HNDIT 3042 Database Management Systems

Week 1 -Overview of Database System

Course Details

- Course Code : HNDIT 3042
- Course Title : Database Management Systems
- Semester : I
- Course Status : Compulsory
- Number of Credits : 3

Learning Outcomes

Upon successful completion of this course, students will be able to:

LO1: Describe database concepts, record storage, file organization and architectures of common database systems used in practice.

LO2. Model databases using ER, EER diagrams.

LO3. Implement a database application using SQL based on a design done with ER and EER diagrams.

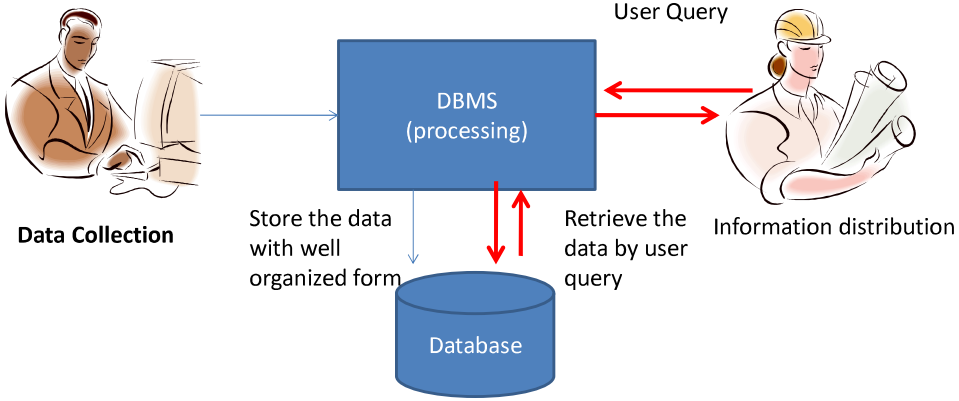
Timetable allocation (per week)

- Lectures : 2 hours
- Tutorials /practical : 2 hours
- Student activities : 6 hours
- Notional hours : 10 hours

Assessment Summary

On-line quizzes	**05%**
Assignment	
Group Assignment	**15%**
Individual Assignment	**20%**
Final Examination (3 hour paper)	**60%**
Total	**100%**

DBMS



DATABASE

- A shared **collection of logically related data** designed to meet the information requirements of an organisation.

Database Management System (DBMS)

- A software system that enables users to define, create and maintain the database and which provides controlled access to the database.

Actors on the Scene

- Database Administrator
- Database Users

Database Users

Users are differentiated by the way they expect to interact with the system

- **Application programmers** – interact with system through DML calls
- **Sophisticated users** – form requests in a database query language
- **Specialized users** – write specialized database applications that do not fit into the traditional data processing framework
- **Naïve users** – invoke one of the permanent application programs that have been written previously
  - Examples, people accessing database over the web, bank tellers, clerical staff

Database Administrator

- Coordinates all the activities of the database system; the database administrator has a good understanding of the enterprise's information resources and needs.
- Database administrator's duties include:
  - Schema definition
  - Storage structure and access method definition
  - Schema and physical organization modification
  - Granting user authority to access the database
  - Specifying integrity constraints
  - Acting as liaison with users
  - Monitoring performance and responding to changes in requirements

Difference between Traditional File based System and DBMS

- Drawbacks of using file systems to store data:
  - Data redundancy and inconsistency
    - Multiple file formats, duplication of information in different files
  - Difficulty in accessing data
    - Need to write a new program to carry out each new task
  - Data isolation — multiple files and formats
  - Integrity problems
    - Integrity constraints (e.g. account balance > 0) become "buried" in program code rather than being stated explicitly
    - Hard to add new constraints or change existing ones

- Atomicity of updates
  - Failures may leave database in an inconsistent state with partial updates carried out
  - Example: Transfer of funds from one account to another should either complete or not happen at all
- Concurrent access by multiple users
  - Concurrent accessed needed for performance
  - Uncontrolled concurrent accesses can lead to inconsistencies
    - Example: Two people reading a balance and updating it at the same time
- Security problems
  - Hard to provide user access to some, but not all, data

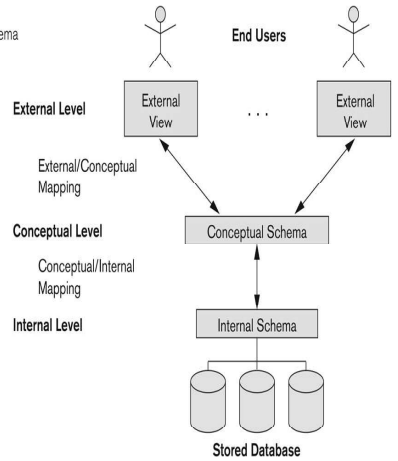
Three-Schema Architecture

- Defines DBMS schemas at ***three*** levels:
  - **Internal schema**
  - **Conceptual schema: entities, data types, relationships, etc**
  - **External schemas** at the external level to describe the various user views.
    - Usually uses the same data model as the conceptual schema.

Data Independence:

- **Logical Data Independence: ability to change conceptual schema without effecting appl. programs**
- **Physical Data Independence: ability to change internal schema without effecting upper layers**

Figure 2.2
The three-schema architecture.



A Physical Centralized Architecture

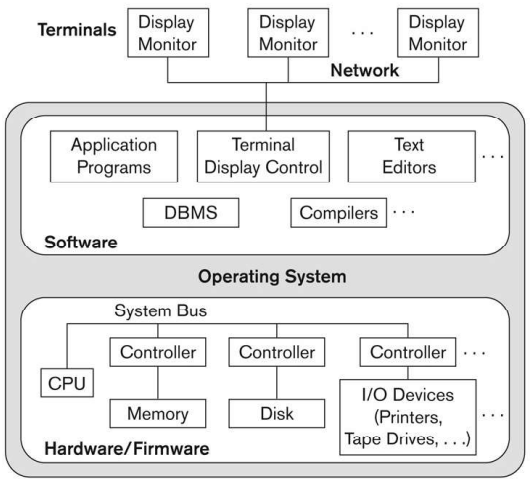
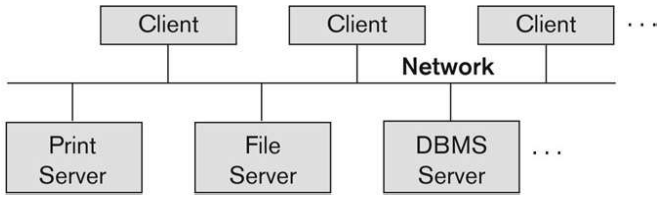


Figure 2.4
A physical centralized architecture.

Logical two-tier client server architecture

Figure 2.5
Logical two-tier client/server architecture.



Three-tier client-server architecture

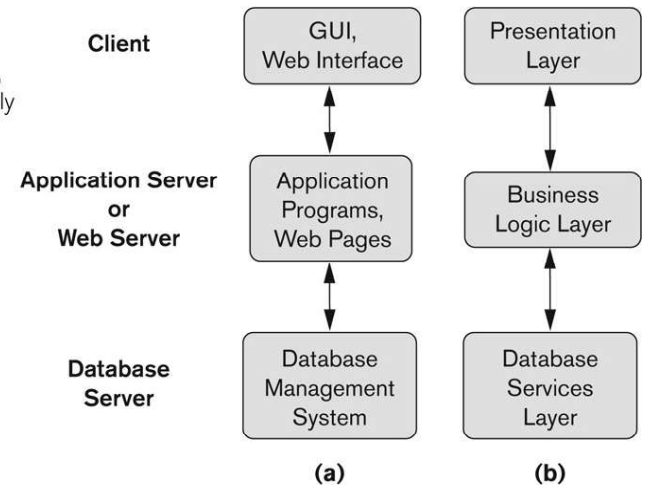


Figure 2.7
Logical three-tier client/server architecture, with a couple of commonly used nomenclatures.

Questions?

The Relational Data Model

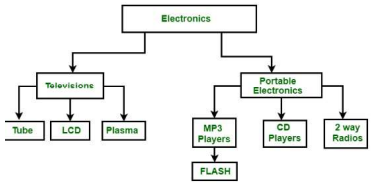
Data Model

- A Data Model in Database Management System (DBMS) is the concept of tools that are developed to summarize the description of the database. Data Models provide us with a transparent picture of data which helps us in creating an actual database. It shows us from the design of the data to its proper implementation of data.

- Types of Relational Models
- Conceptual Data Model
- Representational Data Model
- Physical Data Model

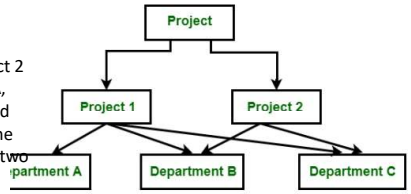
Hierarchical, Network and Relational Data Model

- **Hierarchical Data Model:**
  Hierarchical data model is the oldest type of the data model. It was developed by IBM in 1968. It organizes data in the tree-like structure. Hierarchical model consists of the following :
- It contains nodes which are connected by branches.
- The topmost node is called the root node.
- If there are multiple nodes appear at the top level, then these can be called as root segments.
- Each node has exactly one parent.
- One parent may have many child.

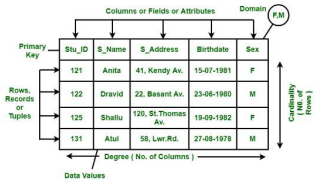


- **Network Data Model:**
  It is the advance version of the hierarchical data model. To organize data it uses directed graphs instead of the tree-structure. In this child can have more than one parent. It uses the concept of the two data structures i.e. Records and Sets.

figure, Project is the root node which has two children i.e. Project 1 and Project 2. Project 1 has 3 children and Project 2 has 2 children. Total there are 5 children i.e Department A, Department B and Department C, they are network related children as we said that this model can have more than one parent. So, for the Department B and Department C have two parents i.e. Project 1 and Project 2.



- **Relational Data Model:**
  The relational data model was developed by E.F. Codd in 1970. There are no physical links as they are in the hierarchical data model. Following are the properties of the relational data model :
- Data is represented in the form of table only.
- It deals only with the data not with the physical structure.
- It provides information regarding metadata.
- At the intersection of row and column there will be only one value for the tuple.
- It provides a way to handle the queries with ease.



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**)

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

Example of a Relation

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.

Informal Definitions

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

Formal Definitions - Schema

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

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

Formal Definitions - Tuple

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

Formal Definitions - Domain

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

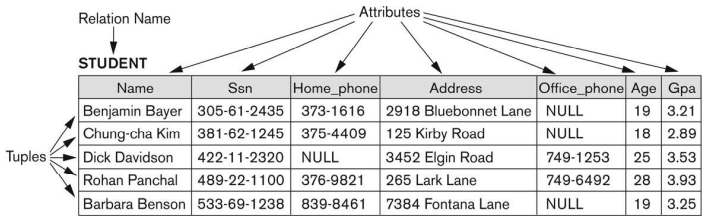
Formal Definitions - State

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

Definition Summary

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

Example – A relation STUDENT



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

Figure 5.1
The attributes and tuples of a relation STUDENT.

Characteristics Of Relations

- Ordering of tuples in a relation r(R):
  - The tuples are *not considered to be ordered*, even though they appear to be in the tabular form.
- Ordering of attributes in a relation schema R (and of values within each tuple):
  - We will consider the attributes in R(A1, A2, ..., An) and the values in t=<v1, v2, ..., vn> to be ordered .
    - (However, a more general alternative definition of relation does not require this ordering).

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

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

STUDENT

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

Characteristics Of Relations

- Values in a tuple:
  - All values are considered atomic (indivisible).
  - Each value in a tuple must be from the domain of the attribute for that column
    - If tuple t = <v1, v2, …, vn> is a tuple (row) in the relation state r of R(A1, A2, …, An)
    - Then each *vi* must be a value from *dom(Ai)*
  - A special **null** value is used to represent values that are unknown or inapplicable to certain tuples.

Characteristics Of Relations

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

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 t1 and t2 in r(R), t1[SK] ≠ t2[SK]
    - This condition must hold in *any valid state* r(R)
- **Key** of R:
  - A "minimal" superkey
  - That is, a key is a superkey K such that removal of any attribute from K results in a set of attributes that is not a superkey (does not possess the superkey uniqueness property)

Key Constraints (continued)

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

Key Constraints (continued)

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

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

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

CAR

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

Relational Database Schema

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

COMPANY Database Schema

EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno

DEPARTMENT

Dname	Dnumber	Mgr_ssn	Mgr_start_date

DEPT_LOCATIONS

Dnumber	Dlocation

PROJECT

Pname	Pnumber	Plocation	Dnum

WORKS_ON

Essn	Pno	Hours

DEPENDENT

Essn	Dependent_name	Sex	Bdate	Relationship

Figure 5.5
Schema diagram for the COMPANY relational database schema.

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] ≠ null for any tuple t in r(R)
    - If PK has several attributes, null is not allowed in any of these attributes
  - Note: Other attributes of R may be constrained to disallow null values, even though they are not members of the primary key.

Referential Integrity

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

Referential Integrity

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

Referential Integrity (or foreign key) Constraint

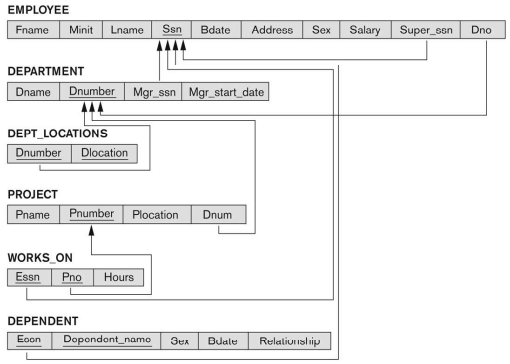
- Statement of the constraint
  - The value in the foreign key column (or columns) FK of the the **referencing relation** R1 can be **either**:
    - (1) a value of an existing primary key value of a corresponding primary key PK in the **referenced relation** R2, or
    - (2) a **null**.
- In case (2),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

Displaying a relational database schema and its constraints

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

Referential Integrity Constraints for COMPANY database

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



Database Management Systems

Data Models

Structure of a database

- Data Model comprises:
  - **a structural part;**
  - **a manipulative part;**
  - **possibly a set of integrity rules.**
- Most commercial database systems are based on relational data model

Database Models

- A set of concepts to describe the structure of a database and certain constraints that the database should obey.
  - Many forms of data models
  - Makes data abstraction possible
    - Hides details of physical data storage from the user

Categories of data models

- *Data models can be categorize according to the types of concepts they use to describe the database structure* :
  - Conceptual (high-level, semantic) data models
  - Physical (low-level, internal) data models
  - Implementation (representational) data models

Conceptual (high-level, semantic) data models

- Provide concepts that are close to the way many users *perceive* data.
- Conceptual models include
  - Entity-relationship database model (ERDBM)
  - Object-oriented model (OODBM)

Physical (low-level, internal) data models

- Provide concepts that describe details of how data is stored in the computer.
  - Record formats, record orderings and access paths
  - Target computer specialists and not typical end users

Conceptual data models, Cont'

- Conceptual data models use concepts such as entities, attributes, and relationships.
- Relationships in Conceptual Models
  - One-to-one (1:1)
  - One-to-many (1:M)
  - Many-to-many (M:N)

Implementation (representational) data models

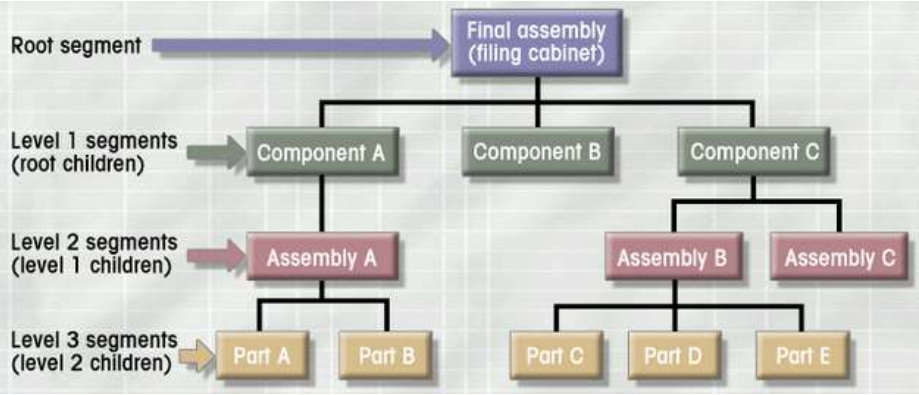
- Emphasizes on how the data is represented in the database or how the data structures are implemented to represent what is modeled:
- Implementation models include
  - Hierarchical database model (HDBM)
  - Network database model (NDBM)
  - Relational database model (RDBM)
  - Object-oriented database model (ODBM)

Hierarchical Database Model (HDBM)

- Logically represented by an upside down tree
  - Each parent can have many children (segment linkage)
  - Each child has only one parent

Hierarchical Database Model

- Logically represented by an upside down tree
  - 1:M relationship



Hierarchical Database Model

- Hierarchical path (beginning from left)
- Left-list hierarchical path, or preorder traversal, or hierarchical sequence

Final assembly->Component A->Assembly A-> -> Part A ->Part B -> Component B -> Component C –Assembly B -> Part C ->Part D

- Re-list sequence, if the segment is frequently accessed
- Bank systems commonly use HD model

Hierarchical Database Model

- Bank systems commonly use the HDBM
  - customer account can be subject to many transactions (1:M relationship)
  - Relationship is fixed (debiting and crediting)
  - Frequently access large amount of transactions

Hierarchical Database Model

- Advantages
  - Conceptual simplicity: relationship between layers is logically simple; design process is simple
  - Database security: enforced uniformly through the system
  - Data integrity
  - Data independence
  - Efficiency in 1:M relationships and when uses require large numbers of transactions
  - Dominant in 1970s , when we used mainframe system with large databases

Hierarchical Database Model

- Disadvantages
  - Complex implementation: physical data storage characteristics; database design is complicated
  - Difficult to manage and lack of standards
  - Lacks structural independence
  - Applications programming and use complexity (pointer based)
  - Implementation limitations, i.e. especially it only handle 1:M type of model

Network Database Model (NDBM)

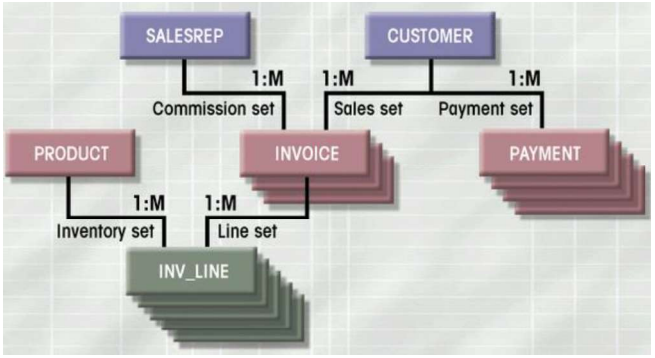
- Each record can have multiple parents
  - Called by Database Task Group (DBTG) to define standards
  - Three crucial database components
    - Network schema: conceptual organization of the entire database
    - Subschema: portion of database as information for application programs
    - Database management language: defining data characteristics and data structure
      - Schema Data definition language (DDL): define schema components
      - Subschema Data definition language
      - Data manipulating language: manipulate data content

Network Database Model

- Each record can have multiple parents
- Introduce set to describe relationship
- Each set has owner record and member record, parallel to parent and child in HDM
  - Member may have several owners
  - One-ownership

Network Database Model

– Member may have several owners



Network Database Model

- Disadvantages
  - System complexity
  - Lack of structural independence

Network Database Model

- Advantages
  - Conceptual simplicity, just lime HDM
  - Handles more relationship types (but all 1:M relationship)
  - Data access flexibility
  - Promotes database integrity
  - Data independence
  - Conformance to standards

