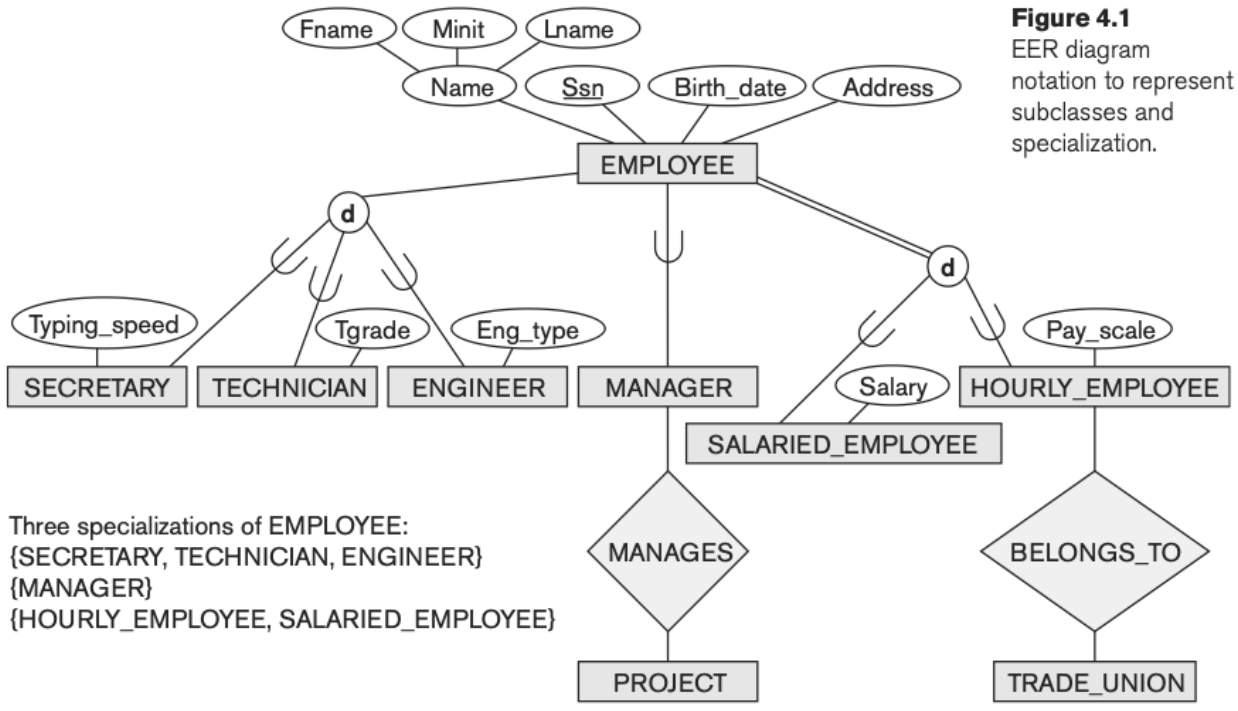


Figure 4.1
EER diagram notation to represent subclasses and specialization.

Specialisation and Generalisation - Notation

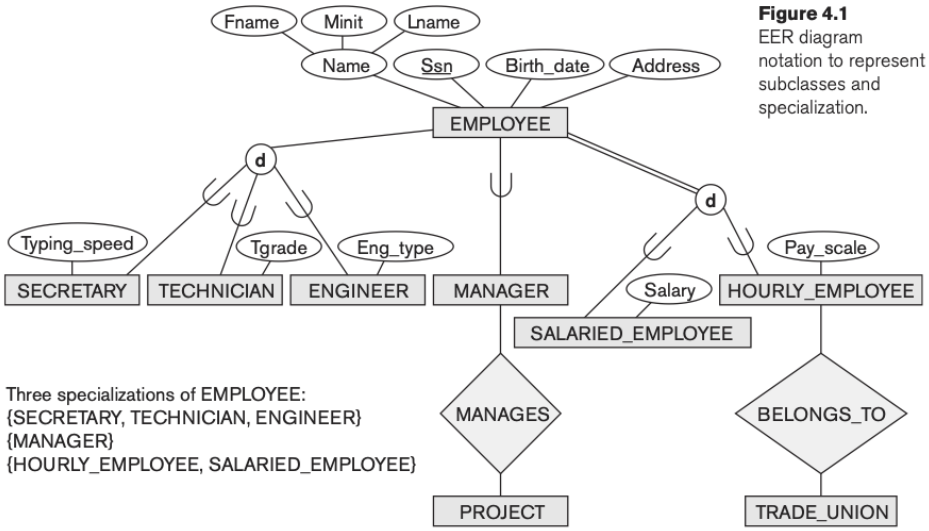


Figure 4.1
EER diagram notation to represent subclasses and specialization.

- Specialisations are linked to the superclass with a line to a circle then another line to the superclass.
- The *proper subset symbol* on the lines shows the direction of the subset/superclass relationship.
- Subsets may include attributes specific to the subclass, referred to as *specific* or *local* attributes.

Specialisation and Generalisation - Notation

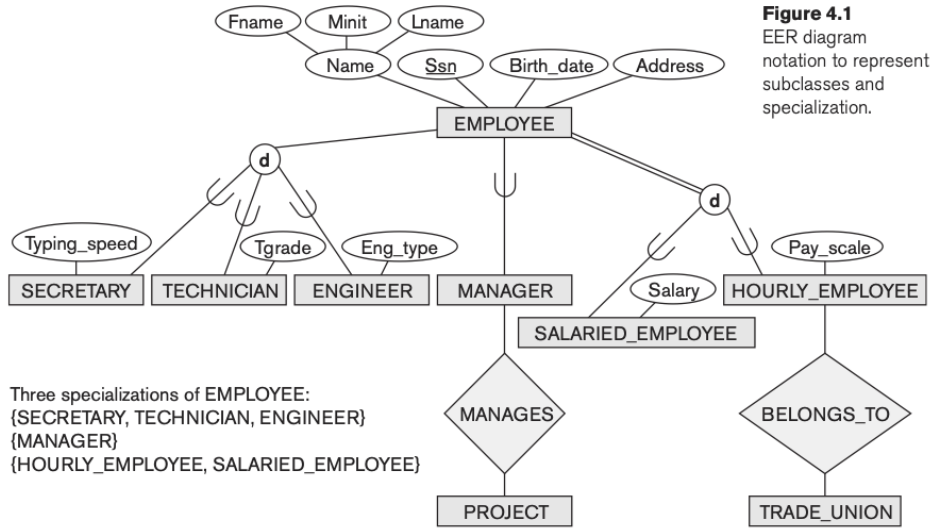


Figure 4.1
EER diagram notation to represent subclasses and specialization.

- A specialisation may comprise of multiple entity types with the same characteristics (SECRETARY,TECHNICIAN and ENGINEER) or a single entity type (MANAGER)
- A specialisation with a single entity type is drawn using only a line. That is, the circle notation is not used.

Specialisation and Generalisation - Notation

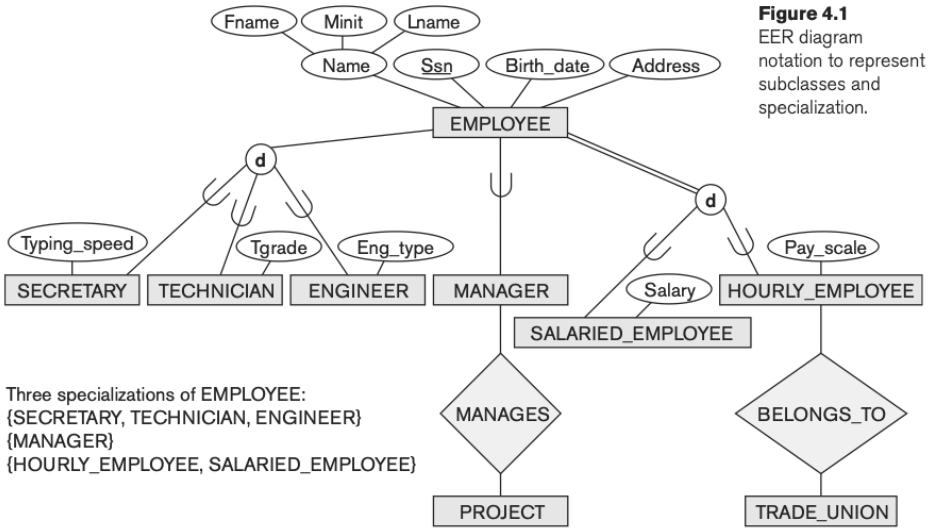


Figure 4.1
EER diagram notation to represent subclasses and specialization.

- Specialisations are used when some relationships may be participated in and others not.
- An HOURLY_EMPLOYEE may belong to a trade union.

Specialisation and Generalisation - Instances of a Specialisation

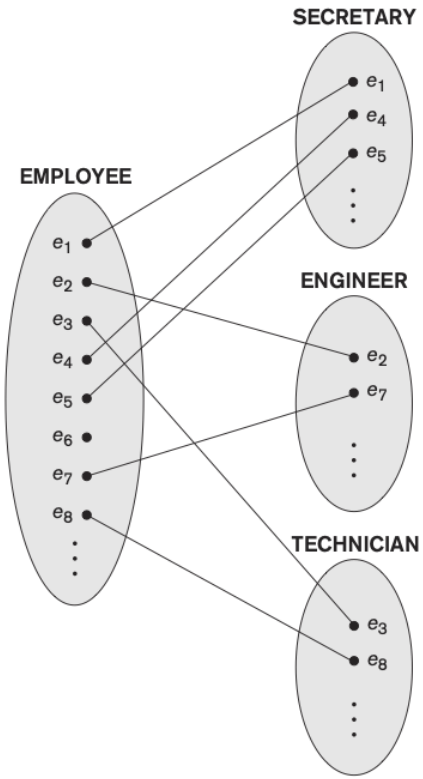


Figure 4.2
Instances of a specialization.

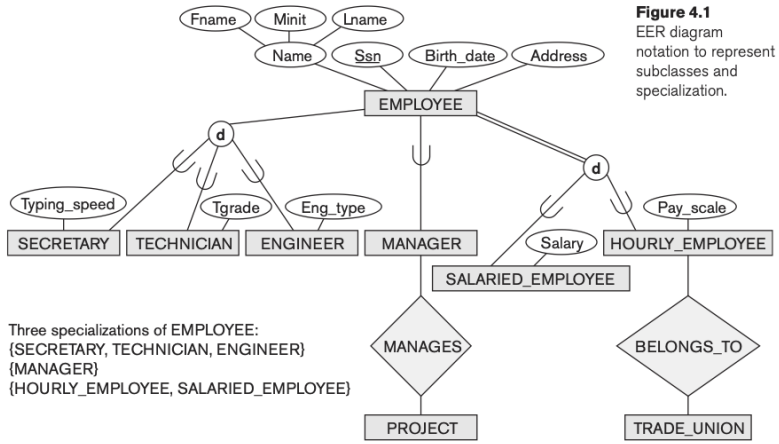
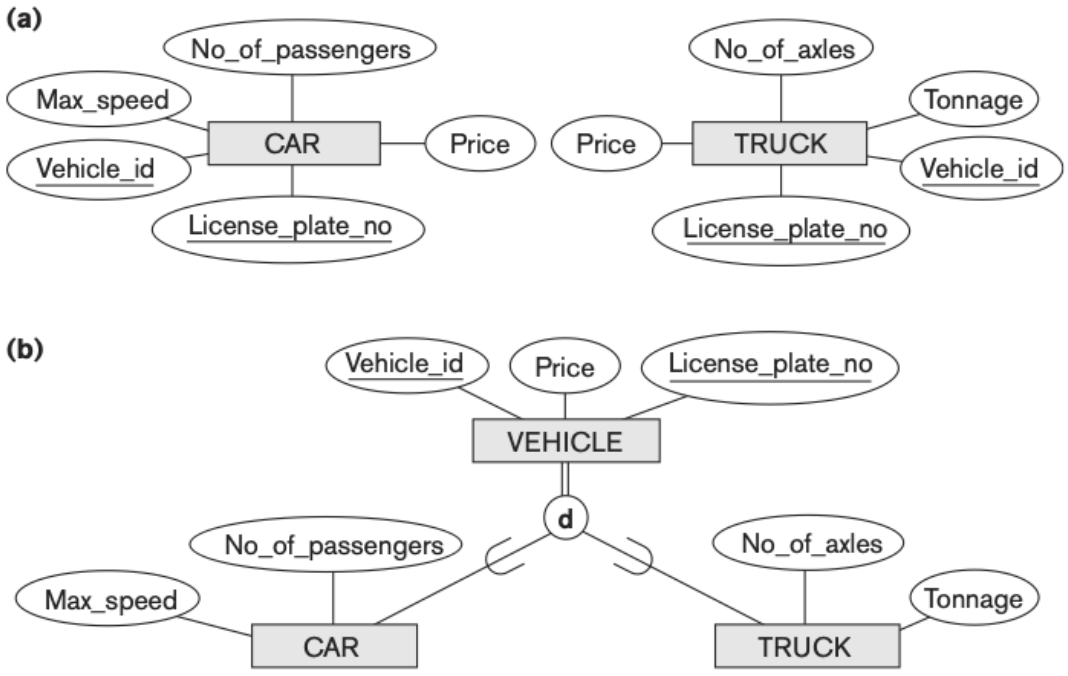


Figure 4.1
EER diagram notation to represent subclasses and specialization.

Specialisation and Generalisation

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



- The reverse process of specialisation is generalisation.
- Given two or more related entities that share common attributes, these common attributes can be generalised (abstracted out) to form the superclass.

Constraints on Specialisation/Generalisation Hierarchies

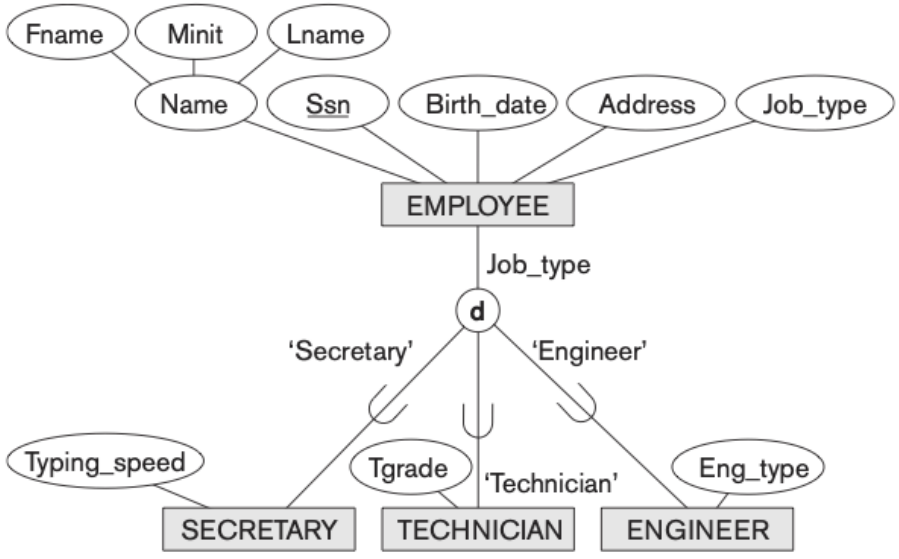


Figure 4.4
EER diagram notation for an attribute-defined specialization on Job_type.

- Placing a constraint on the value of some member of the superclass will allow us to specify exactly which entities will become members of the subclass.
- These subclasses are referred to as *predicate-defined* (or *condition-defined* subclasses.
- `Job_type = 'Secretary'`, is called the *defining predicate* and `Job_type` in EMPLOYEE, the *defining attribute*.
- The specialisation if termed an *attribute-defined specialisation*.

Constraints on Specialisation/Generalisation Hierarchies

- When no condition is defined to determine membership of subclasses, the subclass is called *user-defined*.
- Deciding membership of the subclass in this case is a manual process and determined by the user.
- The process cannot be automated.

Constraints on Specialisation/Generalisation Hierarchies

- Other constraints that may apply to a specialisation
  - Disjointness (or disjointedness)
  - Completeness (or totalness)
- Disjointness and Completeness are independent and therefore the following combinations are possible:
  - Disjoint and Total
  - Disjoint and Partial
  - Complete and Total
  - Complete and Partial

Constraints on Specialisation/Generalisation Hierarchies - Disjointness

- An entity of the specialisation may be a member of at most one subclass of the specialisation.
- An *attribute-defined* specialisation implies disjointedness.
- Disjointedness is specified using a **d** in the circle.
- Sets of entities of subclasses that are not disjoint are said to be *overlapping*. An **o** in the circle is the designation for this, the default, case.

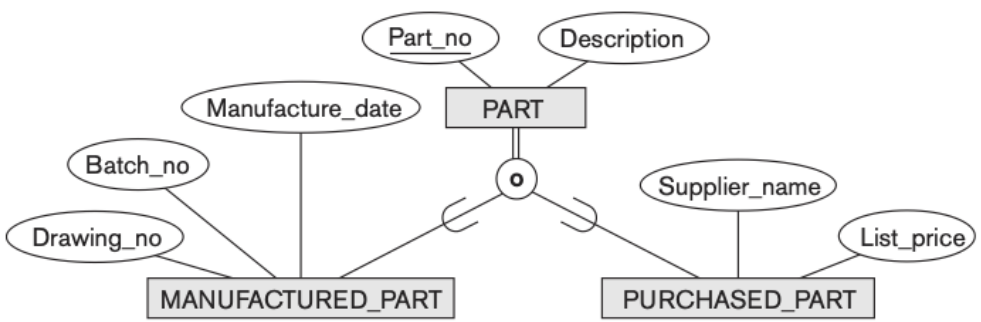


Figure 4.5
EER diagram notation for an overlapping (nondisjoint) specialization.

Constraints on Specialisation/Generalisation Hierarchies - Completeness

The completeness constraint may be *partial* or *total*.

- A *partial specialisation* means that the entity need not be a member of one of its subclasses.
  - This is shown on the ER-diagram using a single line.
  - For example, an EMPLOYEE may be a SECRETARY, ENGINEER, TECHNICIAN or MANAGER.
- A *total specialisation* means that every entity in the superclass **must** be a member of at least one subclass.
  - This is shown using a double line on the ER-diagram.
  - For example, an EMPLOYEE must be either a SALARIED_EMPLOYEE or an HOURLY_EMPLOYEE.

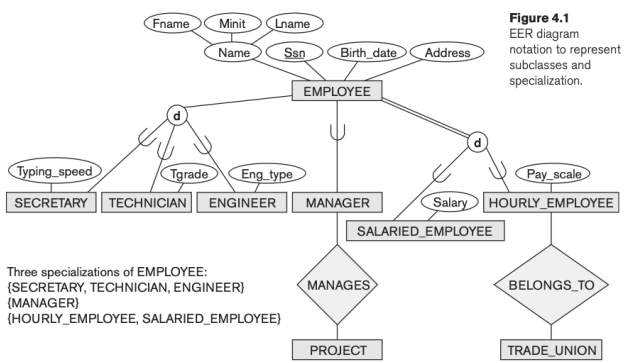


Figure 4.1
EER diagram notation to represent subclasses and specialization.

COS221
L06 - Enhanced Entity-Relationship Modelling (Part 2)

(Chapter 8 in Edition 6 and Chapter 4 in Edition 7)

Linda Marshall

3 March 2023

Recap

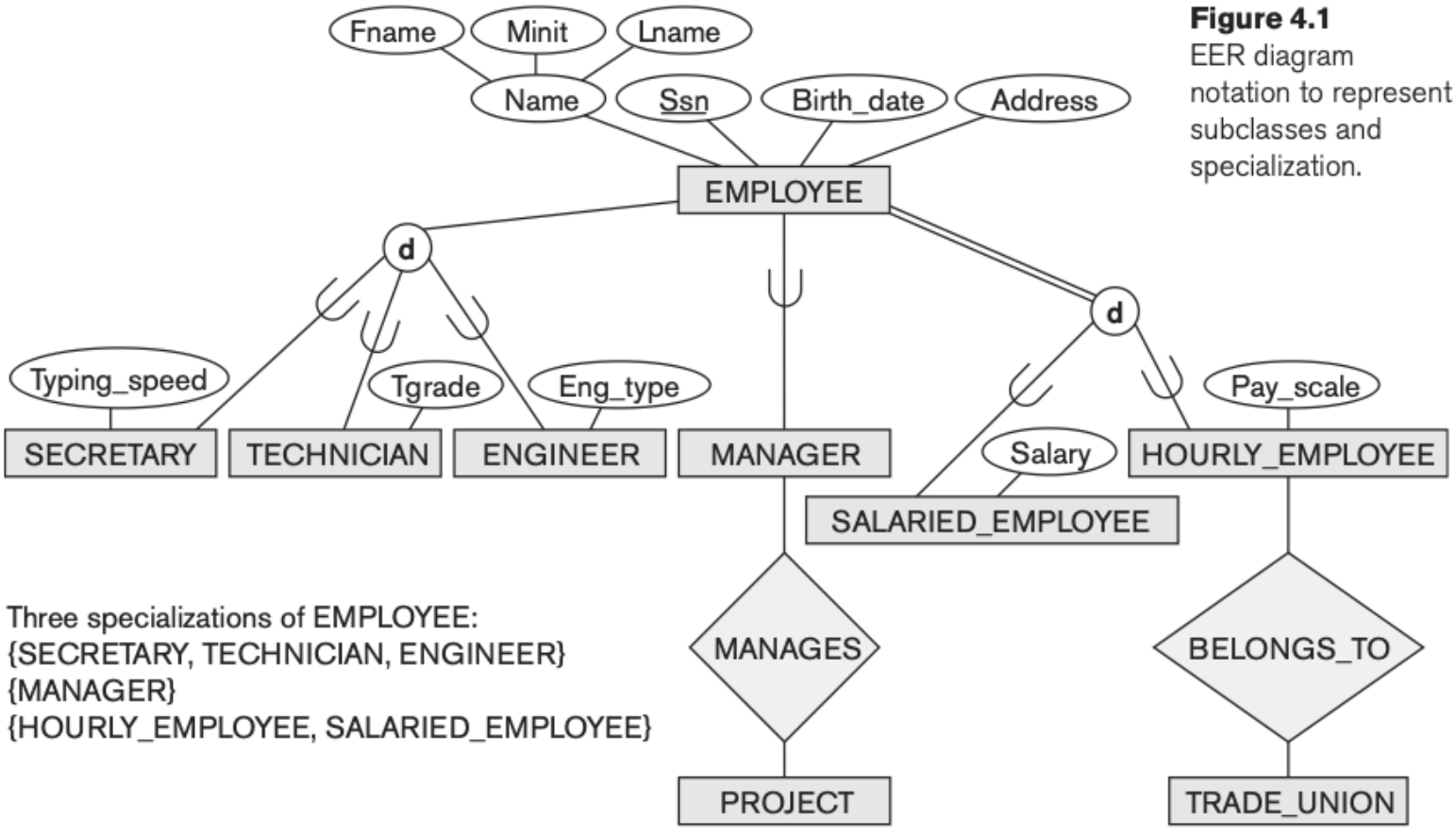


Figure 4.1
EER diagram notation to represent subclasses and specialization.

Definitions for EER concepts

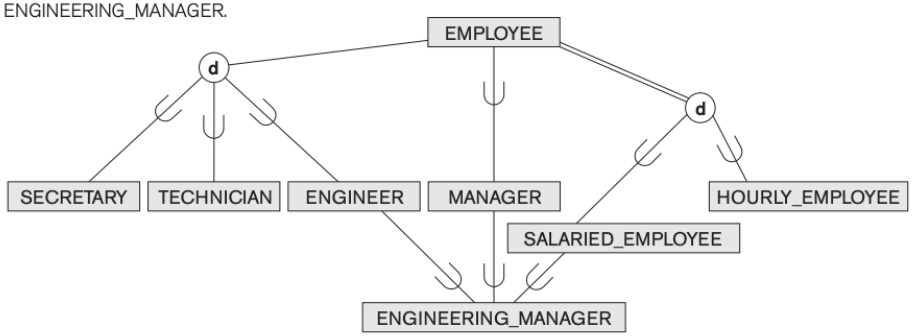
- A *class* is a set or collection of entities of a defined type.
- A *subclass* S is a class whose entities must always be a subset of the entities in another class, called the *superclass* C. S has an *is-a* relationship with C.
- A *specialisation* $Z = S_1, S_2, ..., S_n$ is a set of subclasses that have the same superclass G.
  - Z is said to be *total* if the superclass G is always equal to the union of all the subclasses. Otherwise it is *partial*.
  - Z is said to be *disjoint* if there is never an intersection between the subclasses. Otherwise it is *overlapping*.

