

Principles of Database Design

Dr CK Raju

Muthoot Institute of Technology and Science
(Affiliated to KTU)

rajuck@mgmits.ac.in

February 6, 2019

Overview

1 Course Particulars

- Course Objectives
- Syllabus
- Student Outcomes

2 Syllabus

- Modules

3 Module I

- Introduction

- Database Concepts
- Database Environment
- Database Approach
- Database Users
- End Users
- Advantages
- Implications
- Database Models
- Extending DB Capabilities
- When NOT to use DBMS

Objectives

- To impart basic understanding of the theory and applications of database management systems.
- To give basic level understanding of internals of database systems.
- To expose to some of the recent trends in databases.

Syllabus

- Types of data, database and DBMS, Languages and users.
- Software Architecture, E-R and Extended E-R Modelling,
- Relational Model concepts and languages, relational algebra and tuple relational calculus,
- SQL, views, assertions and triggers, HLL interfaces,
- Relational db design, FDs and normal forms,
- Secondary storage organization, indexing and hashing,
- Query optimization, concurrent transaction processing
- Recovery principles, recent topics.

Outcomes: Students would be able to

- define, explain and illustrate the fundamental concepts of databases,
- construct an Entity-Relationship (E-R) model from specifications and to perform the transformation of the conceptual model into corresponding logical data structures,
- model and design a relational database following the design principles,
- develop queries for relational database in the context of practical applications,
- define, explain and illustrate fundamental principles of data organization, query optimization and concurrent transaction processing,
- appreciate the latest trends in databases.

Modules

- Module I - Introduction, Entity-Relationship Model
- Module II - Relational Model, Database Languages
- Module III - SQL, Views-assertions-triggers, Functions-Procedures-HLL-Interfaces
- Module IV - Relational Database Design
- Module V - Physical Data Organisation, Query Optimization
- Module VI - Transaction Processing Concepts, Recent Topics

Introduction

- Data: structured, semi-structured and unstructured
- DBMS: Concept and Overview
- Data Models
- Database Languages
- DB Admin, DB Users
- DBMS: three schema architecture
- DB architectures and classification

Ref: Elmasri - Ch-1 and Ch-2,
Korth - Ch-1

1. *Journal of Management Studies*, 1997, 34, 1, 1-14.

- Data : Facts that can be stored, processed and also possess some implicit meaning
- Database : Collection of related data, representing some aspect of real world, univ-of-discourse
- Databases are designed, built and populated with data for some specific purpose
- Manual databases exist in some libraries
- DBMS is a software system used to create and maintain a database
- Meta-data : Database definition or descriptive database catalog information
- DBMSs help in designing, creating, manipulating and sharing of databases

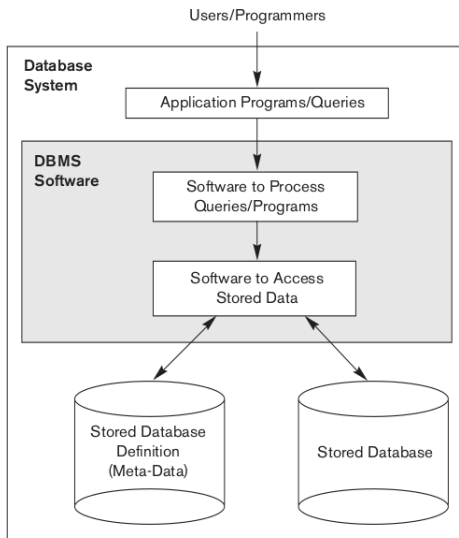
Ref: Elmasri - Ch-1 and Ch-2,
Korth - Ch-1

Database Concepts

- DB Query : S/w program that queries database for some information.
- DB Transaction : An activity that reads or manipulates data in a database.
- Protection : System protection and security protection
- System protection - against h/w, s/w malfunction; Security - against unauthorised access

Ref: Elmasri - Ch-1 and Ch-2,
Korth - Ch-1

A typical Database environment



A student database having course information

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

Ref: Elmasri - Ch-1 and Ch-2,
Korth - Ch-1

Database Approach

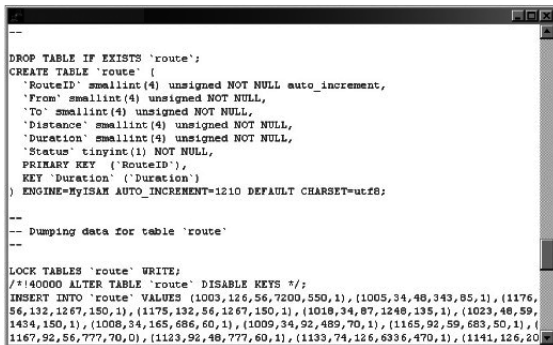
Special Features

- Self-describing notion of database system - meta-data
- Insulation between programs, data and data abstraction
- Support of multiple **views** of data
- Sharing of data and multiuser transaction processing

Ref: Elmasri - Ch-1 and Ch-2,
Korth - Ch-1

Database Approach

- Self-describing notion of database system - meta-data



```
--  
DROP TABLE IF EXISTS `route`;  
CREATE TABLE `route` (  
  `RouteID` smallint(4) unsigned NOT NULL auto_increment,  
  `From` smallint(4) unsigned NOT NULL,  
  `To` smallint(4) unsigned NOT NULL,  
  `Distance` smallint(4) unsigned NOT NULL,  
  `Duration` smallint(4) unsigned NOT NULL,  
  `Status` tinyint(1) NOT NULL,  
  PRIMARY KEY (`RouteID`),  
  KEY `Duration` (`Duration`)  
) ENGINE=MyISAM AUTO_INCREMENT=1210 DEFAULT CHARSET=utf8;  
  
--  
-- Dumping data for table `route`  
--  
  
LOCK TABLES `route` WRITE;  
/*!40000 ALTER TABLE `route` DISABLE KEYS */;  
INSERT INTO `route` VALUES (1003,126,56,7200,550,1),(1005,34,48,343,85,1),(1176,  
56,132,1267,150,1),(1175,132,56,1267,150,1),(1018,34,87,1248,135,1),(1023,48,59,  
1434,150,1),(1008,34,165,686,60,1),(1009,34,92,489,70,1),(1165,92,59,683,50,1),(  
1167,92,56,777,70,0),(1123,92,48,777,60,1),(1133,74,126,6336,470,1),(1141,126,20
```

- Insulation between programs, data and data abstraction
- Support of multiple **views** of data
- Sharing of data and multiuser transaction processing

Database Approach

- Self-describing notion of database system - meta-data
- Insulation between programs, data and data abstraction
 - ① Program-data independence - independence of data from programs
 - ② Program-operation independence
 - ③ Characteristic of data abstraction - program-data and program-operation independence
- Support of multiple **views** of data
- Sharing of data and multiuser transaction processing

Ref: Elmasri - Ch-1 and Ch-2,
Korth - Ch-1

Database Approach

- Self-describing notion of database system - meta-data
- Insulation between programs, data and data abstraction
- Support of multiple **views** of data
 - ① Special perspective of database
 - ② Might contain subset of database in conjunction with other information
- Sharing of data and multiuser transaction processing

Ref: Elmasri - Ch-1 and Ch-2,
Korth - Ch-1

Database Approach

- Self-describing notion of database system - meta-data
- Insulation between programs, data and data abstraction
- Support of multiple **views** of data
- Sharing of data and multiuser transaction processing
 - ① Multi-user DBMS involves **concurrency control**
 - ② Special **online transaction processing** applications
 - ③ Process **isolation** amongst different processes
 - ④ Ensure **atomicity** property of transaction - either whole or none

Ref: Elmasri - Ch-1 and Ch-2,
Korth - Ch-1

Database Users

Users may be divided into

- **Actors on the Scene** - Those who actually use and control the database content, and those who design, develop and maintain database applications
- **Workers Behind the Scene** - Those who design and develop the DBMS software and related tools, and the computer systems operators

Ref: Elmasri - Ch-1 and Ch-2,
Korth - Ch-1

Actors on the scene

- **Database Administrators** - Responsible for authorizing access to the database, for coordinating and monitoring its use, acquiring software and hardware resources, controlling its use and monitoring efficiency of operations.
- **Database Designers** - Responsible to define the content, the structure, the constraints, and functions or transactions against the database. They must communicate with the end-users and understand their needs.

Ref: Elmasri - Ch-1 and Ch-2,
Korth - Ch-1

End Users : Actors on the Scene

- **Casual** - those who access database occasionally when needed
- **Naive or Parametric**
- **Sophisticated**
- **Standalone**

Ref: Elmasri - Ch-1 and Ch-2,
Korth - Ch-1

End Users : Actors on the Scene

- **Casual**
- **Naive or Parametric** - they make up a large section of the end-user population. They use previously well-defined functions in the form of *canned transactions* against the database. Examples are bank-tellers or reservation clerks who do this activity for an entire shift of operations.
- **Sophisticated**
- **Standalone**

Ref: Elmasri - Ch-1 and Ch-2,
Korth - Ch-1

End Users : Actors on the Scene

- **Casual**
- **Naive or Parametric**
- **Sophisticated** - These include business analysts, scientists, engineers, others thoroughly familiar with the system capabilities. Many use tools in the form of software packages that work closely with the stored database.
- **Standalone**

Ref: Elmasri - Ch-1 and Ch-2,
Korth - Ch-1

End Users : Actors on the Scene

- **Casual**
- **Naive or Parametric**
- **Sophisticated**
- **Standalone** - Mostly maintain personal databases using ready-to-use packaged applications. An example is a tax program user that creates its own internal database. Another example is a user that maintains an address book.

Ref: Elmasri - Ch-1 and Ch-2,
Korth - Ch-1

Database Approach - Advantages

- 1 Controlling redundancy in data storage and in development and maintenance efforts. Sharing of data among multiple users.
- 2 Restricting unauthorized access to data.
- 3 Providing persistent storage for program Objects - in Object-oriented DBMSes
- 4 Providing Storage Structures (e.g. indexes) for efficient Query Processing

Ref: Elmasri - Ch-1 and Ch-2,
Korth - Ch-1

Database Approach - Advantages continued

- ① **Potential for enforcing standards:** - This is very crucial for the success of database applications in large organizations. Standards refer to data item names, display formats, screens, report structures, meta-data (description of data), Web page layouts, etc.
- ② **Reduced application development time:** - Incremental time to add each new application is reduced.

Ref: Elmasri - Ch-1 and Ch-2,

Korth - Ch-1

Database Approach - Implications

- ➊ Providing backup and recovery services.
- ➋ Providing multiple interfaces to different classes of users.
- ➌ Representing complex relationships among data.
- ➍ Enforcing integrity constraints on the database.
- ➎ Drawing inferences and actions from the stored data using deductive and active rules

Ref: Elmasri - Ch-1 and Ch-2,
Korth - Ch-1

Database Approach - Implications continued

- ① **Flexibility to change data structures:** - Database structure may evolve as new requirements are defined.
- ② **Availability of current information:** - Extremely important for on-line transaction systems such as airline, hotel, car reservations.
- ③ **Economies of scale:** - Wasteful overlap of resources and personnel can be avoided by consolidating data and applications across departments.

Ref: Elmasri - Ch-1 and Ch-2,
Korth - Ch-1

Database Models

- ❶ **Early Database Applications:** - The Hierarchical and Network Models were introduced in mid 1960s and dominated during the seventies. A bulk of the worldwide database processing still occurs using these models, particularly, the hierarchical model.
- ❷ **Relational Model based Systems**
- ❸ **Object-oriented**
- ❹ **Others**

Ref: Elmasri - Ch-1 and Ch-2,
Korth - Ch-1

Database Models

- ❶ **Early Database Applications**
- ❷ **Relational Model based Systems:** - Relational model was originally introduced in 1970, was heavily researched and experimented within IBM Research and several universities. Relational DBMS Products emerged in the early 1980s.
- ❸ **Object-oriented and others**
- ❹ **Others**

Ref: Elmasri - Ch-1 and Ch-2,
Korth - Ch-1

Database Models

- ❶ **Early Database Applications**
- ❷ **Relational Model based Systems**
- ❸ **Object-oriented Models** - Object-Oriented Database Management Systems (OODBMSs) were introduced in late 1980s and early 1990s to cater to the need of complex data processing in CAD and other applications. Their use has not taken off much. Many relational DBMSs have incorporated object database concepts, leading to a new category called object-relational DBMSs (ORDBMSs)
- ❹ **Others**

Ref: Elmasri - Ch-1 and Ch-2,
Korth - Ch-1

Database Models

- ① **Early Database Applications**
- ② **Relational Model based Systems**
- ③ **Object-oriented Models**
- ④ **Others:** - Extended relational systems add further capabilities (e.g. for multimedia data, XML, and other data types.) Web contains data in HTML (Hypertext markup language) with links among pages. This has given rise to a new set of applications and E-commerce is using new standards like XML (eXtended Markup Language). Script programming languages such as PHP and JavaScript allow generation of dynamic Web pages that are partially generated from a database. Also allow database updates through Web pages.

Ref: Elmasri - Ch-1 and Ch-2,
Korth - Ch-1

Extending DB Capabilities

Additional functionalities are getting added to DBMSes to cater to new needs of

- 1 Scientific Applications
- 2 XML (eXtensible Markup Language)
- 3 Image Storage and Management
- 4 Audio and Video Data Management
- 5 Data Warehousing and Data Mining
- 6 Spatial Data Management
- 7 Time Series and Historical Data Management

This calls for new R and D to support newer data types, complex data structures, new functions, storage and indexing schemes etc.

Ref: Elmasri - Ch-1 and Ch-2,

Korth - Ch-1

When NOT to use DBMS

- ① Some inhibitors - high investment in infrastructure and people
- ② When database application is simple, where changes are less frequent
- ③ If stringent real-time requirements are present
- ④ If access to multiple users is not required
- ⑤ If system is too complex to be modeled to a database system
- ⑥ If database users need special operations, unsupported by DBMS

Ref: Elmasri - Ch-1 and Ch-2,
Korth - Ch-1

Database System Concepts and Architecture

Outline

- Data Models and Their Categories
- History of Data Models
- Schemas, Instances, and States
- Three-Schema Architecture
- Data Independence
- DBMS Languages and Interfaces
- Database System Utilities and Tools
- Centralized and Client-Server Architectures
- Classification of DBMSs

Data Models

- **Data Model:**

- A set of concepts to describe the **structure** of a database, the **operations** for manipulating these structures, and certain **constraints** that the database should obey.

- **Data Model Structure and Constraints:**

- Constructs are used to define the database structure
- Constructs typically include **elements** (and their **data types**) as well as groups of elements (e.g. **entity**, **record**, **table**), and **relationships** among such groups
- Constraints specify some restrictions on valid data; these constraints must be enforced at all times

Data Models (continued)

- **Data Model Operations:**

- These operations are used for specifying database retrievals and updates by referring to the constructs of the data model.
- Operations on the data model may include **basic model operations** (e.g. generic insert, delete, update) and **user-defined operations** (e.g. compute_student_gpa, update_inventory)

Categories within every Data Model

- **Conceptual (high-level, semantic) data models:**
 - Provide concepts that are close to the way many users perceive data.
 - (Also called **entity-based** or **object-based** data models.)
- **Physical (low-level, internal) data models:**
 - Provide concepts that describe details of how data is stored in the computer. These are usually specified in an ad-hoc manner through DBMS design and administration manuals
- **Implementation (representational) data models:**
 - Provide concepts that fall between the above two, used by many commercial DBMS implementations (e.g. relational data models used in many commercial systems).

Schemas versus Instances

- Database Schema:
 - The **description** of a database.
 - Includes descriptions of the database structure, data types, and the constraints on the database.
- Schema Diagram:
 - An **illustrative** display of (most aspects of) a database schema.
- Schema Construct:
 - A **component** of the schema or an object within the schema, e.g., STUDENT, COURSE.