Relational Database Model (RDBM)

- Let's user or database designer to operate human logical environment
- Perceived by user as a collection of tables for data storage, while let RDBMS handles the physical details.
- Tables are a series of row/column intersections
- Tables related by sharing common entity characteristics
- It allows 1:1, 1:M, M:N relationships

Relational Database Model

Table name: AGENT

AGENT_CODE	AGENT_LNAME	AGENT_FNAME	AGENT_INITIAL	AGENT_AREACODE	AGENT_PHONE
501	Alby	Alex	B	713	228-1249
502	Hahn	Leah	F	615	882-1244
503	Okon	John	T	615	123-5589

Link through AGENT code

Table name: CUSTOMER

CUS_CODE	CUS_LNAME	CUS_FNAME	CUS_INITIAL	CUS_AREACODE	CUS_PHONE	CUS_RENEW_DATE	AGENT_CODE
10010	Ramas	Alfred	A	615	844-2573	05-Apr-2002	502
10011	Dunne	Leona	K	713	894-1238	16-Jun-2002	501
10012	Smith	Kathy	W	615	894-2285	29-Jan-2001	502
10013	Olowski	Paul	F	615	894-2180	14-Oct-2002	502
10014	Orlando	Myron		615	222-1672	28-Dec-2002	501
10015	O'Brian	Amy	B	713	442-3381	22-Sep-2002	503
10016	Brown	James	G	615	297-1228	25-Mar-2002	502
10017	Williams	George		615	290-2556	17-Jul-2002	503
10018	Farriss	Anne	G	713	382-7185	03-Dec-2002	501
10019	Smith	Olette	K	615	297-3809	14-Mar-2002	503

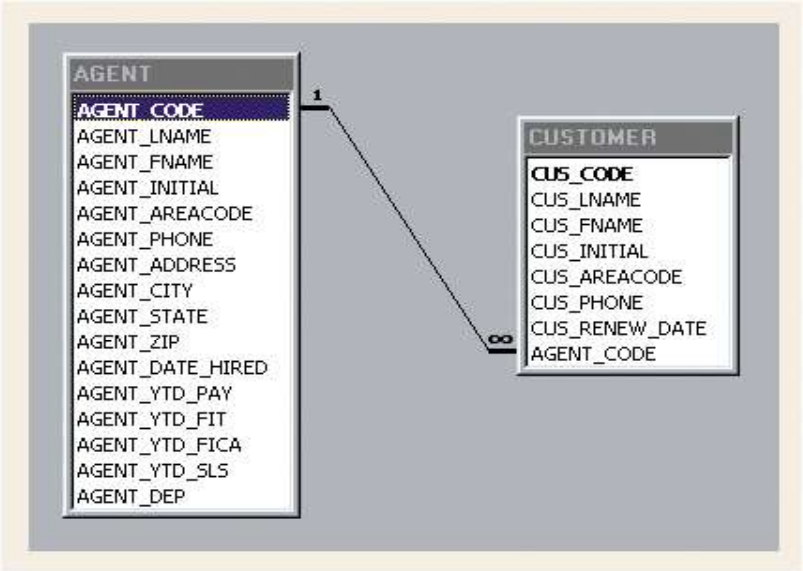


FIGURE 1.12 A RELATIONAL SCHEMA

Relational Database Model

- Advantages
  - Structural independence: data access path is irrelevant to database design; change structure will not affect the database
  - Improved conceptual simplicity
  - Easier database design, implementation, management, and use
  - Ad hoc query capability with SQL (4GL is added)
  - Powerful database management system

Relational Database Model

- Disadvantages
  - Substantial hardware and system software overhead
  - Poor design and implementation is made easy
  - May promote “islands of information” problems

Entity Relationship Database Model (ERDBM)

- Complements the relational data model concepts
- ERDBM introduces a relational graphic representation
- ERDBM is based on several components
  - Entity, tabled entity (in RDM)
- Entity and entity set, a collection of like entities
- Each entity has attributes to describe the entity, which is similar to field in table
- Relationship and connection

Entity Relationship Database Model

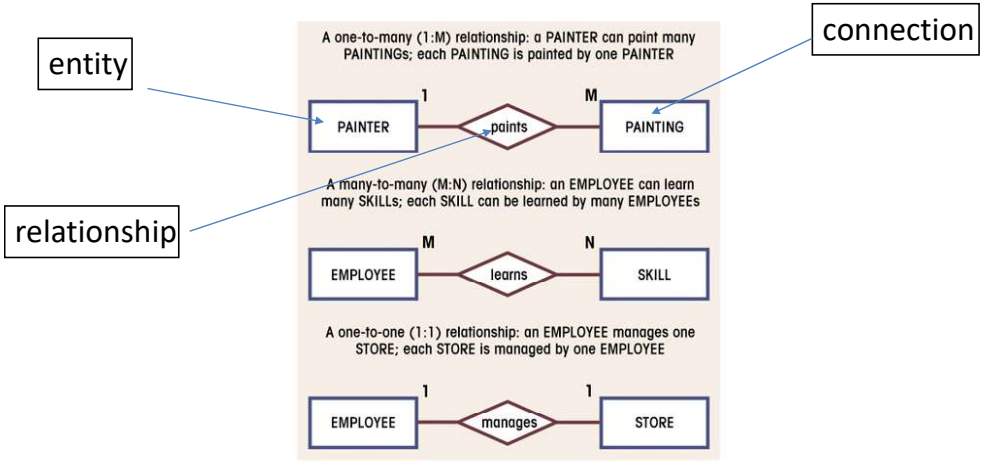


FIGURE 1.13 RELATIONSHIP DEPICTION: THE ERD

Entity Relationship Database Model (ERDBM)

- Represented in an entity relationship diagram (ERD): Chen's ERD model and Crow's Foot ERD
- Based on entities, attributes, and relationships

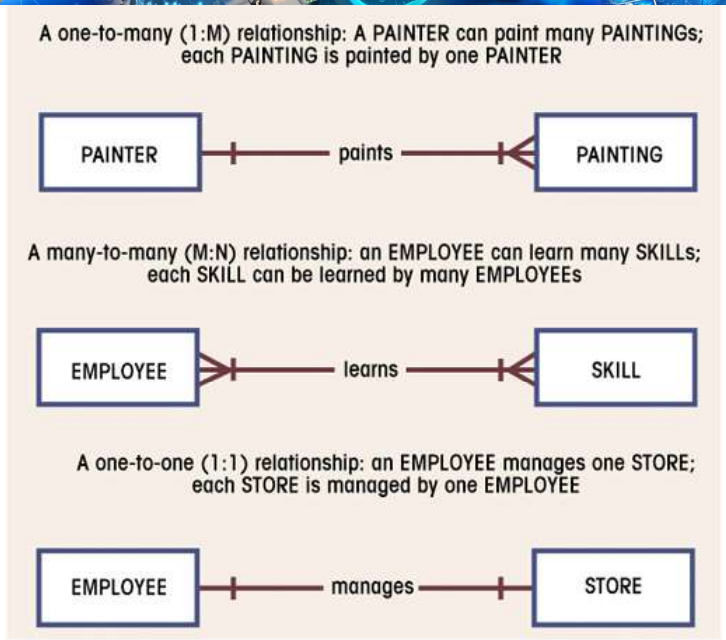


FIGURE 1.14 RELATIONSHIP DEPICTION: THE CROW'S FOOT ERD

Entity Relationship Database Model

- Advantages
  - Exceptional conceptual simplicity
  - Visual representation
  - Effective communication tool
  - Integrated with the relational database model

Entity Relationship Database Model

- Disadvantages
  - Limited constraint representation
  - Limited relationship representation (internal relationship can not be depicted; multiple relationships)
  - No data manipulation language (no complete)
  - Loss of information content

Object-Oriented Database Model (OODBM)

- Semantic Data model (SDM)->Object-oriented Data Model (OODM)
- Object-oriented concept:
- Objects or abstractions of real-world entities are stored
  - Attributes describe properties
  - Collection of similar objects is a class, similar to entity set but contains procedure methods
    - Methods represent real world actions of classes
    - Classes are organized in a class hierarchy
  - Inheritance is the ability of object to inherit attributes and methods of classes above it

Object-Oriented Database Model (OODBM)

- Contains implementation and procedure operation information for more complicated data such as graphics, video, and other metadata
- Support transaction and information
- Reusability
- Portable to powered computing system

Object-Oriented Database Model

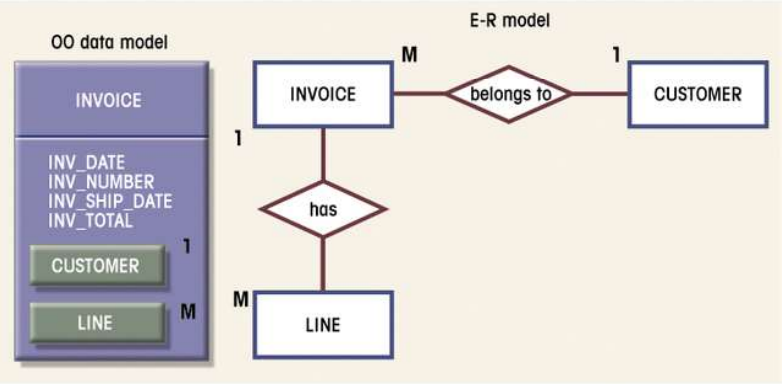


FIGURE 1.15 A COMPARISON OF THE OO DATA MODEL AND THE E-R MODEL

OO Database Model

- Disadvantages
  - Lack of OODM
  - Complex navigational data access
  - Steep learning curve
  - High system overhead slows transactions

OO Database Model

- Advantages
  - Adds semantic content
  - Visual presentation includes semantic content
  - Database integrity
  - Both structural and data independence

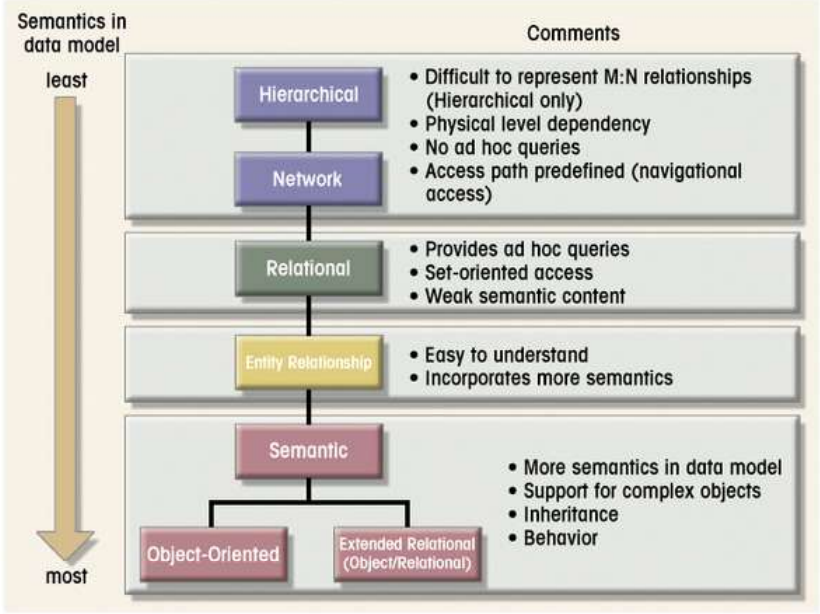


FIGURE 1.16 THE DEVELOPMENT OF DATA MODELS

Database Systems

- A database system is more than just data or data in combination with DBMS software
- A complete database system in an organization consists of four components:
  - Hardware
  - Software
  - Data and
  - People

Database Systems- Software

- A database system includes two types of software
  - General-purpose database management software , usually called the Database Management System
  - Application software that uses DBMS facilities to manipulate the database to achieve a specific business function.

Database Systems- Hardware

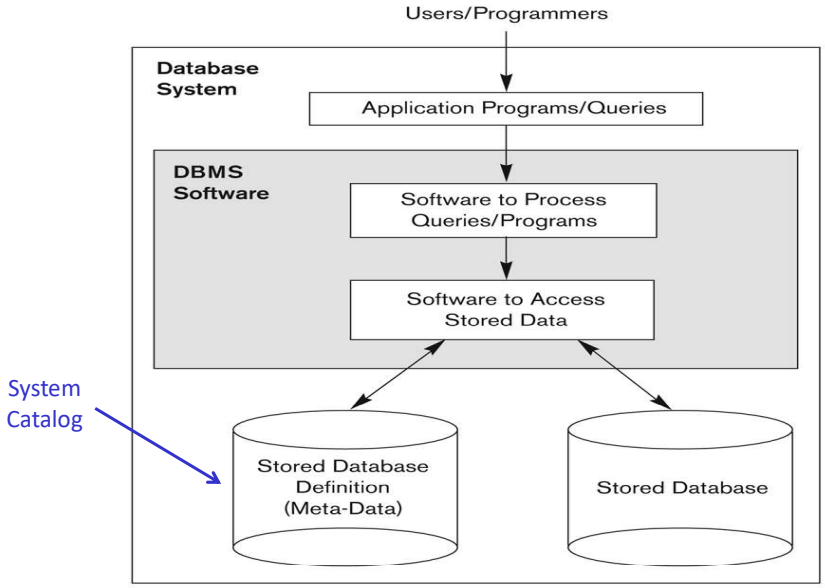
- The set of physical devices on which a database resides.
- The hardware portions of the system consist of:
  - The secondary storage volumes- typically magnetic disks- that are used to hold the stored data; together with the associated I/O devices, device controllers, I/O channels, and so forth.
  - The processor(s) and associated main memory that are used to support the execution of the database system software.

Database Systems- Software

Database Management System

- A software system that enables users to define, create, maintain, and control access to the database
- A DBMS typically provides following services:
  - A centralized data definition & data control facility known as a data dictionary/directory (DD/D) or catalog
  - Data Security and integrity mechanisms
  - Concurrent data access for multiple users
  - User-oriented data query, manipulation, and reporting capabilities
  - Programmer-oriented application system development capabilities

A simplified database system environment



Database Systems - People

- Data Administrator (DA), Database Administrator (DBA), Database Designers (Logical and Physical), Application Programmers, End Users (naive and sophisticated)

Database Systems- Data

- Used by the organization and a description of this data called the schema.
- In general, data in the database – at least in a large system – will be both ***integrated*** and ***shared***.
- Integrated means that the database can be thought of as a unification of several otherwise distinct data files, with any redundancy among those files either wholly or partly eliminated.
- Shared means that individual piece of data in the database can be shared among several different users, in the sense that each of those users can have access to the same piece of data.

References

- Connolly, T. M. & Begg, C. E. Database Systems A Practical Approach to Design, Implementation, and Management.
- Gray W. Hansen, James V. Hansen ,Database Management and Design.
- Ramez Elmasri and Shamkant B. Navathe, Fundamentals of Database Systems

Database Management Systems

Relational Model

THE RELATIONAL MODEL

- The relational model is a broad, flexible model
- Basis for almost all DBMS products
- E.F. Codd defined well-structured "normal forms" of relations, "normalization"

Learning Outcomes

- At the end of the lesson you should be able to describe the fundamental concept of Relational Data Model and its operations

RELATION

- A is a Two-dimensional table containing Rows (***tuples***) *and* Columns (***attributes)*** *of data****.***

Student	Reg_No	Name	ContactNo	Address	Gender	Age	GPA
	1111	C.D. Perera	0372228222	Kurunegala	M	20	3.9
	1112	C. Basnayake	0362255222	Awissawella	M	21	3.5
	1114	N.S. Bandara	0812234567	Kandy	F	20	3.8
	1115	K. Peris	0112876655	Colombo	M	21	3.0
	1116	S. Menike	0253456654	Anuradhapura	F	21	2.6

Relational Model -Properties

- Each relation (or table) in a database has a unique name
- An entry at the intersection of each row and column is atomic (or single-valued); there can be no multi-valued attributes in a relation
- Each row is unique; no two rows in a relation are identical
- Each attribute (or column) within a table has a unique name
- The sequence of columns (left to right) is insignificant; the columns of a relation can be interchanged without changing the meaning or use of the relation
- The sequence of rows (top to bottom) is insignificant; rows of a relation may be interchanged or stored in any sequence

Equivalent Relational Terms

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

Relational Model

- ***The relational model of data has three major components***:
- Relational Database Objects- allows to define data structures
- Relational Operators – allows manipulation of stored data
- Relational Integrity Constraints – allows to define business rules and ensure data integrity

ATTRIBUTE

- Each column in the relation is an attribute of the relation.
- The number of attributes of the relation is called the **DEGREE OF THE RELATION**.
- The set of possible values that an attribute may have is the **DOMAIN** of the attribute.

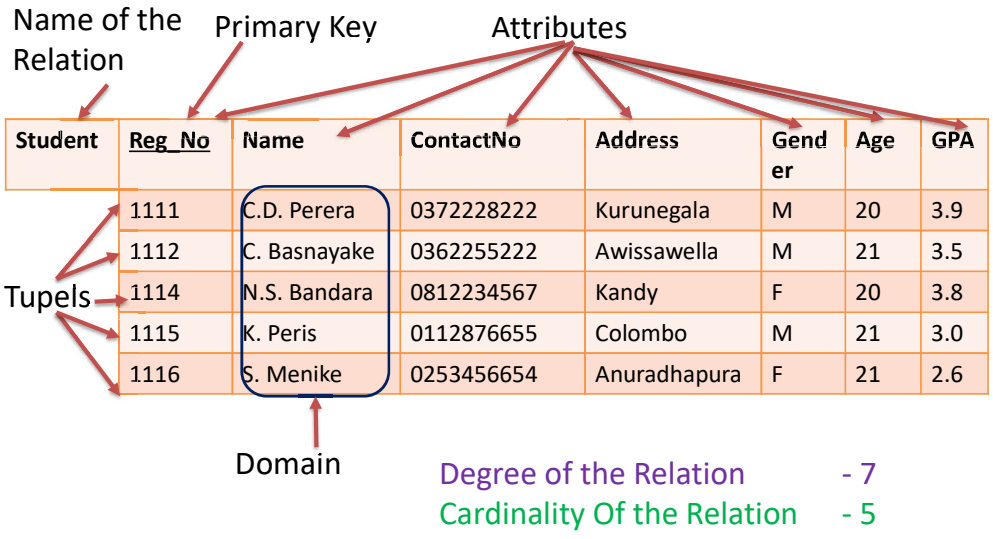
TUPLE

- A tuple is a collection of values that makes up one row of relation.
- A tuple is the equalent of a record.
- Number of tuples in a relation is its **CARDINALITY**.

KEYS

- A key of a relation is an attribute in a relation.
- There are various types of keys in a relation.

Relational Model



Student	Reg_No	Name	ContactNo	Address	Gender	Age	GPA
	1111	C.D. Perera	0372228222	Kurunegala	M	20	3.9
	1112	C. Basnayake	0362255222	Awissawella	M	21	3.5
	1114	N.S. Bandara	0812234567	Kandy	F	20	3.8
	1115	K. Peris	0112876655	Colombo	M	21	3.0
	1116	S. Menike	0253456654	Anuradhapura	F	21	2.6

PRIMARY KEY

- The attribute (or attributes in combination) for which no more than one tuple may have the same value is called the primary key.
- That is a value that can be used to uniquely identify each record.
- The relational data model requires that the primary key of a tuple should not contain null values.

CANDIDATE KEY

- Candidate keys are all attributes that may serve as the primary key. (one key is selected as a primary key)

COMPOSITE/CONCATENATED KEY

- A key that consists of two or more attributes appended (joined) together.

ALTERNATE KEY

- The candidate keys other than the chosen primary key are called alternate keys.