Specialisation and Generalisation - Hierarchies and Lattices

A subclass may have further subclasses. This nesting will either result in a tree-like (hierarchical) or graph-based (lattice) structure.

- In a hierarchical structure, each subclass only has one parent.
- In a lattice structure, each subclass can participate in more than one class/subclass relationship.

Figure 4.6
A specialization lattice with shared subclass ENGINEERING_MANAGER.



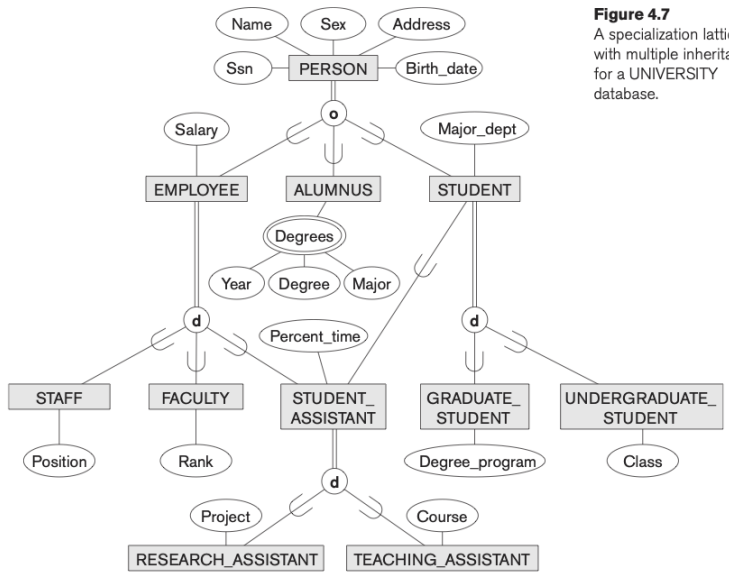


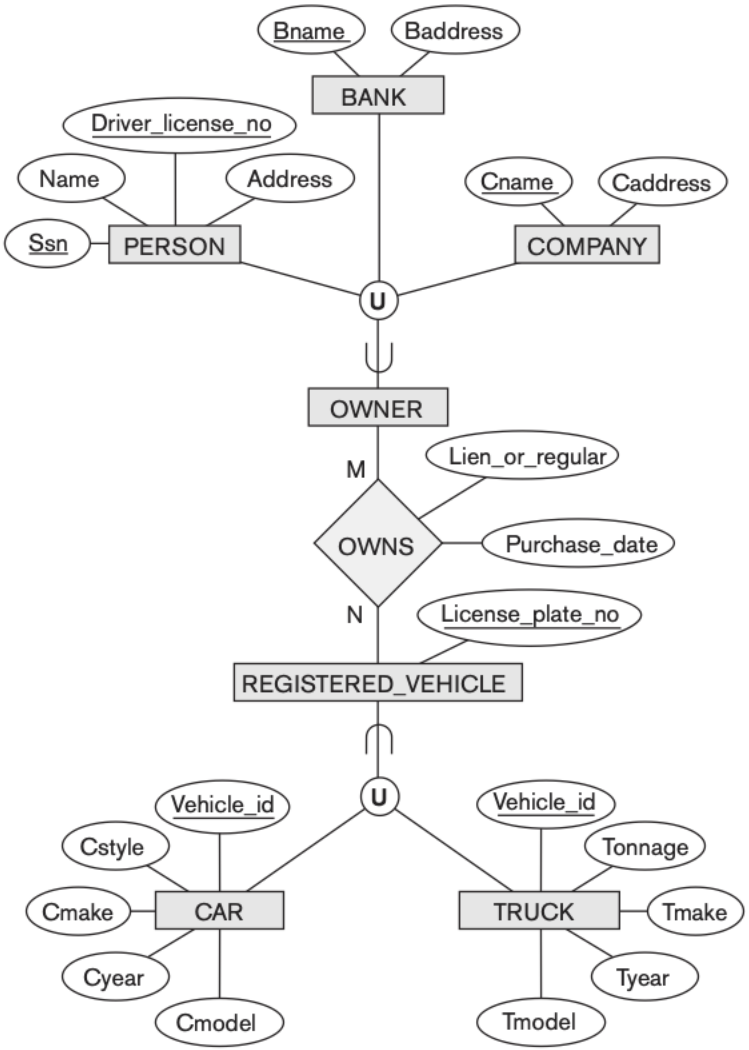
Figure 4.7
A specialization lattice with multiple inheritance for a UNIVERSITY database.

Modelling UNION Types

- Up till now we have considered entities with a single superclass. For example the EMPLOYEE entity type.
- Sometimes a superclass/subclass relationship requires more than one superclass.
- Some tools cannot model UNION types and therefore a work-a-round needs to be defined. This will be discussed in a later chapter.

Modelling UNION Types

- The subclass represents a collection of objects formed by the UNION of distinct entity types.
- The subclass is referred to as the *union type* (or *category*).
- A UNION is denoted in EER-diagrams with by placing a **U** in the circle.



Modelling UNION Types

- OWNER is a union type (or category) of the PERSON or BANK or COMPANY entity types. REGISTERED_VEHICLE is a subclass of CAR or TRUCK.
- An entity that is a member of OWNER, must exist in only one of its superclasses. That is, an OWNER may be a PERSON, BANK or COMPANY.
- Attribute inheritance works selectively for union types. The subclass inherits attributes from one of its superclasses. This means that a superclass of a category may hold different keys. For example, OWNER with PERSON, BANK or COMPANY holds different keys.

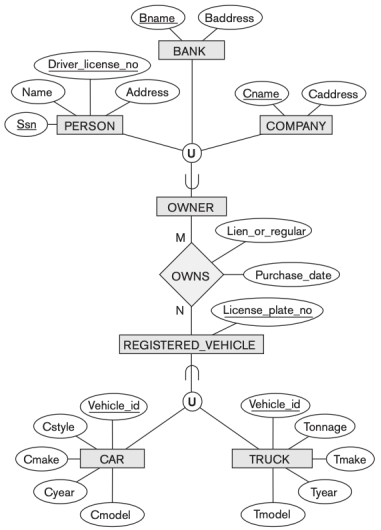


Figure 4.8
Two categories (union types): OWNER and REGISTERED_VEHICLE.

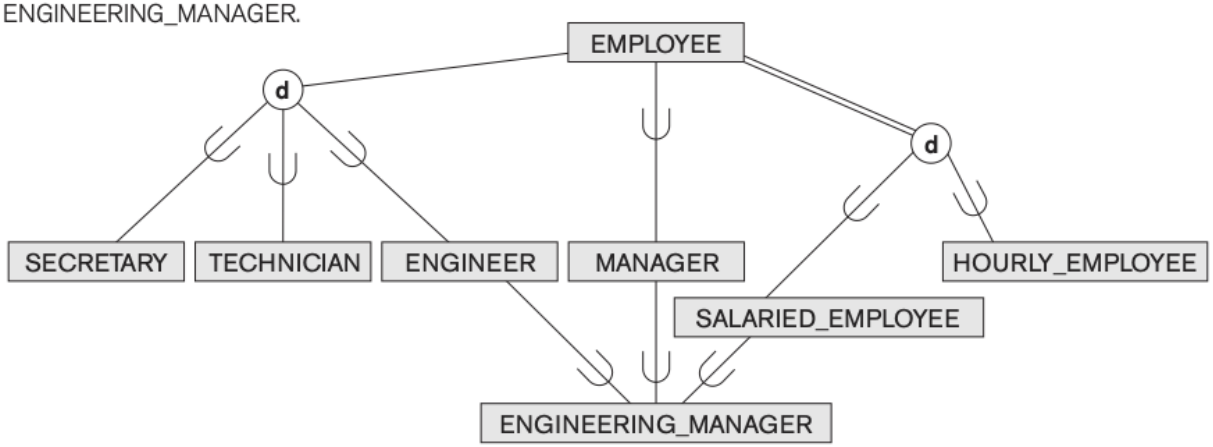
Modelling UNION Types

- A union type may be *partial* or *total*.
  - A *partial* category holds a subset of the union.
  - A *total* category holds the UNION of all entities in its superclass. A double line connecting the category and the circle is used to differentiate the total category.

Modelling UNION Types

Constrast a UNION with the engineering manager. An ENGINEERING_MANAGER must be an ENGINEER and a MANAGER and a SALARIED_EMPLOYEE.

Figure 4.6
A specialization lattice with shared subclass ENGINEERING_MANAGER.



EER Design Choices

- Specialisation and subclasses make the model more *accurate* but increase *clutter*.
- A subclass with **few local attributes** and **no relationship types** can be *merged into superclass*. Could then use 'Type' attribute and 'NULL' values for the non-members of the merged subclass.
- If **all subclasses** meet the requirements above, they can similarly be merged into their superclass.
- **Avoid UNION types** and categories as much as possible. Specialisation/generalisation usually sufficient.
- If the **requirements** do not indicate any particular constraints, the default would generally be *overlapping and partial*, since this does not specify any restrictions on subclass membership.

University Example - Requirements

1. The database keeps track of three types of persons: employees, alumni, and students. A person can belong to one, two, or all three of these types. Each person has a name, SSN, sex, address, and birth date.
2. Every employee has a salary, and there are three types of employees: faculty, staff, and student assistants. Each employee belongs to exactly one of these types. For each alumnus, a record of the degree or degrees that he or she earned at the university is kept, including the name of the degree, the year granted, and the major department. Each student has a major department.
3. Each faculty has a rank, whereas each staff member has a staff position. Student assistants are classified further as either research assistants or teaching assistants, and the percent of time that they work is recorded in the database. Research assistants have their research project stored, whereas teaching assistants have the current course they work on.
4. Students are further classified as either graduate or undergraduate, with the specific attributes degree program (M.S., Ph.D., M.B.A., and so on) for graduate students and class (freshman, sophomore, and so on) for undergraduates.

University Example - Notations

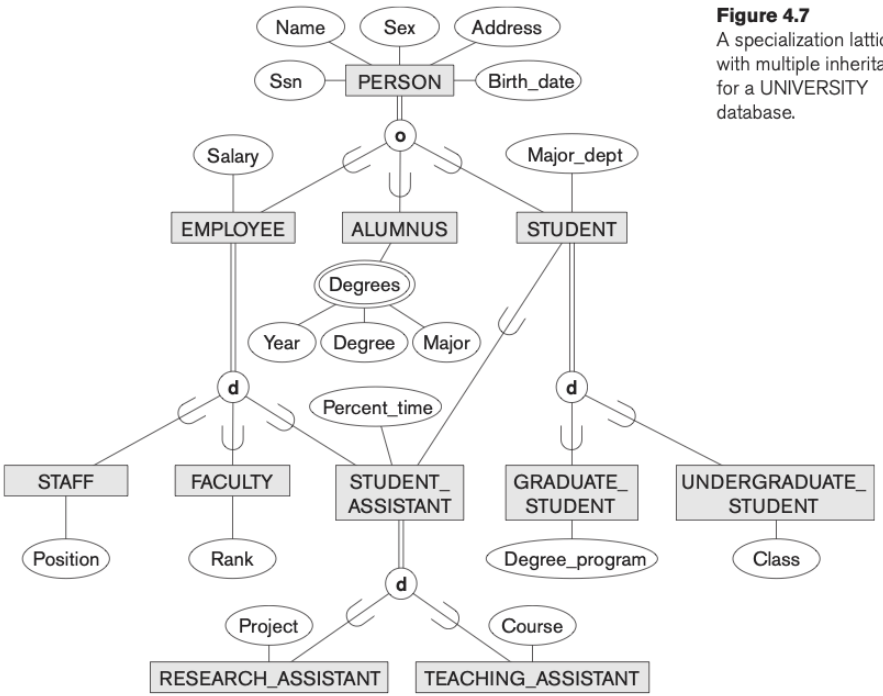


Figure 4.7
A specialization lattice with multiple inheritance for a UNIVERSITY database.

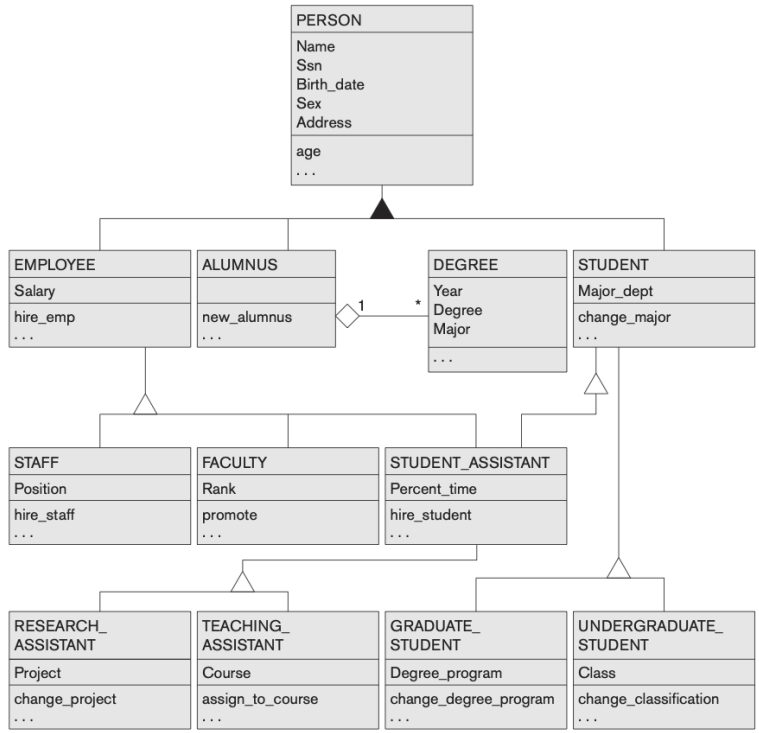


Figure 4.10
A UML class diagram corresponding to the EER diagram in Figure 4.7, illustrating UML notation for specialization/generalization.

COS221
L07 - DB Design using UML
(Chapter 10 in Edition 6 and Chapter 3.8 and 4.6 in Edition 7)

Linda Marshall

6 March 2023

Multiple methodologies exist to represent database design. These include:

- Chen's - extended to include generalisations and specialisations. Good educational notation.
- Information Engineering (IE) - most popular and has the most products that can be used to model and translate the model to SQL
- Barker's - used by Oracle and when using Oracle products
- UML - Only notation (used for OO modelling) that adheres to a standard.

Introduction to UML

- *Class*, comprises of *attributes* and *operations*. Attribute domains (data types) are optional. Operations are not specified on ER diagrams. In UML operations can be viewed as *stored procedures*.
- Modelling complex attributes - *Composite* attributes modelled as structured domains, e.g. Name. *Multivalued* attributes are modelled as a separate class, e.g. LOCATION.
- *Relationships* are modelled using associations.
  - Associations have *multiplicity* and *roles*.
  - There are two variants of association, aggregation and composition.
  - Associations can be unidirectional (arrowed) or bidirectional.
  - *Reflexive* associations link to self.

UML notation used in data modelling

- Adding an association name to the association (dashed line on association) provides *ordering* to the relationship based on the attributes specified in the box.
- Weak entities are modelled using UML *qualified associations/aggregations*. A *descriminator* is used to model the partial key.

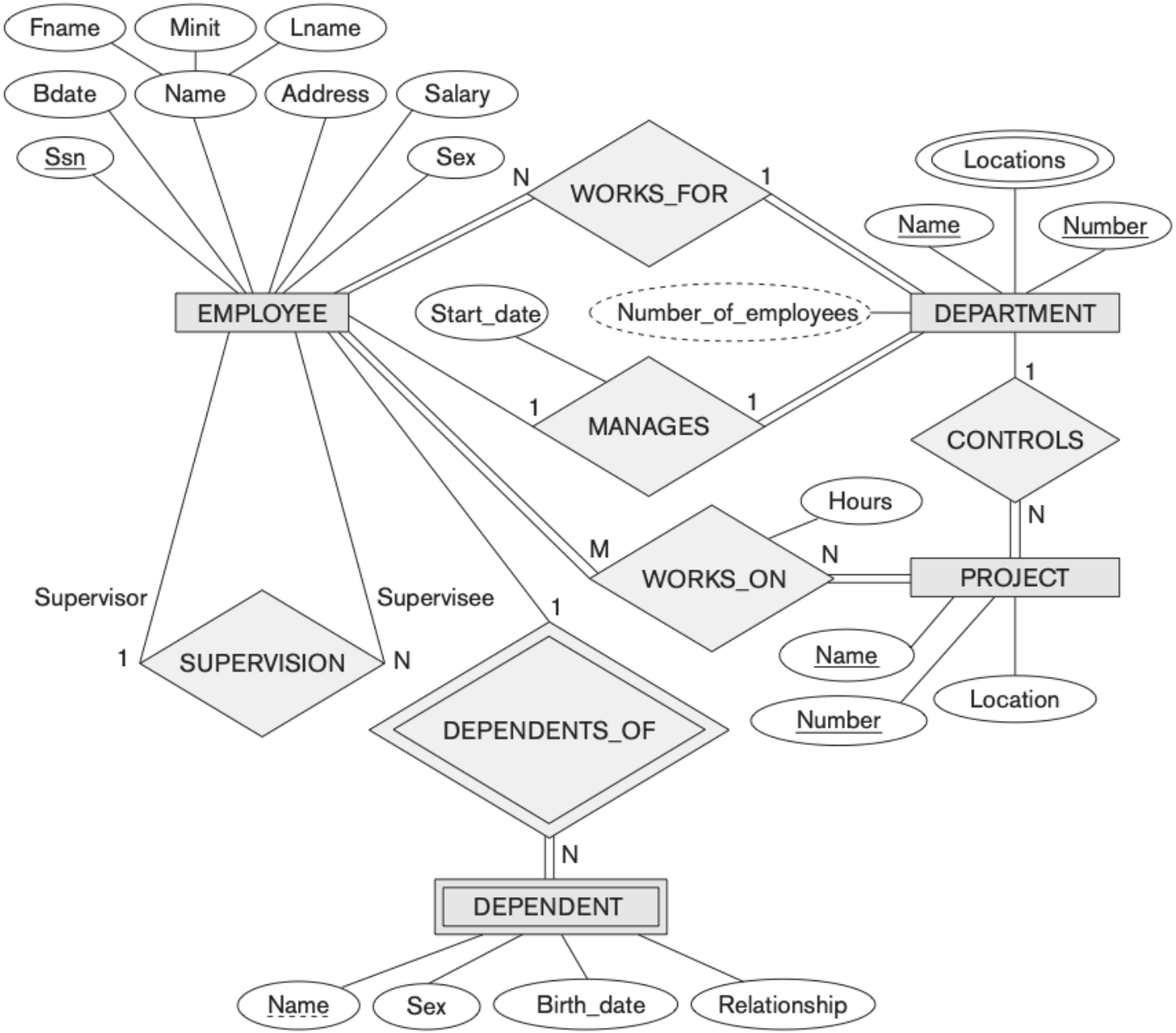


Figure 3.2
An ER schema diagram for the COMPANY database. The diagrammatic notation is introduced gradually throughout this chapter and is summarized in Figure 3.14.

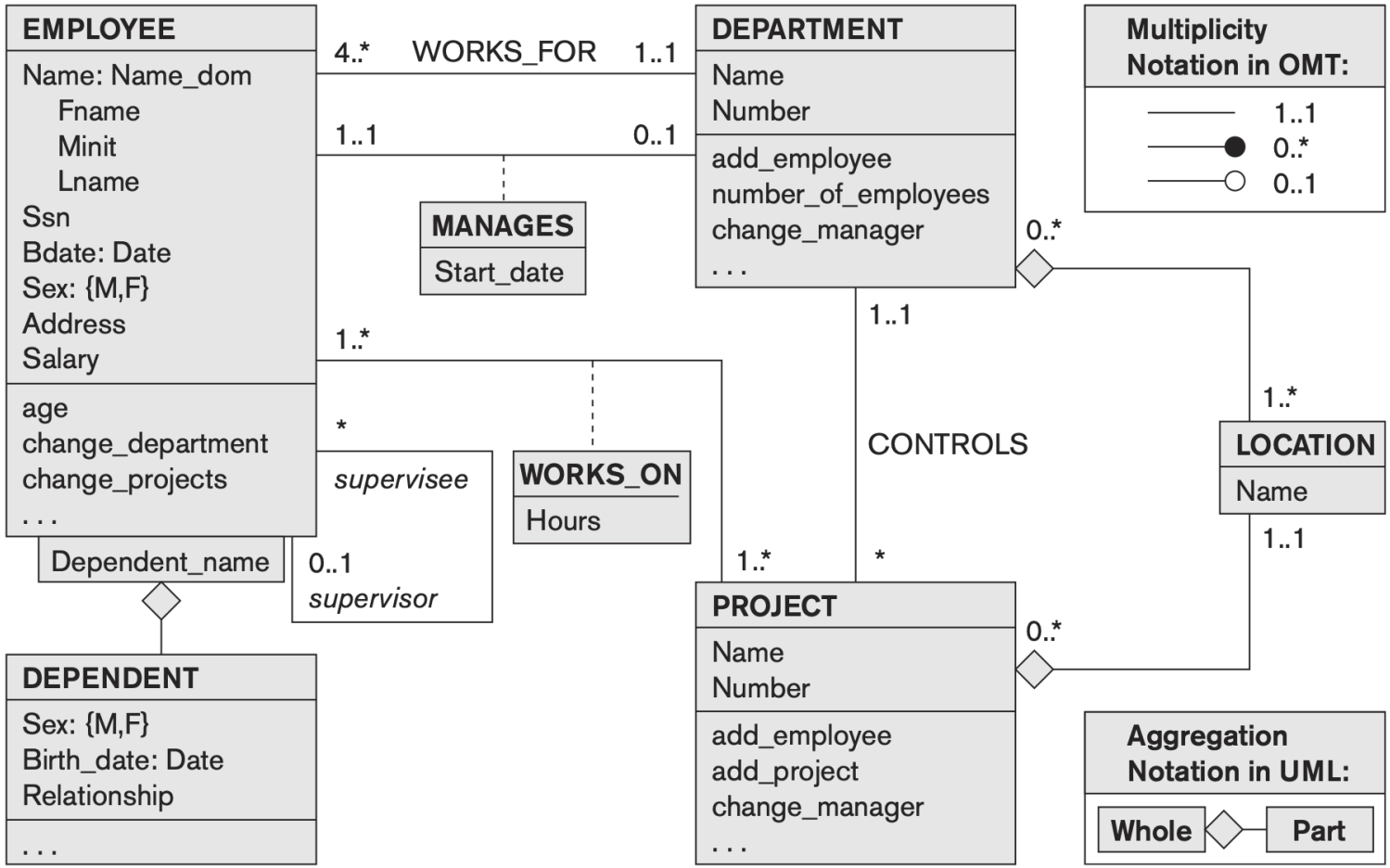


Figure 3.16
The COMPANY conceptual schema in UML class diagram notation.

Modelling Specialisation and Generalisation

- Make use of *inheritance*
  - Extend the UML inheritance notation to make provision for *overlapping* constraints (filled-in triangle). *Disjoint* constraints makes use of the standard inheritance triangle.
  - In UML classes can be modelled as Abstract, Concrete and Template. For database design the models typically make use of Concrete classes for entities.