Schemas versus Instances

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

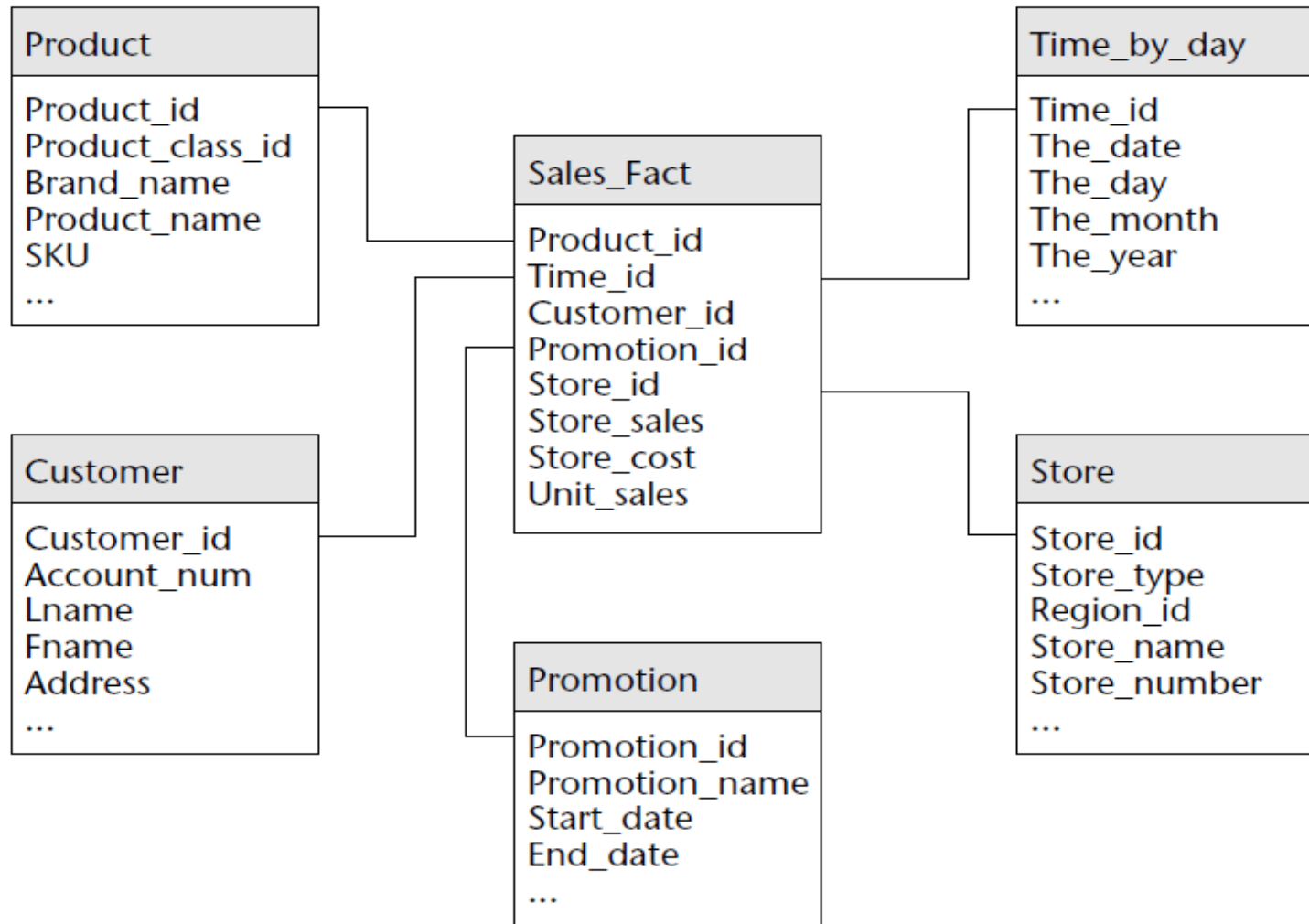
Database Schema vs. Database State

- Database State:
 - Refers to the **content** of a database at a moment in time.
- Initial Database State:
 - Refers to the database state when it is initially loaded into the system.
- Valid State:
 - A state that satisfies the structure and constraints of the database.

Database Schema vs. Database State (continued)

- Distinction
 - The **database schema** changes very infrequently.
 - The **database state** changes every time the database is updated.
- Schema is also called **intension**.
- State is also called **extension**.

Example of a Database Schema



Example of a database state

COURSE

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

SECTION

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

GRADE_REPORT

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

PREREQUISITE

Course_number	Prerequisite_number
CS3380	CS3320
CS3380	MATH2410
CS3320	CS1310

Figure 1.2

A database that stores student and course information.

Three-Schema Architecture

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

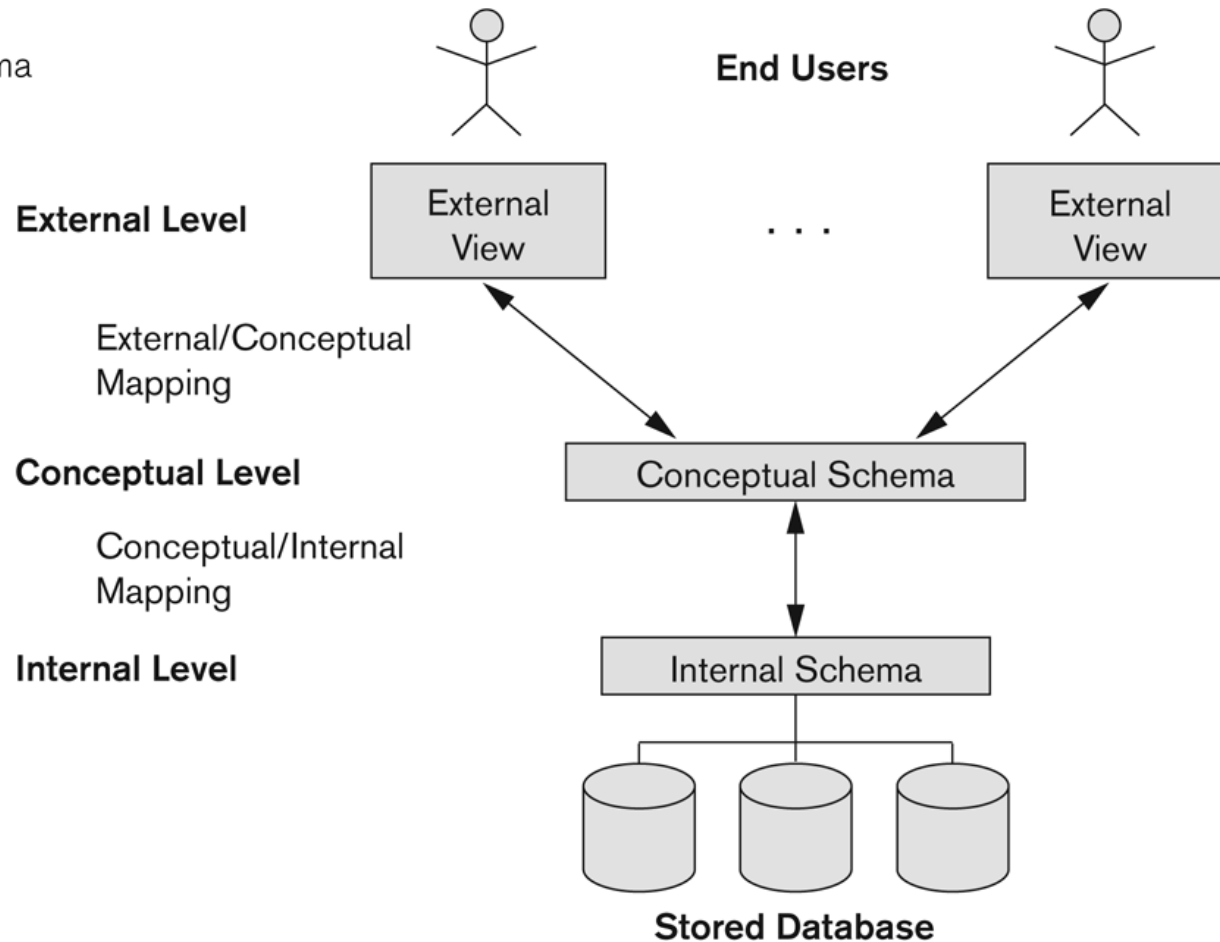
Three-Schema Architecture

- Defines DBMS schemas at **three** levels:
 - **Internal schema** at the internal level to describe physical storage structures and access paths (e.g indexes).
 - Typically uses a **physical** data model.
 - **Conceptual schema** at the conceptual level to describe the structure and constraints for the whole database for a community of users.
 - Uses a **conceptual** or an **implementation** data model.
 - **External schemas** at the external level to describe the various user views.
 - Usually uses the same data model as the conceptual schema.

The three-schema architecture

Figure 2.2

The three-schema architecture.



Three-Schema Architecture

- Mappings among schema levels are needed to transform requests and data.
 - Programs refer to an external schema, and are mapped by the DBMS to the internal schema for execution.
 - Data extracted from the internal DBMS level is reformatted to match the user's external view (e.g. formatting the results of an SQL query for display in a Web page)

Data Independence

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

Data Independence (continued)

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

DBMS Languages

- Data Definition Language (DDL)
- Data Manipulation Language (DML)
 - High-Level or Non-procedural Languages: These include the relational language SQL
 - May be used in a standalone way or may be embedded in a programming language
 - Low Level or Procedural Languages:
 - These must be embedded in a programming language

DBMS Languages

- **Data Definition Language (DDL):**
 - Used by the DBA and database designers to specify the conceptual schema of a database.
 - In many DBMSs, the DDL is also used to define internal and external schemas (views).
 - In some DBMSs, separate **storage definition language (SDL)** and **view definition language (VDL)** are used to define internal and external schemas.
 - SDL is typically realized via DBMS commands provided to the DBA and database designers

DBMS Languages

- **Data Manipulation Language (DML):**
 - Used to specify database retrievals and updates
 - DML commands (data sublanguage) can be embedded in a general-purpose programming language (host language), such as COBOL, C, C++, or Java.
 - A library of functions can also be provided to access the DBMS from a programming language
 - Alternatively, stand-alone DML commands can be applied directly (called a query language).

Types of DML

- **High Level or Non-procedural Language:**
 - For example, the SQL relational language
 - Are “set”-oriented and specify what data to retrieve rather than how to retrieve it.
 - Also called **declarative** languages.
- **Low Level or Procedural Language:**
 - Retrieve data one record-at-a-time;
 - Constructs such as looping are needed to retrieve multiple records, along with positioning pointers.

DBMS Interfaces

- Stand-alone query language interfaces
 - Example: Entering SQL queries at the DBMS interactive SQL interface (e.g. SQL*Plus in ORACLE)
- Programmer interfaces for embedding DML in programming languages
- User-friendly interfaces
 - Menu-based, forms-based, graphics-based, etc.

DBMS Programming Language Interfaces

- Programmer interfaces for embedding DML in a programming languages:
 - **Embedded Approach:** e.g. embedded SQL (for C, C++, etc.), SQLJ (for Java)
 - **Procedure Call Approach:** e.g. JDBC for Java, ODBC for other programming languages
 - **Database Programming Language Approach:** e.g. ORACLE has PL/SQL, a programming language based on SQL; language incorporates SQL and its data types as integral components

User-Friendly DBMS Interfaces

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

Other DBMS Interfaces

- Speech as Input and Output
- Web Browser as an interface
- Parametric interfaces, e.g., bank tellers using function keys.
- Interfaces for the DBA:
 - Creating user accounts, granting authorizations
 - Setting system parameters
 - Changing schemas or access paths

Database System Utilities

- To perform certain functions such as:
 - Loading data stored in files into a database. Includes data conversion tools.
 - Backing up the database periodically on tape.
 - Reorganizing database file structures.
 - Report generation utilities.
 - Performance monitoring utilities.
 - Other functions, such as sorting, user monitoring, data compression, etc.

Other Tools

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

Other Tools

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

Typical DBMS Component Modules

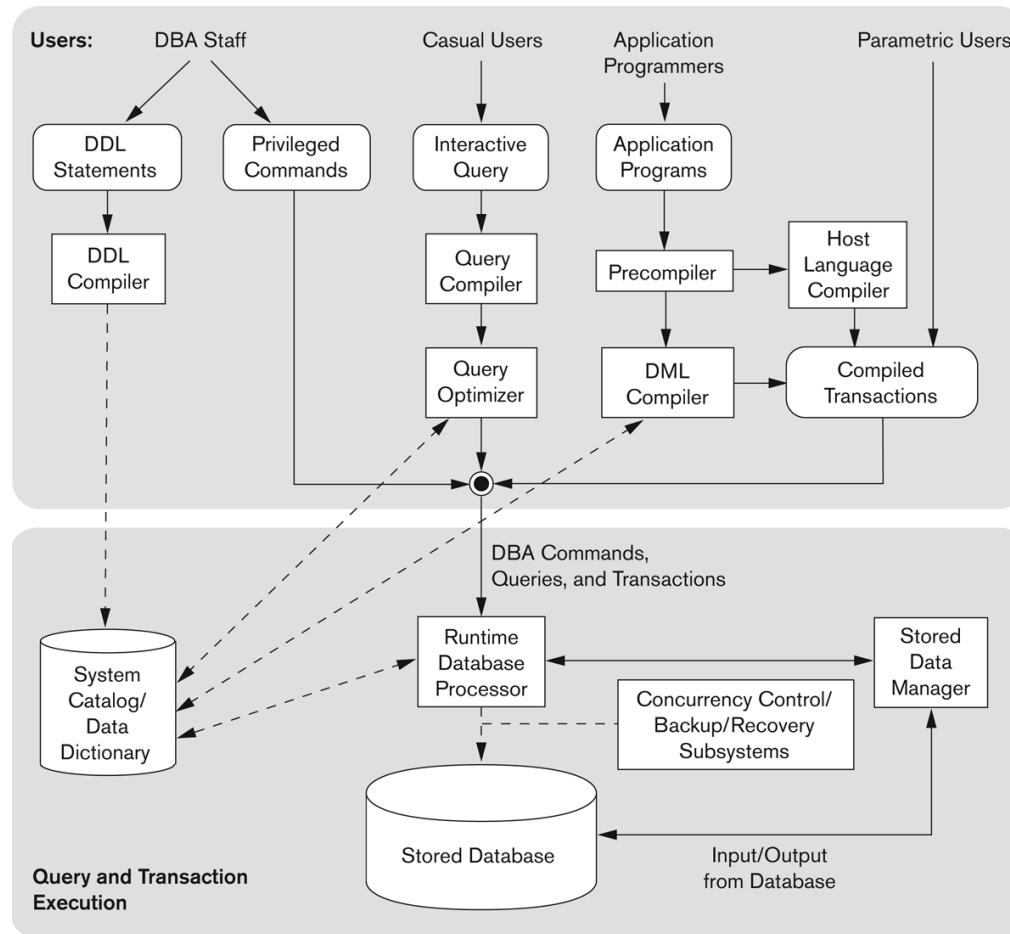


Figure 2.3

Component modules of a DBMS and their interactions.

Centralized and Client-Server DBMS Architectures

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

A Physical Centralized Architecture

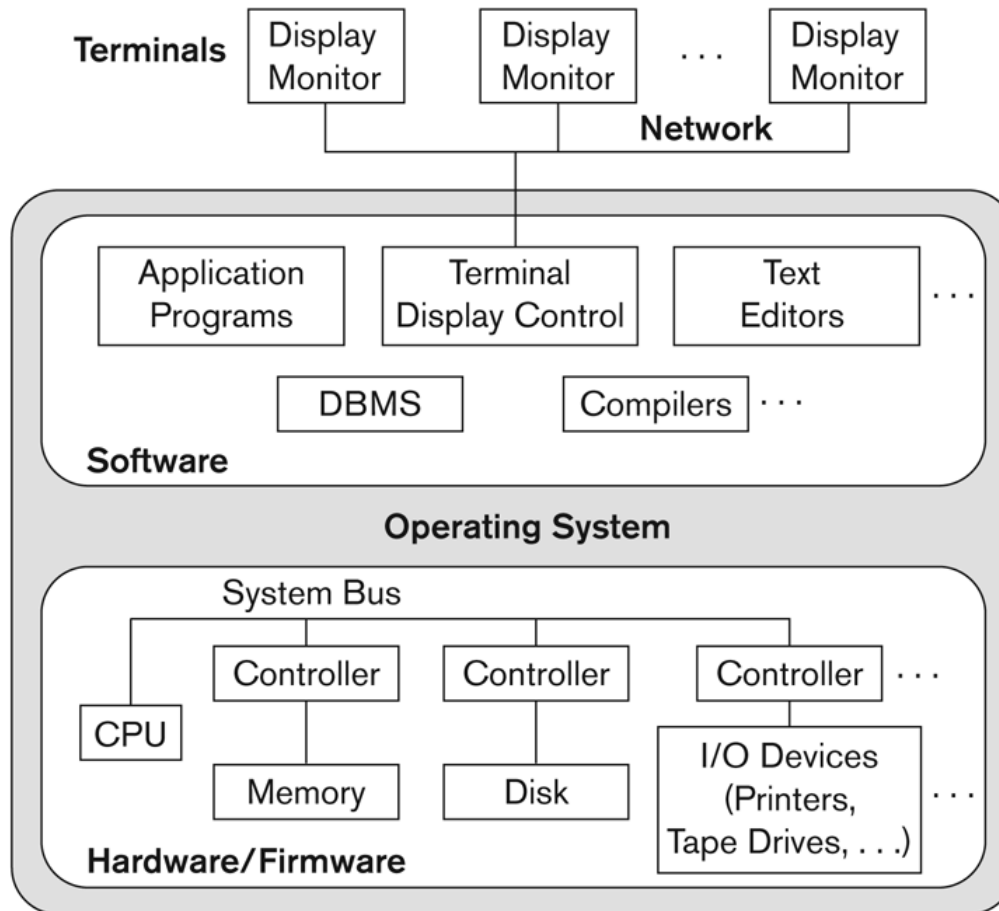


Figure 2.4
A physical centralized architecture.