RELATIONAL INTEGRITY CONSTRAINTS

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

KEY CONSTRAINTS

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

ENTITY INTEGRITY

- **Relational Database Schema**: A set S of relation schemas that belong to the same database. S is the *name* of the **database**.
- S = {R_1, R_2, ..., R_n}
- **Entity Integrity**: The *primary key attributes* PK of each relation schema R in S cannot have null values in any tuple of r(R). This is because primary key values are used to *identify* the individual tuples.
- t[PK] ≠ null for any tuple t in r(R)

KEY CONSTRAINTS

- **Example**: The VEHICAL relation schema:
- VEHICAL(State, Reg#, SerialNo, Make, Model, Year)
- has two keys Key1 = {State, Reg#}, Key2 = {SerialNo}, which are also superkeys. {SerialNo, Make} is a superkey but *not* a key.
- If a relation has *several* candidate **keys**, one is chosen arbitrarily to be the **primary key**. The primary key attributes are *underlined*.

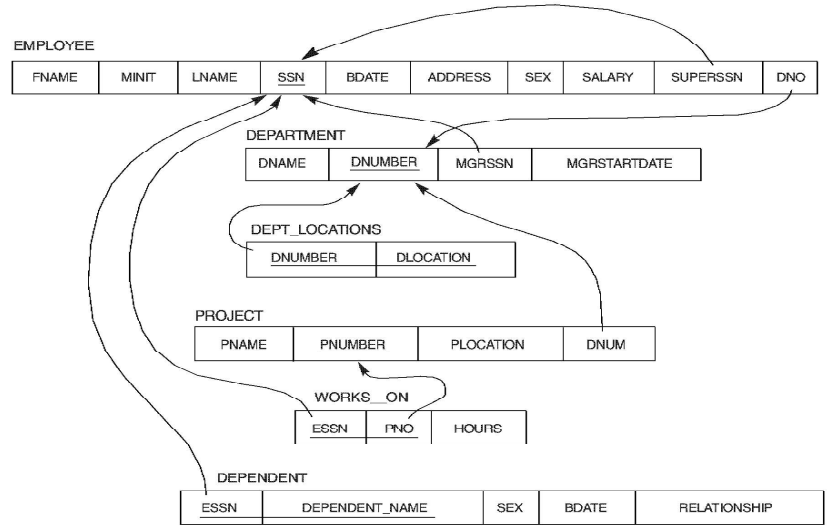
REFERENTIAL INTEGRITY

- 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 t_1 in R_1 is said to **reference** a tuple t_2 in R_2 if $t_1[FK] = t_2[PK]$.
- A referential integrity constraint can be displayed in a relational database schema as a directed arc from R_1.FK to R_2.

Example - Referential integrity constraints displayed on the COMPANY relational database schema diagram.



REFERENTIAL INTEGRITY CONSTRAINT

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

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

(2) a null.

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

Entity-Relationship (E-R) Model

Database Management System

HNDIT 2 YEAR 1 SEMESTER

SLIATE

E-R Model

- ERD is a data modeling technique used in software engineering to produce a conceptual data model of an information system.
- The major activity of this phase is identifying entities, attributes, and their relationships to construct model using the Entity Relationship Diagram.

Agenda

- Characteristics of E-R Model
- Components of E-R Model
- Example of E-R Model
- Enhanced E-R Model

Characteristics of E-R Model

- Semantic data model
- Express the logical properties of an enterprise database
- Design tools and documentation for data base structure
- No physical DBMS

Components of E-R Model

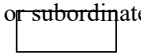
- Entity
- Attribute
- Key
- Relationship
- Structural constraints on relationship

Attribute

- Particular properties that describe entity
- Types
  - **Key attribute:** Uniqueness property must hold for every entity set of the entity type
  - Composite
  - Multi-valued
  - Derived
  - **Stored**
  - **derived** attributes

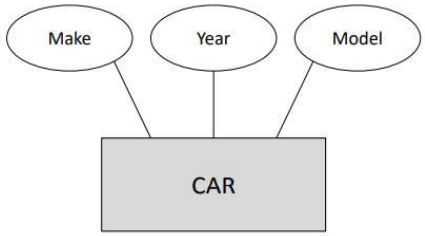
Entity

- Definition
  - An object or concept
  - Thing in real world with independent existence
  - "...anything (people, places, objects, events, etc.) about which we store information (e.g. supplier, machine tool, employee, utility pole, airline seat, etc.)."
  - Tangible: customer, product
  - Intangible: order, accounting receivable
  - Look for singular nouns (beginner)
- Types
  - Strong entity (parent, owner, dominant)
  - Weak entity (child, dependent, or subordinate)
- Diagram Notation
  - Rectangular



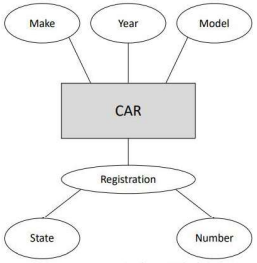
Ex

- All cars have a year, make, and model



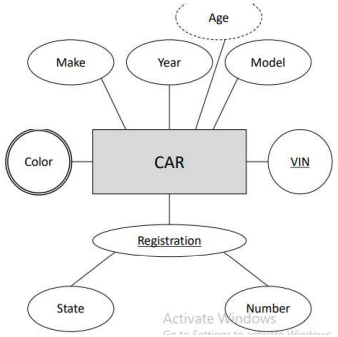
Composite Attributes

- Can be subdivided into
- smaller subparts
- Ex:
  - All cars have a year, make, model, and registration.



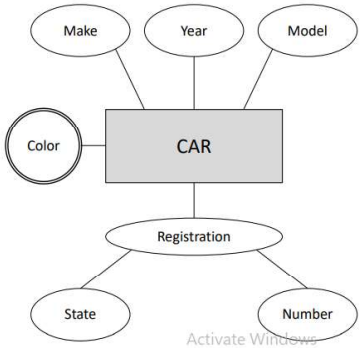
Derived Attributes

- The value can be computed
- Ex
  - All cars have a year, age, make, model, registration (unique), vehicle number (vin; unique), some number of colors.

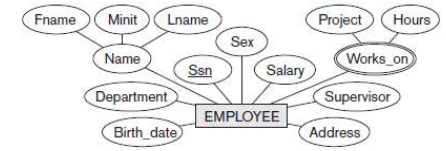


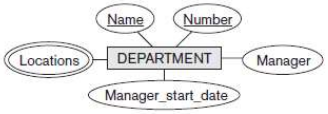
Multivalued Attributes

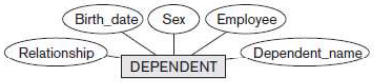
- Can take a [possibly specified] number of values.
- Ex
  - All cars have a year, make, model, registration, and some number of colors.

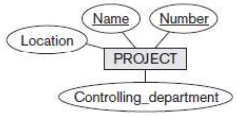


Identify different types of attributes









Draw an ERD for the following description:

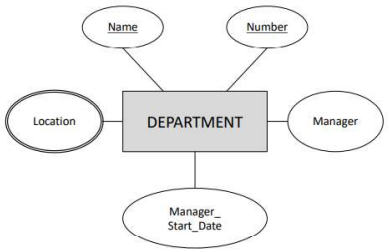
Each department has a unique name, a unique number, and a particular employee who manages the department. We keep track of the start date when that employee began managing the department. A department may have several locations.

Ex continue……

- A department controls a number of projects, each of which has a unique name, a unique number, and a single location.

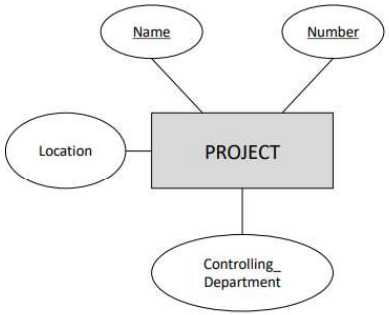
Draw an ERD for the following description:

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.



Ex continue……

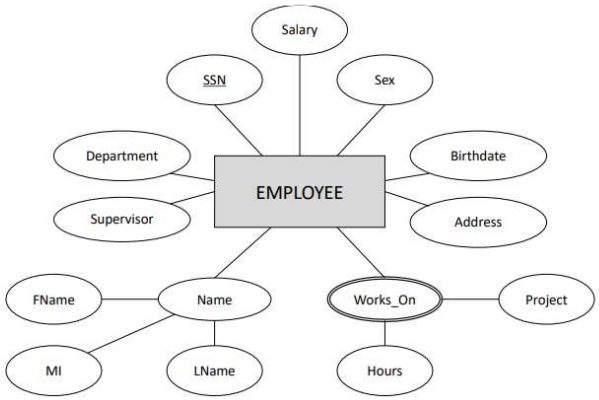
- A department controls a number of projects, each of which has a unique name, a unique number, and a single location.



Ex continue……

- We store each employee's name (first, last, MI), Social Security number (SSN), street 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. We keep track of the current number of hours per week that an employee works on each project. We also keep track of the direct supervisor of each employee (who is another employee).

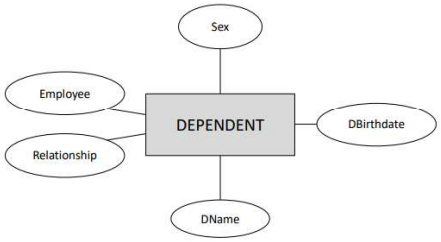
- We store each employee's name (first, last, MI), Social Security number (SSN), street address, salary, sex (gender), and birth date. An employee is assigned to one department, but may work on several projects, which are not necessarily controlled by the same department. We keep track of the current number of hours per week that an employee works on each project. We also keep track of the direct supervisor of each employee (who is another employee).



Ex continue……

- We want to keep track of the dependents of each employee for insurance purposes. We keep each dependent's first name, sex, birth date, and relationship to the employee.

- We want to keep track of the dependents of each employee for insurance purposes. We keep each dependent's first name, sex, birth date, and relationship to the employee.



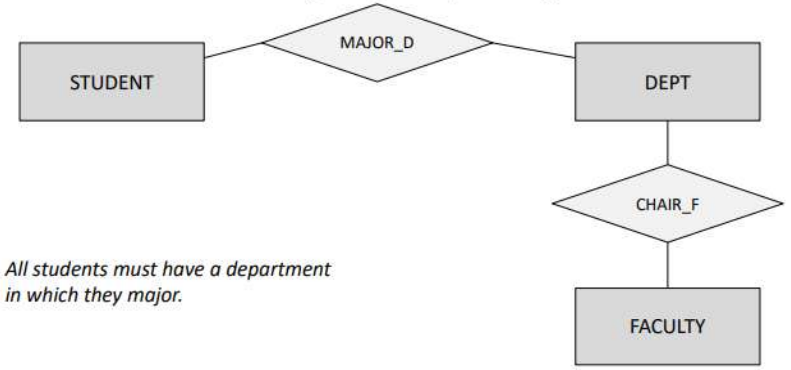
Relationship

- Definition
  - Association among entities
- Diagram Notation
  - Line – (relation or role name)
- Degree of a relationship
  - Number of participating entities
  - Types
    - Unary (recursive relationship)
    - Binary
    - Ternary
    - Quaternary

Relationships

Associates one or more sets of entities

– One = recursive (**role** is important)

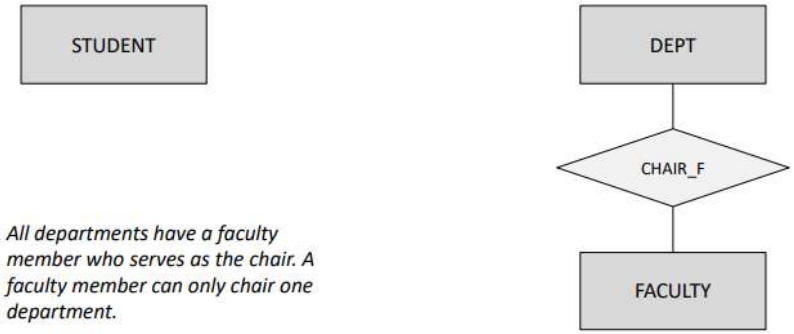


All students must have a department in which they major.

Relationships

Associates one or more sets of entities

– One = recursive (**role** is important)

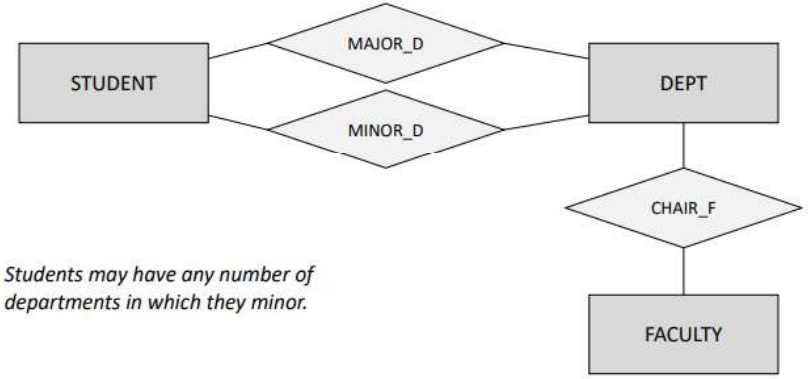


All departments have a faculty member who serves as the chair. A faculty member can only chair one department.

Relationships

Associates one or more sets of entities

– One = recursive (**role** is important)

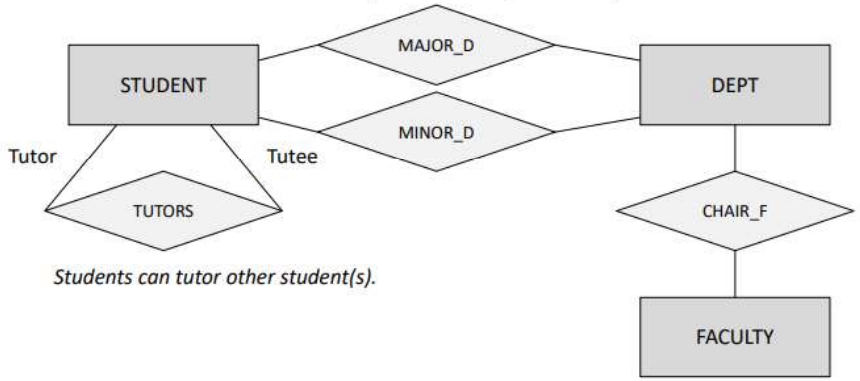


Students may have any number of departments in which they minor.

Relationships

Associates one or more sets of entities

– One = recursive (**role** is important)



Students can tutor other student(s).

Database Management Systems

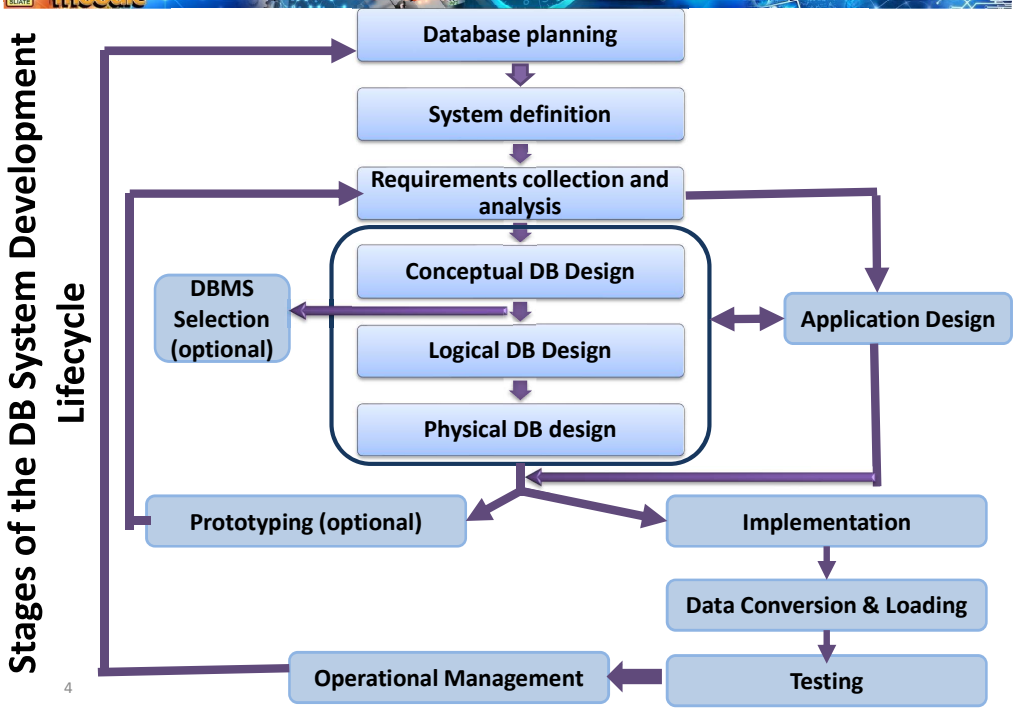
Database Planning, Design and Administration

Database System Development Lifecycle

- Stages of the database system development lifecycle:
  1. **Database planning**
  2. **System definition**
  3. **Requirements collection and analysis**
  4. **Database design**
  5. **DBMS selection (optional)**

Database System Development Lifecycle

6. **Application design**
7. **Prototyping (optional)**
8. **Implementation**
9. **Data conversion and loading**
10. **Testing**
11. **Operational maintenance**



Stage	Main activities
Database planning	Planning how the stages of the lifecycle can be realized most efficiently and effectively.
System definition	Specifying the scope and boundaries of the database system, including the major user views, its users, and application areas.
Requirements collection and analysis	Collection and analysis of the requirements for the new database system.
Database design	Conceptual, logical, and physical design of the database.
DBMS selection (optional)	Selecting a suitable DBMS for the database system.
Application design	Designing the user interface and the application programs that use and process the database.
Prototyping (optional)	Building a working model of the database system, which allows the designers or users to visualize and evaluate how the final system will look and function.
Implementation	Creating the physical database definitions and the application programs.
Data conversion and loading	Loading data from the old system to the new system and, where possible, converting any existing applications to run on the new database.
Testing	Database system is tested for errors and validated against the requirements specified by the users.
Operational maintenance	Database system is fully implemented. The system is continuously monitored and maintained. When necessary, new requirements are incorporated into the database system through the preceding stages of the lifecycle.

Database Planning

- Management activities that allow stages of database system development lifecycle to be realized as efficiently and effectively as possible.
- Must be integrated with overall IS strategy of the organization.

Database Planning

- Database planning should also include development of standards that govern:
- how data will be collected,
- how the format should be specified,
- what necessary documentation will be needed,
- how design and implementation should proceed.

System Definition

- Describes scope and boundaries of database system and the major user views.
- **User view** Defines what is required of a database system from the perspective of a particular job role (such as Manager or Supervisor) or enterprise application area (such as marketing, personnel, or stock control).
- Database application may have one or more user views.

Requirements Collection and Analysis

- Process of collecting and analyzing information about the part of organization to be supported by the database system, and using this information to identify users' requirements of new system.
- **fact-finding techniques are used** for gathering requirements.
- Information is analyzed to identify requirements to be included in new database system. Described in the requirements specification.

Requirements Collection and Analysis

- **requirements specification techniques:**

✓Structured Analysis and Design (SAD) techniques

✓Data Flow Diagrams (DFD)

✓ Hierarchical Input Process Output (HIPO)

✓Computer-Aided Software Engineering (CASE) tools

✓Unified Modeling Language (UML)

Database Design

- Process of creating a design for a database that will support the enterprise's mission statement and mission objectives for the required database system.
- The two main approaches to the design of a database are

✓**Bottom-up** and

✓**Top-down**

Database Design - Bottom-up approach

- This approach begins at the fundamental level of attributes, which through analysis of the associations between attributes, are grouped into relations that represent types of entities and relationships between entities.
- Normalization represents a bottom-up approach to database design.
- appropriate for the design of simple databases with a relatively small number of attributes.