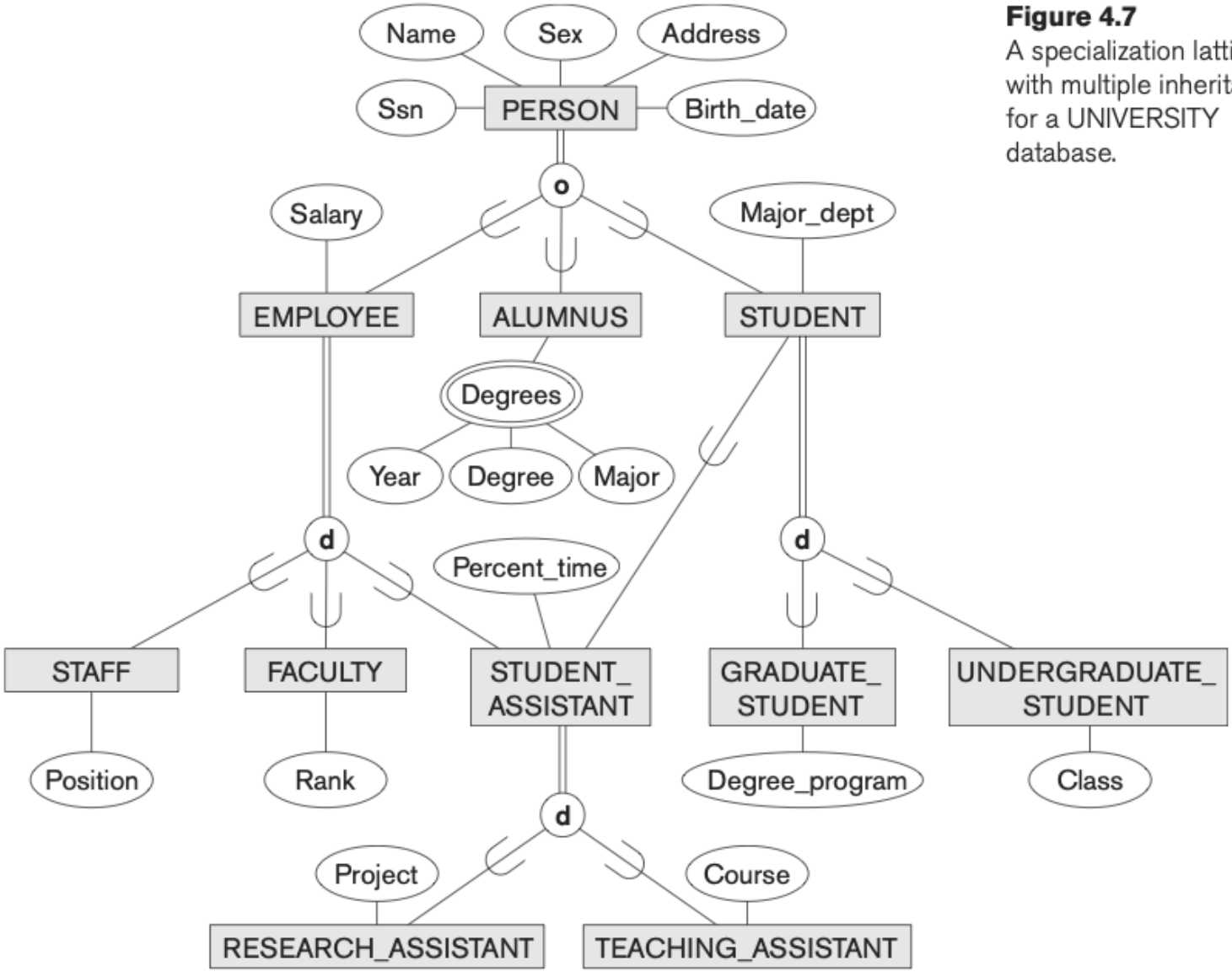


Figure 4.7
A specialization lattice with multiple inheritance for a UNIVERSITY database.

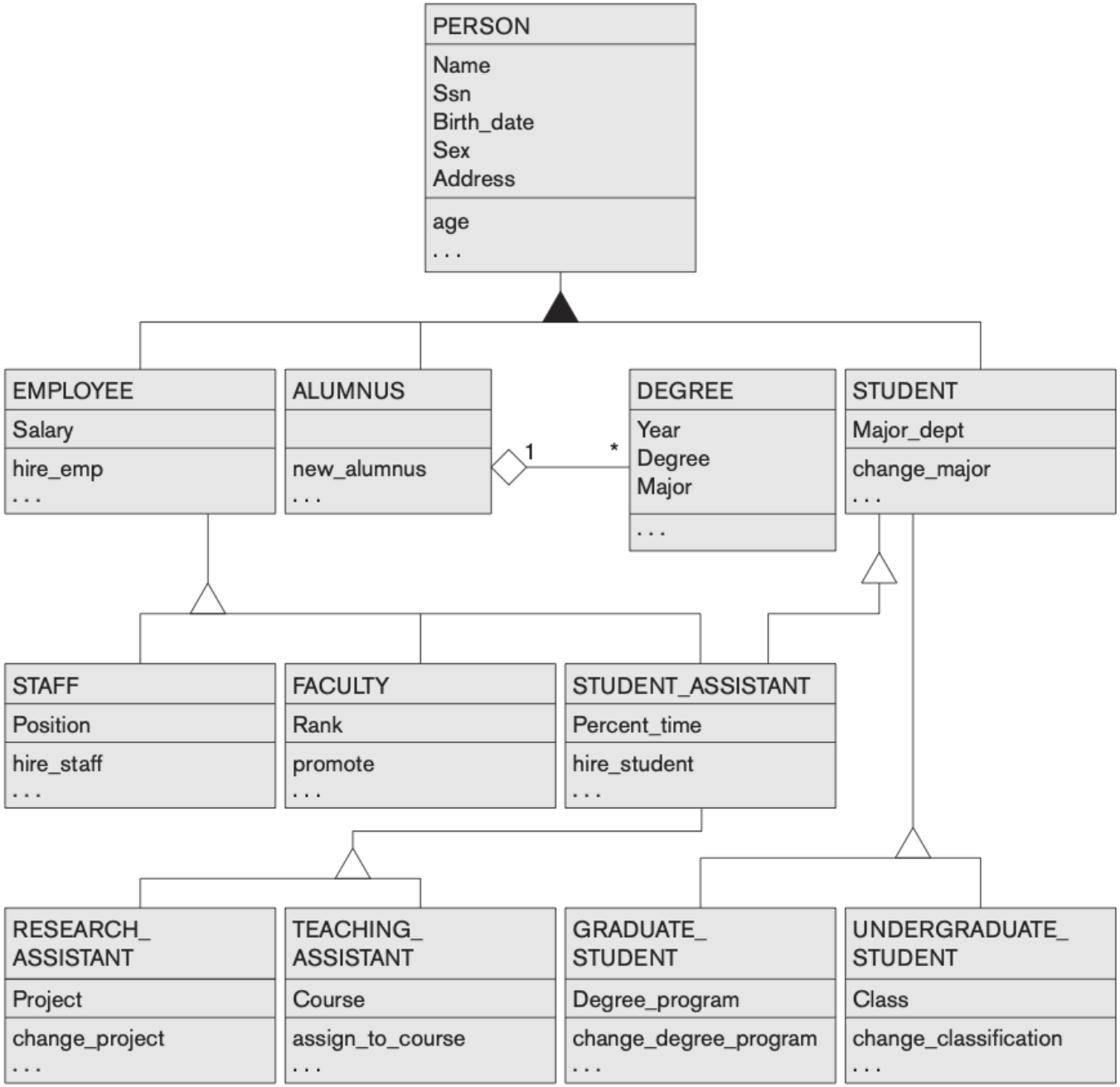
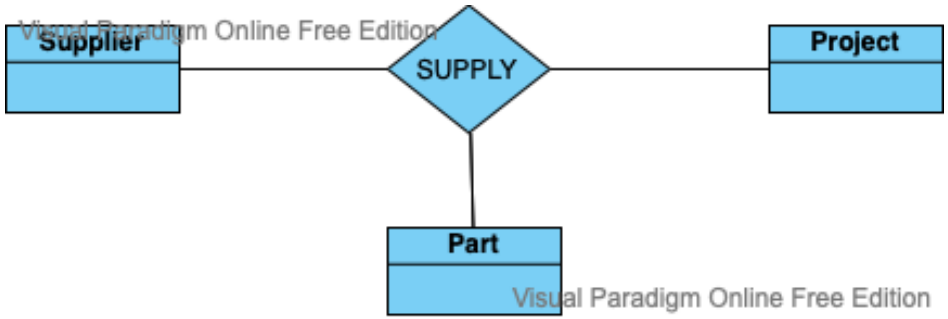


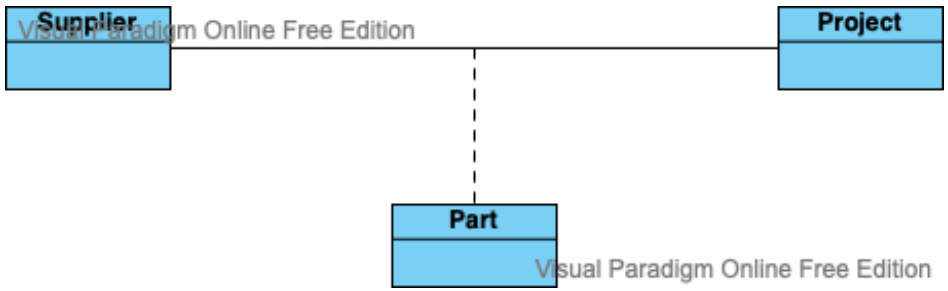
Figure 4.10
A UML class diagram corresponding to the EER diagram in Figure 4.7, illustrating UML notation for specialization/generalization.

N-ary Relationships

- N-ary relationships are modelled in a similar manner to Chen. A diamond links the associations between the classes.



- To transform the relationship to a binary relationship, make use of the dashed line on the association in UML to model the ternary relationship - association relationship.



UML Extensions

- Some tool manufacturers have developed UML extensions to model concepts such as primary keys in the UML database models. The depiction is similar to the Barker notation for primary keys.

COS221
L08 - Relational Model and Constraints
(Chapter 3 in Edition 6 and Chapter 5 in Edition 7)

Linda Marshall

10 March 2023

Relational Model

- First introduced by Codd in 1970
- Became popular because of its simplicity and mathematical foundation
  - Basic building block - a *table of values*
  - Theoretical basis - set theory
- First commercial RDBMSs became available in the 1980's
  - DB2 from IBM
  - Oracle
  - Sybase (now from SAP)
  - SQLServer and Access from Microsoft
  - MySQL, PostgreSQL and MariaDB, all open source

Relational Model Concepts

- A database is represented as a collection of relations
- A relation can be seen as a table (flat file) of values (records)
  - each row (*tuple* in relational database terminology) of a table (*relation*) is collection of related values and typically corresponds to a real-word entity or relationship
  - the table name and column names (*attributes*) are used to interpret the meaning of the values in each row
  - the values in the columns are described by types (*domains*)

Relational Model Concepts - *Relation schema*

A *relation schema*, R, is made up of a relation name, R, and a list of attributes: $A_1, A_2, ..., A_n$ and denoted by $R(A_1, A_2, ..., A_n)$

- Each attribute, A_i, is the name of the role played in domain, D, in the relation schema R.
- The domain of A_i is denoted by $dom(A_i)$
- The *degree* (arity) of a relation is the number of attributes (n) of its relation schema

For example:

```
STUDENT(Name:string, SSn:string, Cell_number:string,
        Age:integer, Gpa:real)
```

Relational Model Concepts - *Relation*

An alternative representation of a relation is as a **predicate**.

- Values in a tuple are interpreted as values satisfying a predicate
- The relational model can then be manipulated by logic languages such as Prolog

The **closed world assumption** states that only true facts in the universe are those present within the extension (state) of the relation(s). This comes in useful when queries based on relational calculus are considered.

Relational Model Concepts - *Relation*

- A relation, r, of $R(A_1, A_2, ..., A_n)$, also denoted by $r(R)$, is the set of n-tuples, $r = \{t_1, t_2, ..., t_n\}$
- Each n-tuple, t, is an ordered list of n values
  $t =< v_1, v_2, ..., v_n >$
- Each v_i is an element $dom(A_i)$, or a special value NULL – that is, not known, not available, undefined or may not apply to the tuple
- The i^{th} value of tuple t, which corresponds to attribute A_i, is referred to as $t[A_i]$,$t.A_i$ or $t[i]$

Relational Model Concepts - *Relation*

- R, the relation schema, is referred to as the **relation intension**, while $r(R)$ – the relation state – is referred to as the **relation extension**
- A tuple can be seen as asserting a fact. Some relations represent facts about *entities* and others about *relationships*. For example, a relation:

  `MAJORS(Student_id, Department_code)`

  asserts that a students major in specific disciplines
- (E)ER conceptual models will be converted to relations where some relations represent the entities and others represents relations in the (E)ER-diagrams

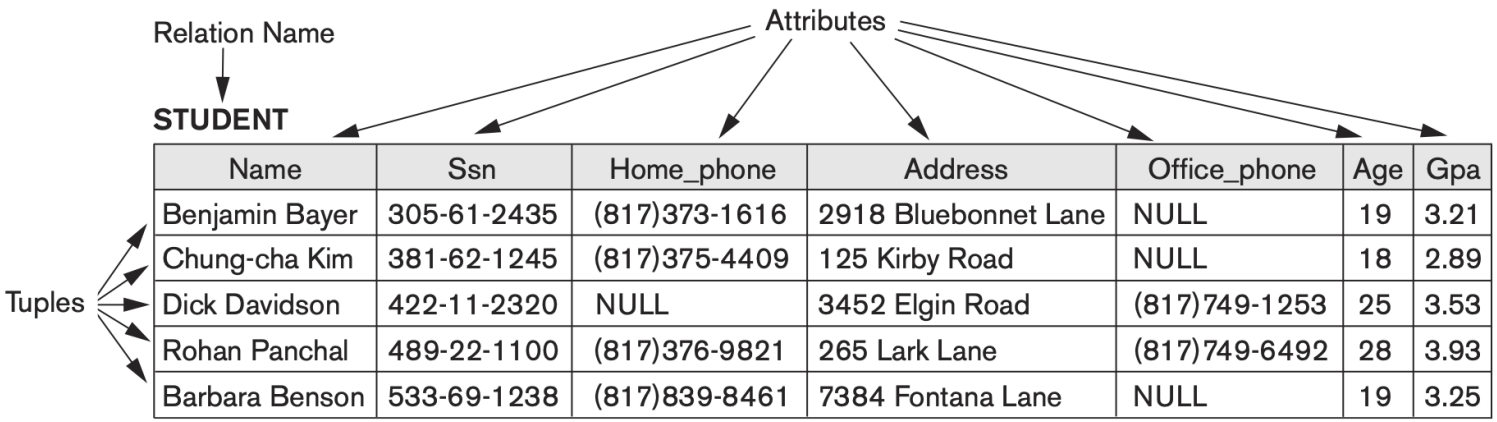
Relational Model Concepts - *Domain*

- A domain, D, is a set of atomic values. That is, they are indivisible in the relational model.
  - Atomic implies that neither composite nor multivalued attributes are allowed in the relational model. This is the reason for the first normal form (1NF) assumption.
- A domain is defined by a name, data type and format.
  - Naming domains helps with the interpretation of the values
  - Data types are used to specify the type of the data values in the domain
  - Format ensures that the data, if need be, adheres to a specific format
- For example: The domain `Telephone_number` is a set of 10 digits. The data type of a domain is a character string of the form `(ddd) ddd-dddd`

Relational Model Concepts - *Domain*

- A relation $r(R)$ of degree n on domains $dom(A_1), dom(A_2), ..., dom(A_n)$ is a subset of the cartesian product of the domains that define R
  $r(R)$ $(dom(A_1) \times dom(A_2) \times ... \times dom(A_n))$
- The total number of tuples in the cartesian product will be: $|dom(A_1)| \times |dom(A_2)| \times ... \times |dom(A_n)|$, where $|D|$ denotes the cardinality of the domain

Relational Model Concepts - *Summary*

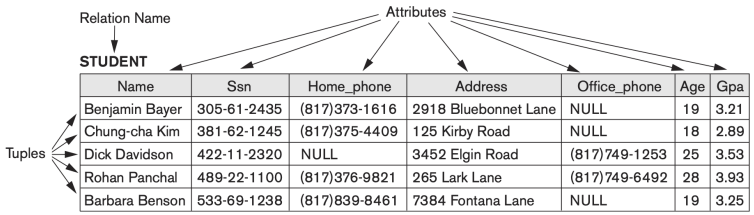


Name	Ssn	Home_phone	Address	Office_phone	Age	Gpa
Benjamin Bayer	305-61-2435	(817)373-1616	2918 Bluebonnet Lane	NULL	19	3.21
Chung-cha Kim	381-62-1245	(817)375-4409	125 Kirby Road	NULL	18	2.89
Dick Davidson	422-11-2320	NULL	3452 Elgin Road	(817)749-1253	25	3.53
Rohan Panchal	489-22-1100	(817)376-9821	265 Lark Lane	(817)749-6492	28	3.93
Barbara Benson	533-69-1238	(817)839-8461	7384 Fontana Lane	NULL	19	3.25

Figure 5.1
The attributes and tuples of a relation STUDENT.

Relational Model Characteristics - Tuples

- A relation is defined as a set of tuples. This means there is no ordering of tuples.
- The relations below are therefore seen as identical



Name	Ssn	Home_phone	Address	Office_phone	Age	Gpa
Benjamin Bayer	305-61-2435	(817)373-1616	2918 Bluebonnet Lane	NULL	19	3.21
Chung-cha Kim	381-62-1245	(817)375-4409	125 Kirby Road	NULL	18	2.89
Dick Davidson	422-11-2320	NULL	3452 Elgin Road	(817)749-1253	25	3.53
Rohan Panchal	489-22-1100	(817)376-9821	265 Lark Lane	(817)749-6492	28	3.93
Barbara Benson	533-69-1238	(817)839-8461	7384 Fontana Lane	NULL	19	3.25

Figure 5.1
The attributes and tuples of a relation STUDENT.

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

STUDENT

Name	Ssn	Home_phone	Address	Office_phone	Age	Gpa
Dick Davidson	422-11-2320	NULL	3452 Elgin Road	(817)749-1253	25	3.53
Barbara Benson	533-69-1238	(817)839-8461	7384 Fontana Lane	NULL	19	3.25
Rohan Panchal	489-22-1100	(817)376-9821	265 Lark Lane	(817)749-6492	28	3.93
Chung-cha Kim	381-62-1245	(817)375-4409	125 Kirby Road	NULL	18	2.89
Benjamin Bayer	305-61-2435	(817)373-1616	2918 Bluebonnet Lane	NULL	19	3.21

Relational Model Characteristics - Attributes

- The ordering of attributes is important. An n-tuple is an ordered list of n values. When manipulating n-tuples, the ordering of the corresponding attributes and its values need to be maintained
- If ordering of attributes is not defined as part of the relation definition, then a tuple can be defined as a set of $(< attribute >, < value >)$ pairs to preserve the mapping from R to D for each t_i. In this mapping, D is the union of domains of the attributes in R.

$t = <$ (Name, Dick Davidson),(Ssn, 422-11-2320),(Home_phone, NULL),(Address, 3452 Elgin Road),
(Office_phone, (817)749-1253),(Age, 25),(Gpa, 3.53)>

$t = <$ (Address, 3452 Elgin Road),(Name, Dick Davidson),(Ssn, 422-11-2320),(Age, 25),
(Office_phone, (817)749-1253),(Gpa, 3.53),(Home_phone, NULL)>

Figure 5.3
Two identical tuples when the order of attributes and values is not part of relation definition.

Relational Model Constraints

- Relational model constraints exist because there are constraints between the tuples of the relations of a database.
- Constraints are divided into the following categories:
  - Implicit or **inherent model-based** constraints
  - Explicit or **schema-based** constraints. These are directly expressed in schemas by specifying them in the DDL (Data Definition Language). These include: domain, key, NULL value, entity integrity and referential integrity constraints.
  - Semantic or **application-based** constraints, also referred to as business rules. The application programmes usually check these when performing updates.
  - **Dependency** constraints such as functional and multivalued dependencies. These are used to test the 'goodness' of the design.

Relational Model Constraints - Schema-based

The following schema-based constraints exist:

- Domain
- Key
- NULL values
- Entity integrity
- Referential integrity

Relational Model Constraints - Schema-based

- **Domain** - An attribute, A, must be an atomic value from the domain, $dom(A)$
- **Key**
  - As all elements of a set must be unique, all tuples in a relation must be unique/distinct.
  - If we can find a subset of attributes (SK, the superkey) of a relation so that for any two distinct tuples t_1 and t_2, in r of R, $t_1[SK]! = t_2[SK]$, then R is distinct.
  - A minimal superkey, or key, is defined as a set of attributes from which the removal of a single attribute no longer renders it unique.
  - A relation may have more than one key. If this is the case, then each of the potential keys is referred to as a **candidate key**. One of the candidate keys is designated the **primary key**. Only the primary keys are *underlined* when drawing a relational schema.
  - A key comprising of multiple attributes, requires all attributes together to provide uniqueness.

Relational Model Constraints - Schema-based

- **NULL values** - An attribute defined as NOT NULL, may not hold the NULL value.
- **Entity integrity** - No primary key value may be NULL.
- **Referential integrity** - Referential integrity constraints are specified between two relations and is used to maintain consistency between the relations.

Relational Model Constraints - Schema-based

- **Referential integrity** - *continued* - To formally specify referential integrity, the concept of a **foreign key** is needed. A set of attributes, FK in a relation schema R_2, is a foreign key of R_2 that references relation R_1 if it satisfies the following:
  - The attributes of FK have the same domain as the PK attributes of R_1. FK references or refers to R_1.
  - A value of FK in the tuple t_2 of the current state $r_2(R_2)$ either occurs as a value of PK for some tuple t_1 in the current state $r_1(R_1)$ or is NULL. If not NULL, we have $t_2[FK] = t_1[PK]$ and t_2 references or refers to tuple t_1.

  R_2 is referred to as the *referencing relation* and R_1 the *referenced relation*. If the conditions hold, a referential integrity constraint from R_2 to R_1 holds. That is, from the FK in R_2 to the PK in R_1.

Relational Database Schema

- A relational database schema, S, is a set of relation schemas, $S = \{R_1, R_2, ..., R_m\}$ and a set of integrity constraints (IC).

EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno

DEPARTMENT

Dname	Dnumber	Mgr_ssn	Mgr_start_date

DEPT_LOCATIONS

Dnumber	Dlocation

PROJECT

Pname	Pnumber	Plocation	Dnum

WORKS_ON

Essn	Pno	Hours

DEPENDENT

Essn	Dependent_name	Sex	Bdate	Relationship

Figure 5.5
Schema diagram for the COMPANY relational database schema.

Note: Attribute names are context specific. We could have had `Name` in both PROJECT and DEPARTMENT. `Dno`, `Dnumber` and `Dnum` all refer to the department number.

Relational Database Schema

- A relational database state (DB of S) is the set of relation states $DB = \{r_1, r_2, ..., r_m\}$.