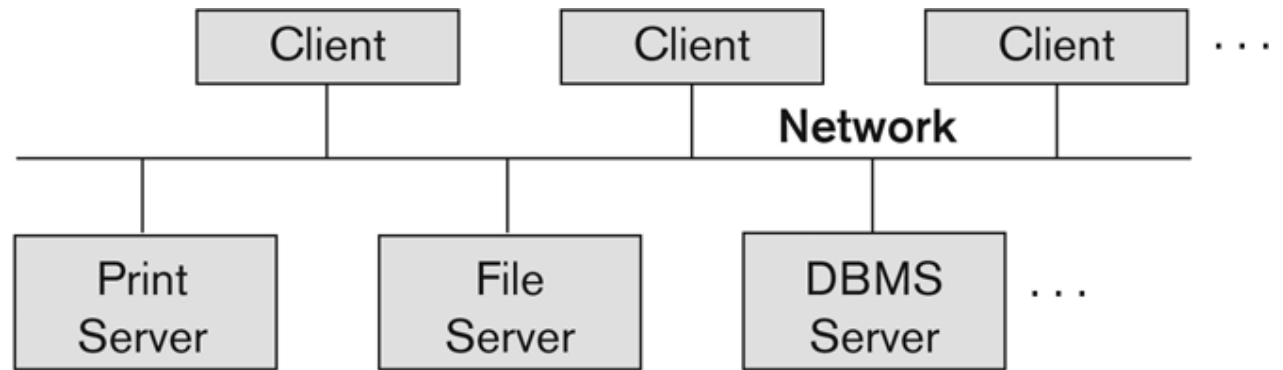
Basic 2-tier Client-Server Architectures

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

Logical two-tier client server architecture

Figure 2.5

Logical two-tier
client/server
architecture.



Clients

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

DBMS Server

- Provides database query and transaction services to the clients
- Relational DBMS servers are often called SQL servers, query servers, or transaction servers
- Applications running on clients utilize an Application Program Interface (**API**) to access server databases via standard interface such as:
 - **ODBC: Open Database Connectivity standard**
 - **JDBC: for Java programming access**
- Client and server must install appropriate client module and server module software for ODBC or JDBC
- See Chapter 9

Two Tier Client-Server Architecture

- A client program may connect to several DBMSs, sometimes called the data sources.
- In general, data sources can be files or other non-DBMS software that manages data.
- Other variations of clients are possible: e.g., in some object DBMSs, more functionality is transferred to clients including data dictionary functions, optimization and recovery across multiple servers, etc.

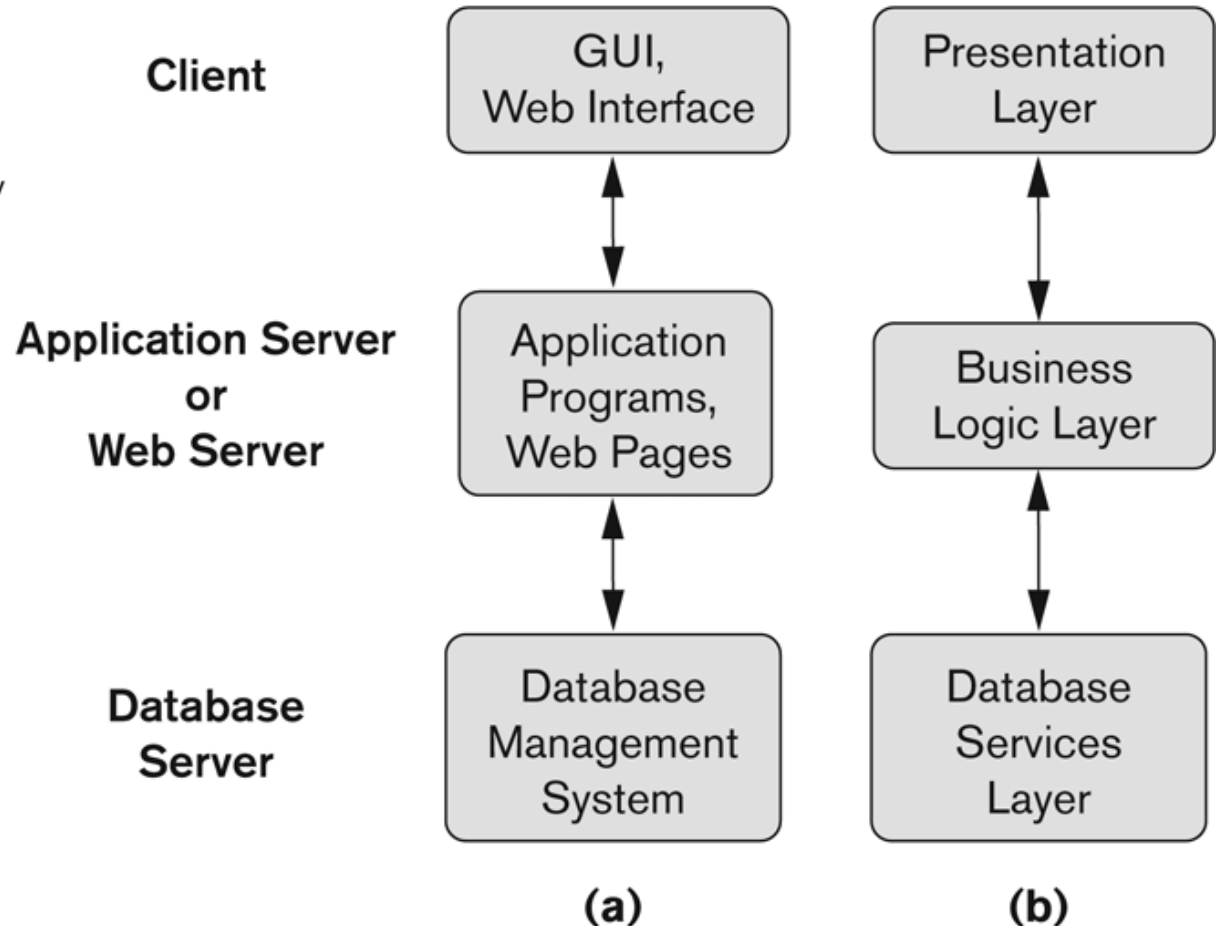
Three Tier Client-Server Architecture

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

Three-tier client-server architecture

Figure 2.7

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



Classification of DBMSs

- Based on the data model used
 - Traditional: Relational, Network, Hierarchical.
 - Emerging: Object-oriented, Object-relational.
- Other classifications
 - Single-user (typically used with personal computers) vs. multi-user (most DBMSs).
 - Centralized (uses a single computer with one database) vs. distributed (uses multiple computers, multiple databases)

Variations of Distributed DBMSs (DDBMSs)

- Homogeneous DDBMS
- Heterogeneous DDBMS
- Federated or Multi-database Systems
- Distributed Database Systems have now come to be known as client-server based database systems because:
 - They do not support a totally distributed environment, but rather a set of database servers supporting a set of clients.

Cost considerations for DBMSs

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

History of Data Models

- Network Model
- Hierarchical Model
- Relational Model
- Object-oriented Data Models
- Object-Relational Models

History of Data Models

■ Network Model:

- The first network DBMS was implemented by Honeywell in 1964-65 (IDS System).
- Adopted heavily due to the support by CODASYL (Conference on Data Systems Languages) (CODASYL - DBTG report of 1971).
- Later implemented in a large variety of systems - IDMS (Cullinet - now Computer Associates), DMS 1100 (Unisys), IMAGE (H.P. (Hewlett-Packard)), VAX - DBMS (Digital Equipment Corp., next COMPAQ, now H.P.).

Example of Network Model Schema

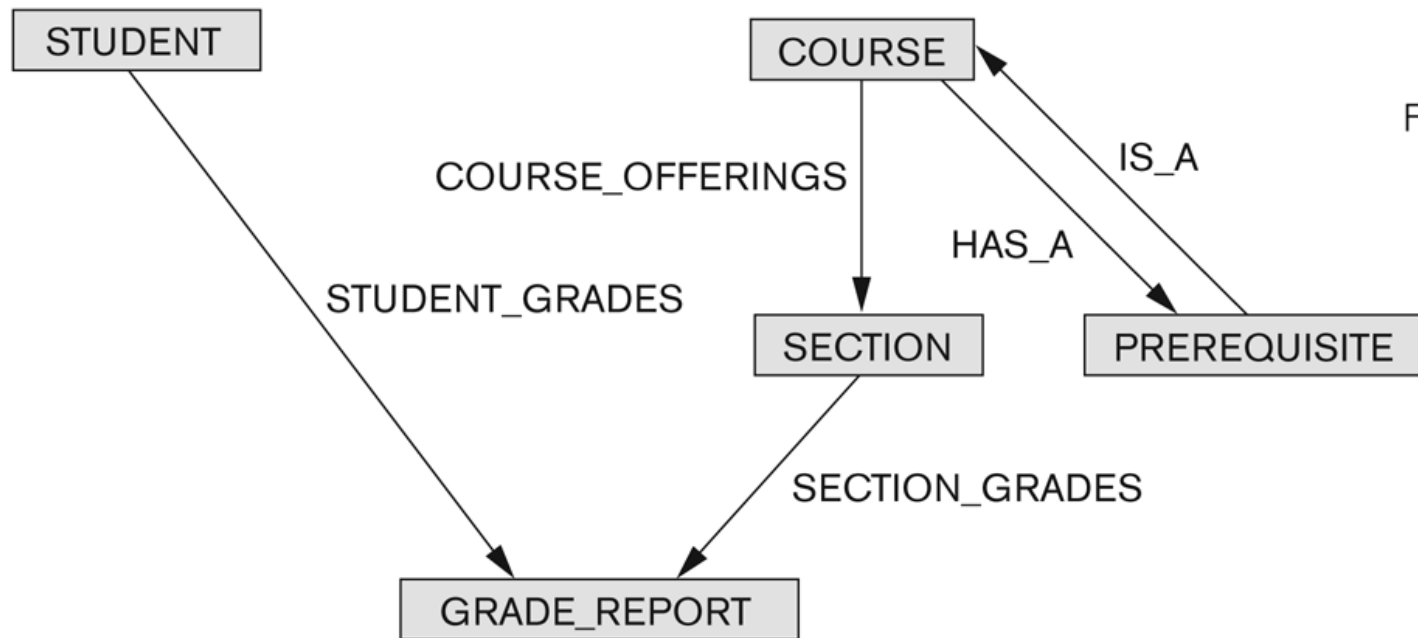


Figure 2.8

The schema of Figure 2.1 in network model notation.

Network Model

- Advantages:
 - Network Model is able to model complex relationships and represents semantics of add/delete on the relationships.
 - Can handle most situations for modeling using record types and relationship types.
 - Language is navigational; uses constructs like FIND, FIND member, FIND owner, FIND NEXT within set, GET, etc.
 - Programmers can do optimal navigation through the database.

Network Model

- Disadvantages:
 - Navigational and procedural nature of processing
 - Database contains a complex array of pointers that thread through a set of records.
 - Little scope for automated “query optimization”

History of Data Models

- **Hierarchical Data Model:**

- Initially implemented in a joint effort by IBM and North American Rockwell around 1965. Resulted in the IMS family of systems.
- IBM's IMS product had (and still has) a very large customer base worldwide
- Hierarchical model was formalized based on the IMS system
- Other systems based on this model: System 2k (SAS inc.)

Hierarchical Model

- Advantages:
 - Simple to construct and operate
 - Corresponds to a number of natural hierarchically organized domains, e.g., organization (“org”) chart
 - Language is simple:
 - Uses constructs like GET, GET UNIQUE, GET NEXT, GET NEXT WITHIN PARENT, etc.
- Disadvantages:
 - Navigational and procedural nature of processing
 - Database is visualized as a linear arrangement of records
 - Little scope for "query optimization"

History of Data Models

■ Relational Model:

- Proposed in 1970 by E.F. Codd (IBM), first commercial system in 1981-82.
- Now in several commercial products (e.g. DB2, ORACLE, MS SQL Server, SYBASE, INFORMIX).
- Several free open source implementations, e.g. MariaDB, MySQL, PostgreSQL
- Currently most dominant for developing database applications.
- SQL relational standards: SQL-89 (SQL1), SQL-92 (SQL2), SQL-99, SQL3, ...

History of Data Models

■ **Object-oriented Data Models:**

- Several models have been proposed for implementing in a database system.
- One set comprises models of persistent O-O Programming Languages such as C++ (e.g., in OBJECTSTORE or VERSANT), and Smalltalk (e.g., in GEMSTONE).
- Additionally, systems like O2, ORION (at MCC - then ITASCA), IRIS (at H.P.- used in Open OODB).
- Object Database Standard: ODMG-93, ODMG-version 2.0, ODMG-version 3.0.
- Chapters 20 and 21 describe this model.

History of Data Models

- **Object-Relational Models:**

- Most Recent Trend. Started with Informix Universal Server.
- Relational systems incorporate concepts from object databases leading to object-relational.
- Exemplified in the latest versions of Oracle-10i, DB2, and SQL Server and other DBMSs.
- Standards included in SQL-99 and expected to be enhanced in future SQL standards.
- Chapter 22 describes this model.

Data Modeling Using the Entity-Relationship (ER) Model

Chapter Outline

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

Overview of Database Design Process

- Two main activities:
 - Database design
 - Applications design
- Focus in this chapter on database design
 - To design the conceptual schema for a database application
- Applications design focuses on the programs and interfaces that access the database
 - Generally considered part of software engineering

Overview of Database Design Process

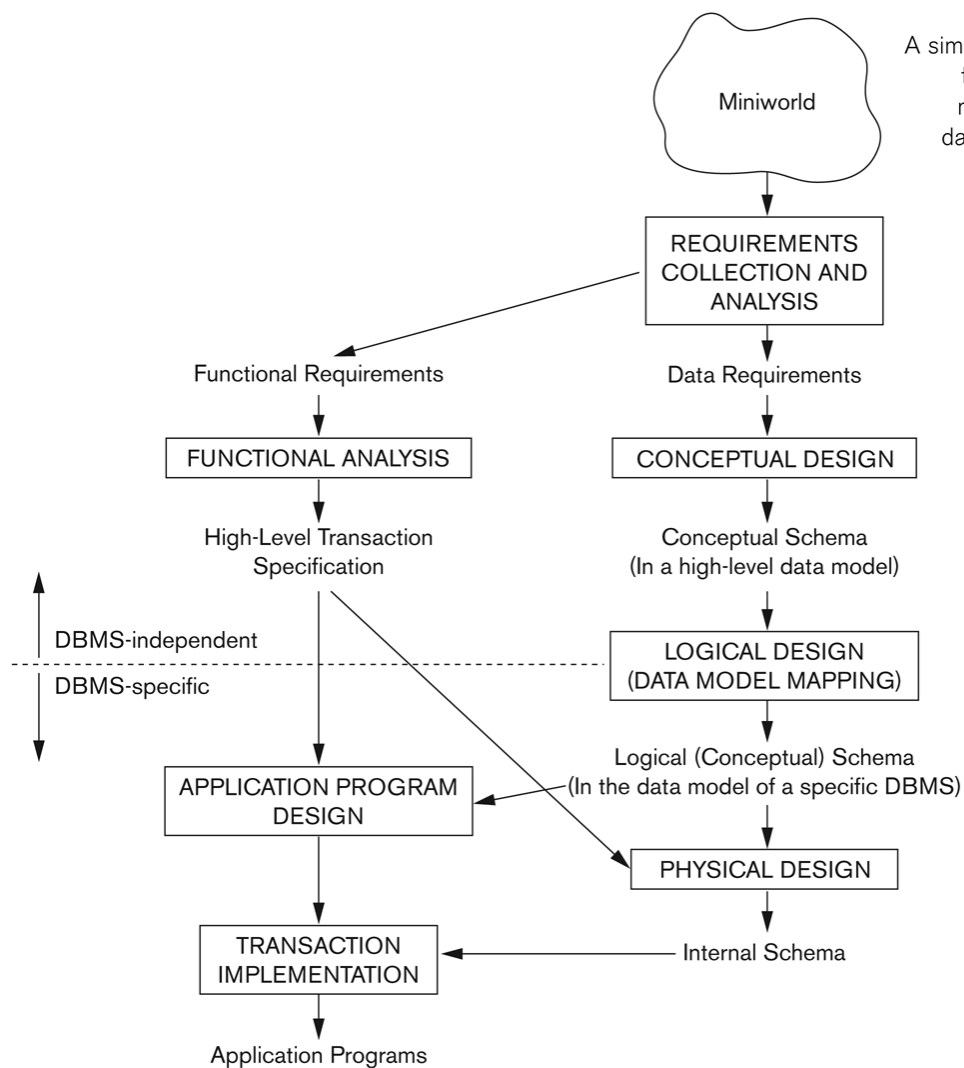


Figure 3.1

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

Example COMPANY Database

- We need to create a database schema design based on the following (simplified) **requirements** of the COMPANY Database:
 - The company is organized into DEPARTMENTS. Each department has a name, number and an employee who *manages* the department. We keep track of the start date of the department manager. A department may have several locations.
 - Each department *controls* a number of PROJECTs. Each project has a unique name, unique number and is located at a single location.