Database Design - Top-down Approach

- A more appropriate strategy for the design of complex databases is to use the **top-down** approach.
- This approach starts with the development of data models that contain a few high-level entities and relationships and then applies successive top-down refinements to identify lower-level entities, relationships, and the associated attributes.
- The top-down approach is illustrated using the concepts of the Entity–Relationship (ER) model

Database Design - Data Modeling

- A data model ensures we understand:
  - each user's perspective of the data;
  - nature of the data itself, independent of its physical representations;
  - use of data across user views.
- ***Criteria to Produce an Optimal Data Model*** :

Structural validity, Simplicity, Expressibility, Nonredundancy, Shareability, Extensibility, Integrity, Diagrammatic representation

Database Design -Data Modeling

- Main purposes of data modeling include:
  - to assist in understanding the meaning (semantics) of the data;
  - to facilitate communication about the information requirements.
- Building data model requires answering questions about entities, relationships, and attributes.

Database Design

- **Three phases of database design:**
  - **Conceptual database design**
  - **Logical database design**
  - **Physical database design.**

Conceptual Database Design

- Process of constructing a model of the data used in an enterprise, independent of *all* physical considerations.
- Data model is built using the information in users' requirements specification.
- Conceptual data model is source of information for logical design phase.

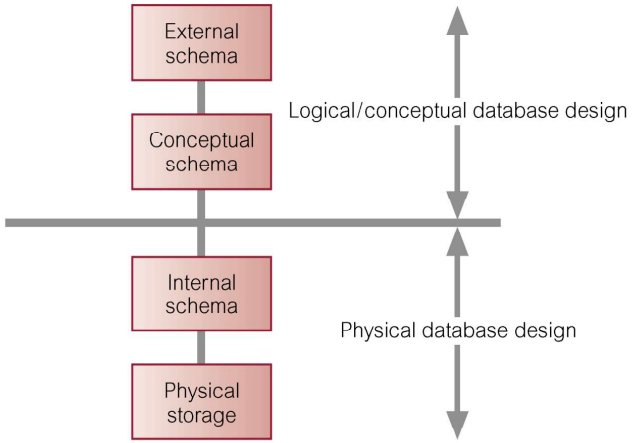
Logical Database Design

- Process of constructing a model of the data used in an enterprise based on a specific data model (e.g. relational), but independent of a particular DBMS and other physical considerations.
- Conceptual data model is refined and mapped on to a logical data model.

Physical Database Design

- Process of producing a description of the database implementation on secondary storage.
- Describes base relations, file organizations, and indexes used to achieve efficient access to data. Also describes any associated integrity constraints and security measures.
- Tailored to a specific DBMS system.

Three-Level ANSI-SPARC Architecture and Phases of Database Design



DBMS Selection

- Selection of an appropriate DBMS to support the database system.
- Undertaken at any time prior to logical design provided sufficient information is available regarding system requirements.
- Main steps to selecting a DBMS:
  - define Terms of Reference of study;
  - shortlist two or three products;
  - evaluate products;
  - recommend selection and produce report.

Example - Evaluation of DBMS Product

DBMS: Sample product
Vendor: Sample vendor

Physical Definition Group

Features	Comments	Rating	Weighting	Score
File structures available	Choice of 4	8	0.15	1.2
File structure maintenance	NOT self-regulating	6	0.2	1.2
Ease of reorganization		4	0.25	1.0
Indexing		6	0.15	0.9
Variable length fields/records		6	0.15	0.9
Data compression	Specify with file structure	7	0.05	0.35
Encryption routines	Choice of 2	4	0.05	0.2
Memory requirements		0	0.00	0
Storage requirements		0	0.00	0
Totals		41	1.0	**5.75**
Physical definition group		5.75	0.25	**1.44**

Application Design

- Design of user interface and application programs that use and process the database.
- Database design and application design are parallel activities.
- Includes two important activities:
  - transaction design;
  - user interface design.

Application Design - Transactions

- An action, or series of actions, carried out by a single user or application program, which accesses or changes content of the database.
- Should define and document the high-level characteristics of the transactions required.

Application Design - Transactions

- Important characteristics of transactions:
  - data to be used by the transaction;
  - functional characteristics of the transaction;
  - output of the transaction;
  - importance to the users;
  - expected rate of usage.
- Three main types of transactions: retrieval, update, and mixed.

Prototyping

- Building working model of a database system.
- Prototypes help
  - to identify features of a system that work well, or are inadequate;
  - to suggest improvements or even new features;
  - to clarify the users' requirements;
  - to evaluate feasibility of a particular system design.

Implementation

- Physical realization of the database and application designs.
  - Use DDL to create database schemas and empty database files.
  - Use DDL to create any specified user views.
  - Use 3GL or 4GL to create the application programs. This will include the database transactions implemented using the DML, possibly embedded in a host programming language.

Data Conversion and Loading

- Transferring any existing data into new database and converting any existing applications to run on new database.
- Only required when new database system is replacing an old system.
  - DBMS normally has utility that loads existing files into new database.
- May be possible to convert and use application programs from old system for use by new system.

Testing

- Process of running the database system with intent of finding errors.
- Use carefully planned test strategies and realistic data.
- Testing cannot show absence of faults; it can show only that software faults are present.
- Demonstrates that database and application programs *appear* to be working according to requirements.

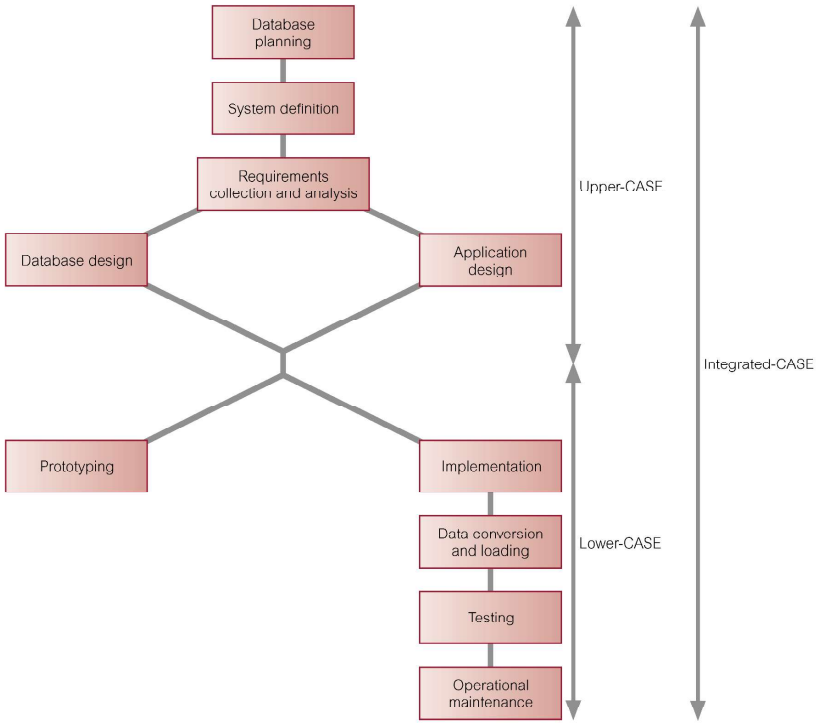
Testing

- Should also test usability of system.
- Evaluation conducted against a usability specification.
- Examples of criteria include:
  - Learnability;
  - Performance;
  - Robustness;
  - Recoverability;
  - Adaptability.

Operational Maintenance

- Process of monitoring and maintaining database system following installation.
- Monitoring performance of system.
  - if performance falls, may require tuning or reorganization of the database.
- Maintaining and upgrading database application (when required).
- Incorporating new requirements into database application.

CASE Tools and DB System Development Lifecycle



Data Administration and Database Administration

- The Data Administrator (DA) and Database Administrator (DBA) are responsible for managing and controlling the corporate data and corporate database, respectively.
- DA is more concerned with early stages of database system development lifecycle and DBA is more concerned with later stages.

Data Administration

- Management of data resource including:
  - database planning,
  - development and maintenance of standards, policies and procedures, and conceptual and logical database design.

Database Administration

- Management of physical realization of a database system including:
  - physical database design and implementation,
  - setting security and integrity controls,
  - monitoring system performance, and reorganizing the database.

Major functions of DBA and DA in the DB development life cycle

	DB planning and System definition- Develop entry charts ,Analyze costs and benefits, develop implementation plan, evaluate and select software/hardware, establish application priorities, develop data standards
	Requirements collection and analysis – define user requirements, develop data definition, develop data dictionary
	Database Design- Design conceptual model, external model, internal models and integrity controls
	DBMS selection ,Application design, Prototyping , Implementation, Data conversion and loading, Testing Specify DB access policies Develop standards for application program, establish security techniques, load DBs, specify test procedures and backup & recovery procedures
	Operational maintenance – Monitor DB Performance, tune and re-organize DBs, enforce standards Support Users , implement change control procedures, plan DB growth and change

Major functions of DBA and DA in the DB development life cycle

Data Administrator	**DB planning and System definition-** Develop entry charts ,Analyze costs and benefits, develop implementation plan, evaluate and select software/hardware, establish application priorities, develop data standards
Data Administrator	**Requirements collection and analysis –** define user requirements, develop data definition, develop data dictionary
	Database Design-
DBA	Design conceptual model, external model
DBA	Design internal models and integrity controls
	DBMS selection ,Application design, Prototyping , Implementation, Data conversion and loading, Testing
DA & DBA	Specify DB access policies
DBA	Develop standards for application program, establish security techniques, load DBs, specify test procedures and backup & recovery procedures
DBA	**Operational maintenance –** Monitor DB Performance, tune and re-organize DBs, enforce standards
DA & DBA	Support Users , implement change control procedures, plan DB growth and change

References

- Connolly, T. M. & Begg, C. E. (2005). Database Systems A Practical Approach to Design, Implementation, and Management (4th Edition)

Entity-Relationship (E-R) Model

Part 2

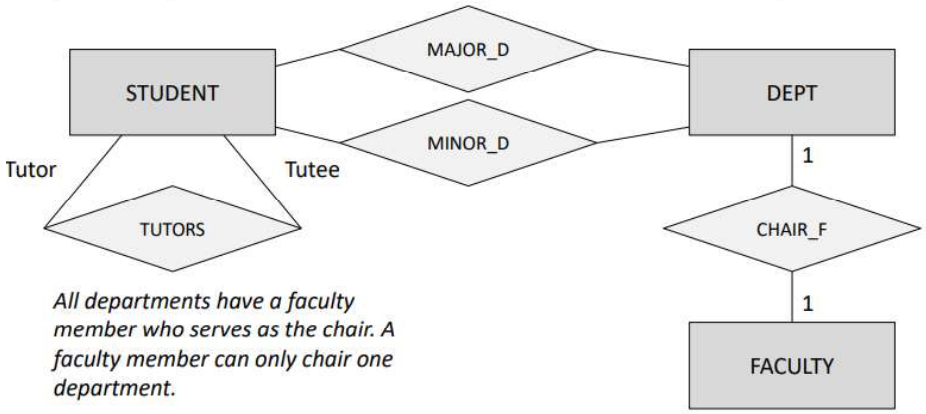
Structural Constraints on Relationship

- Cardinality constraints
  - Zero-to-one 0..1
  - Zero-to-many 0..*
  - One-to-many 1..*
  - Many-to-many *..*
- Participation constraints
  - Total (mandatory, every one involved)
  - Partial (optional, only some involved)
- Improper relationship
  - Fan trap (ambiguous pathway)
  - Chasm trap (missing pathway)

1 to 1
1:1

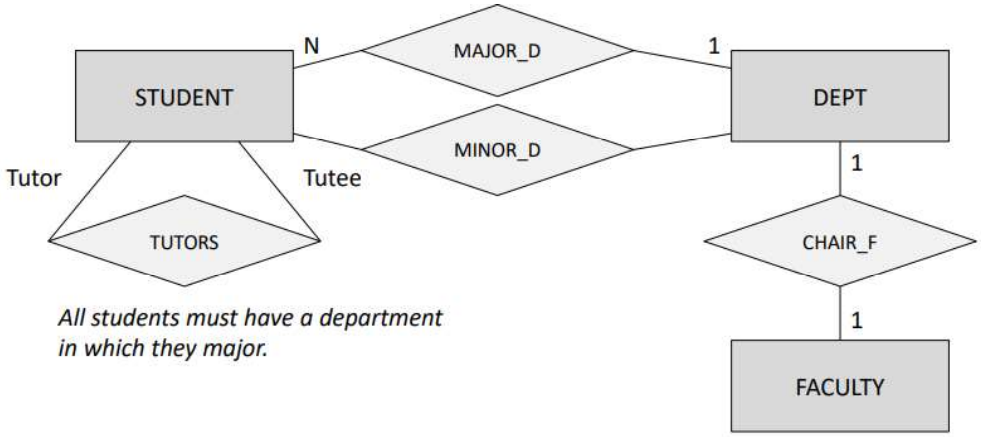
Cardinality Ratios

Constrains the number of entities that can participate in each role of the relationship



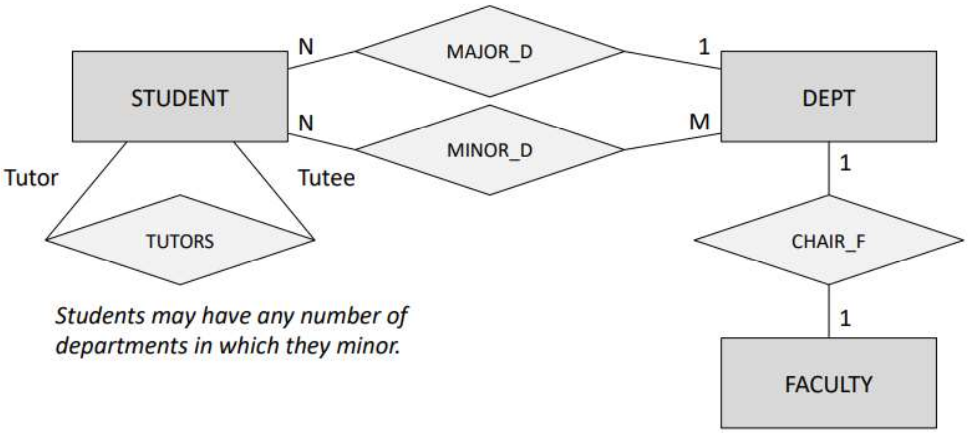
All departments have a faculty member who serves as the chair. A faculty member can only chair one department.

1 to n
1:n



All students must have a department in which they major.

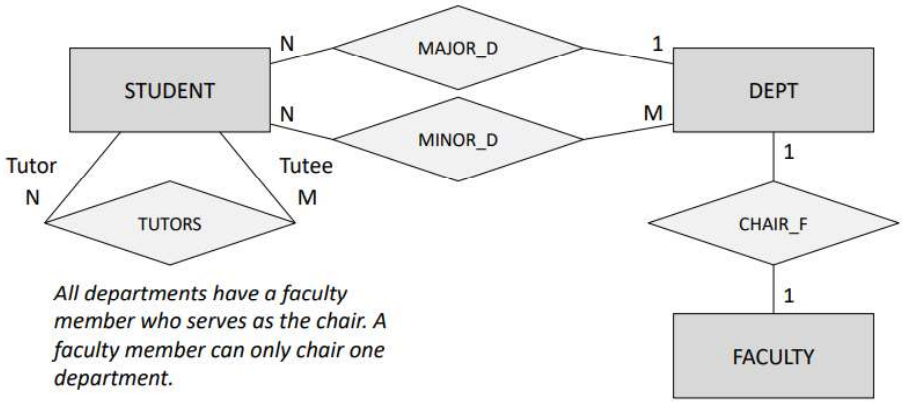
n to m
n:m



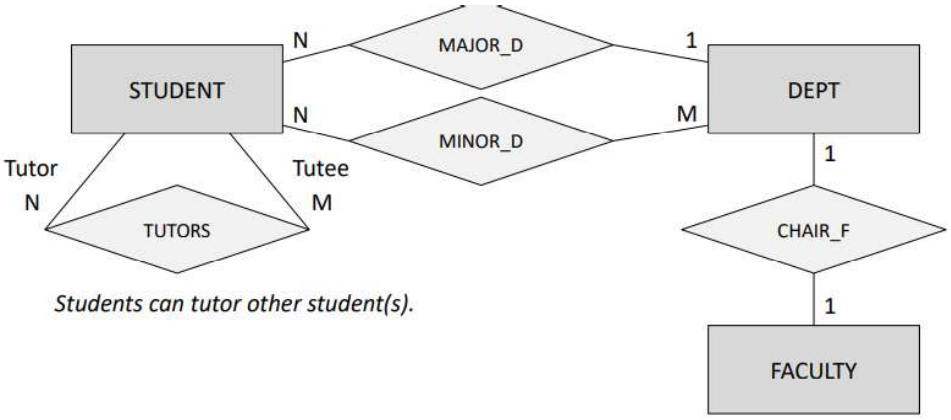
Students may have any number of departments in which they minor.

Structural Constraints

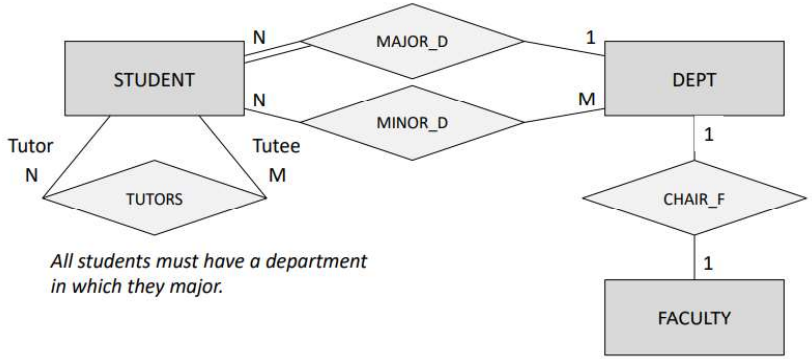
If an entity does not exist unless it appears with an entity in a relationship, the participation is **total** (existence dependency). Else, **partial**.



All departments have a faculty member who serves as the chair. A faculty member can only chair one department.



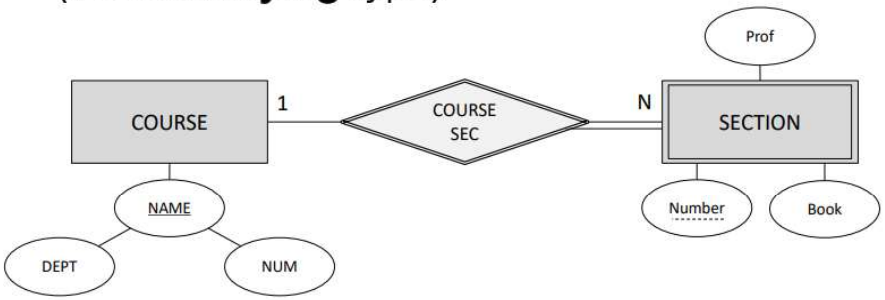
Students can tutor other student(s).



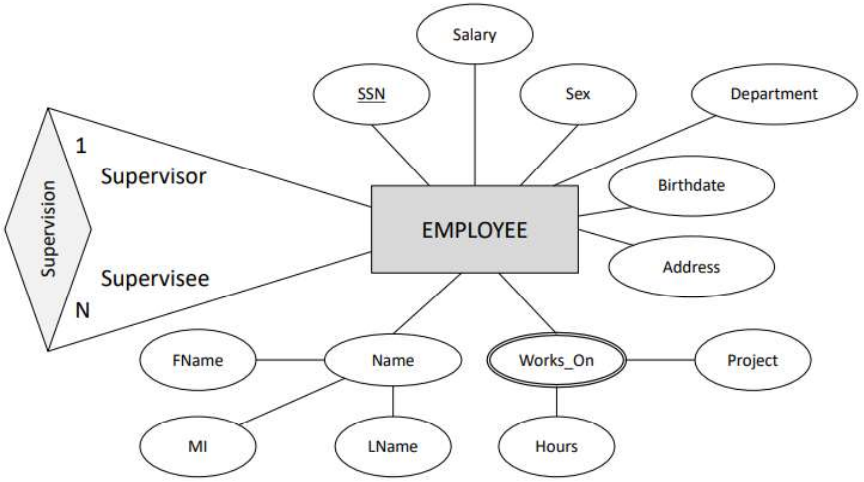
All students must have a department in which they major.