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

EMPLOYEE

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

DEPARTMENT

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

DEPT_LOCATIONS

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

WORKS_ON

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

PROJECT

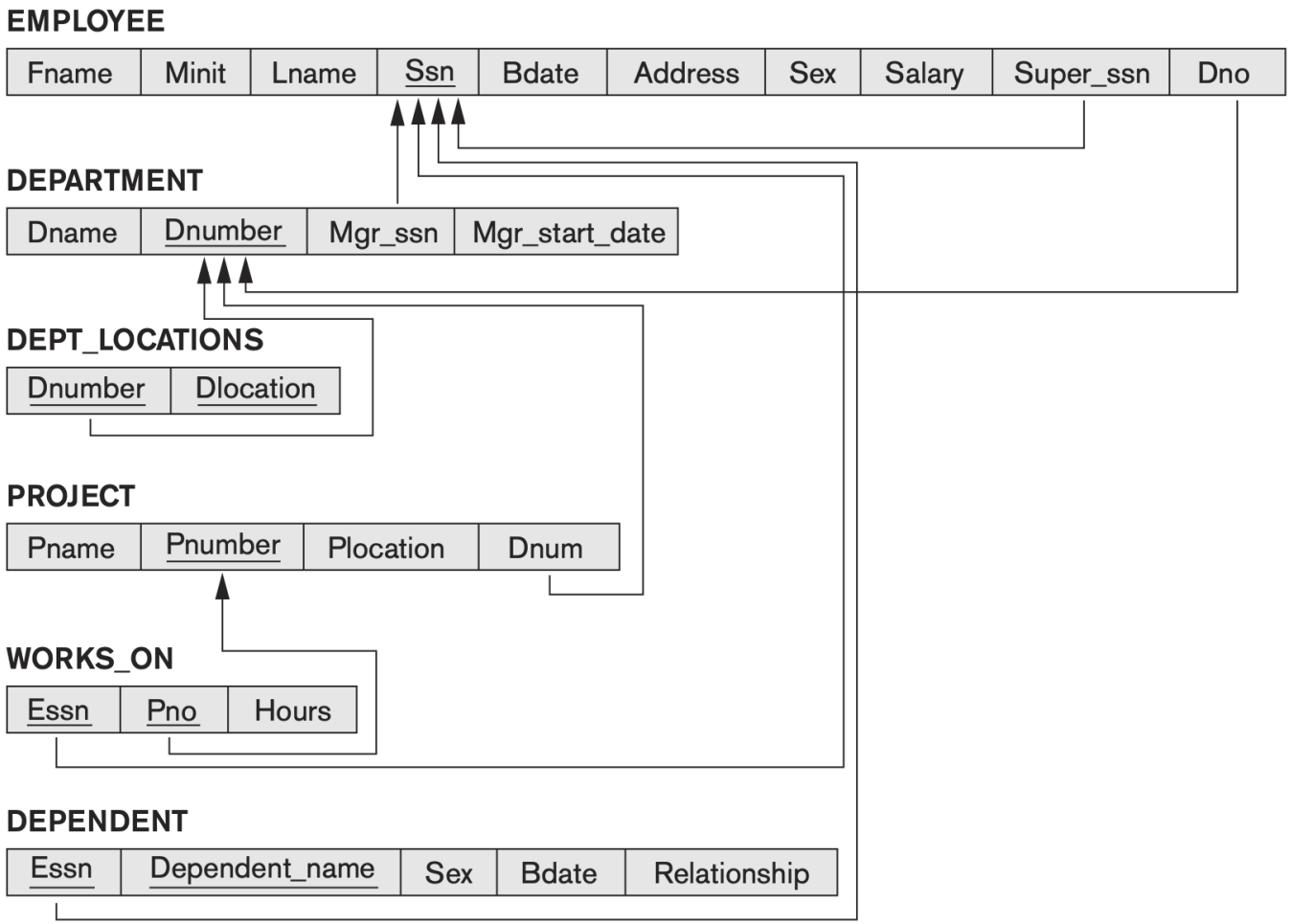
Pname	Pnumber	Plocation	Dnum
ProductX	1	Bellaire	5
ProductY	2	Sugarland	5
ProductZ	3	Houston	5
Computerization	10	Stafford	4
Reorganization	20	Houston	1
Newbenefits	30	Stafford	4

DEPENDENT

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

Relational Database Schema - Referential Integrity

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



Update operations and dealing with constraint violations

Update operations fall into two categories.

- **Retrieval** - makes use of Relational Algebra to query the database. The result relation is the answer to the user's query. This will be covered later.
- **Update** - Changes the state of the relations in the database. Three basic operations exist to change the state of a database.
  - Insert - Used to add one or more tuples into a relation
  - Delete - Used to delete tuples.
  - Update/Modify - Used to change values of some attributes of existing tuples.

Update operations and dealing with constraint violations - Insert

Provides a list of attribute values for a new tuple t. It can violate:

- Domain constraints - attribute value does not appear in the corresponding domain or not of the appropriate data type.
- Key constraints - key value in the new tuple already exists in another tuple in $r(R)$
- Entity integrity constraints - any part of the primary key in the new tuple is NULL
- Referential integrity constraints - the value of a foreign key refers to a tuple that does not exist in the referenced relation

Update operations and dealing with constraint violations - Delete

Delete only violates referential integrity constraint. The following options exist if delete causes a violation:

- Restrict - reject the deletion
- Cascade - propagate the deletion by deleting tuples that reference the the tuple being deleted
- Nullify - set the referencing attributes that cause the violation to NULL or change to reference another default valid tuple. If the referencing tuple is part of the primary key it cannot be set to NULL so as to not violate entity integrity

Update operations and dealing with constraint violations - Update/Modify

- Used to change the values of one or more attributes in the tuple(s) of some relation R
- A condition on attribute(s) of the relation needs to be specified to select the tuple(s) to be modified
- Updating an attribute that is neither part of a primary key or a foreign key causes no problems
- Modifying the primary key is similar to deleting the tuple and adding a new one with the modified values in its place.
- When modifying a foreign key, the value to which it is modified must exist in the relation being referred to, or must be set to NULL

Update operations and dealing with constraint violations

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

EMPLOYEE

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

DEPARTMENT

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

DEPT_LOCATIONS

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

WORKS_ON

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

PROJECT

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

DEPENDENT

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

Transactions and dealing with constraint violations

- A transaction is a program that manipulates - reads and/or updates - the database
- Once a transaction has completed, the database must be in a valid or consistent state that satisfies the constraints specified by the database schema
- In large online transaction processing (OLTP) systems, many transactions run in milliseconds and concurrently and therefore concurrency and recovery from failure needs to be managed

COS221
L09 - Introduction to SQL
(Chapter 4 in Edition 6 and Chapter 6 in Edition 7)

Linda Marshall

16 March 2023

SQL Background

- SQL (or Structured Query Language which is a shortened form of the original name SEQUEL (Structured English QUEry Language)) forms the backbone of Relational Database Systems.
- There are differences in implementations of SQL in available RDBMS's. The fundamentals, however, are the same and most SQL programs are easily transferred between RDBMS's.
- SQL provides a high-level declarative language stating what the query is and not how it must happen.
- SQL is a comprehensive database language. It can be used for data definitions, queries and updates. It is therefore a DDL and a DML. Additionally, views can be defined.
- Over the years, SQL has been extended to provide functionality for data warehousing, data mining, multimedia etc.

Data definitions and data type in SQL - Terminology

SQL uses the terminology of:

- Table (previously seen as Relation in the Relational Model and Entity type in the (E)ER model
- Row (referred to as a Tuple in the Relational Model and part of the Entity set in (E)ER)
- Column (thankfully Attributes in both the Relational Model and (E)ER)

The CREATE statement is used to create schemas, tables, types and domains. Additionally the CREATE statement is used for constructs such as views, assertions and triggers.

Schemas and Catalogues

A *schema* is used to group together tables and other constructs that fall under the same database application

- An SQL schema is identified by a name, an authorisation identifier - the user who owns the schema, and descriptors for each element in the schema.
- Schema elements include tables, types, constraints, views, domains, and other constructs - such as authorisation - to describe the schema.

A *catalogue* is a named collection of schemas

- Most users are moved directly into a schema they work with when they log into the database system
- Contains a schema INFORMATION_SCHEMA. This schema provides information on all the schemas in the catalog and all the element descriptors in these schemas.

Note: Integrity constraints such as referential integrity can be defined between relations only if they exist in schemas within the same catalog. Schemas within the same catalog can also share certain elements, such as type and domain definitions.

Creating a Schema

- Schemas are created with the CREATE SCHEMA statement.
- This statement may include all the elements in the schema or may include only the schema name and authorisation identifier
  CREATE SCHEMA COMPANY AUTHORIZATION 'John.Smith';
- Typically not all users are authorised to create schemas. Permissions to do so are assigned to a user by the DBA.
- Once a schema has been created, tables can be added to the schema.

Creating a Table

- The CREATE TABLE command is used to specify a new table/relation. Each new table is given a name, attributes and constraints.
- Attributes are specified first and are given a name, data type to specify the domain of values and possibly attribute constraints, e.g. NOT NULL.
- Constraints - key, entity integrity and referential integrity - are specified after the attributes are declared. Constraints can also be added later with the ALTER TABLE command.
- The schema name can be included when creating a table. If it is left out, the current schema is used, for example:
  CREATE TABLE COMPANY.EMPLOYEE
  or
  CREATE TABLE EMPLOYEE

Creating a Table

```
CREATE TABLE EMPLOYEE
        ( Fname                 VARCHAR(15)         NOT NULL,
          Minit                 CHAR,
          Lname                 VARCHAR(15)         NOT NULL,
          Ssn                   CHAR(9)             NOT NULL,
          Bdate                 DATE,
          Address               VARCHAR(30),
          Sex                   CHAR,
          Salary                DECIMAL(10,2),
          Super_ssn             CHAR(9),
          Dno                   INT                 NOT NULL,
        PRIMARY KEY (Ssn),
CREATE TABLE DEPARTMENT
        ( Dname                 VARCHAR(15)         NOT NULL,
          Dnumber               INT                 NOT NULL,
          Mgr_ssn               CHAR(9)             NOT NULL,
          Mgr_start_date        DATE,
        PRIMARY KEY (Dnumber),
        UNIQUE (Dname),
        FOREIGN KEY (Mgr_ssn) REFERENCES EMPLOYEE(Ssn) );
CREATE TABLE DEPT_LOCATIONS
        ( Dnumber               INT                 NOT NULL,
          Dlocation             VARCHAR(15)         NOT NULL,
        PRIMARY KEY (Dnumber, Dlocation),
        FOREIGN KEY (Dnumber) REFERENCES DEPARTMENT(Dnumber) );
CREATE TABLE PROJECT
        ( Pname                 VARCHAR(15)         NOT NULL,
          Pnumber               INT                 NOT NULL,
          Plocation             VARCHAR(15),
          Dnum                  INT                 NOT NULL,
        PRIMARY KEY (Pnumber),
        UNIQUE (Pname),
        FOREIGN KEY (Dnum) REFERENCES DEPARTMENT(Dnumber) );
CREATE TABLE WORKS_ON
        ( Essn                  CHAR(9)             NOT NULL,
          Pno                   INT                 NOT NULL,
          Hours                 DECIMAL(3,1)        NOT NULL,
        PRIMARY KEY (Essn, Pno),
        FOREIGN KEY (Essn) REFERENCES EMPLOYEE(Ssn),
        FOREIGN KEY (Pno) REFERENCES PROJECT(Pnumber) );
CREATE TABLE DEPENDENT
        ( Essn                  CHAR(9)             NOT NULL,
          Dependent_name        VARCHAR(15)         NOT NULL,
          Sex                   CHAR,
          Bdate                 DATE,
          Relationship          VARCHAR(8),
        PRIMARY KEY (Essn, Dependent_name),
        FOREIGN KEY (Essn) REFERENCES EMPLOYEE(Ssn) );
```

Figure 6.1
SQL CREATE TABLE data definition statements for defining the COMPANY schema from Figure 5.7.

Creating a Table

- Tables created using CREATE TABLE are referred to as base tables (or base relations) as opposed to virtual tables/relations created using the CREATE VIEW command.

Notes:

- Attributes in a base relation are ordered in the sequence they are specified.
- Some attributes may cause errors when created due to referring to tables that have not been created yet. In such cases, defining the attributes and then adding the constraints with ALTER TABLE solves this problem. Examples of this problem are:
  - creating a foreign key for Super_ssn before the table is created - circular reference; or
  - creating the foreign key for department number in EMPLOYEE before the DEPARTMENT table is created

Basic Datatypes

Basic datatypes include:

- Numeric
- Character string
- Bit string
- Boolean
- Date

Basic Datatypes

- Numeric: INTEGER (or INT), SMALLINT, FLOAT (or REAL), DECIMAL(i,j) (or DEC(i,j)), NUMERIC(i,j) where i is the precision (total number of decimal digits) and j is the scale (number of digits after the decimal point, default is 0)
- Character string: Fixed length - CHAR(n) (or CHARACTER(n), Varying length - VARCHAR(n) (or CHAR VARYING(n)), CLOB(nS), where S is specified in kilobytes (K), ... , gigabytes (G). CHAR's are generally padded with blanks which can be are ignored. A concatenation operator (||) exists to join two 'strings'
- Bit string: These can either be a fixed size (BIT(n)) or varying (BIT VARYING(n)). Literal bit strings are prefixed with B' (B followed by a single quotation mark) and suffixed with a ' (a single quotation mark). BLOBs, or BINARY LARGE OBJECTS are used to for a bitstream as images. Again in K, ..., G.

Basic Datatypes

- Boolean: BOOLEAN can be TRUE, FALSE or UNKNOWN (referred to as three value logic). The latter is necessary because of the presence of NULL values.
- Date: DATE has the form YYYY-MM-DD. Additionally there is a TIME (HH:MM:SS or TIME(i) where i is given in seconds) type which could be qualified with WITH TIME ZONE. If no time zone is specified the SQL session time zone is used. Values of date types can be compared ($<$).

Other data types

- Other data types, which may be system specific, may include:
  - Timestamp
  - Interval - relative value that is used to increment or decrement an absolute value.
- Other than specifying the data type directly, domains can be defined. This improves readability. For example:
  CREATE DOMAIN SSN_TYPE AS CHAR(9);
- It is also possible to create user defined types with the CREATE TYPE command

Specifying constraints in SQL

Basic constraints are checked using the CHECK clause at table creation, these include:

- key and referential integrity,
- restrictions on attribute domains and NULL's
- constraints on individual tuples within a relation.

Constraints can be given a name by using the CONSTRAINT keyword followed by the constraint name. Naming constraints is optional.

Specifying constraints - Key and Referential Integrity

Makes use of the PRIMARY KEY clause to specify if an attribute is a primary key (or an attribute of a multi-attribute primary key).

- To specify Dnumber as the primary key of DEPARTMENT, the following is used:
  Dnumber INT PRIMARY KEY,
- To specify alternate keys (candidate keys), the UNIQUE clause is used, for example:
  Dname VARCHAR(15) UNIQUE,
- Referential integrity is specified using the FOREIGN KEY clause. The default action when an integrity violation occurs is reject the update, this is known as the RESTRICT option. The schema designer may specify alternative actions by attaching a referential integrity action to the foreign key constraint. The options include: SET NULL, CASCADE, and SET DEFAULT. The option must be qualified with ON DELETE or ON UPDATE.

Specifying constraints - Attribute constraints and defaults

- If NULL is not permitted for a specific attribute a constraint NOT NULL may be specified. NOT NULL is always specified for attributes of primary keys.
- Default values are specified by appending the DEFAULT <value> clause to an attribute definition. Default values are used if an explicit value is not provided for that attribute.
  Dnumber INT NOT NULL CHECK (Dnumber $>$ 0 AND Dnumber $<$ 21);
  or create a domain and use it as the attribute type :
  CREATE DOMAIN D_NUM AS INTEGER CHECK (D_NUM $>$ 0 AND D_NUM $<$ 21);

Specifying constraints - Attribute constraints and defaults

```
CREATE TABLE EMPLOYEE
    ( ... ,
      Dno          INT            NOT NULL       DEFAULT 1,
    CONSTRAINT EMPPK
      PRIMARY KEY (Ssn),
    CONSTRAINT EMPSUPERFK
      FOREIGN KEY (Super_ssn) REFERENCES EMPLOYEE(Ssn)
                    ON DELETE SET NULL          ON UPDATE CASCADE,
    CONSTRAINT EMPDEPTFK
      FOREIGN KEY(Dno) REFERENCES DEPARTMENT(Dnumber)
                    ON DELETE SET DEFAULT       ON UPDATE CASCADE);
CREATE TABLE DEPARTMENT
    ( ... ,
      Mgr_ssn CHAR(9)          NOT NULL       DEFAULT '888665555',
      ... ,
    CONSTRAINT DEPTPK
      PRIMARY KEY(Dnumber),
    CONSTRAINT DEPTSK
      UNIQUE (Dname),
    CONSTRAINT DEPTMGRFK
      FOREIGN KEY (Mgr_ssn) REFERENCES EMPLOYEE(Ssn)
                    ON DELETE SET DEFAULT       ON UPDATE CASCADE);
CREATE TABLE DEPT_LOCATIONS
    ( ... ,
    PRIMARY KEY (Dnumber, Dlocation),
    FOREIGN KEY (Dnumber) REFERENCES DEPARTMENT(Dnumber)
                    ON DELETE CASCADE           ON UPDATE CASCADE);
```

Figure 6.2
Example illustrating how default attribute values and referential integrity triggered actions are specified in SQL.

Specifying constraints in SQL

CHECK clauses are placed at the end of the CREATE TABLE statement. These are row-based constraints. They are applied to each row individually and whenever a row is inserted or modified.
CHECK (Dept_create_date <= Mgr_start_date);
CREATE ASSERTION can be used for more general constraints.

Basic retrieval queries in SQL

Retrieval makes use of the SELECT statement. The basic structure of the SELECT statement is:

```
SELECT <attribute list>
FROM <table list>
WHERE <condition>;
```

A basic query in SQL can have up to 4 clauses:

```
SELECT <attribute list>
FROM <table list>
[ WHERE <condition> ]
[ ORDER BY <attribute list> ];
```

Basic retrieval queries in SQL

Substring matching is possible, for example:

```
SELECT Fname, Lname
FROM EMPLOYEE
WHERE Address LIKE '%Houston,TX%';
```

% means 0 or more characters and _ represents a character, for example:

```
SELECT Fname, Lname
FROM EMPLOYEE
WHERE Bdate LIKE '_ _ 7 _ _ _ _ _ _ _';
```

Basic retrieval queries in SQL

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

Q0 - Fig 6.3 (a):

```
SELECT Bdate, Address
FROM EMPLOYEE
WHERE Fname = 'John' AND Minit = 'B' AND Lname = 'Smith';
```

Basic retrieval queries in SQL

Q1 - Fig 6.3 (b):

```
SELECT Fname, Lname, Address
FROM EMPLOYEE, DEPARTMENT
WHERE Dname = 'Research' AND Dnumber = Dno;
```

If the attribute representing the department number was specified as Dnumber in all tables, it would be necessary to qualify which Dnumber was being referred to. Q1 would change as follows:
Q1:

```
SELECT Fname, Lname, Address
FROM EMPLOYEE, DEPARTMENT
WHERE Dname = 'Research' AND
                DEPARTMENT.Dnumber = EMPLOYEE.Dnumber;
```

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

EMPLOYEE

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

DEPARTMENT

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

DEPT_LOCATIONS

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

WORKS_ON

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

PROJECT

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

DEPENDENT

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

Basic retrieval queries in SQL

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

Q8 - Fig 6.3 (d):

```
SELECT E.Fname, E.Lname, S.Fname, S.Lname
FROM EMPLOYEE AS E, EMPLOYEE AS S
WHERE E.Super_ssn = S.Ssn;
```

Q9 - Fig 6.3 (e):

```
SELECT Ssn
FROM EMPLOYEE;
```

The attribute of the table given in Fig 6.3 (e) is not correct for this query.

Basic retrieval queries in SQL

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

Q10 - Fig 6.3 (f):

```
SELECT Ssn, Dname
FROM EMPLOYEE, DEPARTMENT;
```

Q1C - Fig 6.3(g)

```
SELECT *
FROM EMPLOYEE
WHERE Dno = 5;
```

The keyword DISTINCT can be used in the SELECT clause to produce rows with no duplicates. For example:

```
SELECT DISTINCT Salary
FROM EMPLOYEE;
```

INSERT, DELETE and UPDATE in SQL

- INSERT is used to add a Single tuple into a relation
- DELETE removes a tuple(s) from a relation
- UPDATE modifies attribute values in one or more selected tuples

INSERT, DELETE and UPDATE in SQL

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

```
INSERT INTO EMPLOYEE
VALUES ( 'Richard', 'K', 'Marini', '653298653',
'1962-12-30', '98 Oak Forest, Katy, TX', 'M',
37000, '653298653', 4 );

INSERT INTO EMPLOYEE (Fname, Lname, Dno, Ssn)
VALUES ('Richard', 'Marini', 4, '653298653');
```

INSERT, DELETE and UPDATE in SQL

Create a table and populate it from other tables:

```
CREATE TABLE WORKS_ON_INFO
( Emp_name VARCHAR(15),
Proj_name VARCHAR(15),
Hours_per_week DECIMAL(3,1) );


INSERT INTO WORKS_ON_INFO ( Emp_name, Proj_name,
                            Hours_per_week)
SELECT E.Lname, P.Pname, W.Hours
FROM PROJECT P, WORKS_ON W, EMPLOYEE E
WHERE P.Pnumber = W.Pno AND W.Essn = E.Ssn;
```

INSERT, DELETE and UPDATE in SQL

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

Zero, one or more tuples can be deleted by a single DELETE command

```
DELETE FROM EMPLOYEE
WHERE Lname = 'Brown';

DELETE FROM EMPLOYEE
WHERE Ssn = '123456789';

DELETE FROM EMPLOYEE
WHERE Dno=5;

DELETE FROM EMPLOYEE;;
```

INSERT, DELETE and UPDATE in SQL

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

UPDATE uses the SET clause to specify the attributes to be modified

```
UPDATE PROJECT
SET Plocation = 'Bellaire', Dnum = 5
WHERE Pnumber = 10;
```

INSERT, DELETE and UPDATE in SQL

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

UPDATE uses the SET clause to specify the attributes to be modified

```
UPDATE EMPLOYEE
SET Salary = Salary * 1.1
WHERE Dno = 5;
```

COS221
L10 - Advanced SQL
(Chapter 5 in Edition 6 and Chapter 7 in Edition 7)

Linda Marshall

17 March 2023

Comparisons involving NULL and three-valued logic

- Recall, NULL is used to represent missing values. A NULL value could mean value is unknown, unavailable or withheld, and not applicable.
- SQL has three-valued logic with values TRUE, FALSE and UNKNOWN. This means the boolean operators need to make provision for TRUE and FALSE in combination with UNKNOWN.

Table 7.1 Logical Connectives in Three-Valued Logic

(a)

AND	TRUE	FALSE	UNKNOWN
TRUE	TRUE	FALSE	UNKNOWN
FALSE	FALSE	FALSE	FALSE
UNKNOWN	UNKNOWN	FALSE	UNKNOWN

(b)

OR	TRUE	FALSE	UNKNOWN
TRUE	TRUE	TRUE	TRUE
FALSE	TRUE	FALSE	UNKNOWN
UNKNOWN	TRUE	UNKNOWN	UNKNOWN

(c)

NOT	
TRUE	FALSE
FALSE	TRUE
UNKNOWN	UNKNOWN

Comparisons involving NULL and three-valued logic

- In *select-project* queries, only tuples that result in a TRUE for the WHERE clause are selected. There is one exception and that is when considering outer joins.
- The keyword IS or IS NOT is used for comparisons of NULL values.

```
SELECT Fname, Lname
FROM EMPLOYEE
WHERE Super_ssn IS NULL;
```

- SQL considers each NULL value as unique and therefore comparison operators cannot be used. Tuples with the comparison of NULL are therefore not included in the results.

Nested Queries (sub-queries), Tuples, and Set/Multiset Comparisons

- Some queries may require results from one query to be used as input to another query. Nested queries are used in such cases.
- The nested query is usually added in the WHERE clause of the outer query . The nested query is executed first and the result is used by the outer query

```
SELECT DISTINCT Essn
FROM WORKS_ON
WHERE (Pno, Hours) IN
    ( SELECT Pno, Hours
      FROM WORKS_ON
      WHERE Essn = '123456789' );
```

- Generally a nested query results in a table and not a scalar value.