Example COMPANY Database (Contd.)

- We store each EMPLOYEE's social security number, address, salary, sex, and birthdate.
 - Each employee *works for* one department but may *work on* several projects.
 - We keep track of the number of hours per week that an employee currently works on each project.
 - We also keep track of the *direct supervisor* of each employee.
- Each employee may *have* a number of DEPENDENTS.
 - For each dependent, we keep track of their name, sex, birthdate, and relationship to the employee.

ER Model Concepts

■ Entities and Attributes

- Entities are specific objects or things in the mini-world that are represented in the database.
 - For example the EMPLOYEE John Smith, the Research DEPARTMENT, the ProductX PROJECT
- Attributes are properties used to describe an entity.
 - For example an EMPLOYEE entity may have the attributes Name, SSN, Address, Sex, BirthDate
- A specific entity will have a value for each of its attributes.
 - For example a specific employee entity may have Name='John Smith', SSN='123456789', Address ='731, Fondren, Houston, TX', Sex='M', BirthDate='09-JAN-55'
- Each attribute has a *value set* (or data type) associated with it
 - e.g. integer, string, subrange, enumerated type, ...

Types of Attributes (1)

- Simple
 - Each entity has a single atomic value for the attribute. For example, SSN or Sex.
- Composite
 - The attribute may be composed of several components. For example:
 - Address(Apt#, House#, Street, City, State, ZipCode, Country), or
 - Name(FirstName, MiddleName, LastName).
 - Composition may form a hierarchy where some components are themselves composite.
- Multi-valued
 - An entity may have multiple values for that attribute. For example, Color of a CAR or PreviousDegrees of a STUDENT.
 - Denoted as {Color} or {PreviousDegrees}.

Types of Attributes (2)

- In general, composite and multi-valued attributes may be nested arbitrarily to any number of levels, although this is rare.
 - For example, PreviousDegrees of a STUDENT is a composite multi-valued attribute denoted by {PreviousDegrees (College, Year, Degree, Field)}
 - Multiple PreviousDegrees values can exist
 - Each has four subcomponent attributes:
 - College, Year, Degree, Field

Example of a composite attribute

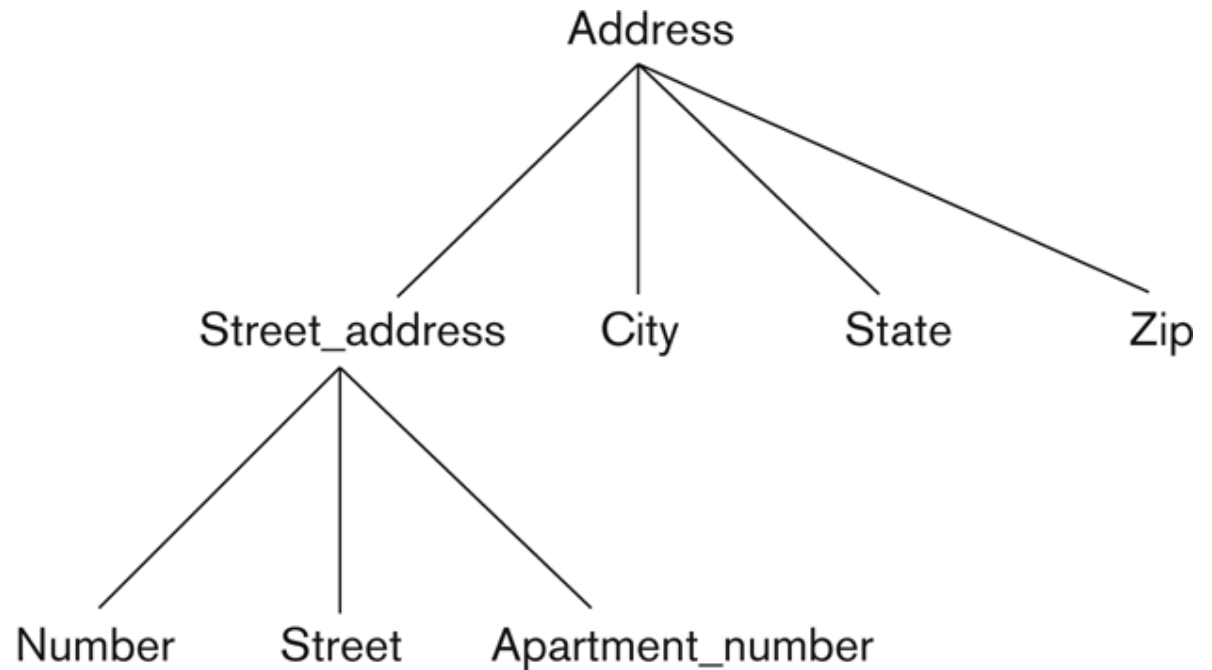


Figure 3.4

A hierarchy of composite attributes.

Entity Types and Key Attributes (1)

- Entities with the same basic attributes are grouped or typed into an entity type.
 - For example, the entity type EMPLOYEE and PROJECT.
- An attribute of an entity type for which each entity must have a unique value is called a key attribute of the entity type.
 - For example, SSN of EMPLOYEE.

Entity Types and Key Attributes (2)

- A key attribute may be composite.
 - VehicleTagNumber is a key of the CAR entity type with components (Number, State).
- An entity type may have more than one key.
 - The CAR entity type may have two keys:
 - VehicleIdentificationNumber (popularly called VIN)
 - VehicleTagNumber (Number, State), aka license plate number.
- Each key is underlined

Displaying an Entity type

- In ER diagrams, an entity type is displayed in a rectangular box
- Attributes are displayed in ovals
 - Each attribute is connected to its entity type
 - Components of a composite attribute are connected to the oval representing the composite attribute
 - Each key attribute is underlined
 - Multivalued attributes displayed in double ovals
- See CAR example on next slide

Entity Type CAR with two keys and a corresponding Entity Set

(a)

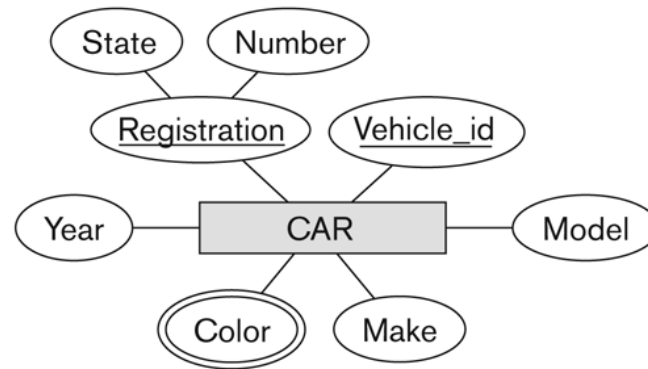


Figure 3.7

The CAR entity type with two key attributes, Registration and Vehicle_id. (a) ER diagram notation. (b) Entity set with three entities.

(b)

CAR

Registration (Number, State), Vehicle_id, Make, Model, Year, {Color}

CAR₁

((ABC 123, TEXAS), TK629, Ford Mustang, convertible, 2004 {red, black})

CAR₂

((ABC 123, NEW YORK), WP9872, Nissan Maxima, 4-door, 2005, {blue})

CAR₃

((VSY 720, TEXAS), TD729, Chrysler LeBaron, 4-door, 2002, {white, blue})

⋮

Entity Set

- Each entity type will have a collection of entities stored in the database
 - Called the **entity set**
- Previous slide shows three CAR entity instances in the entity set for CAR
- Same name (CAR) used to refer to both the entity type and the entity set
- Entity set is the current *state* of the entities of that type that are stored in the database

Initial Design of Entity Types for the COMPANY Database Schema

- Based on the requirements, we can identify four initial entity types in the COMPANY database:
 - DEPARTMENT
 - PROJECT
 - EMPLOYEE
 - DEPENDENT
- Their initial design is shown on the following slide
- The initial attributes shown are derived from the requirements description

Initial Design of Entity Types: EMPLOYEE, DEPARTMENT, PROJECT, DEPENDENT

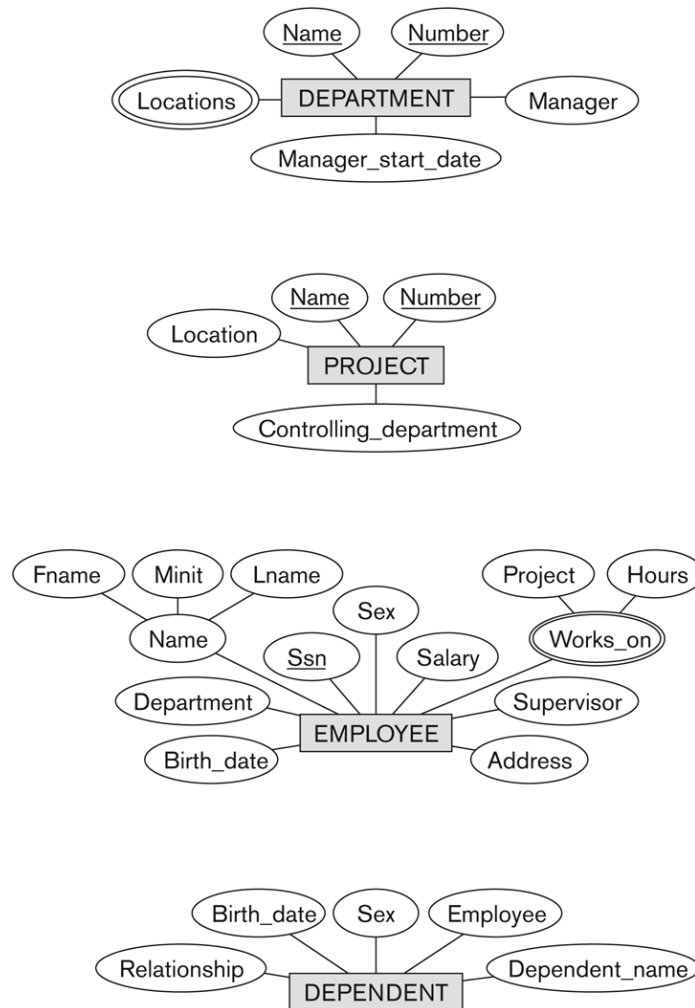


Figure 3.8
Preliminary design of entity
types for the COMPANY
database. Some of the
shown attributes will be
refined into relationships.

Refining the initial design by introducing **relationships**

- The initial design is typically not complete
- Some aspects in the requirements will be represented as **relationships**
- ER model has three main concepts:
 - Entities (and their entity types and entity sets)
 - Attributes (simple, composite, multivalued)
 - Relationships (and their relationship types and relationship sets)
- We introduce relationship concepts next

Relationships and Relationship Types (1)

- A **relationship** relates two or more distinct entities with a specific meaning.
 - For example, EMPLOYEE John Smith *works on* the ProductX PROJECT, or EMPLOYEE Franklin Wong *manages* the Research DEPARTMENT.
- Relationships of the same type are grouped or typed into a **relationship type**.
 - For example, the WORKS_ON relationship type in which EMPLOYEEs and PROJECTs participate, or the MANAGES relationship type in which EMPLOYEEs and DEPARTMENTs participate.
- The degree of a relationship type is the number of participating entity types.
 - Both MANAGES and WORKS_ON are *binary* relationships.

Relationship instances of the WORKS_FOR N:1 relationship between EMPLOYEE and DEPARTMENT

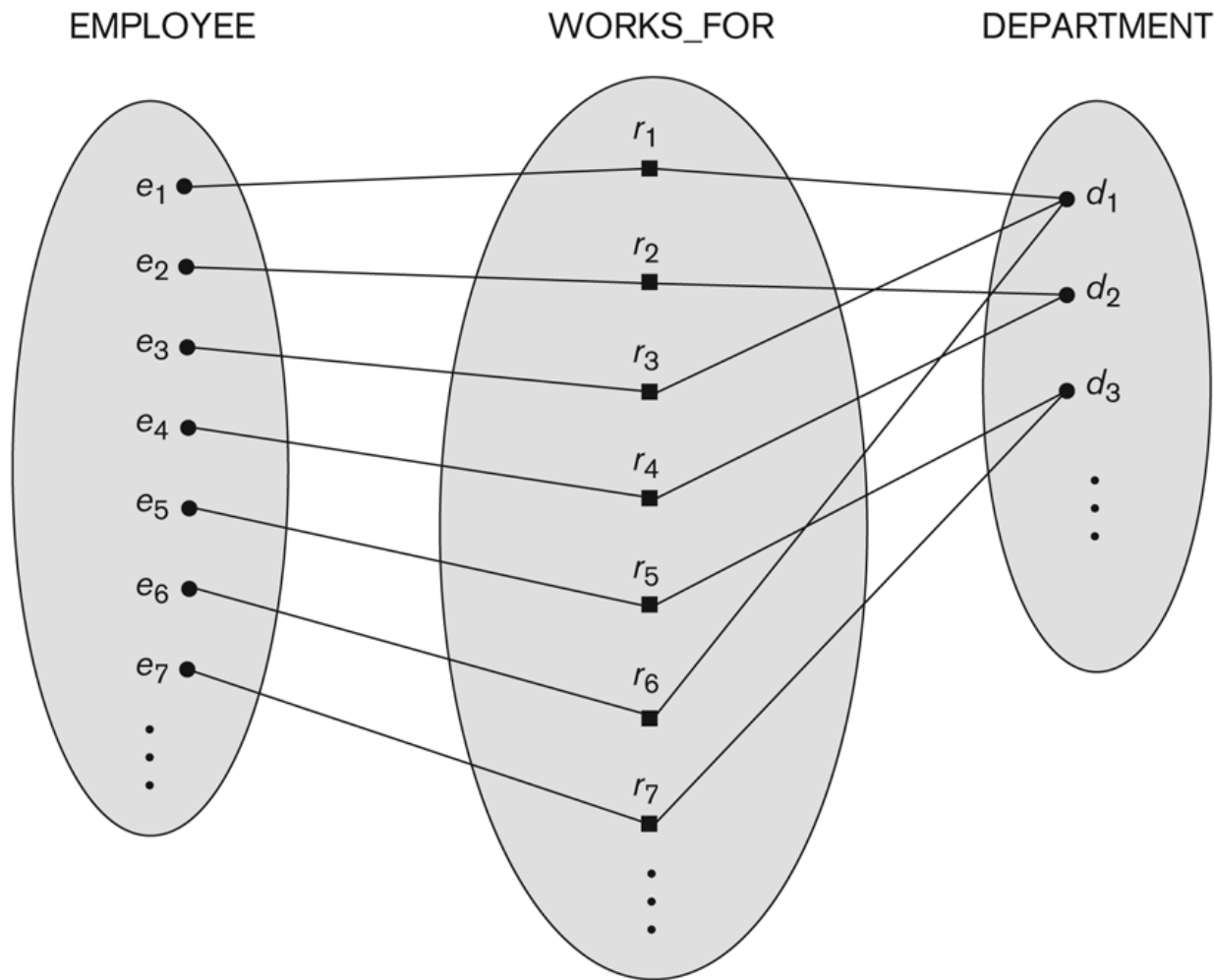


Figure 3.9

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

Relationship instances of the M:N WORKS_ON relationship between EMPLOYEE and PROJECT

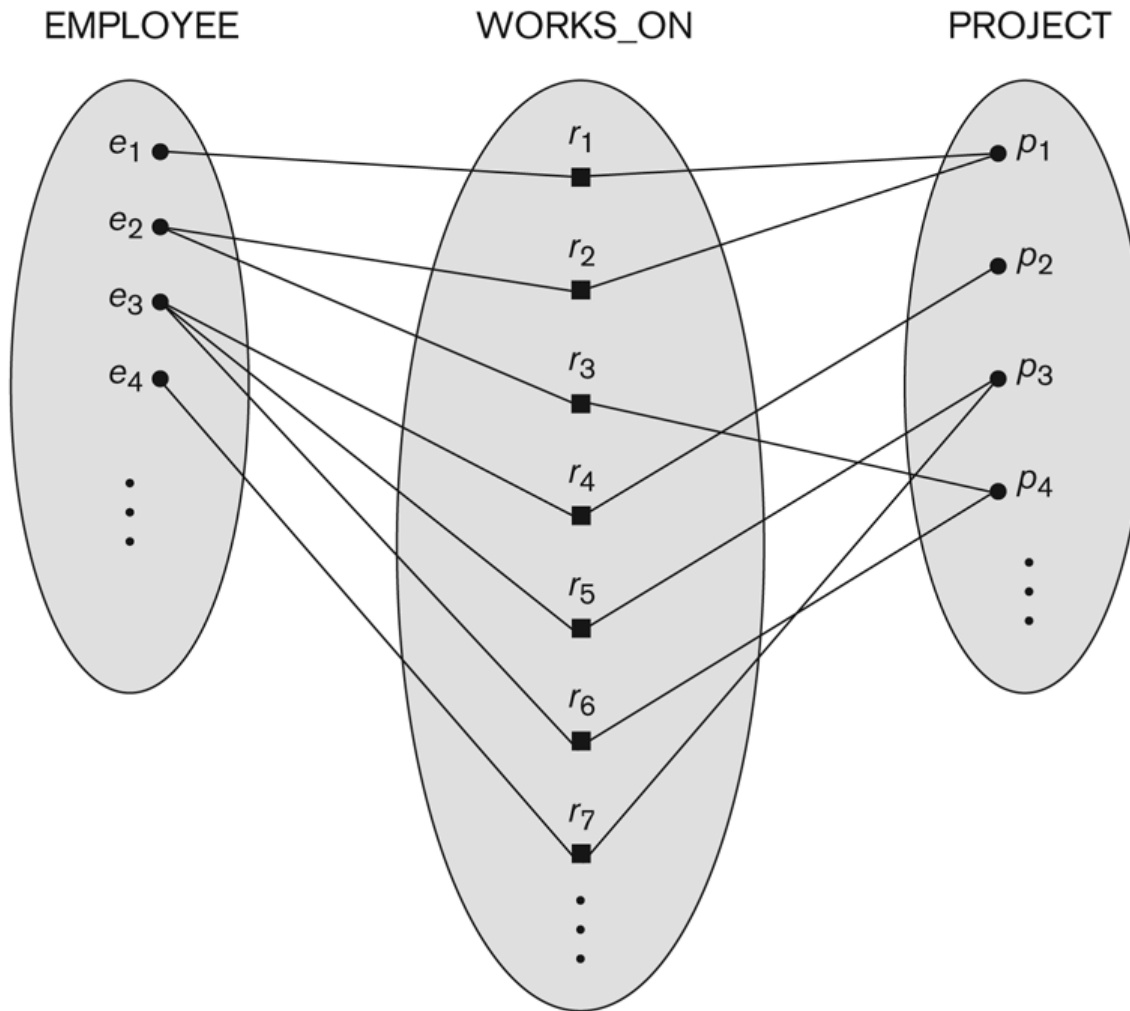


Figure 3.13
An M:N relationship,
WORKS_ON.