Weak Entities

Entity types that do not have key attributes of their own are **weak**; instead identified by relation to specific entity of another type (the **identifying** type)



Answer

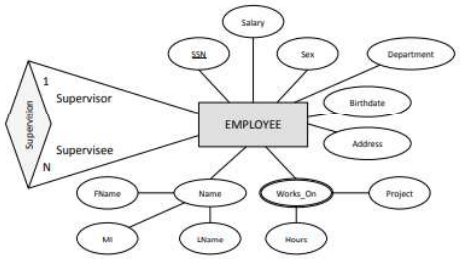


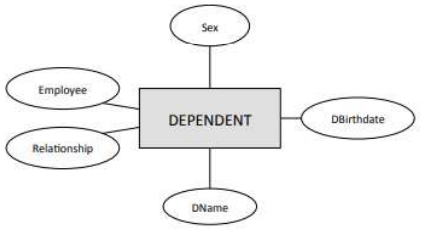
Example

We store each employee's name (first, last, MI), Social Security number (SSN), street 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. We keep track of the current number of hours per week that an employee works on each project. ***We also keep track of the direct supervisor of each employee (who is another employee).***

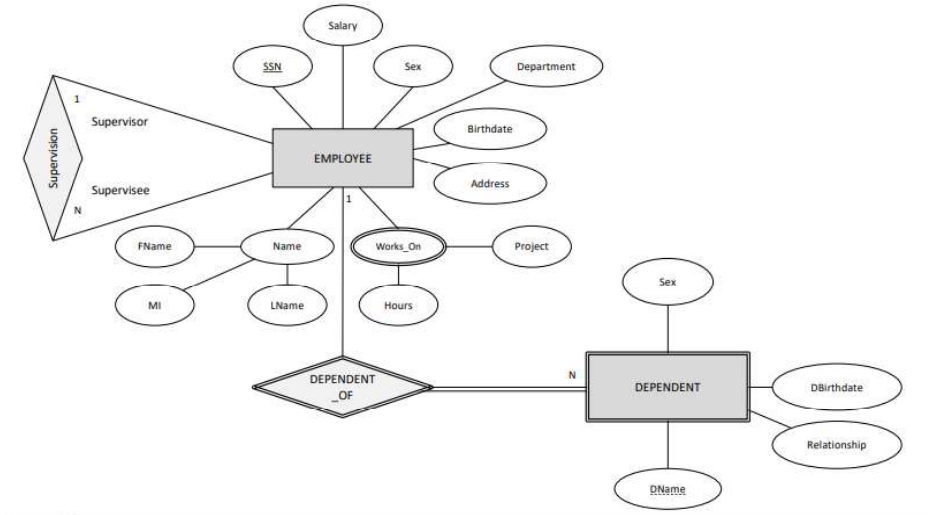
Example

We want to keep track of the dependents ***of each employee*** *for insurance purposes. We keep each dependent's first name, sex, birth date, and relationship to the employee.*

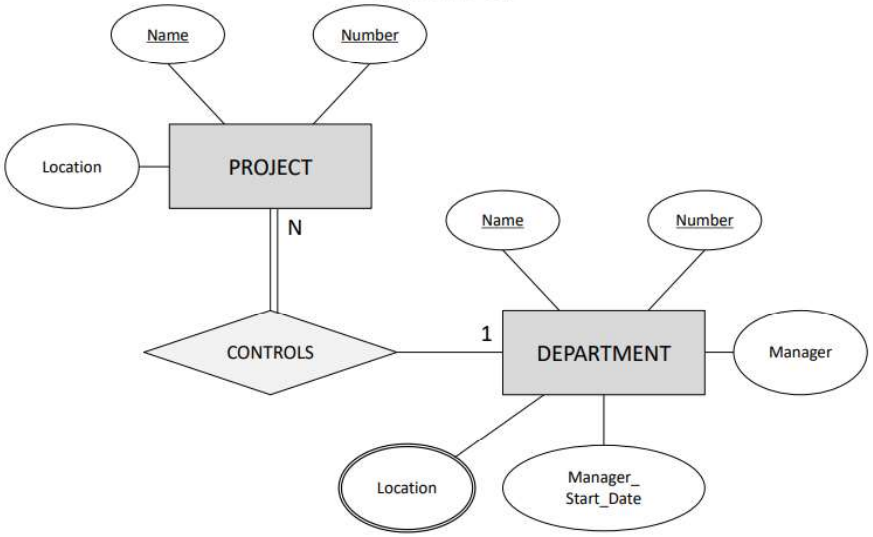




Answer

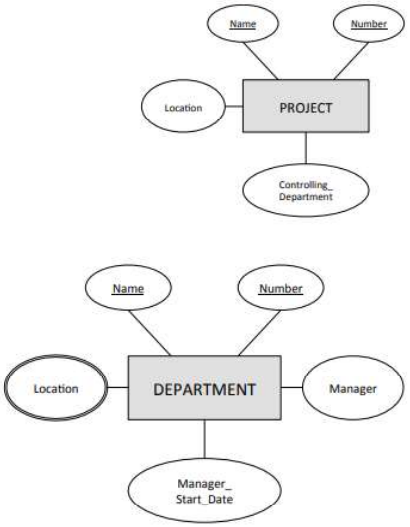


Answer



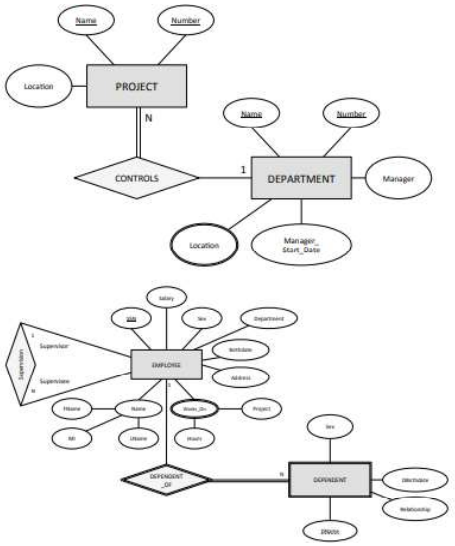
Example

A department controls *a number of projects, each of which has a unique name, a unique number, and a single location.*

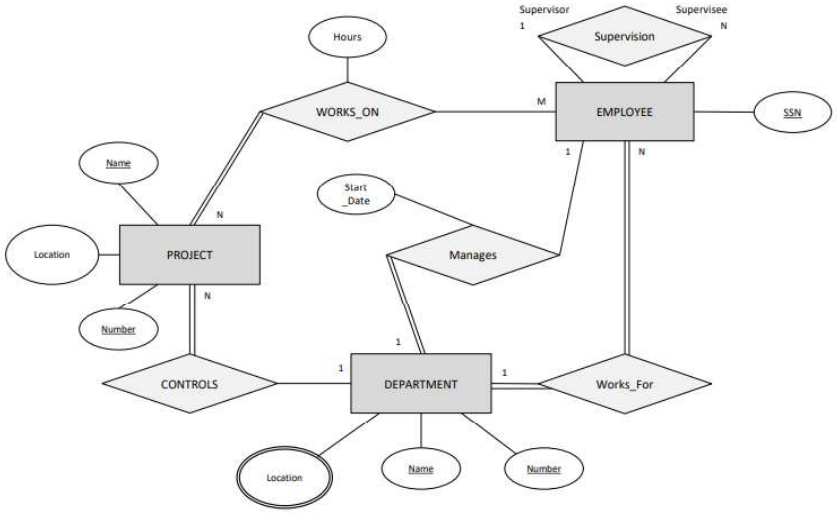


Each department has a … particular employee who manages the department.

An employee is assigned to one department, but may work on several projects, which are not necessarily controlled by the same department. We keep track of the current number of hours per week that an employee works on each project.

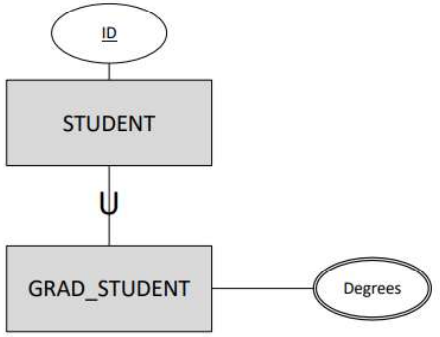


Answer



Specialization/Generalization

Only a subset of entities within a type have certain attributes or participate in certain relationships



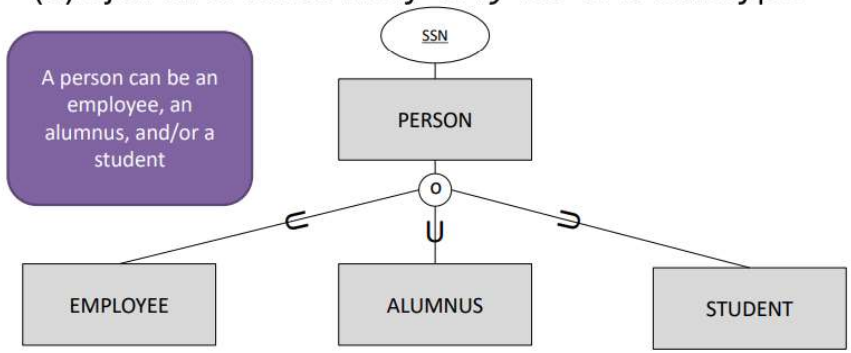
Enhanced Entity-Relationship Model

- Additional entity types
  - Superclass: including one or more distinct subgroups in the data model
  - Subclass: a distinct subgroup of an entity type in the data model
- Attribute Inheritance
  - Specialization hierarchy (specialization: maximizing the differences between members of an entity by identifying their distinguishing characteristics)
  - Generalization hierarchy (generalization: minimizing the differences between entities by identifying their common characteristics)
  - Is-A hierarchy
- Constraints on specialization/generalization
  - Participation (mandatory, optional)
  - Disjoint: disjoint (or), non-disjoint (and)
- Other
  - Aggregation (has a or is part of)
  - Composition (strong ownership of aggregation)

Multiple Subtypes: Disjointedness

(o)verlap: may be more than one

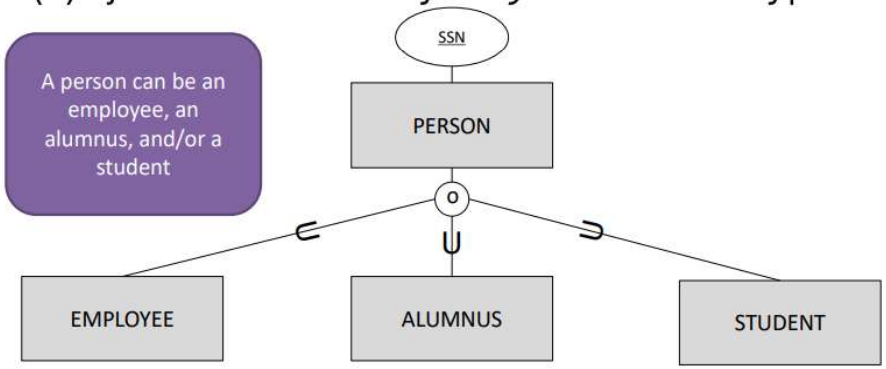
(d)isjoint: entities may *only be one* subtype



Multiple Subtypes: Disjointedness

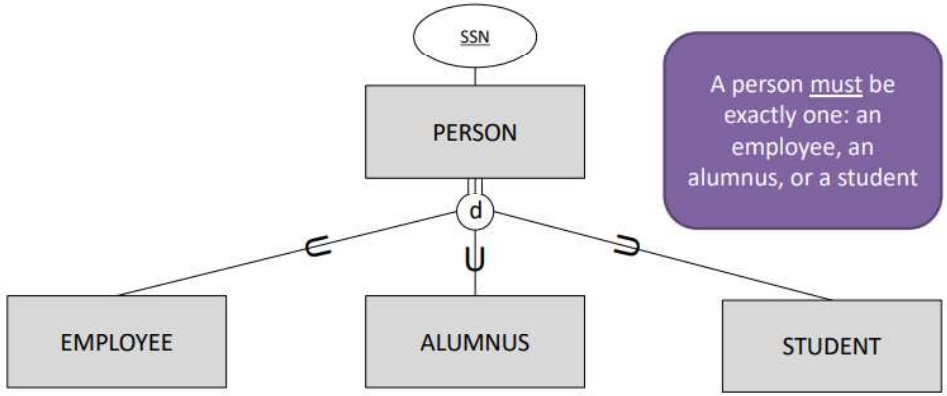
(o)verlap: may be more than one

(d)isjoint: entities may *only be one* subtype



Multiple Subtypes: Completeness

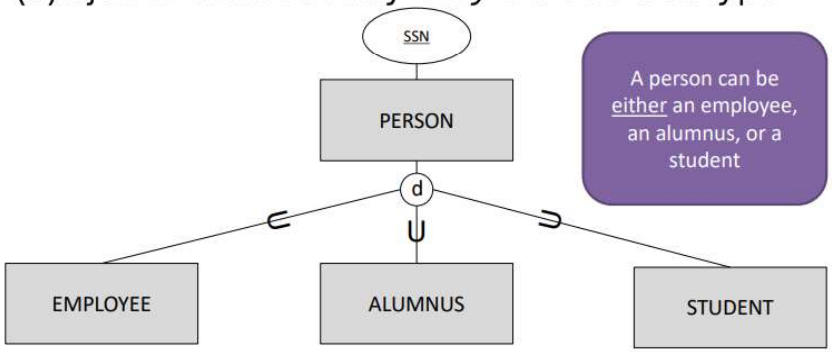
Similar to relationships; can be total (must belong to subtypes) or partial (can belong)



Multiple Subtypes: Disjointedness

(o)verlap: may be more than one

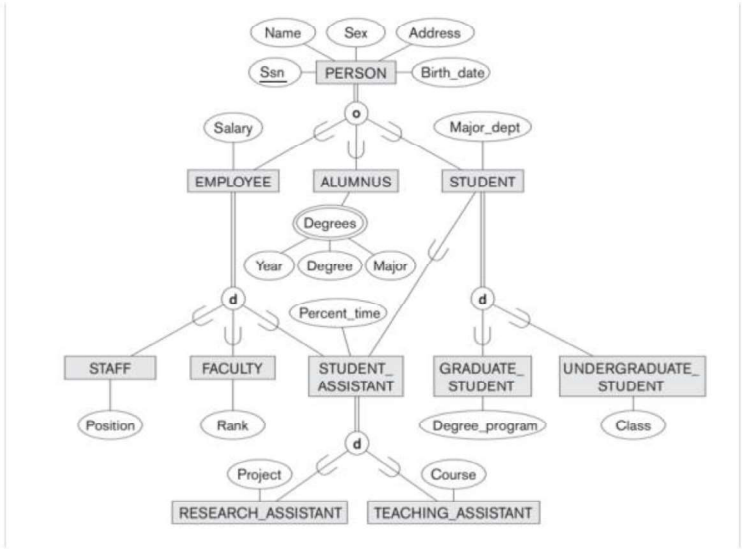
(d)isjoint: entities may *only be one* subtype



Exercise

- 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.
- 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.
- 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.
- 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 under- graduates.

Answer



- Create an enhanced ER diagram for a rental management using following entities:
  - Rental agency
  - Staff
    - Part time
    - Full time
  - Owner
  - Renter
  - Property
    - Business
    - Home

Design Steps

- Identify
  - Entity types, relationship types
  - Cardinality and participation constraints
  - Attributes
  - Keys
  - Specialize/generalize
  - EER diagram
- EER model example

A Sample Database Application

- COMPANY
  - Employees, departments, and projects
  - Company is organized into departments
  - Department controls a number of projects
  - Employee: store each employee's name, Social Security number, address, salary, sex (gender), and birth date
  - Keep track of the dependents of each employee

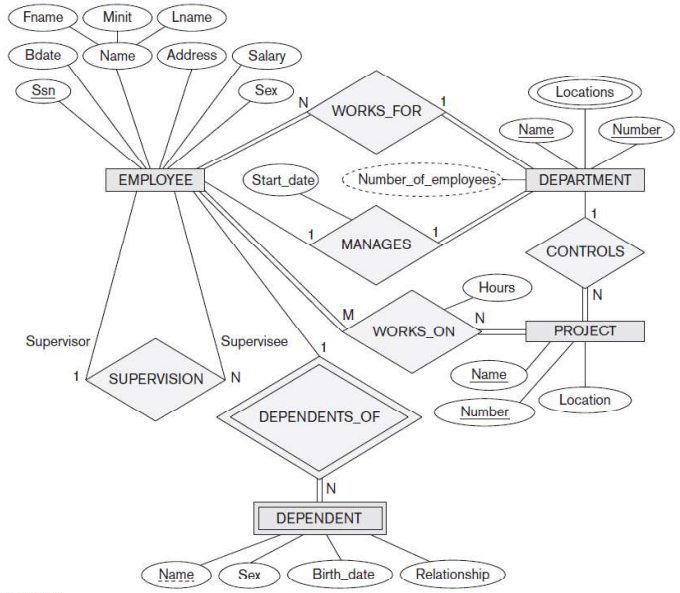


Figure 7.2
An ER schema diagram for the COMPANY database. The diagrammatic notation is introduced gradually throughout this chapter and is summarized in Figure 7.14.

Relational Database Design by ER- and EERR-to-Relational Mapping

HNDIT3042 Database Management Systems

Outline

- **ER-to-Relational Mapping Algorithm**
  - Step 1: Mapping of Regular Entity Types
  - Step 2: Mapping of Weak Entity Types
  - Step 3: Mapping of Binary 1:1 Relation Types
  - Step 4: Mapping of Binary 1:N Relationship Types.
  - Step 5: Mapping of Binary M:N Relationship Types.
  - Step 6: Mapping of Multivalued attributes.
  - Step 7: Mapping of N-ary Relationship Types.
- **Mapping EER Model Constructs to Relations**
  - Step 8: Options for Mapping Specialization or Generalization.
  - Step 9: Mapping of Union Types (Categories).

ER-to-Relational Mapping Algorithm

- Step 1: Mapping of Regular Entity Types.
  - For each regular (strong) entity type E in the ER schema, create a relation R that includes all the simple attributes of E.
  - Choose one of the key attributes of E as the primary key for R.
  - If the chosen key of E is composite, the set of simple attributes that form it will together form the primary key of R.
- Example: We create the relations EMPLOYEE, DEPARTMENT, and PROJECT in the relational schema corresponding to the regular entities in the ER diagram.
  - SSN, DNUMBER, and PNUMBER are the primary keys for the relations EMPLOYEE, DEPARTMENT, and PROJECT as shown.

FIGURE 7.1
The ER conceptual schema diagram for the COMPANY database.