Nested Queries (sub-queries), Tuples, and Set/Multiset Comparisons

- Other than IN, other comparisons can be used to compare a single value, v, to a set or multiset, V.
  - $=$ ANY (or $=$ SOME) is TRUE if v is equal some value in V. This makes it equivalent to IN
  - ANY and SOME can be combined with other operators (other than $=$). The keyword ALL can be used with these operators. For example: $v >$ ALLV returns TRUE if the value v is greater than all the values in V.

```
SELECT Lname, Fname
FROM EMPLOYEE
WHERE Salary > ALL
    ( SELECT Salary
      FROM EMPLOYEE
      WHERE Dno = 5 );
```

Nested Queries (sub-queries), Tuples, and Set/Multiset Comparisons

- Ambiguity of attributes when using nested joins can be a problem. In general it is a good idea to create aliases for all tables in a query.

```
SELECT E.Fname, E.Lame
FROM EMPLOYEE AS E
WHERE E.Ssn IN
    ( SELECT D.Essn
      FROM DEPENDENT AS D
      WHERE E.Fname = D.Dependent_name AND
            E.Sex =D.Sex );
```

Other Nested Query Related Concepts

- **Correlated Nested Query** - When the WHERE clause of a nested query references an attribute of the outer query, the queries are correlated.
- **EXISTS and UNIQUE functions** are used in the WHERE clause as they result in a TRUE/FALSE result. (NOT) EXISTS tests in the nested query is empty or not. UNIQUE returns TRUE if there are no duplicate tuples, otherwise it returns FALSE.
- **Explicit Sets** are specified in the WHERE clause and the value being selected must be in the set.
- Within the SELECT clause, attributes can be **renamed**. This will result in a table with attributes with the new naming.

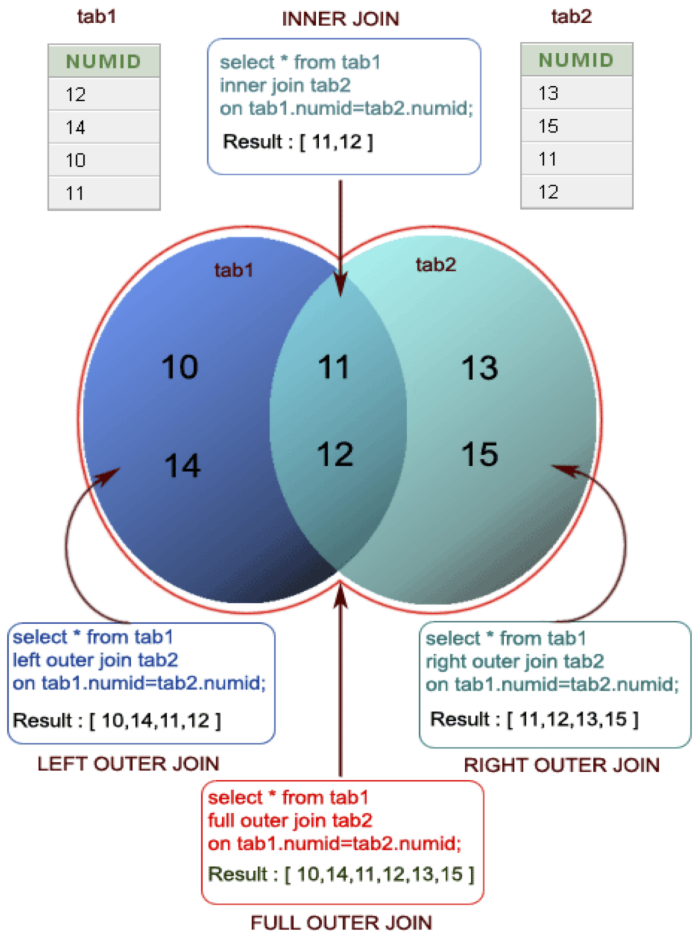
Other Nested Query Related Concepts

- **Joins** form a major part of SQL and where introduced so that joining is specified in the FROM clause.
  - A join is used to combine rows from two or more tables. Joins are left associative. That is A join B join C is completed as the result of A join B is joined to C.

```
SELECT Fname, Lname, Address
FROM (EMPLOYEE JOIN DEPARTMENT ON Dno = Dnumber)
WHERE Dname = 'Research';
```

Other Nested Query Related Concepts

- **Joins** - There are different types of joins: NATURAL JOIN (also referred to as an inner join) and various types of OUTER JOINs. With a NATURAL join there must be identical column names (attributes) in each table being joined.



Other Nested Query Related Concepts

- The attributes of the joined table are all the attributes of the first table followed by all the attributes of the second table.
- When a natural join is used, no join condition is specified. The condition is an implicit Equijoin.

```
SELECT Fname, Lname, Address
FROM (EMPLOYEE NATURAL JOIN
(DEPARTMENT AS DEPT (Dname, Dno, Mssn, Msdate)))
WHERE Dname = 'Research';
```

Other Nested Query Related Concepts

- Inner joins only pair the tuples that match the join condition.
- If all tuples in one relation are required then an outer join is required. There are therefore two types of outer join, where all tuples in the table to the left of the join condition and to the right of the join condition are included.

Other Nested Query Related Concepts

- **Aggregate functions** summerise information from multiple tuples. Examples of aggregate functions are COUNT, SUM, AVG, MIN, MAX...

```
SELECT SUM (Salary), MAX (Salary),
       MIN (Salary), AVG (Salary)
FROM EMPLOYEE;
```

This query returns a single row table with 4 columns. Renaming the columns will make the table more readable.

```
SELECT SUM (Salary) AS Total_Sal,
        MAX (Salary) AS Highest_Sal,
        MIN (Salary) AS Lowest_Sal,
        AVG (Salary) AS Average_Sal
FROM EMPLOYEE;
```

Other Nested Query Related Concepts

- **Grouping** enables an aggregate to be applied to a subgroup of tuples in a relation.

```
SELECT Pnumber, Pname, COUNT (*)
FROM PROJECT, WORKS_ON
WHERE Pnumber = Pno
GROUP BY Pnumber, Pname;
```

- If NULL exists in the table, then a separate group for all NULL values in the grouping attribute is created.
- Adding a HAVING clause, provides a condition on the grouping information.

Specifying Constraints as Assertions and Actions as Triggers

Constraints as Assertions

An assertion is a named constraint that may relate to the content of individual rows of a table, to the entire contents of a table, or to a state required to exist among a number of tables (entire schema or database).

```
CREATE ASSERTION <constraint name>
CHECK (<search condition>)
```

For example:

```
CREATE ASSERTION SALARY_CONSTRAINT
CHECK ( NOT EXISTS (
        SELECT  *
        FROM EMPLOYEE E, EMPLOYEE M, DEPARTMENT D
        WHERE  E.Salary>M.Salary AND E.Dno=D.Dnumber
        AND D.Mgr_ssn=M.Ssn ) );
```

Specifying Constraints as Assertions and Actions as Triggers

Actions as Triggers

A trigger is a set of actions that are run automatically when a specified change operation (SQL INSERT, UPDATE, or DELETE statement) is performed on a specified table.

Uses for triggers:

- Enforce business rules
- Validate input data
- Generate a unique value for a newly-inserted row in a different file.
- Write to other files for audit trail purposes
- Replicate data to different files to achieve data consistency

Specifying Constraints as Assertions and Actions as Triggers

Actions as Triggers

```
CREATE
[DEFINER = { user | CURRENT_USER }]
TRIGGER trigger_name {All triggers must have unique names w
ON tbl_name FOR EACH ROW
trigger_body {the statement to execute when the trigger act
trigger_time: { BEFORE | AFTER }
trigger_event: { INSERT | UPDATE | DELETE }
```

Specifying Constraints as Assertions and Actions as Triggers

Actions as Triggers For example to check whenever an employee's salary is greater than the salary of his or her direct supervisor in the COMPANY database

```
CREATE TRIGGER SALARY_VIOLATION
BEFORE INSERT OR UPDATE OF SALARY, SUPERVISOR_SSN ON EMPLOY
FOR EACH ROW
WHEN ( NEW.SALARY > ( SELECT SALARY FROM EMPLOYEE
WHERE SSN = NEW.SUPERVISOR_SSN ) )
INFORM_SUPERVISOR(NEW.Supervisor_ssn, NEW.Ssn );
```

Views (virtual tables)

A view is a single table that is derived from other tables and is a virtual table. It is usually defined for queries that are run frequently.
To create and remove a view:

```
CREATE VIEW Viewname
AS defining_query;
```

```
DROP view Viewname;
```

A view is up to date, modifying the values in tuples in the base table will update the values in the view. A view is not realised when it is created, but when a query is run on the view.

Views (virtual tables)

Examples of views:

```
CREATE VIEW WORKS_ON1
AS SELECT Fname, Lname, Pname, Hours
FROM EMPLOYEE, PROJECT, WORKS_ON
WHERE Ssn = Essn AND Pno = Pnumber;

CREATE VIEW DEPT_INFO(Dept_name, No_of_emps, Total_sal)
AS SELECT Dname, COUNT (*), SUM (Salary)
FROM DEPARTMENT, EMPLOYEE
WHERE Dnumber = Dno
GROUP BY Dname;
```

Schema change statements

The DROP command can be used on SCHEMA and TABLEs. The ALTER command can be used on TABLES.

```
DROP SCHEMA COMPANY CASCADE;

DROP TABLE DEPENDENT CASCADE;

ALTER TABLE COMPANY.EMPLOYEE
ADD COLUMN Job VARCHAR(12);

ALTER TABLE COMPANY.DEPARTMENT
ALTER COLUMN Mgr_ssn DROP DEFAULT;

ALTER TABLE COMPANY.DEPARTMENT
ALTER COLUMN Mgr_ssn SET DEFAULT ?333445555?;
```

Summary

Table 7.2 Summary of SQL Syntax

CREATE TABLE <table name> (<column name> <column type> [<attribute constraint>] { , <column name> <column type> [<attribute constraint>] } [<table constraint> { , <table constraint> }])
DROP TABLE <table name> ALTER TABLE <table name> ADD <column name> <column type>
SELECT [DISTINCT] <attribute list> FROM (<table name> { <alias> } | <joined table>) { , (<table name> { <alias> } | <joined table>) } [WHERE <condition>] [GROUP BY <grouping attributes> [HAVING <group selection condition>]] [ORDER BY <column name> [<order>] { , <column name> [<order>] }]
<attribute list> ::= (* | (<column name> | <function> (([DISTINCT] <column name> | *))) { , (<column name> | <function> (([DISTINCT] <column name> | *)) }))
<grouping attributes> ::= <column name> { , <column name> }
<order> ::= (ASC | DESC)
INSERT INTO <table name> [(<column name> { , <column name> })] (VALUES (<constant value> , { <constant value> }) { , (<constant value> { , <constant value> }) } | <select statement>)
DELETE FROM <table name> [WHERE <selection condition>]
UPDATE <table name> SET <column name> = <value expression> { , <column name> = <value expression> } [WHERE <selection condition>]
CREATE [UNIQUE] INDEX <index name> ON <table name> (<column name> [<order>] { , <column name> [<order>] }) [CLUSTER]
DROP INDEX <index name>
CREATE VIEW <view name> [(<column name> { , <column name> })] AS <select statement>
DROP VIEW <view name>

NOTE: The commands for creating and dropping indexes are not part of standard SQL.

COS221
L11 - Relational Algebra 1
(Chapter 6 in Edition 6 and Chapter 8 in Edition 7)

Linda Marshall

23 March 2023

Why Relational Algebra (RA)?

- Developed before the SQL language.
- Represents the basic set of operations for the relational model.
- Basic retrieval operations are expressed using relational algebra expressions which results in a new relation being created. A relational algebra expression is defined by a sequence of relational algebra operations.

Why Relational Algebra (RA)?

So why?

- provides a formal mathematical foundation for the relational model
- provides a mechanism to optimise queries which is why we use a relational DBMS (RDBMS)
- concepts are defined within SQL

Why Relational Algebra (RA)?

Relation algebra operations fall into two groups:

- set operations from mathematical set theory
  - Relations are views as a set of tuples.
  - Therefore set operations such as UNION, INTERSECTION, SET DIFFERENCE and CARTESIAN/CROSS PRODUCT can be applied
- relational database specific operations
  - Relational database specific operations include operations such as SELECT, PROJECT and JOIN.
  - SELECT and PROJECT are unary operations while JOIN is a binary operation.

Aggregate operations are defined for instances where set or specific relational algebra operations are not enough.

The SELECT operation

- The SELECT (σ) operation is used to choose a subset of tuples which satisfy a selection condition.
- Syntax of the SELECT operation is given by:

  $\sigma_{<\text{selection\_condition}>}(R)$, where R is a relation

- The resulting relation will have the same attributes as R.
- For example:
  $\sigma_{Dno=4}(EMPLOYEE)$
  $\sigma_{Salary>30000}(EMPLOYEE)$

The SELECT operation

The selection condition is a boolean expression and is made up of a number of clauses:
$<$ attribute_name $><$ comparison_op $><$ constant_value $>$
$<$ attribute_name $><$ comparison_op $><$ attribute_value $>$

where:

$<$ attribute_name $>$ is the name of an attribute in R
$<$ constant_name $>$ is a constant attribute from the attribute domain
$<$ comparison_op $>$ is normally one of $\{=, <, \leq, >\geq, \neq\}$. All operators can be applied to *ordered values*. If the values are unordered, such as chars or strings, then only the first and last operator given above can be applied.

The SELECT operation

Clauses can be connected by the Boolean operators AND, OR and NOT. They follow their normal interpretation:

- AND is true is both conditions are true, otherwise it is false
- OR is true of one of the conditions is true, otherwise it is false
- NOT is true if its condition is false, otherwise it is true

The SELECT operation

- The *selection_condition* is applied independently to each tuple (t) in R. By substituting each attribute (A_i) in the selection condition with its value in the tuple $t[A_i]$, a TRUE result will result in tuple t to be placed in the resultant relation.
- The *degree*, number of attributes, of the resultant relation is the same as the original relation R.
- The number of tuples in the resultant relation is less than or equal to the number of tuples in R.
- Therefore, for any condition c, $|\sigma_c(R)| \leq |R|$

The SELECT operation

- SELECT is *commutative*,
  $\sigma_{<cond1>}(\sigma_{<cond2>}(R)) = \sigma_{<cond2>}(\sigma_{<cond1>}(R))$
- SELECT *cascades*,
  $\sigma_{<cond1>}(\sigma_{<cond2>}(R)) = \sigma_{<cond1> AND <cond2>}(R)$

The SELECT operation

For example:

$\sigma_{Dno=4}(\sigma_{Salary>25000}(EMPLOYEE)) =$

$\sigma_{Dno=4 \text{ AND } Salary>25000}(EMPLOYEE))$

which translates to the SQL query:

```
SELECT *
FROM EMPLOYEE
WHERE Dno = 4 AND Salary > 25000;
```

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

The PROJECT operation

- Where the SELECT operation selected 'rows', the PROJECT operation selects 'columns'. That is, the PROJECT (π) operation partitions the relation vertically.
- The PROJECT operation is used to select certain attributes and discard the rest.
- PROJECT has the following syntax:
  $\pi_{<attribute\_list>}(R)$

The PROJECT operation

- The order of the attributes in the resultant relation are the same as given in the $< attribute\_list >$.
- The degree of the relation of the same as the number of attributes in the $< attribute\_list >$.
- If the $< attribute\_list >$ does not include any key attributes, the duplicate tuples are removed - referred to as *duplicate elimination*
- Duplicate elimination is as a result of a relation being represented as a set of tuples and a *mathematical set not allowing duplicates*.

The PROJECT operation

For example:

$\pi_{Sex,Salary}(EMPLOYEE)$

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

(c)

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

The PROJECT operation

- If a relation had been defined as a *multiset/bag*, then duplicates would have been allowed.
- RA considers a relation as a *set*, whereas SQL allows for the distinction between a set and a multiset.
  - As a set:

    ```
    SELECT DISTINCT Sex, Salary
    FROM EMPLOYEE;
    ```

  - As a multiset or bag:

    ```
    SELECT Sex, Salary
    FROM EMPLOYEE;
    ```

Sequences of operations and the RENAME operation

- Operations may be applied as a nested sequence or one at a time.
- If operations are applied one at a time, the resultant relation must be stored in an intermediate relation.
- This is written in relational algebra as follows:
  $DEPS\_EMPS \leftarrow \sigma_{Dno=5}(EMPLOYEE)$
  $RESULT \leftarrow \pi_{Fname,Lname,Salary}(DEPS\_EMPS)$
  This is equivalent to:
  $\pi_{Fname,Lname,Salary}(\sigma_{Dno=5}(EMPLOYEE))$

Sequences of operations and the RENAME operation

- Sometimes it is necessary to *rename* the attributes of the relation. Renaming is written as follows:
  $TEMP \leftarrow \sigma_{Dno=5}(EMPLOYEE)$
  $R(First\_name, Last\_name, Salary) \leftarrow$
  $\pi_{Fname,Lname,Salary}\, TEMP$

(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

Sequences of operations and the RENAME operation

A formal RENAME (ρ) operation definition for a relation of degree n is given by:

- for renaming the relation and attributes
  $\rho_{S(B_1,B_2,\ldots,B_n)}(R)$
- for renaming the relation only
  $\rho_S(R)$
- for renaming the attributes only
  $\rho_{B_1,B_2,\ldots,B_n}(R)$

In SQL

```
SELECT E.Fname AS First_name
       E.Lname AS Last_name
       E.Salary AS Salary
FROM EMPLOYEE AS E
WHERE DNO = 5;
```

Relational algebra operations from set theory

- UNION, INTERSECTION and MINUS are standard operations from set theory.
- All these operations are binary operations and therefore work on two relations (sets of tuples).
- These operations are applied to sets that are union/type compatible. That is, for two relations $R(A_1, A_2, ..., A_n)$ and $S(B_1, B_2, ..., B_n)$, the degrees of the relations must be the same (both n) and the $dom(A_i) = dom(B_i)$ for $1 \leq i \leq n$.

Relational algebra operations from set theory

For two union compatible relations R and S, the operations have the following meaning:

- UNION ($\cup$) - the union of R and S is the relation of all tuples in (R or S) or (R and S). Duplicates are eliminated.
- INTERSECTION ($\cap$) - the intersection of R and S is the relation of all tuples in both R and S.
- SET DIFFERENCE (-) (or MINUS) - the difference of R and S ($R - S$) is all tuples in R but not in S.

Relational algebra operations from set theory

Figure 6.4

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

(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

COS221
L11 - Relational Algebra 2
(Chapter 6 in Edition 6 and Chapter 8 in Edition 7)

Linda Marshall

23 March 2023

Recap

Set operations — **Relational Algebra** — Relational database specific operations

UNION
INTERSECTION
SET DIFFERENCE

CARTESIAN/CROSS PRODUCT

SELECT
PROJECT

JOIN

Generalised projection
Aggregate
Recursive closure

Recap

Question: Write intersection ($\cap$) in terms of the other set operations

Figure 6.4
The set operations UNION, INTERSECTION, and MINUS. (a) Two union-compatible relations. (b) STUDENT $\cup$ INSTRUCTOR. (c) STUDENT $\cap$ INSTRUCTOR. (d) STUDENT – INSTRUCTOR. (e) INSTRUCTOR – STUDENT.

(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

Cartesian/Cross product

- The CARTESIAN PRODUCT ($\times$) operation is a binary operation where the two relations do not have to be union compatible.
- The operation produces a new relation by combining every tuple in the first relation with every tuple in the second relation.
- The cartesian product of relations R (of degree n) and S (of degree m) given by: $R(A_1, A_2, ..., A_n) \times R(B_1, B_2, ..., B_m)$ results in the relation Q (with degree $m+n$) : $Q(A_1, A_2, ..., A_n, B_1, B_2, ..., B_m)$

Cartesian/Cross product

- The order of the attributes in Q is the concatenation of the attributes in order of R followed by the attributes in the same order as in S.
- The number of tuples in Q will be $n_R * n_S$, where $n_R = |R|$ and $n_S = |S|$
- Applying the CARTESIAN PRODUCT operation by itself is usually meaningless. A combination of the CARTESIAN PRODUCT with PROJECTs and SELECTs provides a powerful sequence of operations.
- In SQL, the Cartesian product is realised using the CROSS JOIN.

Cartesian/Cross product

Figure 8.5
The CARTESIAN PRODUCT (CROSS PRODUCT) operation.

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

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

$$\text{FEMALE\_EMPS} \leftarrow \sigma_{\text{Sex='F'}}(\text{EMPLOYEE})$$
$$\text{EMPNAMES} \leftarrow \pi_{\text{Fname, Lname, Ssn}}(\text{FEMALE\_EMPS})$$
$$\text{EMP\_DEPENDENTS} \leftarrow \text{EMPNAMES} \times \text{DEPENDENT}$$
$$\text{ACTUAL\_DEPENDENTS} \leftarrow \sigma_{\text{Ssn=Essn}}(\text{EMP\_DEPENDENTS})$$
$$\text{RESULT} \leftarrow \pi_{\text{Fname, Lname, Dependent\_name}}(\text{ACTUAL\_DEPENDENTS})$$

The JOIN Operation

- The JOIN operation ($\bowtie$) is used to combine related tuples from two relations, resulting in "longer" tuples.
- By joining, relationships among relations can be processed

Figure 8.6
Result of the JOIN operation DEPT_MGR ← DEPARTMENT $\bowtie_{Mgr\_ssn=Ssn}$ EMPLOYEE.

DEPT_MGR

Dname	Dnumber	Mgr_ssn	...	Fname	Minit	Lname	Ssn	...
Research	5	333445555	...	Franklin	T	Wong	333445555	...
Administration	4	987654321	...	Jennifer	S	Wallace	987654321	...
Headquarters	1	888665555	...	James	E	Borg	888665555	...

- A JOIN can be written as a CARTESIAN PRODUCT followed by a SELECT.
  $RESULT \leftarrow DEPARTMENT \times EMPLOYEE$
  $DPT\_MGR \leftarrow \sigma_{Mgr\_ssn=Ssn}(RESULT)$

The JOIN Operation

- The general form of a JOIN $Q \leftarrow R \bowtie_{<join\_condition>} S$ where $R(A_1, A_2, ..., A_n)$ and $R(B_1, B_2, ..., B_m)$ results in $Q(A_1, A_2, ..., A_n, B_1, B_2, ..., B_m)$
- A join condition is of the form: $< condition > AND < condition > AND...AND < condition >$
- Each $< condition >$ is of the form: $A_i\ \theta\ B_j$ where A_i is in R and B_j is in S is one of the comparison operators
- This is referred as a THETA join.
- Specialisations of the THETA join are EQUIJOIN and NATURAL JOIN.

The JOIN Operation

- EQUIJOIN is when the only comparison operator is $=$.
  One or more pairs of attributes have identical values in a EQUIJOIN. One of these attributes in each of the pairs are superfluous in the results and can be dropped. To make provision for this, a special operator, the NATURAL JOIN is defined.
- A NATURAL JOIN is denoted by $*$.
  A natural join requires that the pair of attributes from each relation participating in the join have the same name in both relations. Renaming of attributes is therefore necessary if they are not the same.
  $DEPT \leftarrow \rho_{(Dname,Dnum,Mgr\_ssn,Mgr\_start\_date)}(DEPARTMENT)$
  $PROJ\_DEPT \leftarrow PROJECT * DEPT$
  $Dnum$ is a join attribute.

The JOIN Operation

- If no combination of tuples satisfies the join condition the result of the join is an empty relation.
- If there is no join condition the join is a Cartesian product.
- A join combining two relations to form a single relation is also called an inner join. That is a join formed by combining a CARTESIAN PRODUCT followed by a SELECT operation.
- In contrast outer joins were developed when all tuples in either R or S or both relations are to be kept regardless of whether or not they have matching tuples in the other relation. Two types of outer joins exist, a LEFT OUTER JOIN ($⟕$) and a RIGHT OUTER JOIN ($⟖$).

The JOIN Operation

- Left outer join example:
  $TEMP \leftarrow EMPLOYEE \ ⟕_{Ssn=Mgs\_ssn} \ DEPARTMENT$
  $RESULT \leftarrow \pi_{Fname,Minit,Lname,Dname}(TEMP)$
- Every tuple in EMPLOYEE is kept. If no matching tuple is found in DEPARTMENT, the attributes are padded with NULL values.

Figure 8.12
The result of a LEFT OUTER JOIN operation.

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

The Complete Set of Relational Algebra Operations

The set of relational operations $\{\sigma, \pi, \cup, \rho, -, \times\}$ is a complete set. That is, any of the other original relational algebra operations can be expressed as a sequence of operations from this set.