Relationship type vs. relationship set (1)

- Relationship Type:

- Is the schema description of a relationship
- Identifies the relationship name and the participating entity types
- Also identifies certain relationship constraints

- Relationship Set:

- The current set of relationship instances represented in the database
- The current *state* of a relationship type

Relationship type vs. relationship set (2)

- Previous figures displayed the relationship sets
- Each instance in the set relates individual participating entities – one from each participating entity type
- In ER diagrams, we represent the *relationship type* as follows:
 - Diamond-shaped box is used to display a relationship type
 - Connected to the participating entity types via straight lines

Refining the COMPANY database schema by introducing relationships

- By examining the requirements, six relationship types are identified
- All are *binary* relationships(degree 2)
- Listed below with their participating entity types:
 - WORKS_FOR (between EMPLOYEE, DEPARTMENT)
 - MANAGES (also between EMPLOYEE, DEPARTMENT)
 - CONTROLS (between DEPARTMENT, PROJECT)
 - WORKS_ON (between EMPLOYEE, PROJECT)
 - SUPERVISION (between EMPLOYEE (as subordinate), EMPLOYEE (as supervisor))
 - DEPENDENTS_OF (between EMPLOYEE, DEPENDENT)

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

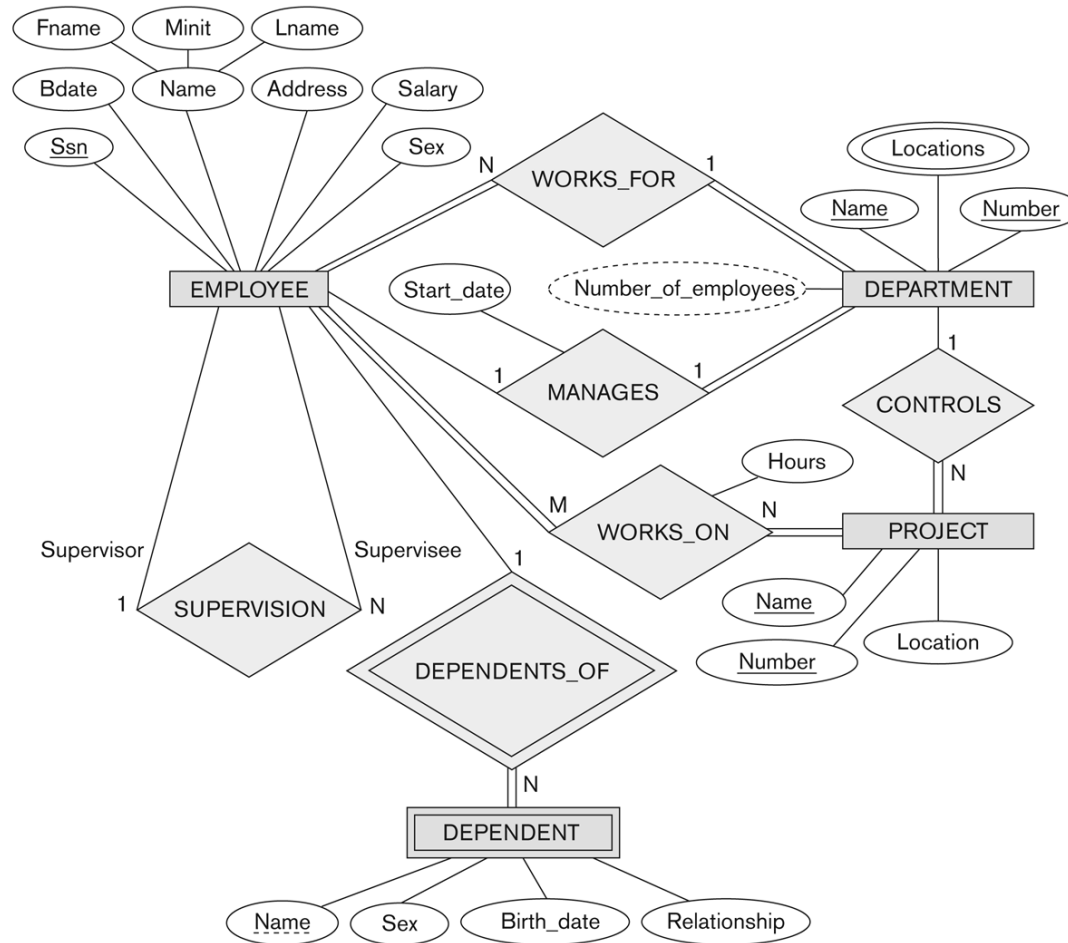


Figure 3.2

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

Discussion on Relationship Types

- In the refined design, some attributes from the initial entity types are refined into relationships:
 - Manager of DEPARTMENT -> MANAGES
 - Works_on of EMPLOYEE -> WORKS_ON
 - Department of EMPLOYEE -> WORKS_FOR
 - etc
- In general, more than one relationship type can exist between the same participating entity types
 - MANAGES and WORKS_FOR are distinct relationship types between EMPLOYEE and DEPARTMENT
 - Different meanings and different relationship instances.

Recursive Relationship Type

- An relationship type whose with the same participating entity type in **distinct roles**
- Example: the SUPERVISION relationship
- EMPLOYEE participates twice in two distinct roles:
 - supervisor (or boss) role
 - supervisee (or subordinate) role
- Each relationship instance relates two distinct EMPLOYEE entities:
 - One employee in *supervisor* role
 - One employee in *supervisee* role

Weak Entity Types

- An entity that does not have a key attribute
- A weak entity must participate in an identifying relationship type with an owner or identifying entity type
- Entities are identified by the combination of:
 - A partial key of the weak entity type
 - The particular entity they are related to in the identifying entity type
- **Example:**
 - A DEPENDENT entity is identified by the dependent's first name, *and* the specific EMPLOYEE with whom the dependent is related
 - Name of DEPENDENT is the *partial key*
 - DEPENDENT is a *weak entity type*
 - EMPLOYEE is its identifying entity type via the identifying relationship type DEPENDENT_OF

Constraints on Relationships

- Constraints on Relationship Types
 - (Also known as ratio constraints)
 - Cardinality Ratio (specifies *maximum* participation)
 - One-to-one (1:1)
 - One-to-many (1:N) or Many-to-one (N:1)
 - Many-to-many (M:N)
 - Existence Dependency Constraint (specifies *minimum* participation) (also called participation constraint)
 - zero (optional participation, not existence-dependent)
 - one or more (mandatory participation, existence-dependent)

Many-to-one (N:1) Relationship

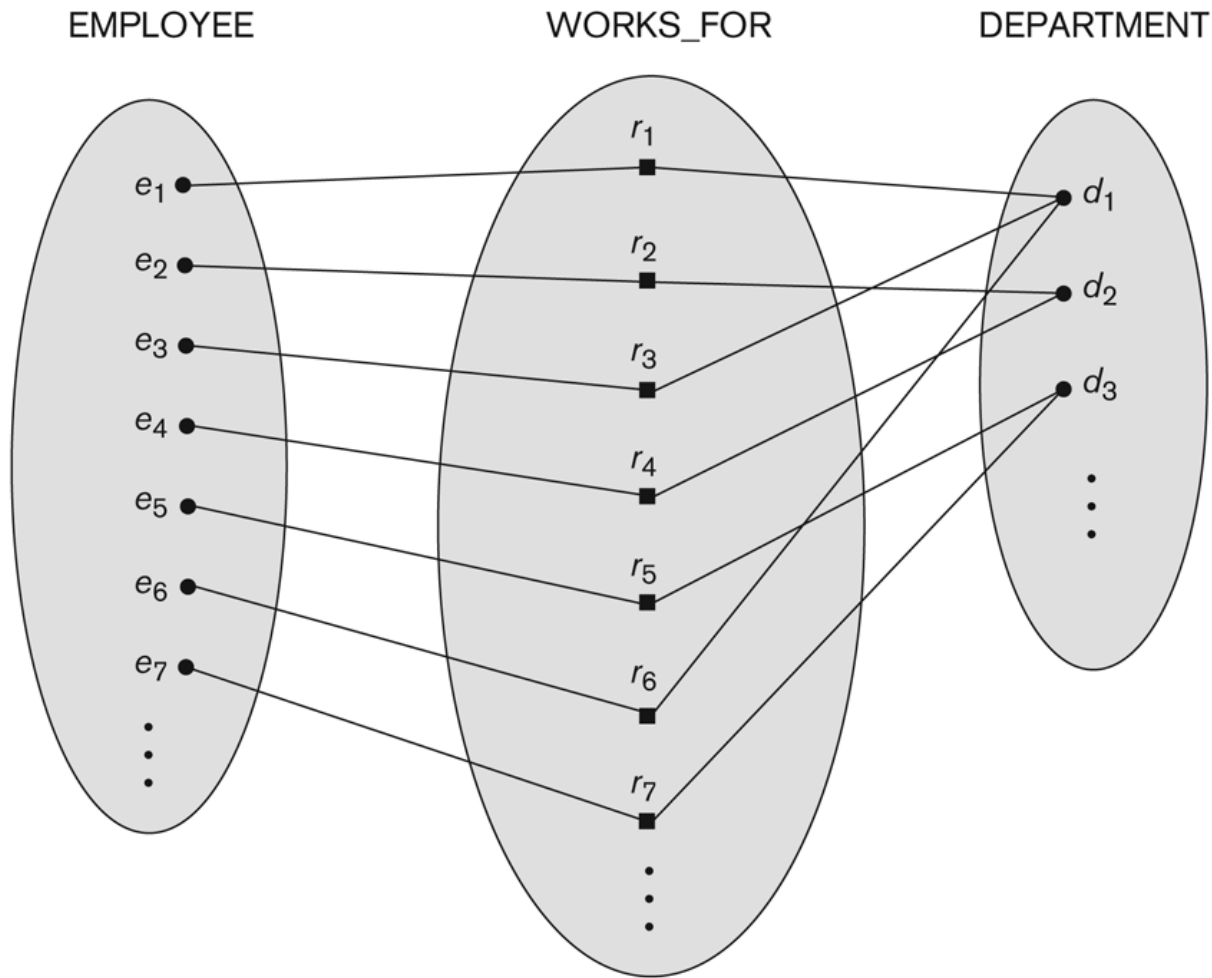


Figure 3.9

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

Many-to-many (M:N) Relationship

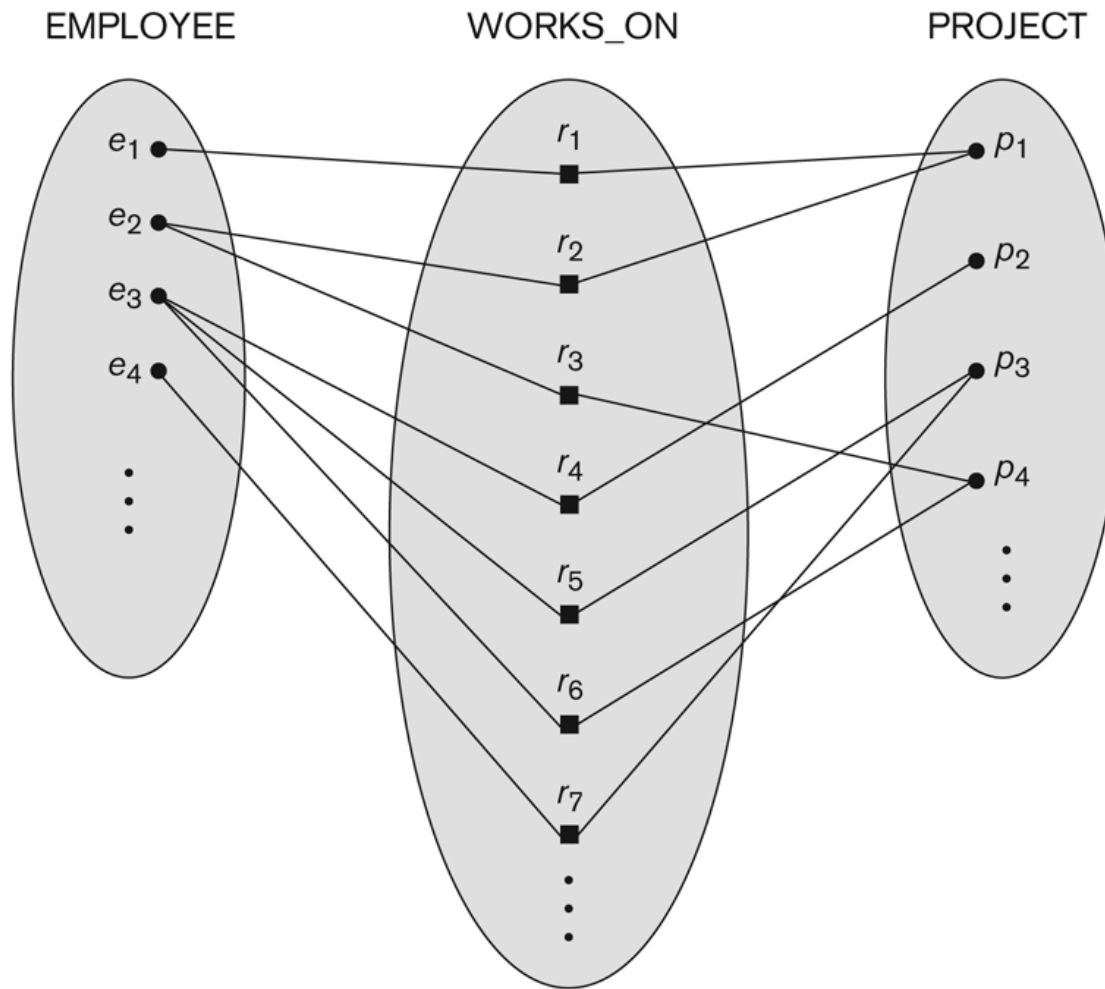


Figure 3.13
An M:N relationship,
WORKS_ON.