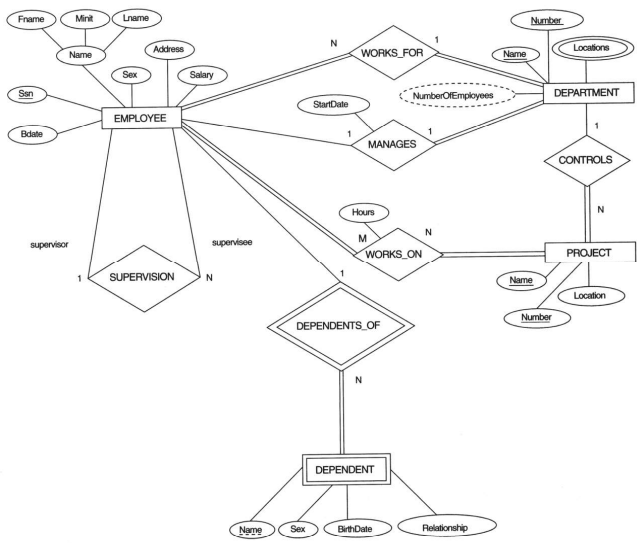


FIGURE 7.2
Result of mapping the COMPANY ER schema into a relational schema.

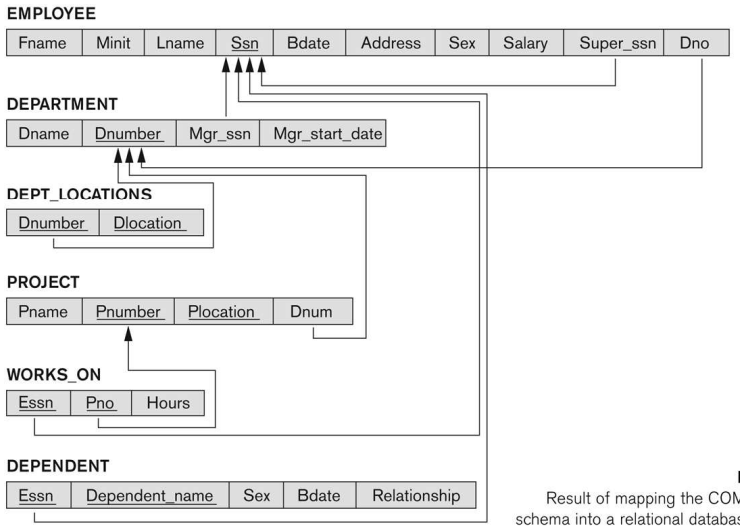


Figure 7.2
Result of mapping the COMPANY ER schema into a relational database schema.

ER-to-Relational Mapping Algorithm (contd.)

- **Step 3: Mapping of Binary 1:1 Relation Types**
  - For each binary 1:1 relationship type R in the ER schema, identify the relations S and T that correspond to the entity types participating in R.
- There are three possible approaches:
  1. **Foreign Key approach:** Choose one of the relations-say S-and include a foreign key in S the primary key of T. It is better to choose an entity type with total participation in R in the role of S.
     - Example: 1:1 relation MANAGES is mapped by choosing the participating entity type DEPARTMENT to serve in the role of S, because its participation in the MANAGES relationship type is total.
  2. **Merged relation option:** An alternate mapping of a 1:1 relationship type is possible by merging the two entity types and the relationship into a single relation. This may be appropriate when both participations are total.
  3. **Cross-reference or relationship relation option:** The third alternative is to set up a third relation R for the purpose of cross-referencing the primary keys of the two relations S and T representing the entity types.

ER-to-Relational Mapping Algorithm (contd.)

- **Step 2: Mapping of Weak Entity Types**
  - For each weak entity type W in the ER schema with owner entity type E, create a relation R & include all simple attributes (or simple components of composite attributes) of W as attributes of R.
  - Also, include as foreign key attributes of R the primary key attribute(s) of the relation(s) that correspond to the owner entity type(s).
  - The primary key of R is the *combination of* the primary key(s) of the owner(s) and the partial key of the weak entity type W, if any.
- **Example:** Create the relation DEPENDENT in this step to correspond to the weak entity type DEPENDENT.
  - Include the primary key SSN of the EMPLOYEE relation as a foreign key attribute of DEPENDENT (renamed to ESSN).
  - The primary key of the DEPENDENT relation is the combination {ESSN, DEPENDENT_NAME} because DEPENDENT_NAME is the partial key of DEPENDENT.

ER-to-Relational Mapping Algorithm (contd.)

- Step 4: Mapping of Binary 1:N Relationship Types.
  - For each regular binary 1:N relationship type R, identify the relation S that represent the participating entity type at the N-side of the relationship type.
  - Include as foreign key in S the primary key of the relation T that represents the other entity type participating in R.
  - Include any simple attributes of the 1:N relation type as attributes of S.
- Example: 1:N relationship types WORKS_FOR, CONTROLS, and SUPERVISION in the figure.
  - For WORKS_FOR we include the primary key DNUMBER of the DEPARTMENT relation as foreign key in the EMPLOYEE relation and call it DNO.

ER-to-Relational Mapping Algorithm (contd.)

- **Step 5: Mapping of Binary M:N Relationship Types.**
  - For each regular 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; *their combination will form the primary key* of S.
  - Also include any simple attributes of the M:N relationship type (or simple components of composite attributes) as attributes of S.
- Example: The M:N relationship type WORKS_ON from the ER diagram is mapped by creating a relation WORKS_ON in the relational database schema.
  - The primary keys of the PROJECT and EMPLOYEE relations are included as foreign keys in WORKS_ON and renamed PNO and ESSN, respectively.
  - Attribute HOURS in WORKS_ON represents the HOURS attribute of the relation type. The primary key of the WORKS_ON relation is the combination of the foreign key attributes {ESSN, PNO}.

ER-to-Relational Mapping Algorithm (contd.)

- **Step 6: Mapping of Multivalued attributes.**
  - For each multivalued attribute A, create a new relation R.
  - This relation R will include an attribute corresponding to A, plus the primary key attribute K-as a foreign key in R-of the relation that represents the entity type of relationship type that has A as an attribute.
  - The primary key of R is the combination of A and K. If the multivalued attribute is composite, we include its simple components.
- **Example:** The relation DEPT_LOCATIONS is created.
  - The attribute DLOCATION represents the multivalued attribute LOCATIONS of DEPARTMENT, while DNUMBER-as foreign key-represents the primary key of the DEPARTMENT relation.
  - The primary key of R is the combination of {DNUMBER, DLOCATION}.

ER-to-Relational Mapping Algorithm (contd.)

- **Step 7: Mapping of N-ary Relationship Types.**
  - For each n-ary relationship type R, where n>2, create a new relationship S to represent R.
  - Include as foreign key attributes in S the primary keys of the relations that represent the participating entity types.
  - Also include any simple attributes of the n-ary relationship type (or simple components of composite attributes) as attributes of S.
- **Example:** The relationship type SUPPY in the ER on the next slide.
  - This can be mapped to the relation SUPPLY shown in the relational schema, whose primary key is the combination of the three foreign keys {SNAME, PARTNO, PROJNAME}

FIGURE 4.11
Ternary relationship types. (a) The SUPPLY relationship.

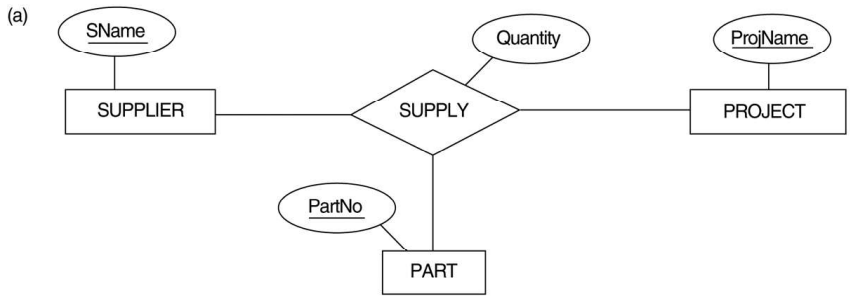
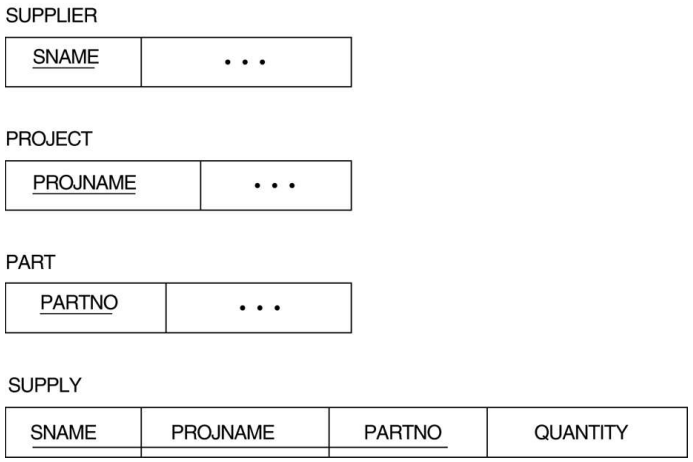


FIGURE 7.3
Mapping the *n*-ary relationship type SUPPLY from Figure 4.11a.



Summary of Mapping constructs and constraints

Table 7.1 Correspondence between ER and Relational Models

ER Model	Relational Model
Entity type	"Entity" relation
1:1 or 1:N relationship type	Foreign key (or "relationship" relation)
M:N relationship type	"Relationship" relation and two foreign keys
n-ary relationship type	"Relationship" relation and n foreign keys
Simple attribute	Attribute
Composite attribute	Set of simple component attributes
Multivalued attribute	Relation and foreign key
Value set	Domain
Key attribute	Primary (or secondary) key

Mapping EER Model Constructs to Relations

- **Step8: Options for Mapping Specialization or Generalization.**
  - Convert each specialization with m subclasses {S1, S2,….,Sm} and generalized superclass C, where the attributes of C are {k,a1,…an} and k is the (primary) key, into relational schemas using one of the four 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

Mapping EER Model Constructs to Relations

- **Option 8A: Multiple relations-Superclass and subclasses**
  - Create a relation L for C with attributes Attrs(L) = {k,a1,…an} and PK(L) = k. Create a relation Li for each subclass Si, 1 < i < m, with the attributesAttrs(Li) = {k} U {attributes of Si} and PK(Li)=k. This option works for any specialization (total or partial, disjoint of over-lapping).
- **Option 8B: Multiple relations-Subclass relations only**
  - Create a relation Li for each subclass Si, 1 < i < m, with the attributes Attr(Li) = {attributes of Si} U {k,a1…,an} and PK(Li) = k. This option only works for a specialization whose subclasses are total (every entity in the superclass must belong to (at least) one of the subclasses).

FIGURE 4.4
EER diagram notation for an attribute-defined specialization on JobType.

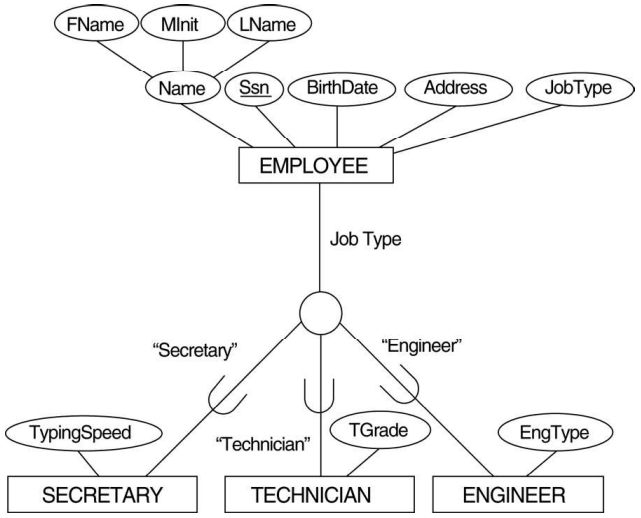


FIGURE 4.3
Generalization. (b) Generalizing CAR and TRUCK into the superclass VEHICLE.

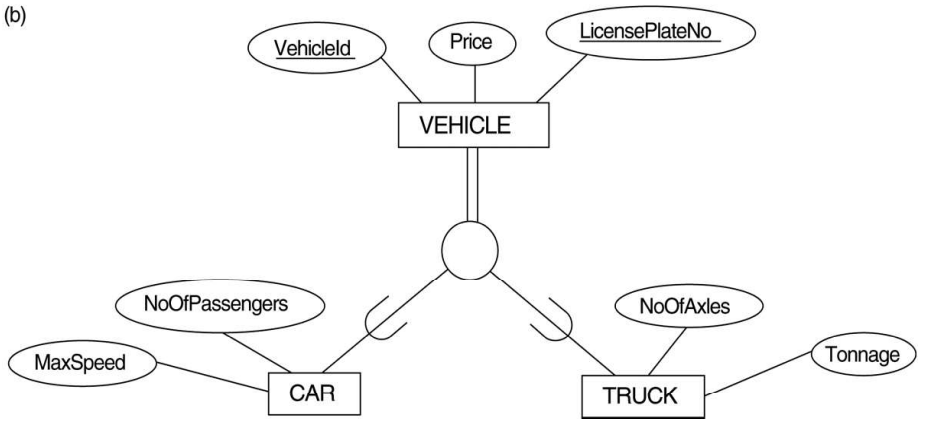


FIGURE 7.4
Options for mapping specialization or generalization.
(a) Mapping the EER schema in Figure 4.4 using option 8A.

(a) EMPLOYEE

SSN	FName	MInit	LName	BirthDate	Address	JobType

SECRETARY

SSN	TypingSpeed

TECHNICIAN

SSN	TGrade

ENGINEER

SSN	EngType

FIGURE 7.4
Options for mapping specialization or generalization.
(b) Mapping the EER schema in Figure 4.3b using option 8B.

(b) CAR

VehicleId	LicensePlateNo	Price	MaxSpeed	NoOfPassengers

TRUCK

VehicleId	LicensePlateNo	Price	NoOfAxles	

Mapping EER Model Constructs to Relations (contd.)

- **Option 8C: Single relation with one type attribute**
  - Create a single relation L with attributes Attrs(L) = {k,a_1,…a_n} U {attributes of S_1} U…U {attributes of S_m} U {t} and PK(L) = k. The attribute t is called a type (or **discriminating**) attribute that indicates the subclass to which each tuple belongs
- **Option 8D: Single relation with multiple type attributes**
  - Create a single relation schema L with attributes Attrs(L) = {k,a_1,…a_n} U {attributes of S_1} U…U {attributes of S_m} U {t_1, t_2,…,t_m} and PK(L) = k. Each t_i, 1 < I < m, is a Boolean type attribute indicating whether a tuple belongs to the subclass S_i.

FIGURE 4.4
EER diagram notation for an attribute-defined specialization on JobType.

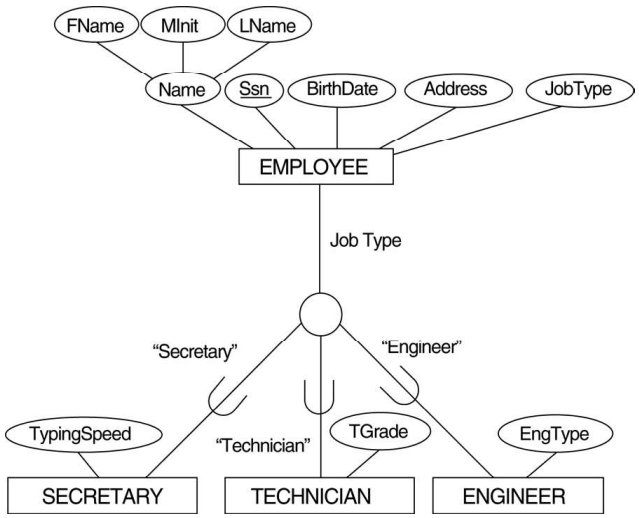


FIGURE 7.4
Options for mapping specialization or generalization.
(c) Mapping the EER schema in Figure 4.4 using option 8C.

(c) EMPLOYEE

SSN	FName	MInit	LName	BirthDate	Address	JobType	TypingSpeed	TGrade	

FIGURE 4.5
EER diagram notation for an overlapping (non-disjoint) specialization.

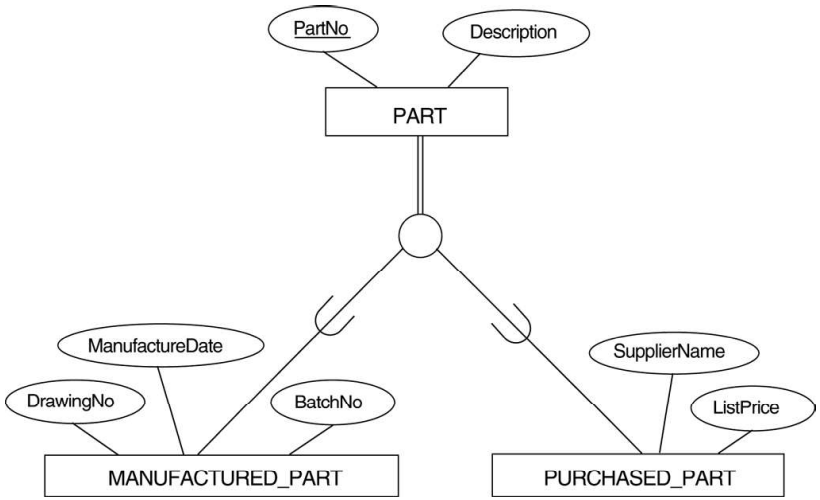


FIGURE 7.4
Options for mapping specialization or generalization. (d) Mapping Figure 4.5 using option 8D with Boolean type fields Mflag and Pflag.

(d) PART

PartNo	Description	MFlag	DrawingNo	ManufactureDate	DatchNo	PFlag	SupplierName	ListPrice

Mapping EER Model Constructs to Relations (contd.)

- Mapping of Shared Subclasses (Multiple Inheritance)
  - A shared subclass, such as STUDENT_ASSISTANT, is a subclass of several classes, indicating multiple inheritance. These classes must all have the same key attribute; otherwise, the shared subclass would be modeled as a category.
  - We can apply any of the options discussed in Step 8 to a shared subclass, subject to the restriction discussed in Step 8 of the mapping algorithm. Below both 8C and 8D are used for the shared class STUDENT_ASSISTANT.

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

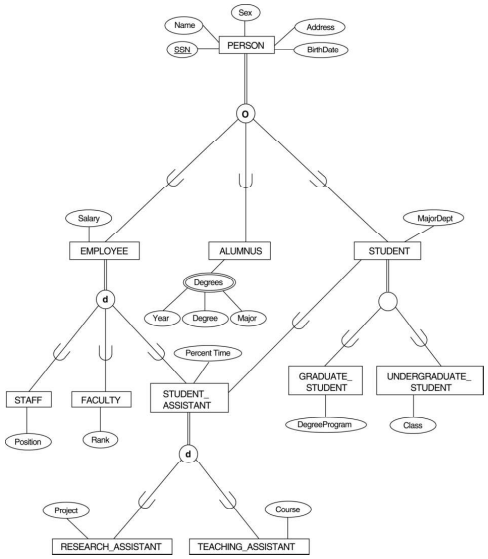


FIGURE 7.5
Mapping the EER specialization lattice in Figure 4.6 using multiple options.

PERSON

SSN	Name	BirthDate	Sex	Address

EMPLOYEE

SSN	Salary	EmployeeType	Position	Rank	PercentTime	RAFlag	TAFlag	Project	

ALUMNUS

SSN

ALUMNUS_DEGREES

SSN	Year	Degree	

STUDENT

SSN	MajorDept	GradFlag	UndergradFlag	DegreeProgram	Class	StudAssistFlag

Mapping EER Model Constructs to Relations (contd.)

- **Step 9: Mapping of Union Types (Categories).**
  - For mapping a category whose defining superclass have different keys, it is customary to specify a new key attribute, called a surrogate key, when creating a relation to correspond to the category.
  - In the example below we can create a relation OWNER to correspond to the OWNER category and include any attributes of the category in this relation. The primary key of the OWNER relation is the surrogate key, which we called OwnerId.

FIGURE 7.6
Mapping the EER categories (union types) in Figure 4.7 to relations.