- The INTERSECTION operation can be expressed using UNION and SET DIFFERENCE.
  $R \cap S \equiv (R \cup S) - ((R - S) \cup (S - R))$
- The JOIN operation is a CARTESIAN PRODUCT followed by a SELECT.
  $R \bowtie_{<condition>} S \equiv \sigma_{<condition>}(R \times S)$
- A NATURAL JOIN is a RENAME of attributes followed by a CARTESIAN PRODUCT, a SELECT and a PROJECT.
  $(R1 \leftarrow \rho_{(A_1, A_2, \ldots, A_n)}(R)$
  $S1 \leftarrow \rho_{(B_1, B_2, \ldots, B_n)}(S)$
  $TEMP \leftarrow S1 \times S2$
  $RESULT \leftarrow \pi_{(C_1, C_2, \ldots, C_k)}(\sigma_{<condition>}(TEMP))$
  $) \equiv R * S$

The Complete Set of Relational Algebra Operations

- The DIVISION ($\div$) operation is a combination of PROJECT, CARTESIAN PRODUCT and SET DIFFERENCE. The DIVISION operation is applied to two relations
  (
  $T1 \leftarrow \pi_Y(R)$
  $T2 \leftarrow \pi_Y((S \times T1) - R)$
  $T \leftarrow T1 - T2$
  $) \equiv (R \div S)$
  Where the attributes of R are a subset of the attributes of S. The result of the division is a relation T that for every tuple t in T, the values of t must appear in R in combination with every tuple in S.

The Complete Set of Relational Algebra Operations

- For example: *Retrieve the names of all employees who work on all the projects that John Smith works on.*

$$SMITH \leftarrow \sigma_{Fname='John' \text{ AND } Lname='Smith'}(EMPLOYEE)$$

$$SMITH\_PNOS \leftarrow \pi_{Pno}(WORKS\_ON \bowtie_{Essn=Ssn} SMITH)$$

$$SSN\_PNOS \leftarrow \pi_{Essn, Pno}(WORKS\_ON)$$

$$SSNS(Ssn) \leftarrow SSN\_PNOS \div SMITH\_PNOS$$

$$RESULT \leftarrow \pi_{Fname, Lname}(SSNS * EMPLOYEE)$$

Figure 6.8
The DIVISION operation. (a) Dividing SSN_PNOS by SMITH_PNOS. (b) $T \leftarrow R \div S$.

(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

Additional Operations

- Generalised projection adds functions on attributes to be included in the projection list. The general form is given by: $\pi_{F_1,F_2,\ldots,F_n}(R)$ where F_i is a function over the attributes. These functions may include arithmetic operations and constants

  For example:

  Assume EMPLOYEE (Ssn, Salary, Deduction, Years_service)
  Require the following:
  - NetSalary = Salary - Deduction
  - Bonus = 2000 * Years_service; and
  - Tax = 0.25 * Salary

REPORT $\leftarrow \rho_{(Ssn, Net\_salary, Bonus, Tax)}$ (

$\pi_{Ssn, Salary - Deduction, 2000 * Years\_service, 0.25 * Salary}$ (EMPLOYEE))

Additional Operations

- Aggregate functions ($\mathscr{F}$ - Script F) on collections, for example average or total salary. Common functions included are: SUM, AVERAGE, MAXIMUM, MINIMUM, COUNT. The general form of the function is given by:
  $<grouping\_attributes>\mathscr{F}<function\_list>R$
  Where $< grouping\_attributes >$ is the list of attributes in R to which the aggregate is to be applied, $< function\_list >$ the list of pairs ($< function >< attribute >$) specifying the attributes and functions to be calculated with respect to the grouping attributes. The resulting relation will include the grouping attributes and an attribute per pair in the function list.

Additional Operations

- Aggregate example: *Retrieve each department number, the number of employees in each department and their average salary*

(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 8.10
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. $\Im \text{ COUNT Ssn, AVERAGE Salary}(\text{EMPLOYEE})$.

Additional Operations

- Recursive closure operations which are applied between tuples of the same type, for example: the supervisor-employee relationship. In many cases, this type of relationship is not just on one level. An employee may indirectly manage another employee because of a management hierarchy. To identify multiple levels of relationship, a transitive closure is computed. The SQL3 standard includes recursive closure.

Find all employees directly supervised by "James Borg".

$$\text{BORG\_SSN} \leftarrow \pi_{Ssn}(\sigma_{Fname='James' \text{ AND } Lname='Borg'}(\text{EMPLOYEE}))$$

$$\text{SUPERVISION}(Ssn1, Ssn2) \leftarrow \pi_{Ssn, Super\_ssn}(\text{EMPLOYEE})$$

$$\text{RESULT1}(Ssn) \leftarrow \pi_{Ssn1}(\text{SUPERVISION} \bowtie_{Ssn2=Ssn} \text{BORG\_SSN})$$

Now, find all employees supervised by an employee who is supervised by "Borg".

$$\text{RESULT2}(Ssn) \leftarrow \pi_{Ssn1}(\text{SUPERVISION} \bowtie_{Ssn2=Ssn} \text{RESULT1})$$

$$\text{RESULT} \leftarrow \text{RESULT2} \cup \text{RESULT1}$$

Examples

Using the COMPANY database:

- Query 1: Retrieve the name and address of all employees who work for the 'Research' department.
- Query 2: For every project located in 'Stafford', list the project number, the controlling department number, and the departments manager's last name, address and birth date.
- Query 3: Find the names of employees who work on all the projects controlled by department number 5.
- Query 4: Make a list of project numbers for projects that involve an employee who's last name is 'Smith', either as a worker or as a manager of the department that controls the project.
- Query 5: List the names of all employees with two or more dependents.
- Query 6: Retrieve the names of employees who have no dependents.
- Query 7: List the names of managers who have at least one dependent.

COS221
L12 - Relational Calculus
(Chapter 6 in Edition 6 and Chapter 8 in Edition 7)

Linda Marshall

24 March 2023

Overview

- Relational Calculus is another formal query language for the relational model.
- Two variations exist
  - tuple relational calculus, and
  - domain relational calculus
- In both variations, a declarative expression is written to specify a retrieval request.
- No description of how, or in what order to evaluate the query is given. Just what is be retrieved is given. Relational calculus is therefore a nonprocedural language. That is, unlike with relational algebra where steps were given, no steps are given here.

Overview

- Queries written in relational algebra can be written in relational calculus and vice versa. This means the expressive power of the two languages is the same, leading to the definition of relational completeness.
- A relational query language L is relationally complete if any query expressed in L can be expressed in relational calculus.
- Relational calculus has a basis in mathematical logic and SQL has some of its foundations in tuple relational calculus

Tuple Relational Calculus

- Tuple relational calculus specifies a number of *tuple variables*. A tuple variable, ranges over a database relation - the *range relation*. That is, it gets its value from any individual tuple in the relation.
- A query has the general form:
  $\{t|COND(t)\}$
  where t is a tuple variable and $COND(t)$ is a boolean expression involving t. The result of the query is all the tuples that satisfy the condition, that is the selected combinations satisfy the query. A set of attributes can be retrieved and are referred to as the *requested attributes*.

Tuple Relational Calculus

- For example:
  $\{t|EMPLOYEE(t)$ AND $t.Salary > 50000\}$
  where the range relation of the query is $EMPLOYEE(t)$ and the condition to be satisfied is $t.Salary > 50000$
- This query retrieves **all** the attributes of each selected EMPLOYEE tuple t. The following query selects specific attributes:
  $\{.t.Fname, t.Lname|EMPLOYEE(t)$ AND $t.Salary > 50000\}$

Tuple Relational Calculus

- The general expression for tuple relational calculus is therefore given A condition or formula is made up of atoms which can be one of the following:
  $\{t_1.A_j, t_2.A_k, ..., t_n.A_m | COND(t_1, t_2, ..., t_n, t_{n+1}, t_{n+2}, ..., t_{n+m})\}$
  where A_i is an attribute on which t_i , the tuple variable, ranges and $COND$ is a condition/formula.
- A condition or formula is made up of atoms which can be one of the following:
  - $R(t)$
  - $(t_i.A$ op $t_j.B)$ where op is a comparison operator, t_i and t_j are tuple variables and A and B are attributes on which the tuple variable respectively range.
  - $(t_i.A$ op $c)$ or $(c$ op $t_j.B)$ where c is a constant.

Tuple Relational Calculus

- Each atom evaluates to a truth value.
- A formula is made up of one or more atoms connected with logical operator AND, OR and. NOT.
- The following rules are defined recursively:
  - Rule 1 - Every atom is a formula
  - Rule 2 - For two formulae, F_1 and F_2, (F_1 AND F_2), (F_1 OR F_2), (NOT F_1) and (NOT F_2) are also formulas and adhere to the standard boolean truth values.

Tuple Relational Calculus

- Two additional special quantifier symbols, the universal quantifier ($\forall$) and the existential quantifier ($\exists$) are defined.
- A tuple variable, t, is bound if it appears in an ($\exists t$) or ($\forall t$), otherwise it is *free*. The following rules determine whether t is free or bound:
  - tuple variables that are atoms are free
  - by adding logical connectors (AND, OR and NOT) the status (free or bound) or the tuple variable participating in the connection dictates the status of the tuple variable in the connection.
  - All free occurrences of t in F are bound in F' for $F' = (\exists)(F)$ or $F' = (\forall t)(F)$

Tuple Relational Calculus

- For example:
  $F_1 : d.Dname = \text{'Research'}$
  $F_2 : (\exists t)(d.Number = t.Dno)$
  $F_3 : (\forall d)(d.Mgr\_ssn =' 333445555'$
  from F_1 and F_2, d is free, in F_2, t is bound to $(\exists)$, and in F_3, d is bound to $(\forall)$
- The following rules can now be added:
  - Rule 3 - If F is a formula, then so is $F' = (\exists t)(F)$. F' is true if F evaluates to true for at least one tuple assigned to free occurrences of t in F.
  - Rule 4 - If F is a formula, then so is $F' = (\forall t)(F)$. F' is true if F evaluates to true for every tuple assigned to free occurrences of t in F.

Transforming the Universal and Existential Quantifiers

- The following equivalences exist
  - $(\forall x)(P(x)) \equiv NOT(\exists x)(NOT(P(x))$
  - $(\exists x)(P(x)) \equiv NOT(\forall x)(NOT(P(x))$
- The following implications are true
  - $(\forall x)(P(x)) \implies (\exists x)(P(x))$
  - $NOT(\exists x)(P(x)) \implies NOT(\forall x)(P(x))$

Safe and Unsafe Expressions

- Expressions in relational calculus must be safe. That is, it must yield a finite number of tuples in its result.
- The following example is unsafe, it yields all tuples in the universe that are not in EMPLOYEE.
  $\{t|NOT(EMPLOYEE(t))\}$

Domain Relational Calculus

- Domain calculus was proposed after the development of QBE (Query-by-Example).
- Domain relational calculus differs from tuple relational calculus in that instead of variables ranging over tuples, variables range over single values from domains of attributes.
- To form a relation of degree n from a query, n domain variables are required - one per attribute.
- An expression in domain relational calculus takes the form: $\{x_1, x_2, ..., x_n | COND(x_1, x_2, ..., x_n, x_{n+1}, x_{n+2}, ..., x_{n+m}\}$
- The properties of a condition or formula are similar to those of Tuple Relational Calculus.
- Note, when referring to relations, the commas can be dropped. That is:
  $\{x_1, x_2, ..., x_n | R(x_1, x_2, ..., x_n)$ AND ... $\} \equiv$
  $\{x_1, x_2, ..., x_n | R(x_1 x_2 ... x_n)$ AND ... $\}$

Domain Relational Calculus

- An example: *List the birth date and address of employee whose name is 'John B Smith'*
  $\{u, v | (\exists q)(\exists r)(\exists s)(\exists t)(\exists w)(\exists x)(\exists y)(\exists z)(EMPLOYEE(qrstuvwxyz)$
  $q =$ 'John' AND $r =$ 'B' AND $s =$ 'Smith')$\}$
  - 10 variables are required for the EMPLOYEE relation. Of these 10, only u and v are free.
  - A shorthand notation for this query is:
    $\{u, v | (EMPLOYEE('\text{John}','\text{B}','\text{Smith}', t, u, v, w, x, y, z)\}$

Examples

Using the COMPANY database to write the following queries in RA, RC (tuple and domain)

- Query 1: Retrieve the name and address of all employees who work for the 'Research' department.
- Query 2: For every project located in 'Stafford', list the project number, the controlling department number, and the departments manager's last name, address and birth date.
- Query 3: Find the names of employees who work on all the projects controlled by department number 5.
- Query 4: Make a list of project numbers for projects that involve an employee who's last name is 'Smith', either as a worker or as a manager of the department that controls the project.
- Query 5: List the names of all employees with two or more dependents.
- Query 6: Retrieve the names of employees who have no dependents.
- Query 7: List the names of managers who have at least one

Examples

Query 1: Retrieve the name and address of all employees who work for the 'Research' department.

Figure 3.6
One possible database state for the COMPANY relational database schema.

EMPLOYEE

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

DEPARTMENT

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

DEPT_LOCATIONS

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

WORKS_ON

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

PROJECT

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

DEPENDENT

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

Examples

Query 2: For every project located in ‘Stafford’, list the project number, the controlling department number, and the departments manager’s last name, address and birth date.

Figure 3.6
One possible database state for the COMPANY relational database schema.

EMPLOYEE

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

DEPARTMENT

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

DEPT_LOCATIONS

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

WORKS_ON

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

PROJECT

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

DEPENDENT

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

Examples

Query 7: List the names of managers who have at least one dependent.

Figure 3.6
One possible database state for the COMPANY relational database schema.

EMPLOYEE

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

DEPARTMENT

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

DEPT_LOCATIONS

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

WORKS_ON

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

PROJECT

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

DEPENDENT

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

COS221
L14 - (E)ER to Relational Mapping
(Chapter 9 in Editions 6 and 7)

Linda Marshall

13 March 2023

(E)ER-to-Relational Mapping Algorithm

Step 1: Mapping of regular (strong) entity types

Step 2: Mapping of weak entity types

Step 3: Mapping of binary 1:1 relationships

Step 4: Mapping of binary 1:N relationships

Step 5: Mapping of binary M:N relationships

Step 6: Mapping of multivalued attributes

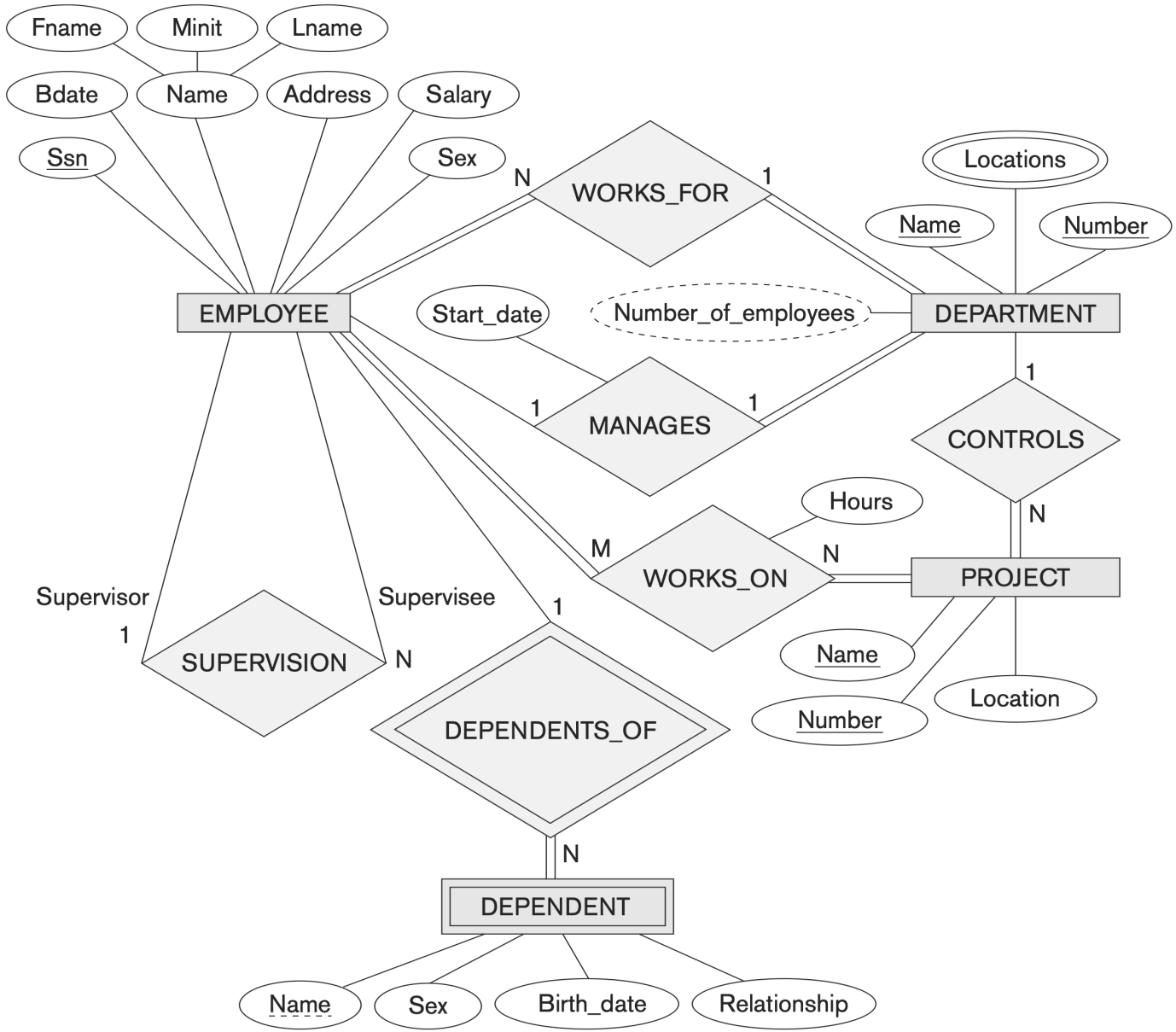
Step 7: Mapping of N-ary relationships

Step 8: Mapping specialisation and generalisation

Step 9: Mapping unions

The example

Figure 9.1
The ER conceptual schema diagram for the COMPANY database.

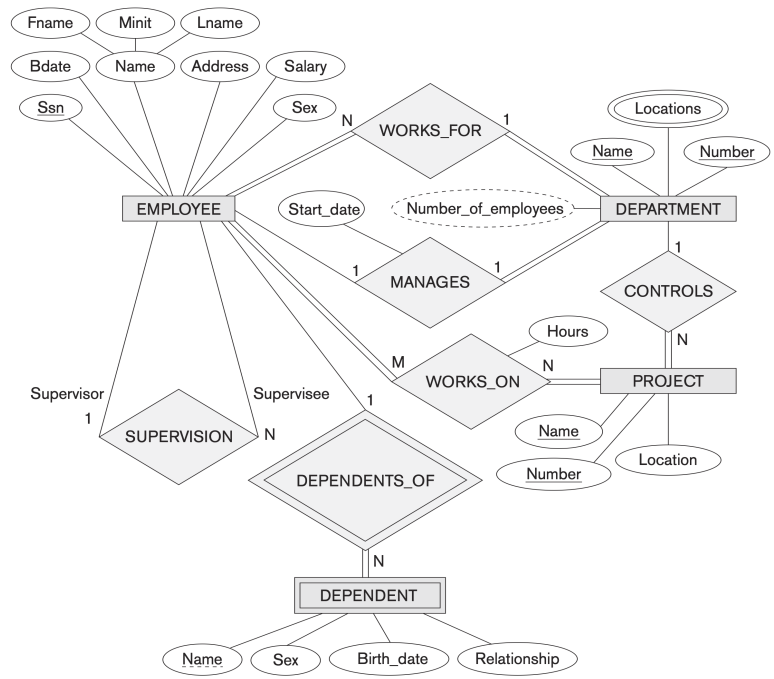


Step 1: Mapping of regular (strong) entity types

For each regular entity type, E, in the ER schema

- Create a relation R
- Add simple attributes of E to R
- Include the simple component attributes of composite attributes
- Choose one of the key attributes of E for R. If the attribute is a composite, then use all the simple attributes of the composite for the primary key of R.

Figure 9.1
The ER conceptual schema diagram for the COMPANY database.



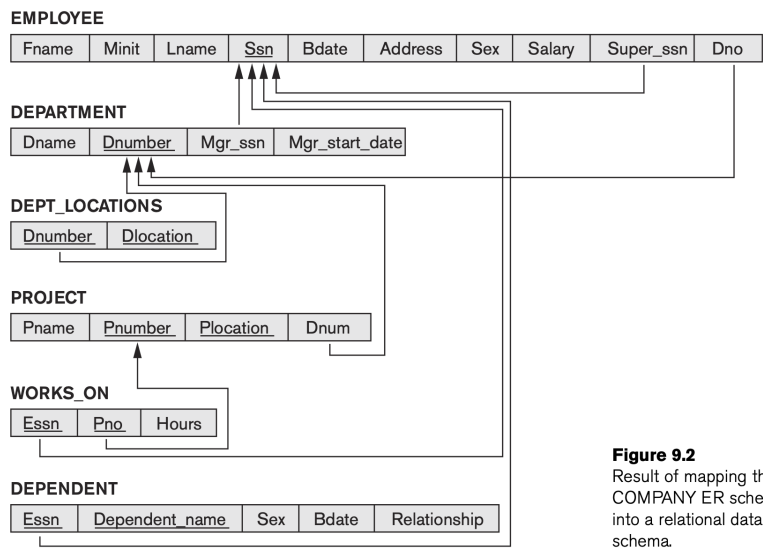


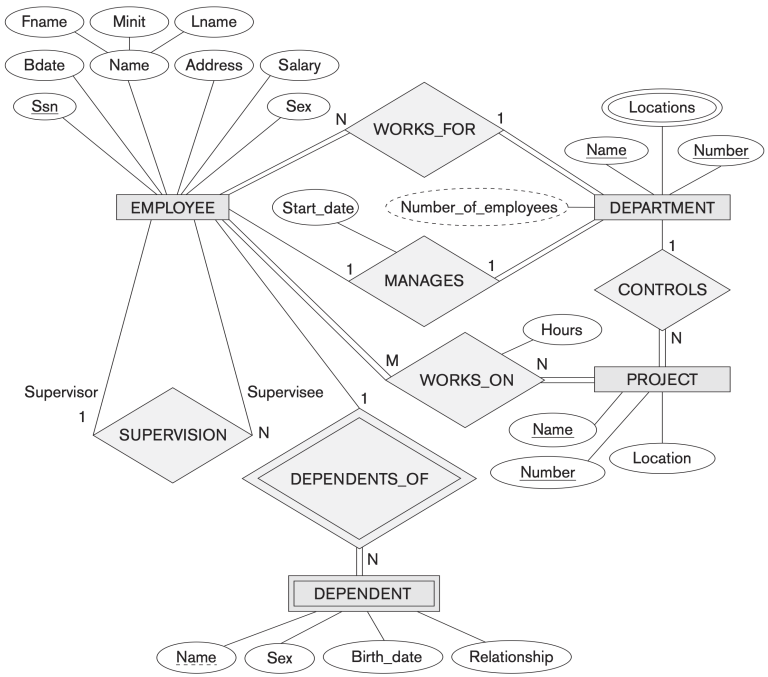
Figure 9.2
Result of mapping the COMPANY ER schema into a relational database schema.

Step 2: Mapping of weak entity types

For each weak entity type, W, in the ER schema with owner entity E

- Create a relation R
- Add the simple attributes of W to R
- Include as foreign key attributes of R, the primary key of the relations that correspond to the owner entity type
- The primary key of R is a combination of the primary key(s) of the owner(s) and the partial key of W, if any
- If there is a weak entity type E_1, whose owner is also a weak entity type, E_2, then E_1 should be mapped before E_2 to determine the primary keys first.

Figure 9.1
The ER conceptual schema diagram for the COMPANY database.