Displaying a recursive relationship

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

A Recursive Relationship Supervision`

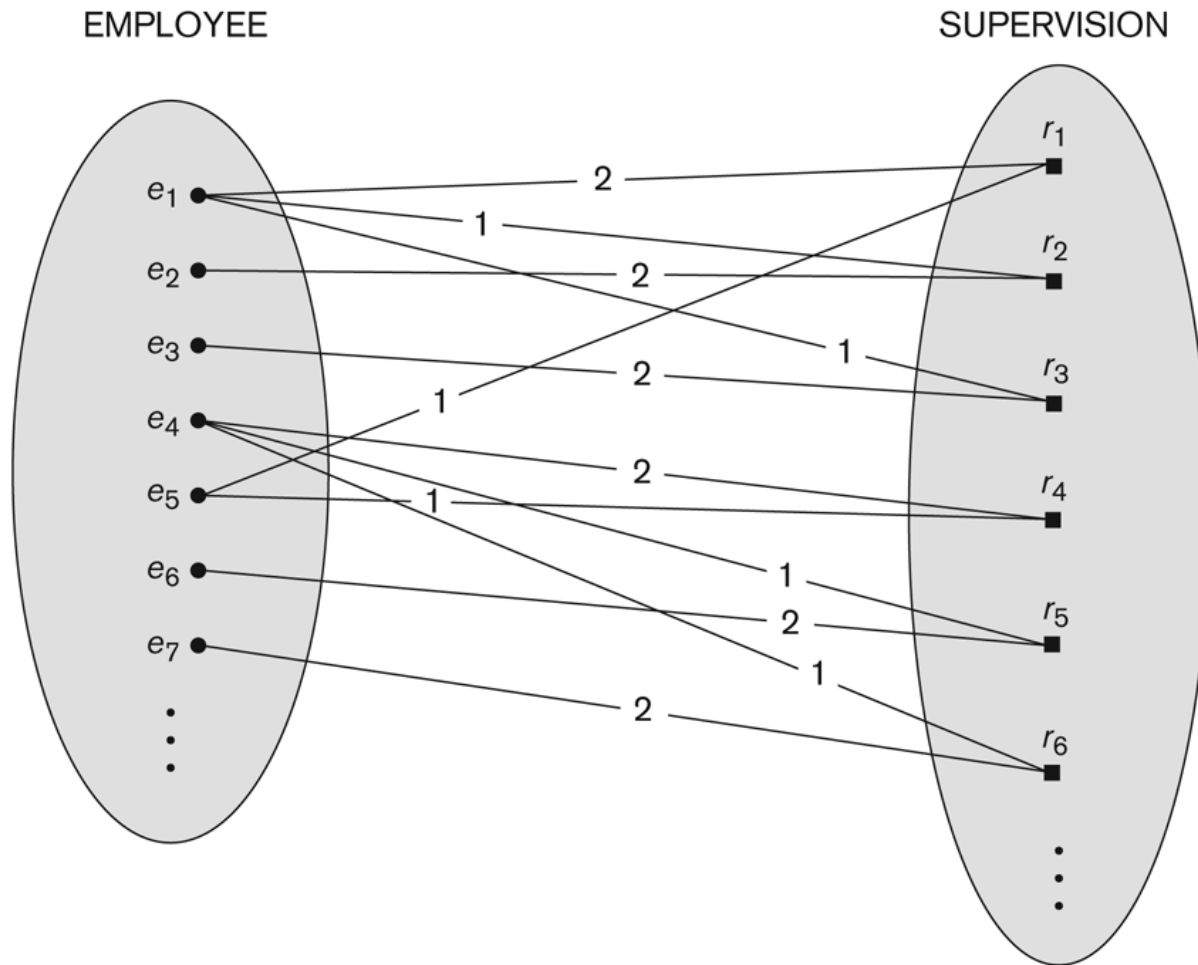


Figure 3.11

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

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

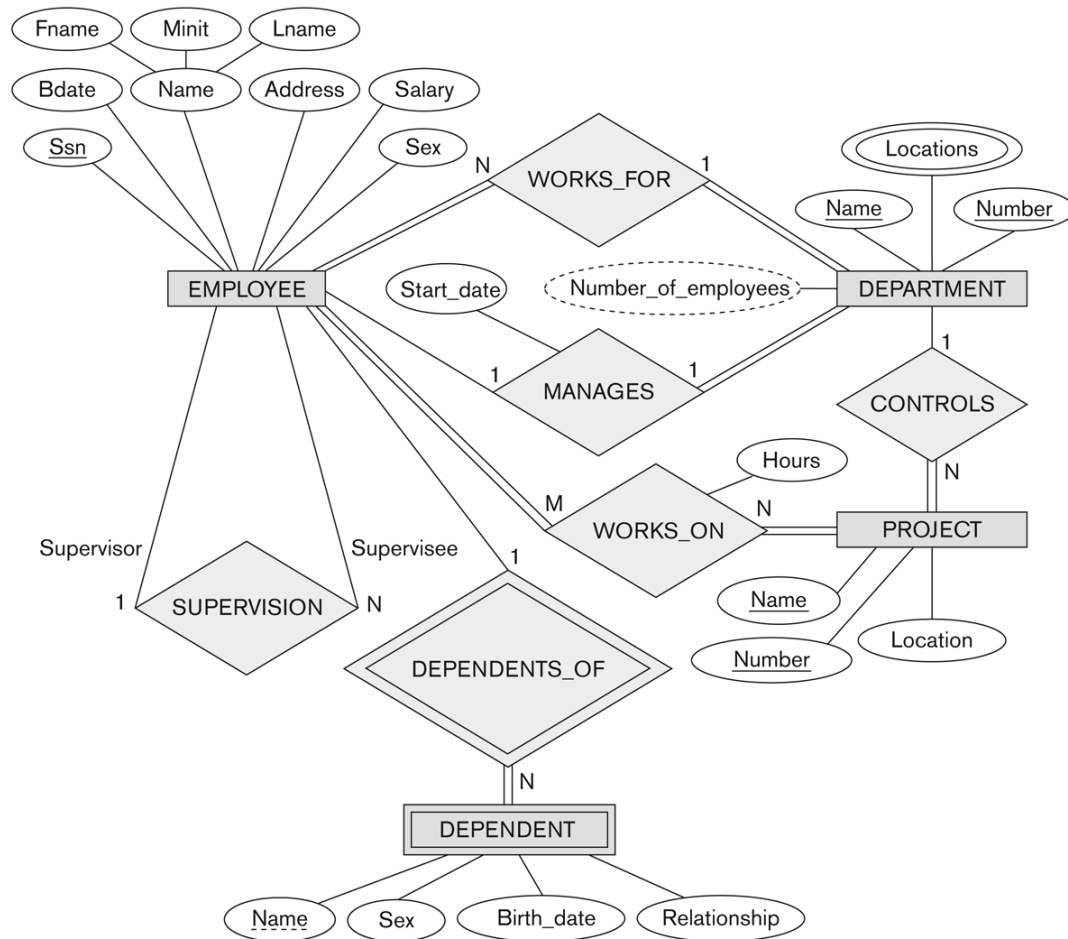


Figure 3.2

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

Attributes of Relationship types

- A relationship type can have attributes:
 - For example, HoursPerWeek of WORKS_ON
 - Its value for each relationship instance describes the number of hours per week that an EMPLOYEE works on a PROJECT.
 - A value of HoursPerWeek depends on a particular (employee, project) combination
- Most relationship attributes are used with M:N relationships
 - In 1:N relationships, they can be transferred to the entity type on the N-side of the relationship

Example Attribute of a Relationship Type: Hours of WORKS_ON

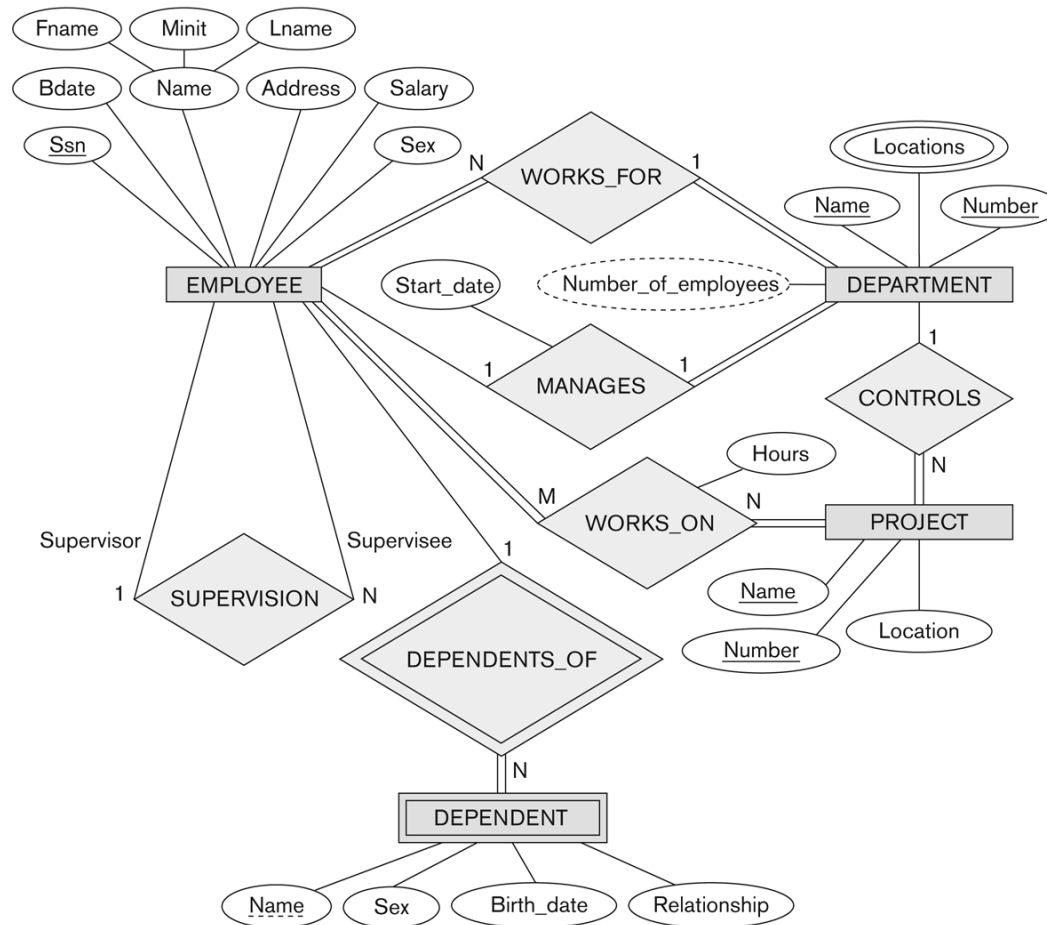


Figure 3.2

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

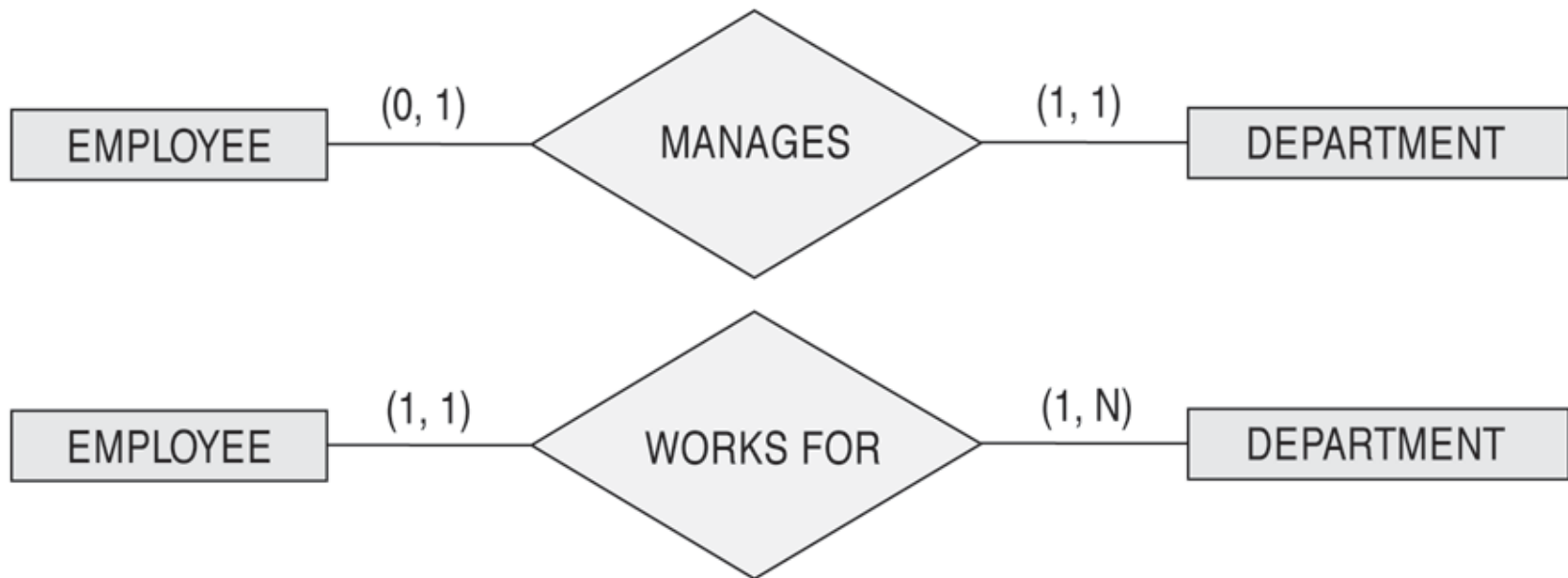
Notation for Constraints on Relationships

- Cardinality ratio (of a binary relationship): 1:1, 1:N, N:1, or M:N
 - Shown by placing appropriate numbers on the relationship edges.
- Participation constraint (on each participating entity type): total (called existence dependency) or partial.
 - Total shown by double line, partial by single line.
- NOTE: These are easy to specify for Binary Relationship Types.

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

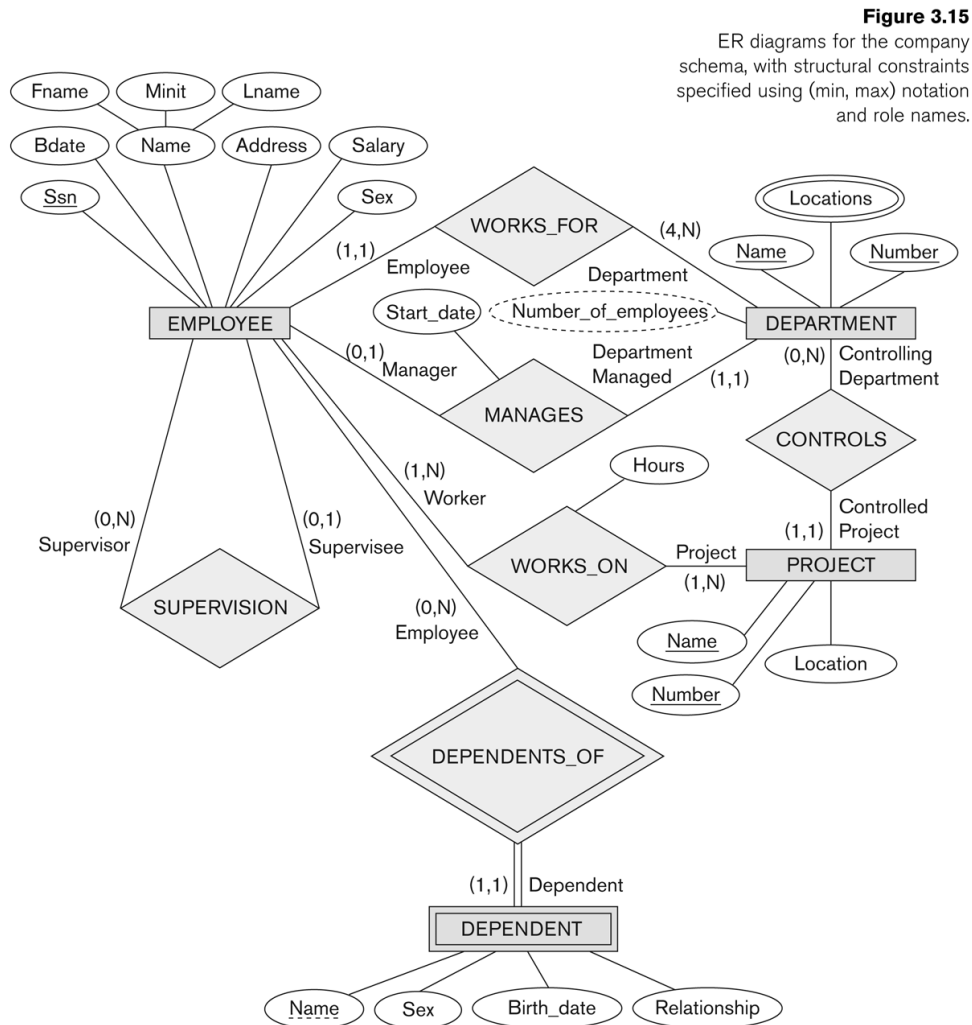
- Specified on each participation of an entity type E in a relationship type R
- Specifies that each entity e in E participates in at least *min* and at most *max* relationship instances in R
- Default(no constraint): min=0, max=n (signifying no limit)
- Must have min max, min 0, max 1
- Derived from the knowledge of mini-world constraints
- Examples:
 - A department has exactly one manager and an employee can manage at most one department.
 - Specify (0,1) for participation of EMPLOYEE in MANAGES
 - Specify (1,1) for participation of DEPARTMENT in MANAGES
 - An employee can work for exactly one department but a department can have any number of employees.
 - Specify (1,1) for participation of EMPLOYEE in WORKS_FOR
 - Specify (0,n) for participation of DEPARTMENT in WORKS_FOR

The (min,max) notation for relationship constraints



Read the min,max numbers next to the entity type and looking **away from** the entity type

COMPANY ER Schema Diagram using (min, max) notation


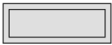







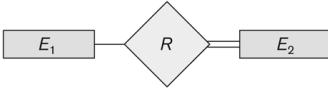
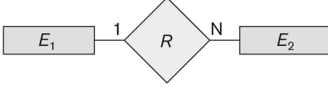
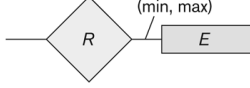


Alternative diagrammatic notation

- ER diagrams is one popular example for displaying database schemas
- Many other notations exist in the literature and in various database design and modeling tools
- Appendix A illustrates some of the alternative notations that have been used
- UML class diagrams is representative of another way of displaying ER concepts that is used in several commercial design tools

Summary of notation for ER diagrams

Figure 3.14
Summary of the
notation for ER
diagrams.