PERSON

SSN	DriverLicenseNo	Name	Address	

BANK

BName	BAddress	OwnerId

COMPANY

CName	CAddress	OwnerId

OWNER

OwnerId

REGISTERED_VEHICLE

VehicleId	LicensePlateNumber

CAR

VehicleId	CStyle	CMake	CModel	

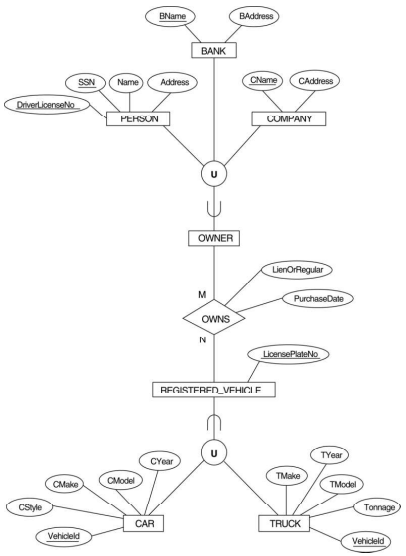
TRUCK

VehicleId	TMake	TModel	Tonnage	TYear

OWNS

OwnerId	VehicleId	PurchaseDate	LienOrRegular

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



Mapping Exercise

Exercise 7.4.

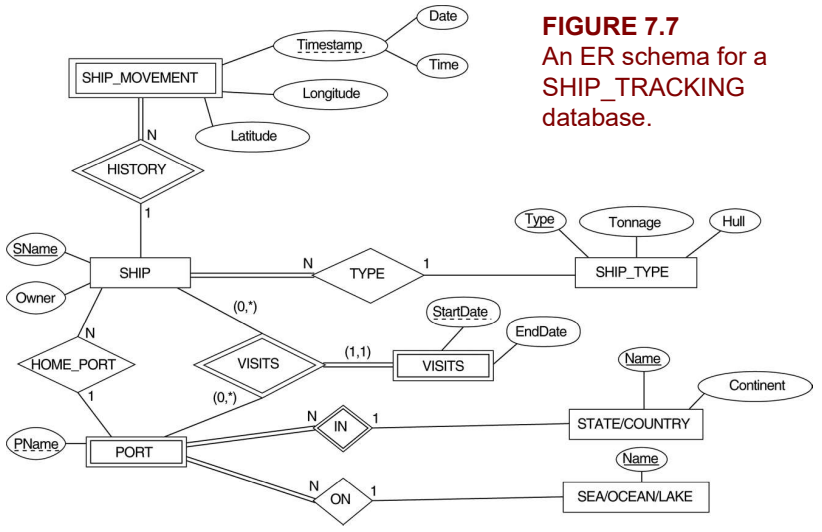


FIGURE 7.7
An ER schema for a SHIP_TRACKING database.

DataBase Management System

Database Design

Introduction

- Relations derived from ER model may be 'faulty'
  - Subjective process.
  - May cause data redundancy, and insert/delete/update anomalies.
- We use some mathematical (semantic?) properties of relations to
  - locate these faults and
  - fix them
- Process is called **Normalization.**

Database Design

- Database design process can be divided into 6 major steps:
  1. Requirements Analysis
  2. Conceptual Database Design
  3. Logical Database Design
  4. Schema Refinement
  5. Physical Database Design
  6. Security Design

Data Redundancy

- Major aim of relational database design is
  - to group attributes into relations to minimize data redundancy and
  - to reduce file storage space required by base relations.
- Data redundancy is undesirable because of the following anomalies
  - 'Insert' anomalies
  - 'Delete' anomalies
  - 'Update' anomalies

Anomalies

Too many attributes…

For example,

LECTURER (id, name, address, salary, department, building)

Anomalies (contd.)

- Inserting a department with no teacher
  - (Impossible – b/c null values for *id* is not allowed)

Anomalies (contd.)

Insertion Anomaly…

1. Inserting a new lecturer to the Teacher table

   - Department information is repeated (ensure that correct department information is inserted).

Teacher (id, name, address, salary, department, building)

Anomalies (contd.)

Deletion Anomalies…

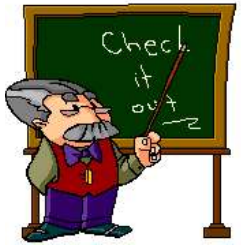
- Deleting the last lecturer from the department will lose information about the department.

LECTURER (id, name, address, salary, department, building)

Anomalies (contd.)

Updating Anomalies…

- Updating the department's building needs to be done for all lecturers working for that department.



Teacher (id, name, address, salary, department, building)

Loss-less join property

Decomposing the relation into too smaller relations…

- **Loss-less join property**: we might lose information if we decompose relations…

Decomposition of Relations

- Staff and Branch relations which are obtained by decomposing StaffBranch do not suffer from these anomalies.
- Two important properties of decomposition
  - **Lossless-join property** enables us to find any instance of original relation from corresponding instances in the smaller relations.
  - **Dependency preservation property** enables us to enforce a constraint on original relation by enforcing some constraint on each of the smaller relations.

Loss-less join property (contd.)

For example,

S

S	P	D
S1	P1	D1
S2	P2	D2
S3	P1	D3

R_1

S	P
S1	P1
S2	P2
S3	P1

R_2

P	D
P1	D1
P2	D2
P1	D3

Loss-less join property (contd.)

Joining them together, we get spurious tuples…

$R_1 \bowtie R_2$

S	P	D
S1	P1	D1
S1	P1	D3
S2	P2	D2
S3	P1	D1
S3	P1	D3

Functional dependency

- A functional dependency, denoted by X→ Y,
  - X functionally determines Y
  - Y is functionally dependent on X
- where X and Y are sets of attributes in relation R, specifies the following constraint:
  Let t_1 and t_2 be tuples of relation R for any given instance
  Whenever $t_1[X] = t_2[X]$ then $t_1[Y] = t_2[Y]$

  where $t_i[X]$ represents the values for X in tuple t_i

The Process of Normalization

- Formal technique for analyzing a relation based on its primary key and functional dependencies between its attributes.
- Often executed as a series of steps. Each step corresponds to a specific normal form, which has known properties.
- As normalization proceeds, relations become progressively more restricted (stronger) in format and also less vulnerable to update anomalies.
- Given a relation, use the following cycle
  - Find out what normal form it is in
  - Transform the relation to the next higher form by decomposing it to form simpler relations
  - You may need to refine the relation further if decomposition resulted in undesirable properties

Normalization is based on Functional Dependencies

Functional dependency (contd.)

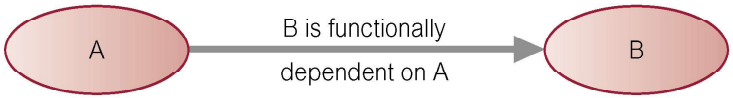
TEACH

STUDENT	COURSE	TEACHER
Narayana	Database	ABC
Sumith	Database	ABC
Nalin	Operating Systems	Samantha
Kamal	Mathematics	Chandrika
Janith	Database	ABC
Ranil	Operating Systems	Samantha
Saman	Mathematics	Chandrika
Ruwan	Database	ABC

TEACHER → COURSE

Functional Dependency

- Diagrammatic representation:



- *Determinant* of a functional dependency refers to attribute or group of attributes on left-hand side of the arrow.
- If the determinant can maintain the functional dependency with a minimum number of attributes, then we call it full functional dependency.

Key Terms

- **Candidate Key**: Each key of a relation is called a candidate key.
- **Primary Key**: A candidate key is chosen to be the primary key.
- **Prime Attribute**: an attribute which is a member of a candidate key.
- **Nonprime Attribute**: An attribute which is not prime.

Key Terms

Review of some terms…

- **Superkey**: Set of attributes S in relation R that can be used to identify each tuple uniquely.
- **Key**: A key is a superkey with the additional property that removal of any attributes from the key will not satisfy the key condition.

Unnormalized Form (UNF)

- A table that contains one or more repeating groups.
- To create an unnormalized table:
  - Transform data from information source (e.g. form) into table format with columns and rows.

Example 1 – address and name fields are composite

Name	Address	Phone
Sally Singer	123 Broadway New York, NY, 11234	(111) 222-3345
Jason Jumper	456 Jolly Jumper St. Trenton NJ, 11547	(222) 334-5566

Another example of UNF

Example 2 – repeating columns for each client & composite name field

Rep ID	Representative	Client 1	Time 1	Client 2	Time 2	Client 3	Time 3
TS-89	Gilroy Gladstone	US Corp.	14 hrs	Taggarts	26 hrs	Kilroy Inc.	9 hrs
RK-56	Mary Mayhem	Italiana	67 hrs	Linkers	2 hrs		

Normalization (contd.)

1st Normal Form

- A relation R is in first normal form (1NF) if domains of all attributes in the relation are *atomic* (simple & indivisible).

- Avoid multivalued & composite attributes.

UNF to 1NF

- Remove repeating group by:
  - entering appropriate data into the empty columns of rows containing repeating data ('flattening' the table).

Or by

  - placing repeating data along with copy of the original key attribute(s) into a separate relation.

Normalization (contd.)

For example…

DEPARTMENT (Dname,Dnumber, DMGRSSN, DLocation)

DEPARTMENT

DNAME	DNUMBER	DMGRSSN	DLOCATIONS
Research	5	333445555	{Mathara, Kandy, Metro}
Administratio n	4	987654321	{Malabe}
Headquarters	1	888665555	{Metro}

- Department relation not in 1NF
- How to take into 1NF ?

Normalization (contd.)

- Solution 1: Create a separate DLOCATION relation with foreign key.
- Solution 2: If max number of locations is known, create a column for each location (may have lots of null values).
- Solution 3: Repeat the same info (redundancy + new key attribute).

Normalization (contd.)

Solution 2:

DEPARTMENT

DNAME	DNUMBER	DMGRSSN	DLOC1	DLOC2	DLOC3
Research	5	333445555	Mathara	Kandy	Metro
Administration	4	987654321	Malabe	Null	Null
Headquarters	1	888665555	Metro	Null	Null

- Need to know max number of locations.
- create a column for each location.
- may have lots of null values.

Normalization (contd.)

Solution 1:

DEPARTMENT

DNAME	DNUMBER	DMGRSSN	DLOCATIONS
Research	5	333445555	Mathara
Research	5	333445555	Kandy
Research	5	333445555	Metro
Administration	4	987654321	Malabe
Headquarters	1	888665555	Metro

1NF relation with redundancy

- Expand the key so that there will be a separate tuple in the original DEPARTMENT relation for each location of the DEPARTMENT.
- This solution has the disadvantage of introducing redundancy in the relation.

Normalization (contd.)

Solution 3:

DEPT_LOCATIONS

DNUMBER	DLOCATIONS
1	Metro
4	Malabe
5	Mathara
5	Kandy
5	Metro

DEPARTMENT

DNAME	DNUMBER	DMGRSSN
Research	5	333445555
Administration	4	987654321
Headquarters	1	888665555

- Remove the attribute DLOCTION and place it in a separate relation DEPT_LOCTIONS along with the primary key DNUMBER of DEPARTMENT.
- The PK is the combination {DNUMBER, DLOCTION}
- This decompose the non-INF relation into two INF relation.

Normalization (contd.)

- **A functional dependency, X → Y is a full functional dependency if removal of any attribute A from X means that the dependency does not hold (i.e. (X –{A}) → Y does not hold)**

TEACH

STUDENT	COURSE	TEACHER	CAMPUS
Narayan	Database	ABC	Metro
Smith	Database	XYZ	Malabe
Nalin	Operating Systems	Samantha	Metro
Kamal	Operating Systems	ABC	Malabe
Janith	Database	ABC	Metro
Ranil	Operating Systems	Samantha	Metro
Saman	Operating Systems	ABC	Malabe
Ruwan	Database	XYZ	Malabe

{Teacher, Campus} → Course

Normalization (contd.)

- Lossless join decomposition:
  - Decomposition of R into X and Y is *lossless-join* w.r.t. a set of FDs F if, for every instance *r* that satisfies F:
    - $\Pi_X(r) \times \Pi_Y(r) = r$

This condition holds if attributes common to X and Y contains a key for either X or Y

Normalization (contd.)

2nd Normal Form:

- A relation R is in second normal form (2NF) if every nonprime attribute A in R is not **partially dependent** on any key of R.

Example: Not in 2NF

TEACHER	CAMPUS	COURSE	ADDRESS
ABC	Metro	Database	BoC Merchant Tower
XYZ	Malabe	Database	Malabe Campus
Samantha	Metro	Operating Systems	BoC Merchant Tower
ABC	Malabe	Operating Systems	Malabe Campus

Removing partial dependency

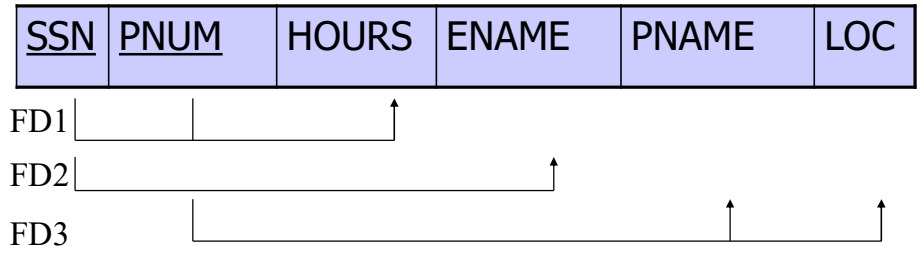
- Place the attributes that create the partial dependency in a separate table.
- Make sure that the new table's primary key is left in the original table.

Normalization (contd.)

TEACHER	CAMPUS	COURSE	ADDRESS
ABC	Metro	Database	BoC Merchant Tower
XYZ	Malabe	Database	Malabe Campus
Samantha	Metro	Operating Systems	BoC Merchant Tower
ABC	Malabe	Operating Systems	Malabe Campus

Another Example

EMP_PROJ



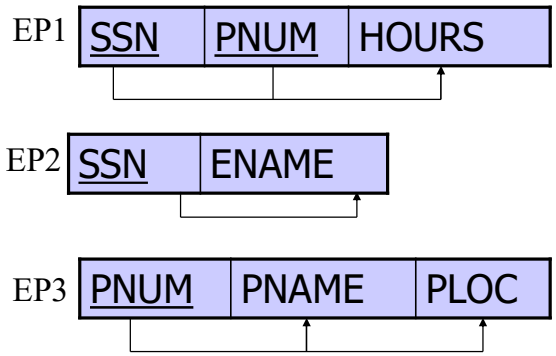
Normalization (contd.)

Example: After normalized into 2NF

TEACHER	CAMPUS	COURSE
ABC	Metro	Database
XYZ	Malabe	Database
Samantha	Metro	Operating Systems
ABC	Malabe	Operating Systems

CAMPUS	ADDRESS
Metro	BoC Merchant Tower
Malabe	Malabe Campus

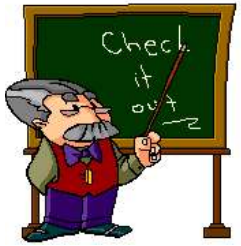
Normalization (contd.)



Normalization (contd.)

3rd Normal Form:

- A relation R is in 3rd normal form (3NF) if every
  - R is in 2NF, and
  - No nonprime attribute is **transitively dependent** on any key.



Removing transitive dependency

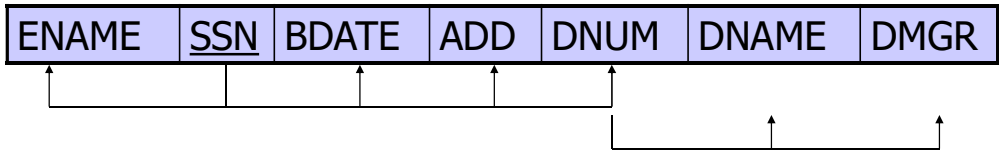
- Place the attributes that create the transitive dependency in a separate table.
- Make sure that the new table's primary key attribute is the foreign key in the original table.

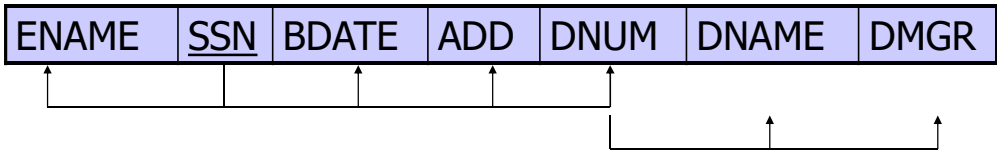
Transitive dependency

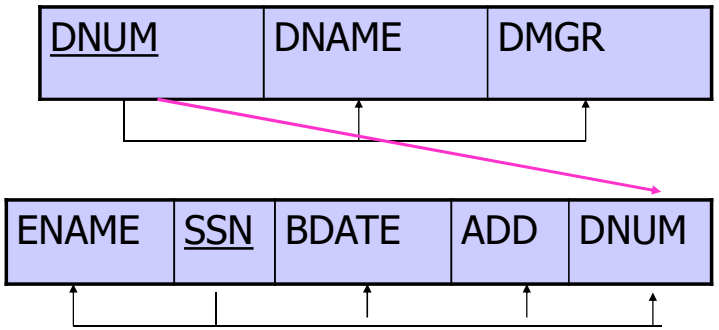
Attribute is dependent on another attribute that is not part of the primary key.

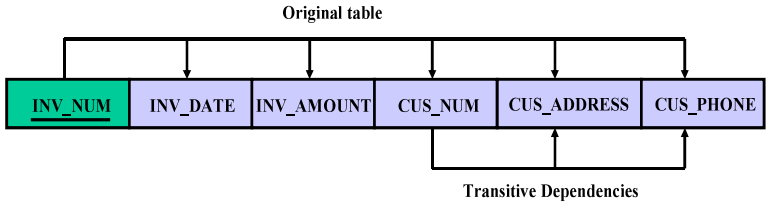
Requires the decomposition of the table containing the transitive dependency.

EMP_DEPT









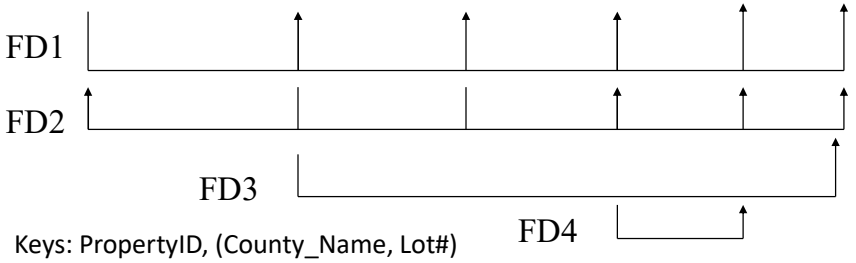
Is the table in 3NF?

Why?

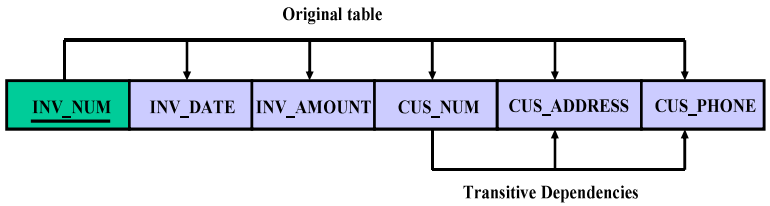
Remove Transitive Dependency

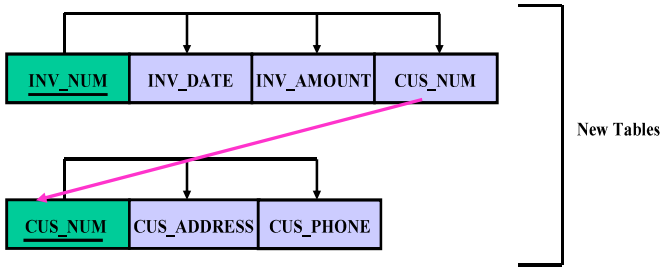
Normalization (contd.)