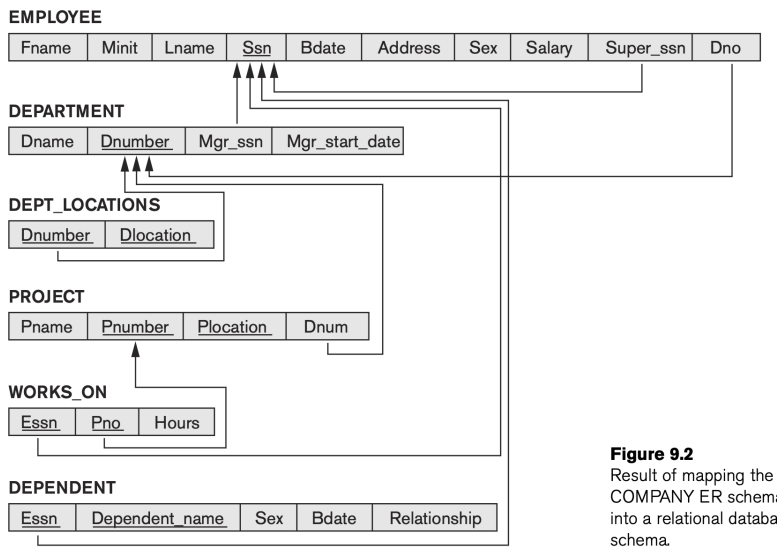


Figure 9.2
Result of mapping the COMPANY ER schema into a relational database schema.

Step 3: Mapping of binary 1:1 relationships

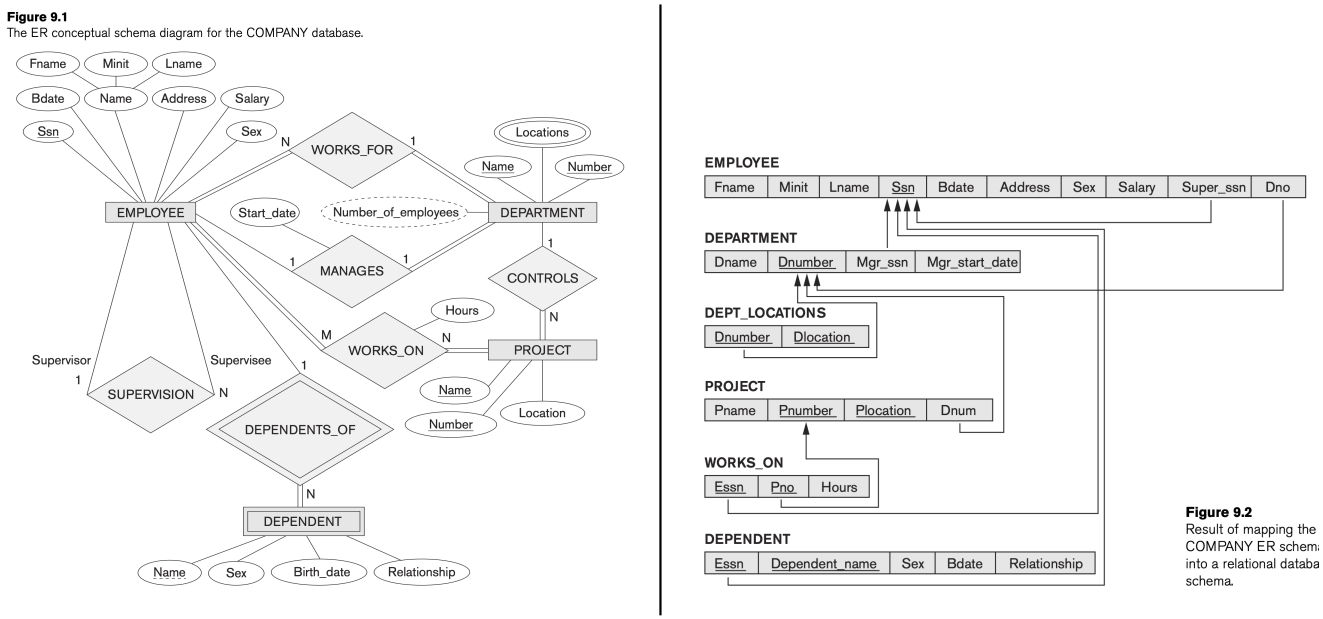
For each binary 1:1 relationship type R, identify the relations S and T that correspond to the entity types participating in R

1. *Foreign key approach* - most useful and followed unless special conditions exist
2. *Merged relation approach* - Works when both relations are total and can be merged. That is, when both relations have the same number of tuples at all times.
3. *Cross-reference or relationship relation approach* - Sets up a third relation, U, for the purpose of cross-referencing the primary keys of S and T. The relation U becomes a look-up table because it relates one tuple from S with one tuple from T.

Step 3: Mapping of binary 1:1 relationships (Approach 1)

For each binary 1:1 relationship type R, identify the relations S and T that correspond to the entity types participating in R, Choose one of the relations, say S - preferably the one with the total participation in R

- include as foreign key in S, the primary key of T
- include all the simple attributes (or simple components of the composite attributes) of the 1:1 relationship type R in S



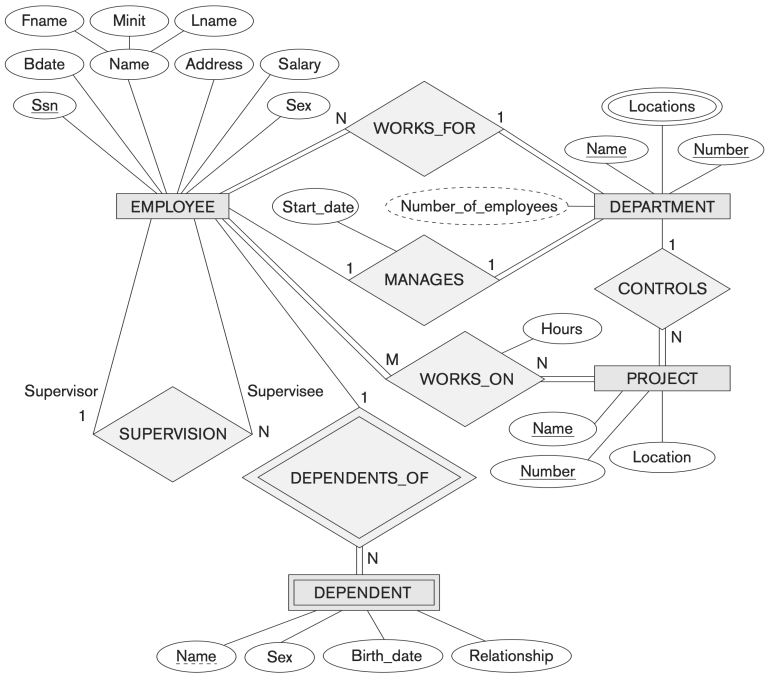
Choose DEPARTMENT as S because total in the MANAGES relation (every department has a manager). If EMPLOYEE was chosen as S, `Dept_managed` would have been included in EMPLOYEE. Where an employee is not a manager, a NULL value would need to have been included.

Step 4: Mapping of binary 1:N relationships

For each binary 1:N relationship type R, identify the relation S on the N side

- include in S the primary key of relation T (the other entity on the relationship).
- include any simple attributes (or simple components of composite attributes) of the 1:N relationship type as attributes of S.

Figure 9.1
The ER conceptual schema diagram for the COMPANY database.



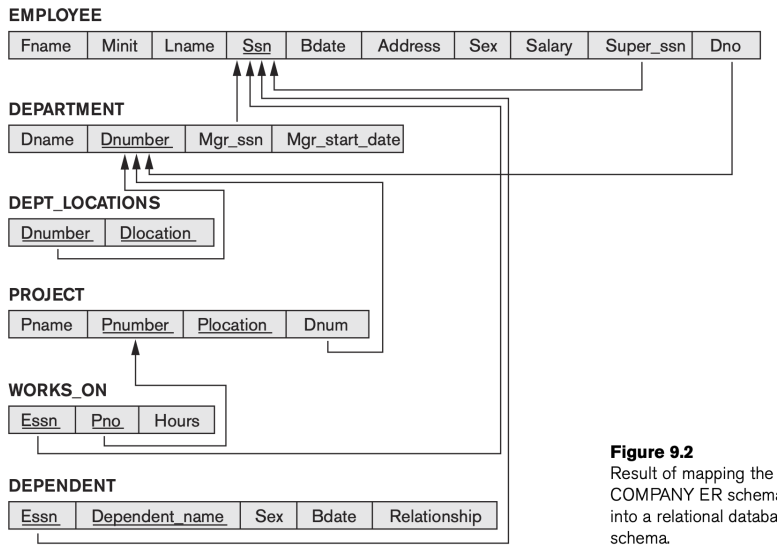


Figure 9.2
Result of mapping the COMPANY ER schema into a relational database schema.

Step 4: Mapping of binary 1:N relationships (Alternative approach)

An alternative approach is to use the relationship relation (cross-reference) approach, discussed as the third option of the 1:1 relationships. In this approach

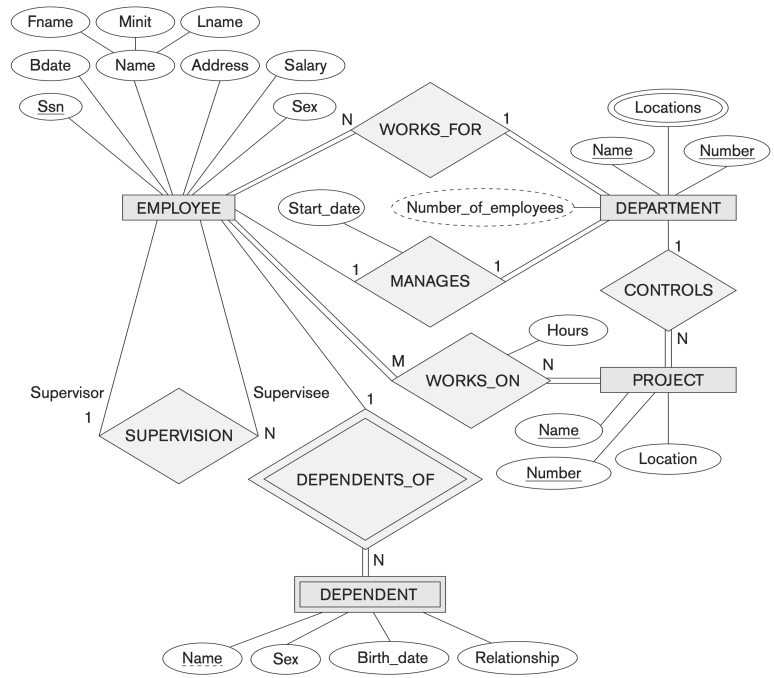
- Create a separate relation with attributes being the primary keys of the two relations involved in the relationship which will act as foreign keys for the respective relations.
- The primary key in this relation is the same as that of the first relation.
- This approach can lead to NULL values in the foreign keys.

Step 5: Mapping of binary M:N relationships

For each binary M:N relationship type R,

- create a new relation S to represent R.
- include as foreign key attributes in S, the primary keys of the relations that represent the participating entity types
- make the combination of these primary keys the primary key
- include the simple attributes (or the composite as simple components of the relationship

Figure 9.1
The ER conceptual schema diagram for the COMPANY database.



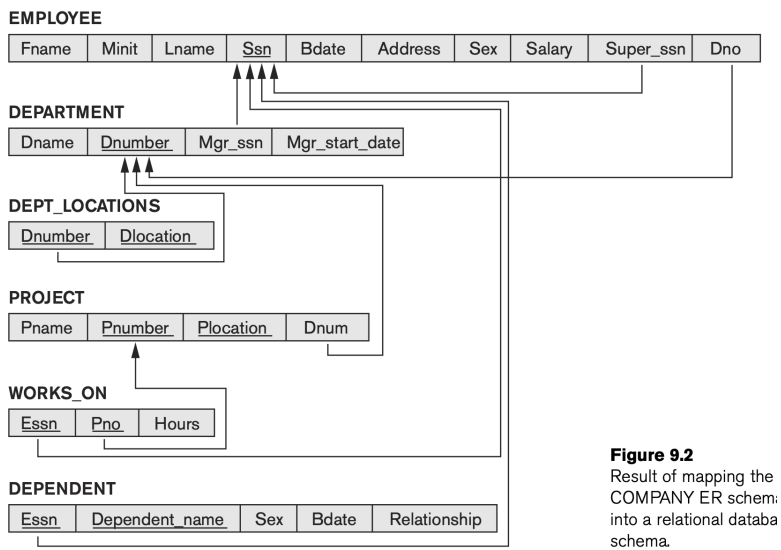


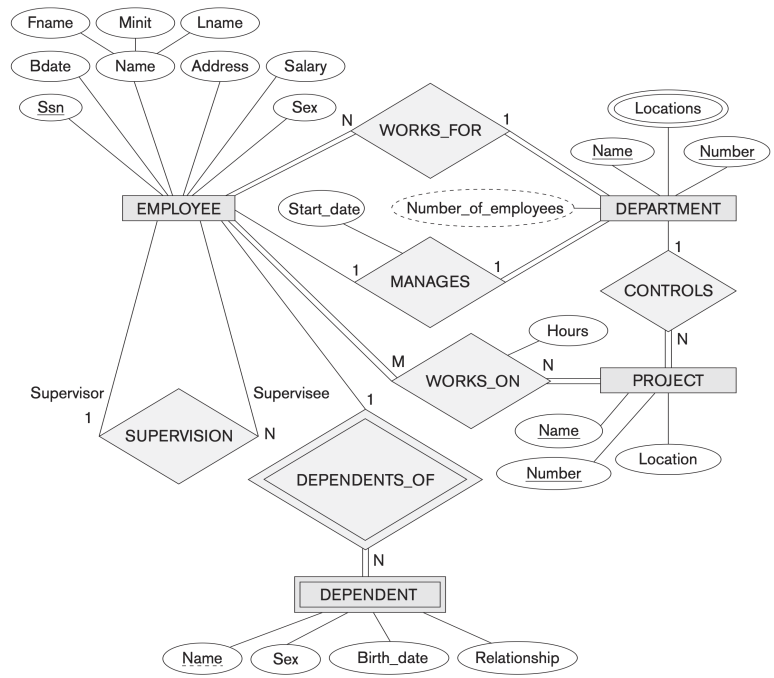
Figure 9.2
Result of mapping the COMPANY ER schema into a relational database schema.

Step 6: Mapping of multivalued attributes

For each multivalued attribute A, create a new relation R

- R will include an attribute corresponding to A and a primary key K as foreign key in R of the relation that represents the entity type that has A as a multivalued attribute
- the primary key R is a combination of A and K (if the multivalued attribute is composite, include its simple components).

Figure 9.1
The ER conceptual schema diagram for the COMPANY database.



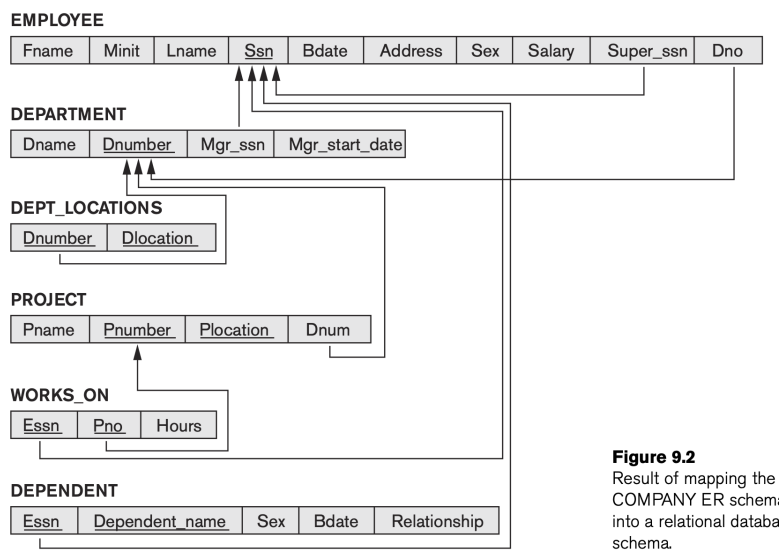


Figure 9.2
Result of mapping the COMPANY ER schema into a relational database schema.

Step 7: Mapping of N-ary relationships

For each N-ary relationship type R, where n ¿ 2, create a new relation S to represent R

- include as foreign attributes in S as primary key attributes that participate in the relationship
- include any simple attributes (or simple components of composites) as attributes of S
- the primary of S is a combination of all foreign keys that reference the relations participating in the relationship
- of the cardinality constraints on any of the entity types E participating in relation R is 1, then the primary key of S should not include the foreign key attribute that reference E' corresponding to E

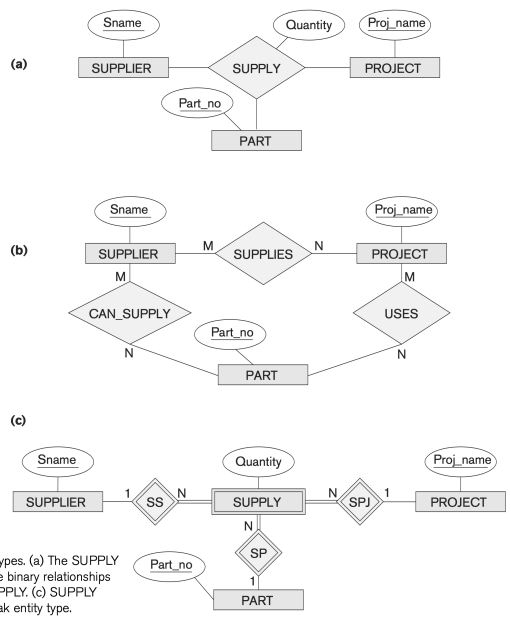
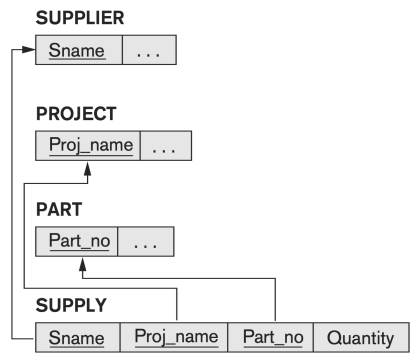


Figure 7.17
Ternary relationship types. (a) The SUPPLY relationship. (b) Three binary relationships not equivalent to SUPPLY. (c) SUPPLY represented as a weak entity type.

Figure 9.4
Mapping the *n*-ary relationship type SUPPLY from Figure 7.17(a).



Step 8: Mapping specialisation and generalisation

Convert each specialisation with m subclasses $\{S_1, S_2, \ldots, S_m\}$ and generalised superclass $C(k, a_1, \ldots, a_n)$, where k is the primary key into relation schemas using one of the following options:

- **Option 8A: Multiple relations - superclass and subclasses**
  - create a relation L for C with attributes of C and the primary key k
  - create a relation L_i for each subclass S_i with the attributes $\{k\} \cup \{\text{attributes of } S_i\}$ and k as primary key
  - works for total or partial, disjoint or overlapping specialisations
- **Option 8B: Multiple relations - subclass relations only**
  - create a relation L_i for each subclass S_i with attributes $\{\text{attributes of } S_i\} \cup \{k, a_1, \ldots, a_n\}$ with primary key of k
  - only works for specialisations where the subclasses are total, also recommended for specialisations with a disjointedness constraint
- **Option 8C: Single relation with one type attribute**
  - create a relation L with attributes $\{k, a_1, \ldots, a_n\} \cup \{\text{attributes of } S_i\} \cup \ldots. \{\text{attributes of } S_m\} \cup \{t\}$ and primary key of k
  - t is a type or discriminating attribute whose value indicates the subclass to which the tuple belongs
  - used for subclasses that are disjoint
  - can generate NULL values
- **Option 8D: Single relation with multiple type attributes**
  - create a relation L with attributes $\{k, a_1, \ldots, a_n\} \cup \{\text{attributes of } S_i\} \cup \ldots. \{\text{attributes of } S_m\} \cup \{t_1, t_2, \ldots, t_m\}$ and primary key of k, where t_i is a Boolean type attribute indicating whether the tuple belongs to subclass S_i
  - used for specialisations whose subclasses are overlapping, also works for disjoint

Step 8: Mapping specialisation and generalisation

Convert each specialisation with m subclasses $\{S_1, S_2, \ldots, S_m\}$ and generalised superclass $C(k, a_1, \ldots, a_n)$, where k is the primary key into relation schemas using one of the following options:

- **Option 8A: Multiple relations - superclass and subclasses**
- **Option 8B: Multiple relations - subclass relations only**
- **Option 8C: Single relation with one type attribute**
- **Option 8D: Single relation with multiple type attributes**

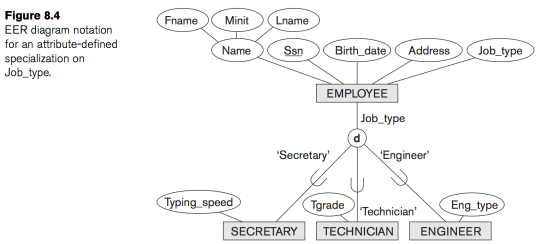


Figure 8.4
EER diagram notation for an attribute-defined specialization on Job_type.

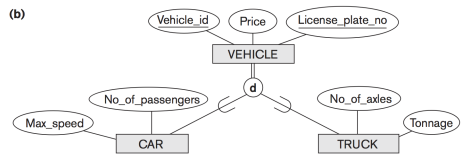


Figure 8.3
Generalization. (a) Two entity types, CAR and TRUCK. (b) Generalizing CAR and TRUCK into the superclass VEHICLE.

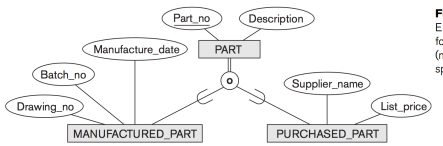


Figure 8.5
EER diagram notation for an overlapping (nondisjoint) specialization.

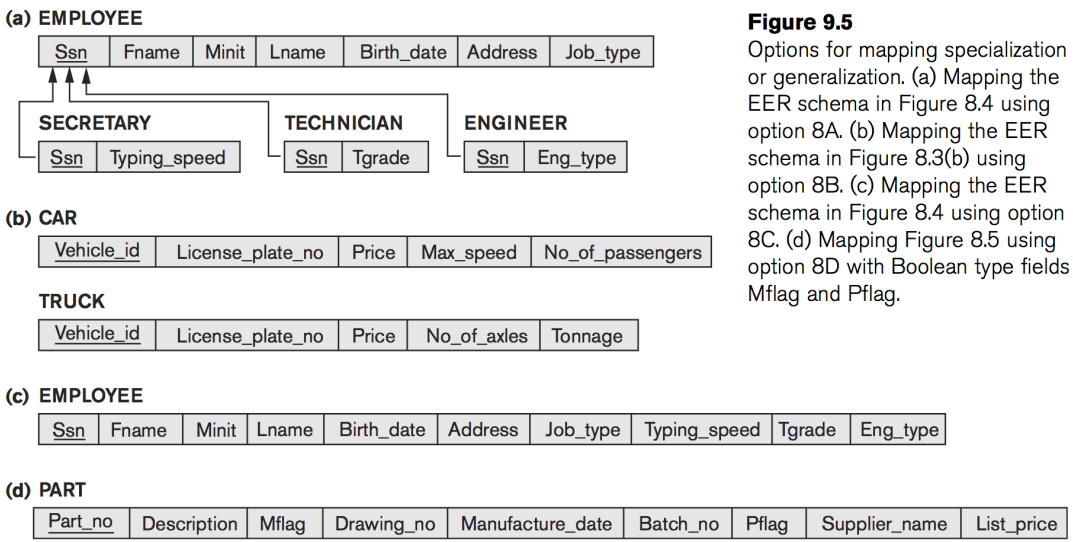


Figure 9.5
Options for mapping specialization or generalization. (a) Mapping the EER schema in Figure 8.4 using option 8A. (b) Mapping the EER schema in Figure 8.3(b) using option 8B. (c) Mapping the EER schema in Figure 8.4 using option 8C. (d) Mapping Figure 8.5 using option 8D with Boolean type fields Mflag and Pflag.