Symbol	Meaning
	Entity
	Weak Entity
	Relationship
	Identifying Relationship
	Attribute
	Key Attribute
	Multivalued Attribute
	Composite Attribute
	Derived Attribute
	Total Participation of E_2 in R
	Cardinality Ratio 1: N for $E_1:E_2$ in R
	Structural Constraint (min, max) on Participation of E in R

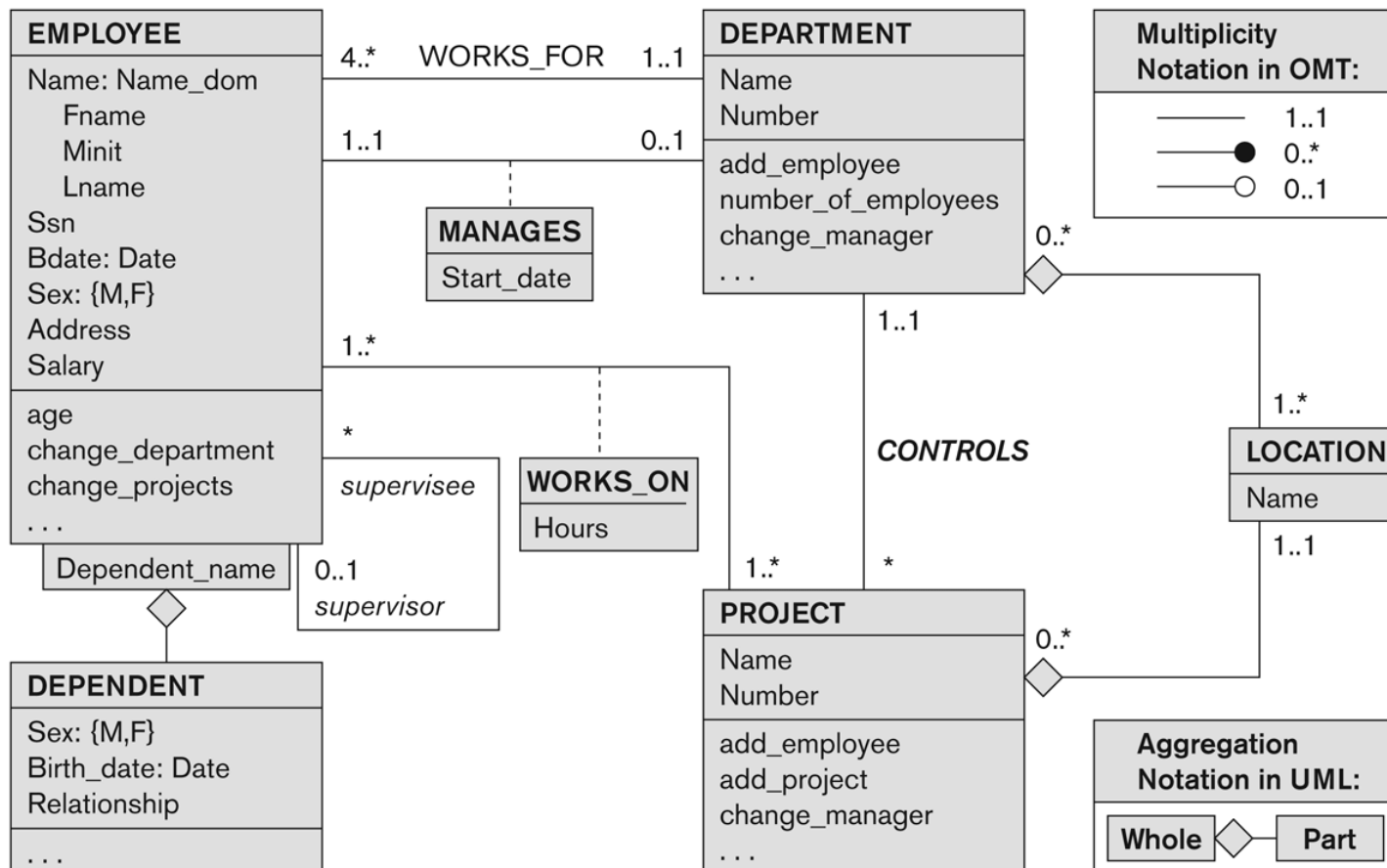
UML class diagrams

- Represent classes (similar to entity types) as large rounded boxes with three sections:
 - Top section includes entity type (class) name
 - Second section includes attributes
 - Third section includes class operations (operations are not in basic ER model)
- Relationships (called associations) represented as lines connecting the classes
 - Other UML terminology also differs from ER terminology
- Used in database design and object-oriented software design
- UML has many other types of diagrams for software design (see Chapter 12)

UML class diagram for COMPANY database schema

Figure 3.16

The COMPANY conceptual schema in UML class diagram notation.



Other alternative diagrammatic notations

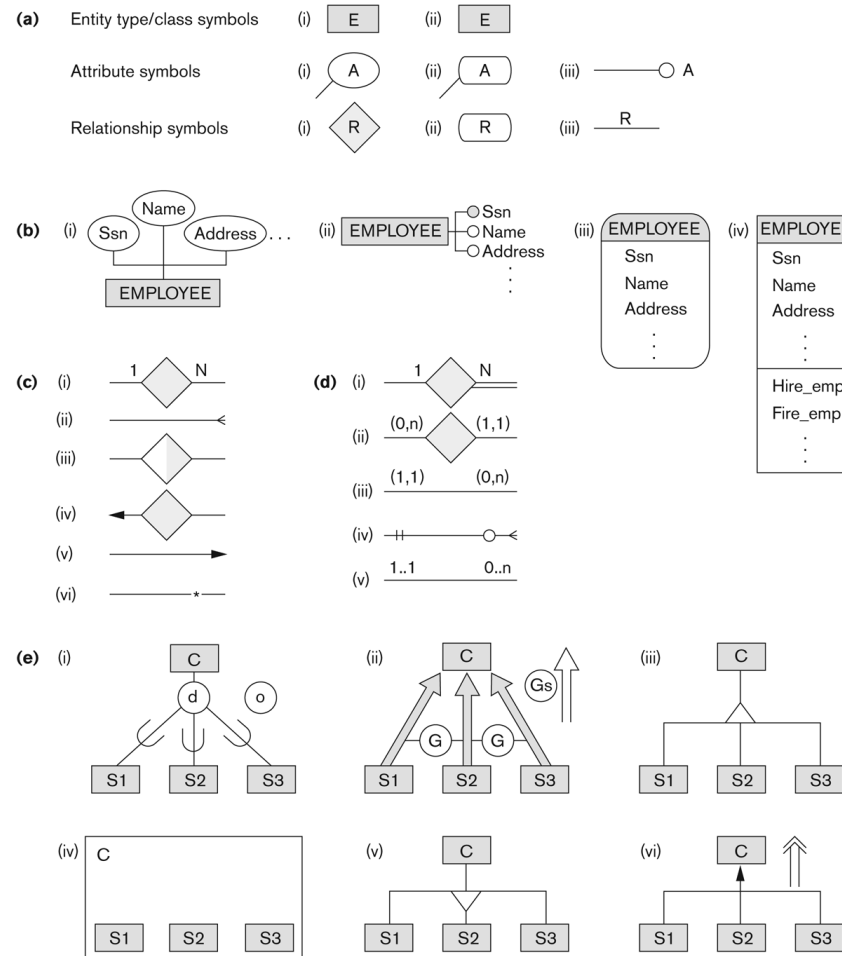


Figure A.1

Alternative notations. (a) Symbols for entity type/class, attribute, and relationship. (b) Displaying attributes. (c) Displaying cardinality ratios. (d) Various (min, max) notations. (e) Notations for displaying specialization/generalization.

Relationships of Higher Degree

- Relationship types of degree 2 are called binary
- Relationship types of degree 3 are called ternary and of degree n are called n -ary
- In general, an n -ary relationship is not equivalent to n binary relationships
- Constraints are harder to specify for higher-degree relationships ($n > 2$) than for binary relationships

Discussion of n-ary relationships ($n > 2$)

- In general, 3 binary relationships can represent different information than a single ternary relationship (see Figure 3.17a and b on next slide)
- If needed, the binary and n-ary relationships can all be included in the schema design (see Figure 3.17a and b, where all relationships convey different meanings)
- In some cases, a ternary relationship can be represented as a weak entity if the data model allows a weak entity type to have multiple identifying relationships (and hence multiple owner entity types) (see Figure 3.17c)

Example of a ternary relationship

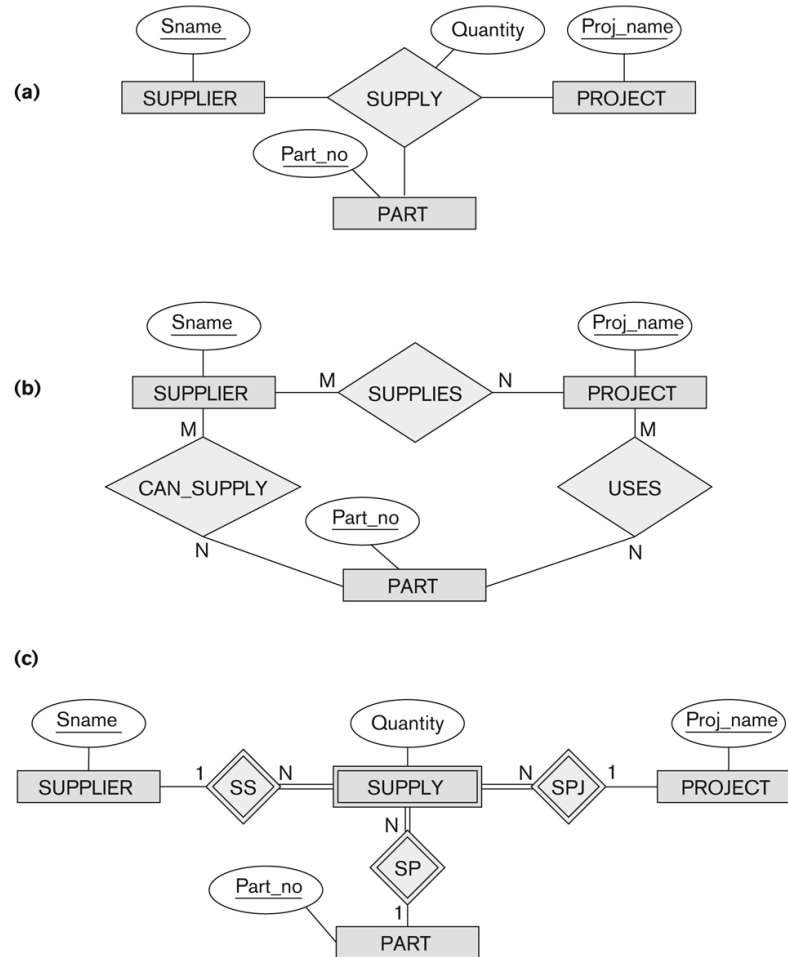


Figure 3.17

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

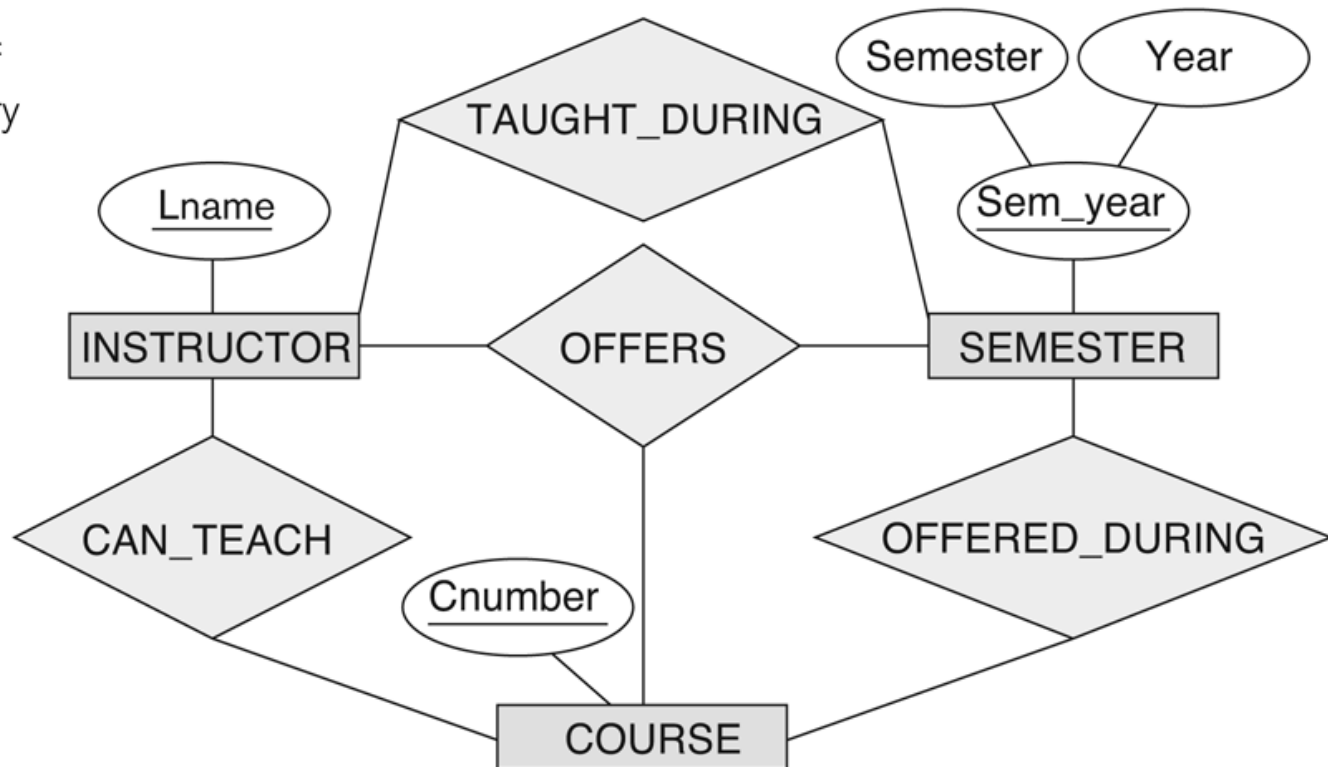
Discussion of n-ary relationships ($n > 2$)

- If a particular binary relationship can be derived from a higher-degree relationship at all times, then it is redundant
- For example, the TAUGHT_DURING binary relationship in Figure 3.18 (see next slide) can be derived from the ternary relationship OFFERS (based on the meaning of the relationships)

Another example of a ternary relationship

Figure 3.18

Another example of ternary versus binary relationship types.