Step 8: Mapping specialisation and generalisation - Lattices

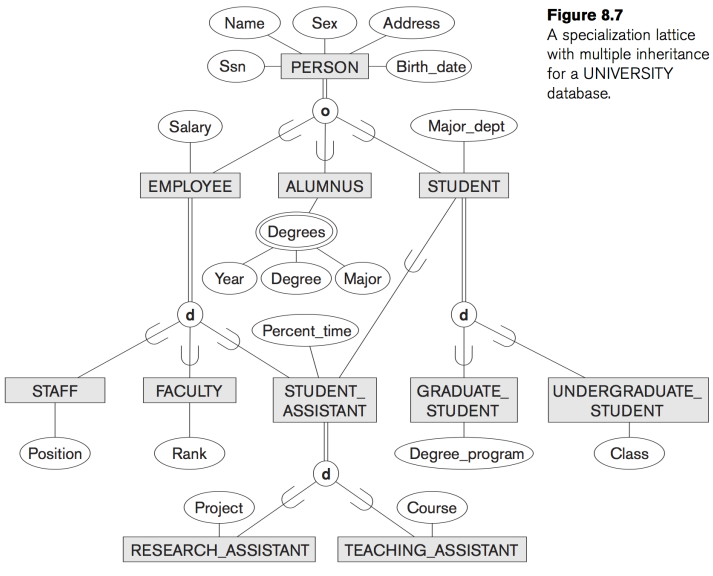


Figure 8.7
A specialization lattice with multiple inheritance for a UNIVERSITY database.

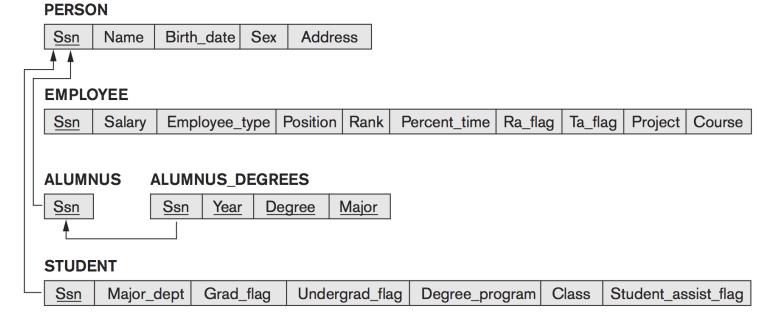


Figure 9.6
Mapping the EER specialization lattice in Figure 8.8 using multiple options.

Step 9: Mapping unions

When mapping classes whose superclasses have different keys, a surrogate key is defined when creating a relation which corresponds to the category. This key is the foreign key in the other relations participating in the relationship.

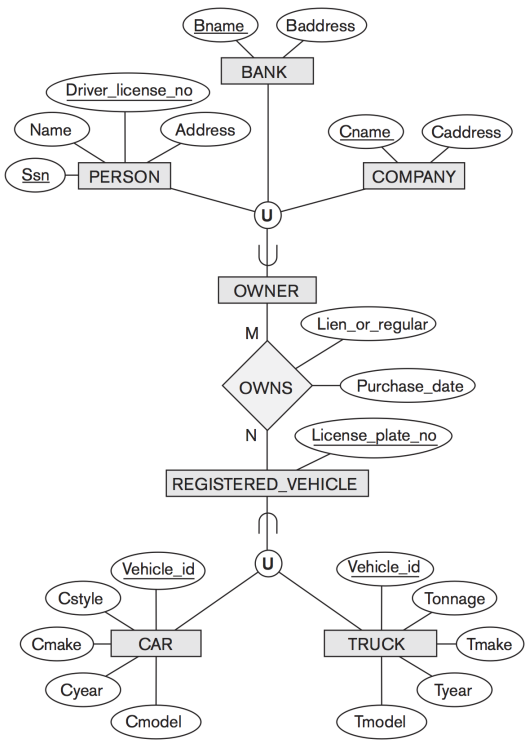
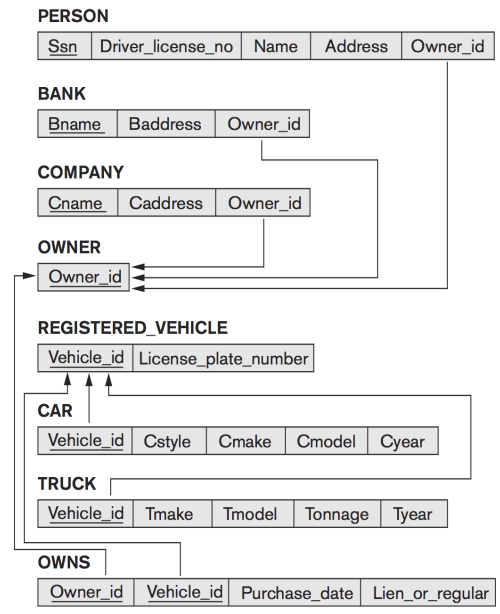


Figure 8.8
Two categories (union types): OWNER and REGISTERED_VEHICLE.

Figure 9.7
Mapping the EER categories (union types) in Figure 8.8 to relations.



`Owner_id` is the surrogate key.

COS221
L15 an L16 - SQL Database Programming
(Chapter 13 in Edition 6 and Chapter 10 in Edition 7)

Linda Marshall

27 and 30 March 2023

Accessing data from application programs

Two categories of application programmes exist:

- **Web-based applications** - An application of the *three-tier architecture*
  - Web interface the client
  - Application program, the middle layer, implements the business logic
  - Database server forms the bottom layer.

  An application does not necessarily link to a single database, it may link to multiple database systems.
- **Canned applications**
  - SQL Workbench is an example of an interactive interface.
  - A file of SQL commands can be presented to the database system.

Accessing data from application programs - *Web-based applications*

Typical sequence of interaction with the database (e.g. access to Wheatley):

- establish or open a connection to the database server
  – URL
  – login account name
  – password for database access
- program interacts with the database by submitting queries, updates, and other database commands.
- program terminates or closes the connection after using the access.

Accessing data from application programs - *Canned applications*

Impedence mismatch occurs when there is a difference between the programming language model and the database model. This occurs when:

- the programming language type does not correspond to the database type. Need to bind types between the programming language and the database.
- mapping between database set/multiset and a data structure in the programming language

Accessing data from application programs - *Canned applications*

Approaches to include database interactions in application programs:

- **Embed database commands in a general-purpose programming language** - database statements are embedded into the host programming language, but they are identified by a special prefix and scanned by a pre-compiler.
- **Use a library of database functions or classes** - makes use of an API to access the database. For example, there could be functions to connect to a database, execute a query, execute an update, and so on.
- **Design a new language** - a database programming language is designed from scratch to be compatible with the database model and query language.

Canned Approach 1 - Embed database commands in a general-purpose programming language

- In this embedded approach, the programming language is called the host language.
- Examples of host languages include C/C++, Java or C#
  - Embedded SQL's host language is C/C++
  - Dynamic SQL
  - SQLJ's host language is Java

Canned Approach 1 - Embed database commands in a general-purpose programming language - Embedded SQL in C/C++

Embedded SQL statements include data or constraint definitions, queries, updates, or view definitions.
Defining tuples

- **Identifying Embedded SQL statements**
  – Prefix SQL statements with EXEC SQL for the preprocessor (or precompiler) to separate SQL statements from the host language code.
  – The SQL statements within a program are terminated by a matching END-EXEC or by a semicolon (;).
- **Shared variables** - used in both the C program and the embedded SQL statements - prefixed by a colon (:)
  – (:) distinguishes program variable names from the names of database schema constructs such as column names and table names.
  – (:) allows program variables to have the same names as attribute names
  – COMPANY database (EMPLOYEE and DEPARTMENT) program variables declared to match the types of the database attributes that the program will process.
- **Data types**
  – The SQL types INTEGER, SMALLINT, REAL, and DOUBLE are mapped to the C types long, short, float, and double, respectively
  – Although `varchar` is not a standard C data type, it is permitted when C is used for SQL database programming

Canned Approach 1 - Embed database commands in a general-purpose programming language - Embedded SQL in C/C++

```
0) int loop ;
1) EXEC SQL BEGIN DECLARE SECTION ;
2) varchar dname [16], fname [16], lname [16], address [31] ;
3) char ssn [10], bdate [11], sex [2], minit [2] ;
4) float salary, raise ;
5) int dno, dnumber ;
6) int SQLCODE ; char SQLSTATE [6] ;
7) EXEC SQL END DECLARE SECTION ;
```

Figure 10.1
C program variables used in the embedded SQL examples E1 and E2.

- Line 1 and 7 – embedded SQL commands. (tell precompiler about shared variables)
- Lines 2 through 5 – regular C program declarations
- Line 6 – SQLCODE and SQLSTATE are used to communicate errors and exception conditions

SQLCODE - integer variable.
For every command executed, DBMS returns a value in SQLCODE. i.e
0 - statement executed successfully
100 - no more data available in query result
$<$ 0 - indicates some error has occurred

SQLSTATE - String of five characters.
'00000' = no error or exception
Other values indicate various errors or exceptions.
For example, '02000' indicates 'no more data'

Canned Approach 1 - Embed database commands in a general-purpose programming language - Embedded SQL in C/C++

Making database connections

- Connecting to the database

  CONNECT TO <server name> AS <connection name>
  AUTHORIZATION <user account name and password> ;

- Change connection

  SET CONNECTION <connection name> ;

- Terminate connection

  DISCONNECT <connection name> ;

```
   //Program Segment E1:
0) loop = 1 ;
1) while (loop) {
2)   prompt("Enter a Social Security Number: ", ssn) ;
3)   EXEC SQL
4)     SELECT Fname, Minit, Lname, Address, Salary
5)     INTO :fname, :minit, :lname, :address, :salary
6)     FROM EMPLOYEE WHERE Ssn = :ssn ;
7)   if (SQLCODE = = 0) printf(fname, minit, lname, address, salary)
8)     else printf("Social Security Number does not exist: ", ssn) ;
9)   prompt("More Social Security Numbers (enter 1 for Yes, 0 for No): ", loop) ;
10)  }
```

Figure 10.2
Program segment E1, a C program segment with embedded SQL.

Canned Approach 1 - Embed database commands in a general-purpose programming language - Embedded SQL in C/C++

Processing query results using cursors

- A cursor is a variable that refers to a single tuple from a query result that retrieves a collection of tuples.
- A cursor is used to loop over the query result, one record at a time.
- General cursor options

DECLARE <cursor name> [**INSENSITIVE**] [**SCROLL**] **CURSOR**
[**WITH HOLD**] **FOR** <query specification>
[**ORDER BY** <ordering specification>]
[**FOR READ ONLY** | **FOR UPDATE** [**OF** <attribute list>]] ;

Canned Approach 1 - Embed database commands in a general-purpose programming language - Embedded SQL in C/C++

Processing query results using cursors

Figure 10.3
Program segment E2, a C program segment that uses cursors with embedded SQL for update purposes.

```
    //Program Segment E2:
 0) prompt("Enter the Department Name: ", dname) ;
 1) EXEC SQL
 2)   SELECT Dnumber INTO :dnumber
 3)   FROM DEPARTMENT WHERE Dname = :dname ;
 4) EXEC SQL DECLARE EMP CURSOR FOR
 5)   SELECT Ssn, Fname, Minit, Lname, Salary
 6)   FROM EMPLOYEE WHERE Dno = :dnumber
 7)   FOR UPDATE OF Salary ;
 8) EXEC SQL OPEN EMP ;
 9) EXEC SQL FETCH FROM EMP INTO :ssn, :fname, :minit, :lname, :salary ;
10) while (SQLCODE = = 0) {
11)   printf("Employee name is:", Fname, Minit, Lname) ;
12)   prompt("Enter the raise amount: ", raise) ;
13)   EXEC SQL
14)     UPDATE EMPLOYEE
15)     SET Salary = Salary + :raise
16)     WHERE CURRENT OF EMP ;
17)   EXEC SQL FETCH FROM EMP INTO :ssn, :fname, :minit, :lname, :salary ;
18)   }
19) EXEC SQL CLOSE EMP ;
```

Canned Approach 1 - Embed database commands in a general-purpose programming language - Embedded SQL in C/C++

Processing query results using cursors

- OPEN CURSOR is used fetches the query result and sets the cursor pointer to before the first entry
- FETCH moves the cursor to the next row in the result of the query and copies attribute values into the C program variables specified in the FETCH command by an INTO clause
- CLOSE CURSOR is issued to indicate that we are done with processing the result of the query associated with that cursor
- FOR UPDATE OF lists the names of any attributes that will be updated by the program. If rows are to be deleted, FOR UPDATE must be added without specifying any attributes
- WHERE CURRENT OF ¡cursor name¿ specifies the current tuple referenced by the cursor that is to be deleted or updated

Canned Approach 1 - Embed database commands in a general-purpose programming language - Dynamic SQL

- Moves the specification of queries from compile-time to run-time.
- Is necessary when the user is allowed to specify their own query
- Updates are relatively easy to include, dynamic retrievals are a little more tricky due to not knowing the attributes or their types when writing the program. Therefore, a complex data structure is required to manage the results of a query.

Canned Approach 1 - Embed database commands in a general-purpose programming language - SQLJ in Java

SQLJ's host language is Java and is the standard adopted by several vendors for embedding SQL in Java. SQLJ translates SQL into Java that are executed through the JDBC interface to an Oracle DBMS. It is therefore necessary to install a JDBC driver when using SQLJ.

- Import several class libraries

```
import java.sql.* ;
import java.io.* ;
import sqlj.runtime.* ;
```

- Uses exceptions for error handling
  SQLException is used to return errors or exception conditions

Canned Approach 1 - Embed database commands in a general-purpose programming language - SQLJ in Java

- Establishing a connection

```
1) import java.sql.* ;
2) import java.io.* ;
3) import sqlj.runtime.* ;
4) import sqlj.runtime.ref.* ;
5) import oracle.sqlj.runtime.* ;
   ...
6) DefaultContext cntxt =
7) oracle.getConnection("<url name>", "<user name>", "<password>", true) ;
8) DefaultContext.setDefaultContext(cntxt) ;
   ...
```

Figure 10.5
Importing classes needed for including SQLJ in Java programs in Oracle, and establishing a connection and default context.

- Java program variables

Figure 10.6
Java program variables used in SQLJ examples J1 and J2.

```
1) string dname, ssn , fname, fn, lname, ln,
   bdate, address ;
2) char sex, minit, mi ;
3) double salary, sal ;
4) integer dno, dnumber ;
```

Canned Approach 1 - Embed database commands in a general-purpose programming language - SQLJ in Java

- Accessing the data

```
   //Program Segment J1:
 1) ssn = readEntry("Enter a Social Security Number: ") ;
 2) try {
 3)   #sql { SELECT Fname, Minit, Lname, Address, Salary
 4)     INTO :fname, :minit, :lname, :address, :salary
 5)     FROM EMPLOYEE WHERE Ssn = :ssn} ;
 6) } catch (SQLException se) {
 7)     System.out.println("Social Security Number does not exist: " + ssn) ;
 8)     Return ;
 9)   }
10) System.out.println(fname + " " + minit + " " + lname + " " + address
      + " " + salary)
```

Figure 10.7
Program segment J1, a Java program segment with SQLJ.

Canned Approach 1 - Embed database commands in a general-purpose programming language - SQLJ in Java

- Processing query results with iterators
  An iterator is an object associated with a set/multiset, the query result.
  There are two types of iterators:
  - **Named iterator** - Associated with a query result by listing attribute names and types in query result
  - **Positional iterator** - Lists only attribute types in query result. Makes use of FETCH.

Canned Approach 1 - Embed database commands in a general-purpose programming language - SQLJ in Java

Named iterator

Figure 10.8
Program segment J2A, a Java program segment that uses a named iterator to print employee information in a particular department.

```
    //Program Segment J2A:
 0) dname = readEntry("Enter the Department Name: ") ;
 1) try {
 2)   #sql { SELECT Dnumber INTO :dnumber
 3)     FROM DEPARTMENT WHERE Dname = :dname} ;
 4) } catch (SQLException se) {
 5)   System.out.println("Department does not exist: " + dname) ;
 6)   Return ;
 7)   }
 8) System.out.printline("Employee information for Department: " + dname) ;
 9) #sql iterator Emp(String ssn, String fname, String minit, String lname,
      double salary) ;
10) Emp e = null ;
11) #sql e = { SELECT ssn, fname, minit, lname, salary
12)   FROM EMPLOYEE WHERE Dno = :dnumber} ;
13) while (e.next()) {
14)   System.out.printline(e.ssn + " " + e.fname + " " + e.minit + " " +
        e.lname  + " " + e.salary) ;
15) } ;
16) e.close() ;
```

Canned Approach 1 - Embed database commands in a general-purpose programming language - SQLJ in Java

Positional iterator

Figure 10.9
Program segment J2B, a Java program segment that uses a positional iterator to print employee information in a particular department.

```
    //Program Segment J2B:
 0) dname = readEntry("Enter the Department Name: ") ;
 1) try {
 2)   #sql { SELECT Dnumber INTO :dnumber
 3)     FROM DEPARTMENT WHERE Dname = :dname} ;
 4) } catch (SQLException se) {
 5)   System.out.println("Department does not exist: " + dname) ;
 6)   Return ;
 7)   }
 8) System.out.printline("Employee information for Department: " + dname) ;
 9) #sql iterator Emppos(String, String, String, String, double) ;
10) Emppos e = null ;
11) #sql e = { SELECT ssn, fname, minit, lname, salary
12)   FROM EMPLOYEE WHERE Dno = :dnumber} ;
13) #sql { FETCH :e INTO :ssn, :fn, :mi, :ln, :sal} ;
14) while (!e.endFetch()) {
15)   System.out.printline(ssn + " " + fn + " " + mi + " " + ln + " " + sal) ;
16)   #sql { FETCH :e INTO :ssn, :fn, :mi, :ln, :sal} ;
17) } ;
18) e.close() ;
```

Canned Approach 2 - Using a library of database functions or classes

Till now, the programming has been static. The query text has been fixed and cannot be changed. Libraries (Application Programming Interface - API) allow for dynamic database querying.

Two function call interfaces will be discussed:

- SQL Call Level Interface (SQL/CLI), part of the SQL standard – provides a standardisation of the Open DataBase Connection (ODBC) library
- JDBC^{TM}, an interface for accessing the database from Java

Canned Approach 2 - Using a library of database functions or classes - SQL/CLI

SQL/CLI and ODBC using C as the host language SQL statements are dynamically created and passed as string parameters in the function calls.
There is a need to keep track of the information about host program interactions with the database. This is done using 4 records which are defined as C structs and are accessed through a C pointer (handle) to the struct.

- Environment record - Tracks one or more database connections and sets environment information
- Connection record - Keeps track of information needed for a particular database connection
- Statement record - Keeps track of the information needed for one SQL statement
- Description record - Keeps track of information about tuples or parameters

Canned Approach 2 - Using a library of database functions or classes - SQL/CLI

SQL/CLI and ODBC using C as the host language

To create a record and return its handle, the following SQL/CLI function is used:

`SQLAllocHandle(<handle_type>, <handle_1>, <handle_2>)`

- `<handle_type>` indicates the type of record being created. The possible values for this parameter are the keywords SQL_HANDLE_ENV, SQL_HANDLE_DBC, SQL_HANDLE_STMT, or SQL_HANDLE_DESC, for an environment, connection, statement, or description record, respectively.
- `<handle_1>` indicates the container within which the new handle is being created.
- `<handle_2>` is the pointer to the newly created record of type `<handle_type>`.

Canned Approach 2 - Using a library of database functions or classes - SQL/CLI

```
    //Program CLI1:
 0) #include sqlcli.h ;
 1) void printSal() {
 2) SQLHSTMT stmt1 ;
 3) SQLHDBC con1 ;
 4) SQLHENV env1 ;
 5) SQLRETURN ret1, ret2, ret3, ret4 ;
 6) ret1 = SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &env1) ;
 7) if (!ret1) ret2 = SQLAllocHandle(SQL_HANDLE_DBC, env1, &con1) else exit ;
 8) if (!ret2) ret3 = SQLConnect(con1, "dbs", SQL_NTS, "js", SQL_NTS, "xyz",
      SQL_NTS) else exit ;
 9) if (!ret3) ret4 = SQLAllocHandle(SQL_HANDLE_STMT, con1, &stmt1) else exit ;
10) SQLPrepare(stmt1, "select Lname, Salary from EMPLOYEE where Ssn = ?",
      SQL_NTS) ;
11) prompt("Enter a Social Security Number: ", ssn) ;
12) SQLBindParameter(stmt1, 1, SQL_CHAR, &ssn, 9, &fetchlen1) ;
13) ret1 = SQLExecute(stmt1) ;
14) if (!ret1) {
15)   SQLBindCol(stmt1, 1, SQL_CHAR, &lname, 15, &fetchlen1) ;
16)   SQLBindCol(stmt1, 2, SQL_FLOAT, &salary, 4, &fetchlen2) ;
17)   ret2 = SQLFetch(stmt1) ;
18)   if (!ret2) printf(ssn, lname, salary)
19)     else printf("Social Security Number does not exist: ", ssn) ;
20)   }
21) }
```

Figure 10.10
Program segment CLI1, a C program segment with SQL/CLI.

Canned Approach 2 - Using a library of database functions or classes - JDBC

JDBC - SQL Function Calls for Java Programming Java program with JDBC function calls can access any RDBMS that has a JDBC driver available.

- Import the JDBC class libraries, which are called java.sql.*.
- Single Java program can connect to several different databases. Databases are referCred to as data sources accessed by the Java program
- Class.forName("com.mysql.jdbc.Driver"); Loads a JDBC driver explicitly (for mysql).

Canned Approach 2 - Using a library of database functions or classes - JDBC

```
import java.sql.*;

class MysqlCon{
public static void main(String args[]){
try{
Class.forName("com.mysql.jdbc.Driver");

Connection con=DriverManager.getConnection
("jdbc:mysql://localhost:3306/sonoo","root","root");
//here sonoo is the database name, root is the username and root is the password
Statement stmt=con.createStatement();

ResultSet rs=stmt.executeQuery("select * from emp");

while(rs.next())
System.out.println(rs.getInt(1)+"   "+rs.getString(2)+"
  "+rs.getString(3));

con.close();

}catch(Exception e){ System.out.println(e);}

}
}
```

More information is available from: - https://docs.oracle.com/javase/tutorial/jdbc/

Canned Approach 3 - Design a new language

Additions are made to SQL. There are two basic additions:

- the concept of **Stored Procedures** (or Functions) - Persistent Stored Modules (SQL/PSM); and
- the inclusion of **constructs for selection and looping** in SQL.

Design a new language

Stored Procedures

```
CREATE PROCEDURE <procedure name> (<parameters>)
<local declarations>
<procedure body> ;
```

```
CREATE FUNCTION <function name> (<parameters>)
                                    RETURNS <return type>
<local declarations>
<function body> ;
```

Calling a procedure or function:

```
CALL <procedure or function name> (<argument list>) ;
```

The procedure does not have to be defined in the DBMS, it can be in a "typical" programming language.

```
CREATE PROCEDURE <procedure name> (<parameters>)
  LANGUAGE <programming language name>
  EXTERNAL NAME <file path name> ;
```

Canned Approach 3 - Design a new language

Selection and Looping

```
IF <condition> THEN <statement list>
ELSEIF <condition> THEN <statement list> ...
ELSEIF <condition> THEN <statement list>
ELSE <statement list>
END IF
```

```
WHILE <condition> DO <statement list>
END WHILE  REPEAT
<statement list> UNTIL <condition>
END REPEAT
```

```
FOR <loop name> AS <cursor name> CURSOR FOR <query> DO
<statement list>
END FOR
```

Breaking out of the loop is done by using the `LEAVE <loop name>` statement.

(Dis)Advantages of the Approaches

- **Embed database commands in a general-purpose programming language**
  – *Advantage* - Query text checked for syntax errors and validated against database schema at compile time
  – *Disadvantage* - The query cannot be change at runtime. This means that for complex applications where queries have to be generated at runtime, the library of database functions approach is more suitable
- **Using a library of database functions or classes**
  – *Advantage* - More flexibility as queries can be generated at runtime.
  – *Disadvantages* - More complex programming and there is no checking of syntax done at compile time.
- **Design a new language**
  – *Advantage* - Does not suffer from the impedance mismatch problem, programming language and database types are the same
  – *Disadvantage* - Programmers must learn a new language which may differ between DBMS vendors