Displaying constraints on higher-degree relationships

- The (min, max) constraints can be displayed on the edges
 - however, they do not fully describe the constraints
- Displaying a 1, M, or N indicates additional constraints
 - An M or N indicates no constraint
 - A 1 indicates that an entity can participate in at most one relationship instance *that has a particular combination of the other participating entities*
- In general, both (min, max) and 1, M, or N are needed to describe fully the constraints

Data Modeling Tools

- A number of popular tools that cover conceptual modeling and mapping into relational schema design.
 - Examples: ERWin, S- Designer (Enterprise Application Suite), ER- Studio, etc.
 - POSITIVES:
 - Serves as documentation of application requirements, easy user interface - mostly graphics editor support
 - NEGATIVES:
 - Most tools lack a proper distinct notation for relationships with relationship attributes
 - Mostly represent a relational design in a diagrammatic form rather than a conceptual ER-based design
- (See Chapter 12 for details)

Some of the Currently Available Automated Database Design Tools

COMPANY	TOOL	FUNCTIONALITY
Embarcadero Technologies	ER Studio	Database Modeling in ER and IDEF1X
	DB Artisan	Database administration, space and security management
Oracle	Developer 2000/Designer 2000	Database modeling, application development
Popkin Software	System Architect 2001	Data modeling, object modeling, process modeling, structured analysis/design
Platinum (Computer Associates)	Enterprise Modeling Suite: Erwin, BPWin, Paradigm Plus	Data, process, and business component modeling
Persistence Inc.	Pwertier	Mapping from O-O to relational model
Rational (IBM)	Rational Rose	UML Modeling & application generation in C++/JAVA
Resolution Ltd.	Xcase	Conceptual modeling up to code maintenance
Sybase	Enterprise Application Suite	Data modeling, business logic modeling
Visio	Visio Enterprise	Data modeling, design/reengineering Visual Basic/C++

Extended Entity-Relationship (EER) Model (in next chapter)

- The entity relationship model in its original form did not support the specialization and generalization abstractions
- Next chapter illustrates how the ER model can be extended with
 - Type-subtype and set-subset relationships
 - Specialization/Generalization Hierarchies
 - Notation to display them in EER diagrams

Chapter Summary

- ER Model Concepts: Entities, attributes, relationships
- Constraints in the ER model
- Using ER in step-by-step conceptual schema design for the COMPANY database
- ER Diagrams - Notation
- Alternative Notations – UML class diagrams, others

The Relational Data Model and Relational Database Constraints

Chapter Outline

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

Relational Model Concepts

- The relational Model of Data is based on the concept of a **Relation**
 - The strength of the relational approach to data management comes from the formal foundation provided by the theory of relations
- The essentials of the **formal relational model** are reviewed
- In **practice**, there is a **standard model** based on SQL (will be discussed later)
- **Note:** There are several important differences between the **formal** model and the **practical** model, as we shall see

Relational Model Concepts

- A Relation is a mathematical concept based on the ideas of sets
- The model was first proposed by Dr E. F. Codd of IBM Research in 1970 in the following paper:
 - **"A Relational Model for Large Shared Data Banks"**,
Communications of the ACM, June 1970
- The above paper caused a major revolution in the field of database management and earned Dr. Codd the coveted ACM Turing Award

Informal Definitions

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

Example of a Relation

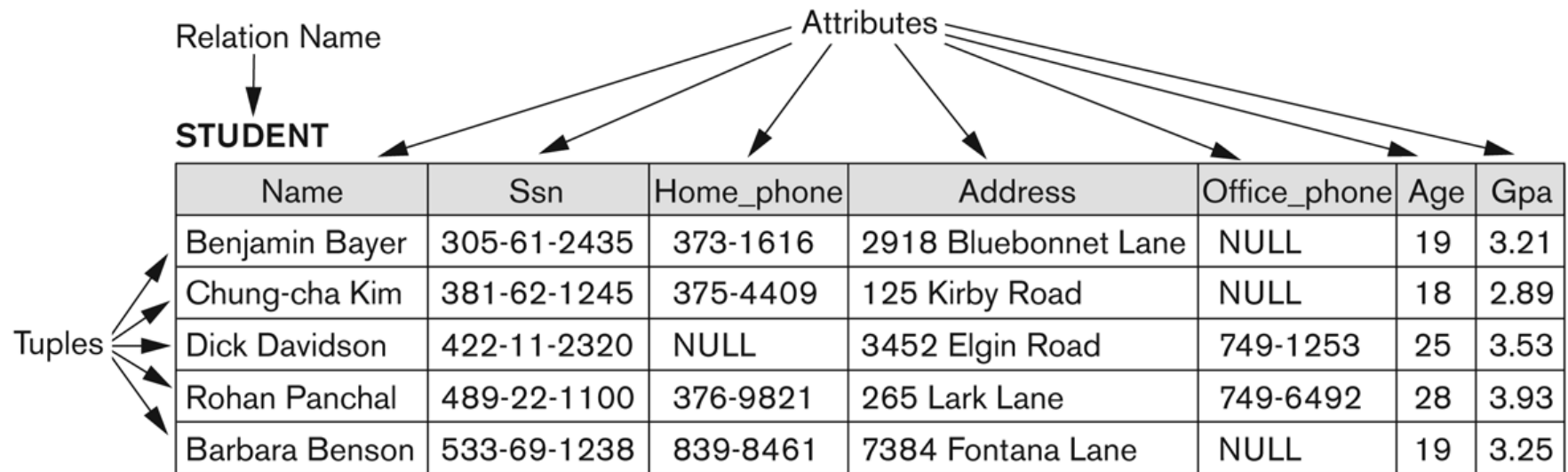


Figure 5.1

The attributes and tuples of a relation STUDENT.

Informal Definitions

- Key of a Relation:
 - Each row has a value of a data item (or set of items) that uniquely identifies that row in the table
 - Called the **key**
 - In the STUDENT table, SSN is the key
- Sometimes row-ids or sequential numbers are assigned as keys to identify the rows in a table
 - Called **artificial key** or **surrogate key**

Formal Definitions - Schema

- The **Schema** (or description) of a Relation:
 - Denoted by $R(A_1, A_2, \dots, A_n)$
 - R is the **name** of the relation
 - The **attributes** of the relation are A_1, A_2, \dots, A_n

- Example:

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

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

Formal Definitions - Tuple

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

Formal Definitions - Domain

- A **domain** has a logical definition:
 - Example: “USA_phone_numbers” are the set of 10 digit phone numbers valid in the U.S.
- A domain also has a data-type or a format defined for it.
 - The USA_phone_numbers may have a format: (ddd)ddd-dddd where each d is a decimal digit.
 - Dates have various formats such as year, month, date formatted as yyyy-mm-dd, or as dd mm, yyyy etc.
- The attribute name designates the role played by a domain in a relation:
 - Used to interpret the meaning of the data elements corresponding to that attribute
 - Example: The domain Date may be used to define two attributes named “Invoice-date” and “Payment-date” with different meanings

Formal Definitions - State

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

Formal Definitions - Summary

The earlier definition of a relation can be *restated* more formally using set theory concepts as follows. A relation (or relation state) $r(R)$ is a **mathematical relation** of degree n on the domains $\text{dom}(A_1)$, $\text{dom}(A_2)$, ..., $\text{dom}(A_n)$, which is a **subset** of the **Cartesian product** (denoted by \times) of the domains that define R :

$$r(R) \subseteq (\text{dom}(A_1) \times \text{dom}(A_2) \times \dots \times \text{dom}(A_n))$$

-
- $R(A_1, A_2, \dots, A_n)$ is the **schema** of the relation
- R is the **name** of the relation
- A_1, A_2, \dots, A_n are the **attributes** of the relation
- $r(R)$: a specific **state** (or "value" or "population") of relation R – this is a *set of tuples* (rows)
 - $r(R) = \{t_1, t_2, \dots, t_n\}$ where each t_i is an n -tuple
 - $t_i = \langle v_1, v_2, \dots, v_n \rangle$ where each v_j *element-of* $\text{dom}(A_j)$

Formal Definitions - Example

- Let $R(A1, A2)$ be a relation schema:
 - Let $\text{dom}(A1) = \{0,1\}$
 - Let $\text{dom}(A2) = \{a,b,c\}$
- The cartesian product of $\text{dom}(A1) \times \text{dom}(A2)$ is all possible combinations: $\{ \langle 0,a \rangle , \langle 0,b \rangle , \langle 0,c \rangle , \langle 1,a \rangle , \langle 1,b \rangle , \langle 1,c \rangle \}$

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

Definition Summary

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

Example – A relation STUDENT

Relation Name

STUDENT

Attributes

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

Tuples

Figure 5.1

The attributes and tuples of a relation STUDENT.

Characteristics Of Relations

- Ordering of tuples in a relation $r(R)$:
 - The tuples are not considered to be **ordered**, even though they appear to be in the tabular form.
- Ordering of attributes in a relation schema R (and of values within each tuple):
 - The values in $t = \langle v_1, v_2, \dots, v_n \rangle$ needs to be in order with the attributes in $R(A_1, A_2, \dots, A_n)$.

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

Figure 5.2

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

STUDENT

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

Characteristics Of Relations

- Values in a tuple:
 - All values are considered atomic (indivisible).
 - Each value in a tuple must be from the domain of the attribute for that column
 - If tuple $t = \langle v_1, v_2, \dots, v_n \rangle$ is a tuple (row) in the relation state r of $R(A_1, A_2, \dots, A_n)$
 - Then each v_i must be a value from $dom(A_i)$
- A special **null** value is used to represent values that are unknown or inapplicable to certain tuples.
(Note: Two NULL values cannot be equated to be same.)

Characteristics Of Relations

- Notation:
 - We refer to **component values** of a tuple t by:
 - $t[A_i]$ or $t.A_i$
 - This is the value v_i of attribute A_i for tuple t
 - Similarly, $t[A_u, A_v, \dots, A_w]$ refers to the sub-tuple of t containing the values of attributes A_u, A_v, \dots, A_w , respectively in t

Relational Integrity Constraints

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

Key Constraints

- **Superkey of R:**
 - Is a set of attributes SK of R with the following condition:
 - No two tuples in any valid relation state $r(R)$ will have the same value for SK
 - That is, for any distinct tuples t_1 and t_2 in $r(R)$, $t_1[SK] \neq t_2[SK]$
 - This condition must hold in **any valid state** $r(R)$
- **Key of R:**
 - A "minimal" superkey
 - That is, a key is a superkey K such that removal of any attribute from K results in a set of attributes that is not a superkey (does not possess the superkey uniqueness property)

Key Constraints (continued)

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

Key Constraints (continued)

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

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

CAR

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

Figure 5.4

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

Relational Database Schema

- **Relational Database Schema:**
 - A set S of relation schemas that belong to the same database.
 - S is the name of the whole **database schema**
 - $S = \{R_1, R_2, \dots, R_n\}$
 - R_1, R_2, \dots, R_n are the names of the individual **relation schemas** within the database S
- Following slide shows a COMPANY database schema with 6 relation schemas

COMPANY Database Schema

EMPLOYEE

Fname	Minit	Lname	<u>Ssn</u>	Bdate	Address	Sex	Salary	Super_ssn	Dno
-------	-------	-------	------------	-------	---------	-----	--------	-----------	-----

DEPARTMENT

Dname	<u>Dnumber</u>	Mgr_ssn	Mgr_start_date
-------	----------------	---------	----------------

DEPT_LOCATIONS

<u>Dnumber</u>	<u>Dlocation</u>
----------------	------------------

PROJECT

Pname	<u>Pnumber</u>	Plocation	Dnum
-------	----------------	-----------	------

WORKS_ON

<u>Essn</u>	<u>Pno</u>	Hours
-------------	------------	-------

DEPENDENT

<u>Essn</u>	<u>Dependent_name</u>	Sex	Bdate	Relationship
-------------	-----------------------	-----	-------	--------------

Figure 5.5
Schema diagram for
the COMPANY
relational database
schema.

Entity Integrity

- **Entity Integrity:**
 - The **primary key attributes** PK of each relation schema R in S cannot have null values in any tuple of $r(R)$.
 - This is because primary key values are used to **identify** the individual tuples.
 - $t[PK]$ cannot be null for any tuple t in $r(R)$
 - If PK has several attributes, null is not allowed in any of these attributes
 - Note: Other attributes of R may be constrained to disallow null values, even if they are not members of the primary key.

Referential Integrity

- A constraint involving **two** relations
 - The previous constraints involve a single relation.
- Used to specify a **relationship** among tuples in two relations:
 - The **referencing relation** and the **referenced relation**.

Referential Integrity

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

Referential Integrity (or foreign key) Constraint

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

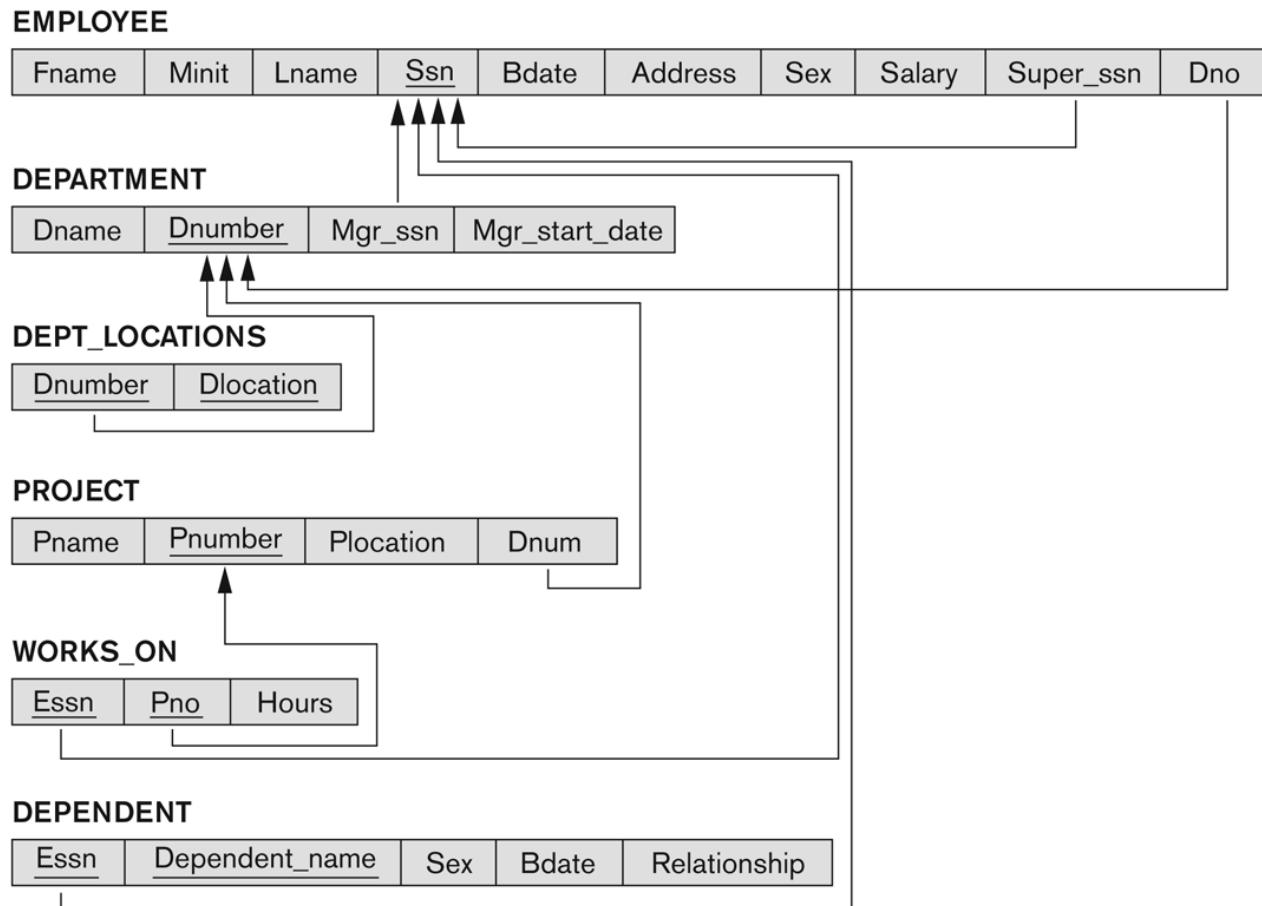
Displaying a relational database schema and its constraints

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

Referential Integrity Constraints for COMPANY database

Figure 5.7

Referential integrity constraints displayed on the COMPANY relational database schema.



Other Types of Constraints

- Semantic Integrity Constraints:
 - based on application semantics and cannot be expressed by the model per se
 - Example: “the max. no. of hours per employee for all projects he or she works on is 56 hrs per week”
- A **constraint specification** language may have to be used to express these
- SQL-99 allows triggers and **ASSERTIONS** to express for some of these

Populated database state

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

Populated database state for COMPANY

Figure 5.6

One possible database state for the COMPANY relational database schema.

EMPLOYEE

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

DEPARTMENT

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

DEPT_LOCATIONS

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

WORKS_ON

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

PROJECT

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

DEPENDENT

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

Update Operations on Relations

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

Update Operations on Relations

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

Possible violations for each operation

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

Possible violations for each operation

- DELETE may violate only referential integrity:
 - If the primary key value of the tuple being deleted is referenced from other tuples in the database
 - Can be remedied by several actions: RESTRICT, CASCADE, SET NULL (see Chapter 8 for more details)
 - RESTRICT option: reject the deletion
 - CASCADE option: propagate the new primary key value into the foreign keys of the referencing tuples
 - SET NULL option: set the foreign keys of the referencing tuples to NULL
 - One of the above options must be specified during database design for each foreign key constraint

Possible violations for each operation

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

Summary

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

In-Class Exercise

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

STUDENT(SSN, Name, Major, Bdate)

COURSE(Course#, Cname, Dept)

ENROLL(SSN, Course#, Quarter, Grade)

BOOK_ADOPTION(Course#, Quarter, Book_ISBN)

TEXT(Book_ISBN, Book_Title, Publisher, Author)

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