PROPERTY_ ID	COUNTY _NAME	LOT#	AREA	PRICE	TAX_ RATE



Keys: PropertyID, (County_Name, Lot#)



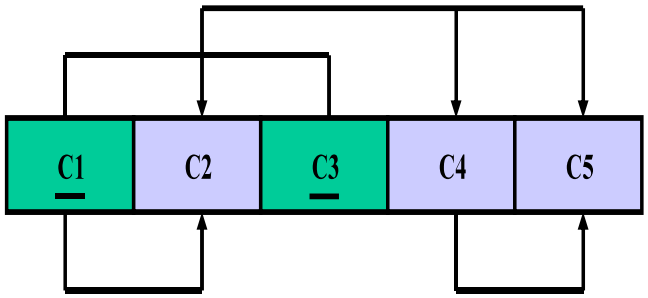


Normalization (contd.)

- 1NF, 2NF & 3NF guarantee to preserve **lossless join property.**

Your Turn !!

Dependency diagram



Identify the dependencies shown in the above diagram

C1->C2	**partial dependency**
C4 ->C5	**transitive dependency**
C1,C3 -> C2,C4,C5	**functional dependency**

Create a database whose tables are at least in 3NF

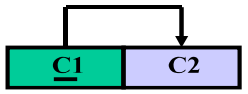


Table 1
Primary key: C1
Foreign key: None
Normal form: 3NF

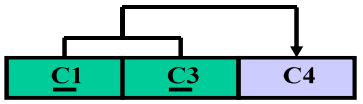


Table 2
Primary key: C1 + C3
Foreign key: C1 (to Table 1)
C4 (to Table 3)
Normal form: 3NF

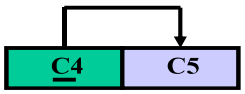


Table 3
Primary key: C4
Foreign key: None
Normal form: 3NF

- **Create a database whose tables are at least in 2NF, showing the dependency diagrams for each table.**

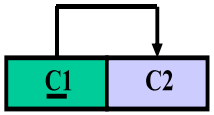


Table 1

Primary key: C1
Foreign key: None
Normal form: 3NF

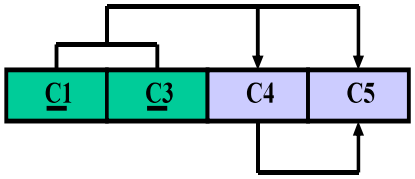


Table 2

Primary key: C1 + C3
Foreign key: C1 (to Table 1)
Normal form: 2NF, because the table exhibits the transitive dependencies C4 ⟶ C5

Normalization (contd.)

- Denormalization…

Sometime for performance reasons, database designer may leave the relation in a lower normal form. This process is known as **denormalization**.

Normalization

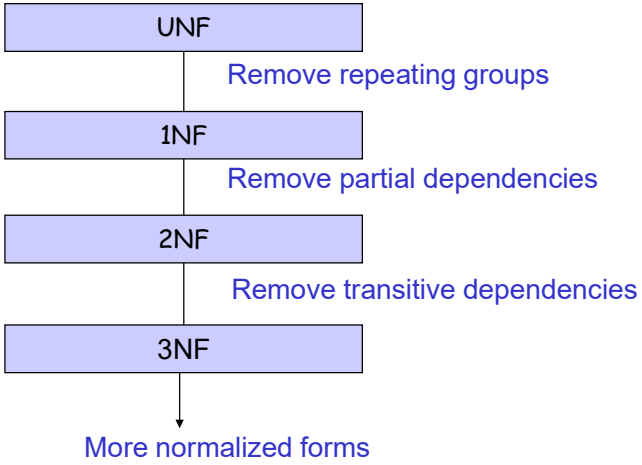
- Normalization complete ☺
- Any questions ???

Your Turn!

Student Results Table

Course Code	Course Title	Student Code	Student Name	Date of Birth	Tutor Code	Tutor Name	Grade	Result
SYA	Systems Analysis	A2345 A7423 B3472 A3472 B9843	Smith Barker Green Harris Green	20/08/69 03/04/59 23/02/70 17/07/69 10/11/68	1746 1746 1330 1746 1330	Jones Jones Jarvis Jones Jarvis	A C D F B	Dist. Pass Pass Fail Merit
COB	COBOL	A7423 A4217 B8238	Barker Morris Carter	03/04/69 17/01/68 09/12/69	1520 1520 1520	Hooper Hooper Hooper	E B C	Fail Merit Pass
PAS	Pascal	A4217 B9843 A3393 A4247	Morris Green White Cross	17/01/68 10/11/68 30/09/69 25/12/69	1520 1283 1283 1520	Hooper Trotter Trotter Hooper	A B E C	Dist. Merit Fail Pass

Normalization Flow



Learning Outcomes

At the end of the module the student will be able to:

- Describe the fundamental concepts in databases and data processing
- Analyze and systematically represent relationships in data records and perform data modeling
- Create, populate and manage relational databases in system
- Use query languages to populate, update and retrieve data from databases
- Implement basic security in database systems

Outline Syllabus

- An introduction to data processing and databases
- Database management systems
- Data analysis and Data modeling (ER diagrams and conceptual modeling)
- Relational models and normalizations
- Creating databases using GUI tools
- Query Languages (Standard Query language)
- Introduction to database security simple report creations.

Assessment & Weighting

- In class assignments and tutorials 25%
- Simple database design 25%
- End of semester examination Structured examination paper 50%

Questions?

Database Management System

Normalization

Unnormalized Form (UNF)

- A table that contains one or more repeating groups.
- To create an unnormalized table:
  - Transform data from information source (e.g. form) into table format with columns and rows.

Example 1 – address and name fields are composite

Name	Address	Phone
Sally Singer	123 Broadway New York, NY, 11234	(111) 222-3345
Jason Jumper	456 Jolly Jumper St. Trenton NJ, 11547	(222) 334-5566

Another example of UNF

Example 2 – repeating columns for each client & composite name field

Rep ID	Representative	Client 1	Time 1	Client 2	Time 2	Client 3	Time 3
TS-89	Gilroy Gladstone	US Corp.	14 hrs	Taggarts	26 hrs	Kilroy Inc.	9 hrs
RK-56	Mary Mayhem	Italiana	67 hrs	Linkers	2 hrs		

UNF to 1NF

- Remove repeating group by:
  - entering appropriate data into the empty columns of rows containing repeating data ('flattening' the table).

Or by

  - placing repeating data along with copy of the original key attribute(s) into a separate relation.

Normalization (contd.)

1st Normal Form

- A relation R is in first normal form (1NF) if domains of all attributes in the relation are *atomic* (simple & indivisible).

- Avoid multivalued & composite attributes.

Normalization (contd.)

- Solution 1: Create a separate DLOCATION relation with foreign key.
- Solution 2: If max number of locations is known, create a column for each location (may have lots of null values).
- Solution 3: Repeat the same info (redundancy + new key attribute).

Normalization

For example…

DEPARTMENT (Dname,Dnumber, DMGRSSN, DLocation)

DEPARTMENT

DNAME	DNUMBER	DMGRSSN	DLOCATIONS
Research	5	333445555	{Mathara, Kandy, Metro}
Administratio	4	987654321	{Malabe}
Headquarters	1	888665555	{Metro}

- Department relation not in 1NF
- How to take into 1NF ?

Normalization (contd.)

Solution 1:

DEPARTMENT

DNAME	DNUMBER	DMGRSSN	DLOCATIONS
Research	5	333445555	Mathara
Research	5	333445555	Kandy
Research	5	333445555	Metro
Administration	4	987654321	Malabe
Headquarters	1	888665555	Metro

1NF relation with redundancy

- Expand the key so that there will be a separate tuple in the original DEPARTMENT relation for each location of the DEPARTMENT.
- This solution has the disadvantage of introducing redundancy in the relation.

Normalization (contd.)

Solution 2:

DEPARTMENT

DNAME	DNUMBER	DMGRSSN	DLOC1	DLOC2	DLOC3
Research	5	333445555	Mathara	Kandy	Metro
Administration	4	987654321	Malabe	Null	Null
Headquarters	1	888665555	Metro	Null	Null

- Need to know max number of locations.
- create a column for each location.
- may have lots of null values.

Normalization (contd.)

- **A functional dependency, X → Y is a full functional dependency if removal of any attribute A from X means that the dependency does not hold**
  (i.e. (X –{A}) → Y does not hold)

TEACH

STUDENT	COURSE	TEACHER	CAMPUS
Narayan	Database	ABC	Metro
Smith	Database	XYZ	Malabe
Nalin	Operating Systems	Samantha	Metro
Kamal	Operating Systems	ABC	Malabe
Janith	Database	ABC	Metro
Ranil	Operating Systems	Samantha	Metro
Saman	Operating Systems	ABC	Malabe
Ruwan	Database	XYZ	Malabe

{Teacher, Campus} → Course

Normalization (contd.)

Solution 3:

DEPT_LOCATIONS

DNUMBER	DLOCATIONS
1	Metro
4	Malabe
5	Mathara
5	Kandy
5	Metro

DEPARTMENT

DNAME	DNUMBER	DMGRSSN
Research	5	333445555
Administration	4	987654321
Headquarters	1	888665555

- Remove the attribute DLOCTION and place it in a separate relation DEPT_LOCTIONS along with the primary key DNUMBER of DEPARTMENT.
- The PK is the combination {DNUMBER, DLOCTION}
- This decompose the non-INF relation into two INF relation.

Normalization (contd.)

2nd Normal Form:

- A relation R is in second normal form (2NF) if every nonprime attribute A in R is not **partially dependent** on any key of R.

Example: Not in 2NF

TEACHER	CAMPUS	COURSE	ADDRESS
ABC	Metro	Database	BoC Merchant Tower
XYZ	Malabe	Database	Malabe Campus
Samantha	Metro	Operating Systems	BoC Merchant Tower
ABC	Malabe	Operating Systems	Malabe Campus

Normalization (contd.)

- Lossless join decomposition:
  - Decomposition of R into X and Y is *lossless-join* w.r.t. a set of FDs F if, for every instance *r* that satisfies F:
    - $\Pi_X(r) \times \Pi_Y(r) = r$

This condition holds if attributes common to X and Y contains a key for either X or Y

Normalization (contd.)

TEACHER	CAMPUS	COURSE	ADDRESS
ABC	Metro	Database	BoC Merchant Tower
XYZ	Malabe	Database	Malabe Campus
Samantha	Metro	Operating Systems	BoC Merchant Tower
ABC	Malabe	Operating Systems	Malabe Campus

Removing partial dependency

- Place the attributes that create the partial dependency in a separate table.
- Make sure that the new table's primary key is left in the original table.

Normalization (contd.)

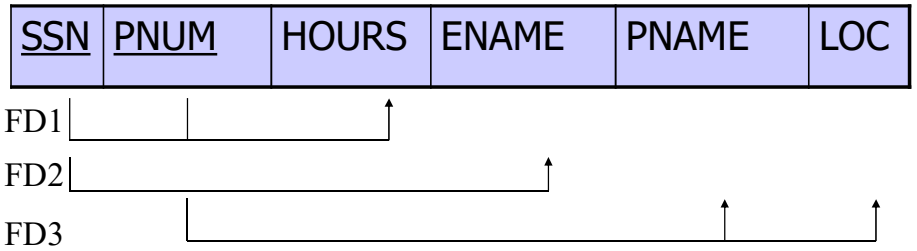
Example: After normalized into 2NF

TEACHER	CAMPUS	COURSE
ABC	Metro	Database
XYZ	Malabe	Database
Samantha	Metro	Operating Systems
ABC	Malabe	Operating Systems

CAMPUS	ADDRESS
Metro	BoC Merchant Tower
Malabe	Malabe Campus

Another Example

EMP_PROJ

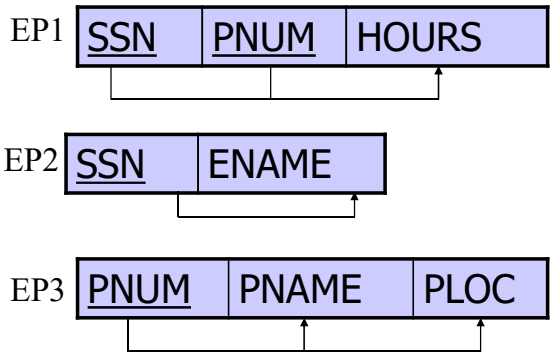


Normalization (contd.)

3rd Normal Form:

- A relation R is in 3rd normal form (3NF) if every
  - R is in 2NF, and
  - No nonprime attribute is **transitively dependent** on any key.

Normalization (contd.)

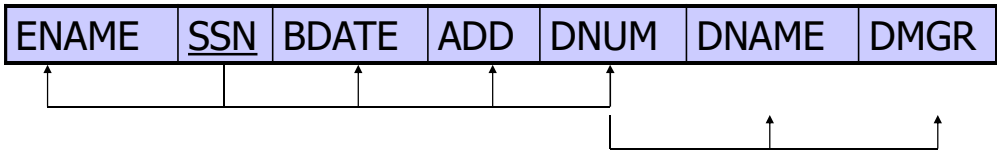


Transitive dependency

Attribute is dependent on another attribute that is not part of the primary key.

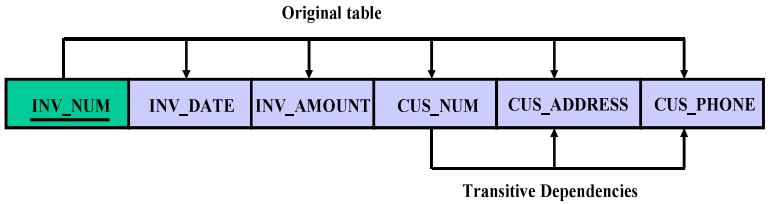
Requires the decomposition of the table containing the transitive dependency.

EMP_DEPT



Removing transitive dependency

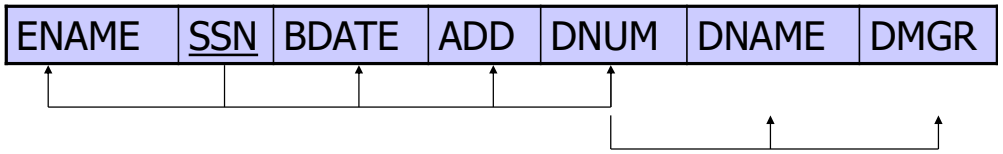
- Place the attributes that create the transitive dependency in a separate table.
- Make sure that the new table's primary key attribute is the foreign key in the original table.

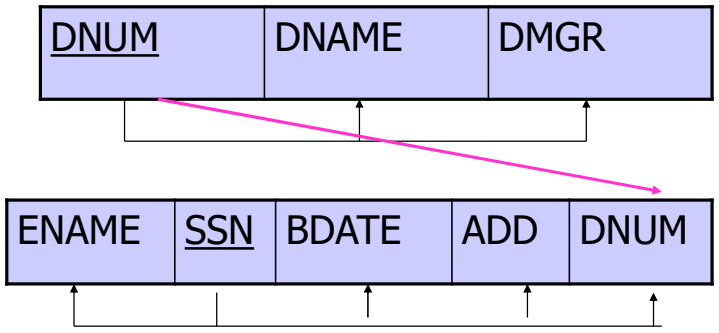


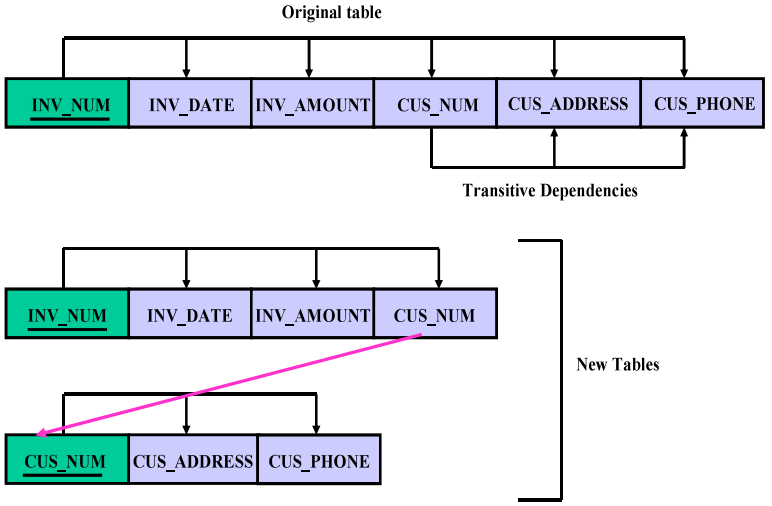
Is the table in 3NF?

Why?

Remove Transitive Dependency

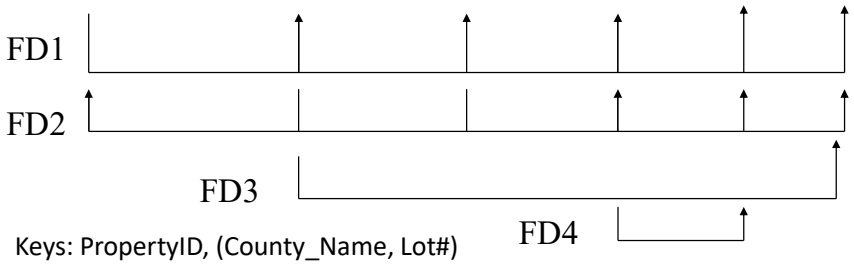






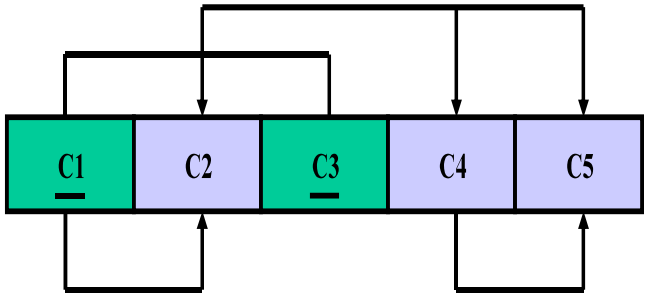
Normalization (contd.)

PROPERTY_ID	COUNTY_NAME	LOT#	AREA	PRICE	TAX_RATE



Keys: PropertyID, (County_Name, Lot#)

Dependency diagram



Identify the dependencies shown in the above diagram

C1->C2	**partial dependency**
C4 ->C5	**transitive dependency**
C1,C3 -> C2,C4,C5	**functional dependency**

Normalization (contd.)

- 1NF, 2NF & 3NF guarantee to preserve **lossless join property.**

- **Create a database whose tables are at least in 2NF, showing the dependency diagrams for each table.**

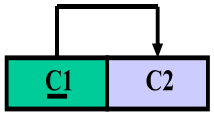


Table 1

Primary key: C1
Foreign key: None
Normal form: 3NF

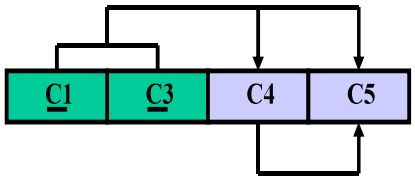


Table 2

Primary key: C1 + C3
Foreign key: C1 (to Table 1)
Normal form: 2NF, because the table exhibits the transitive dependencies C4 ⟶ C5

Create a database whose tables are at least in 3NF

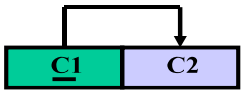


Table 1
Primary key: C1
Foreign key: None
Normal form: 3NF

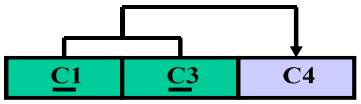


Table 2
Primary key: C1 + C3
Foreign key: C1 (to Table 1)
C4 (to Table 3)
Normal form: 3NF

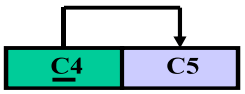


Table 3
Primary key: C4
Foreign key: None
Normal form: 3NF

Normalization

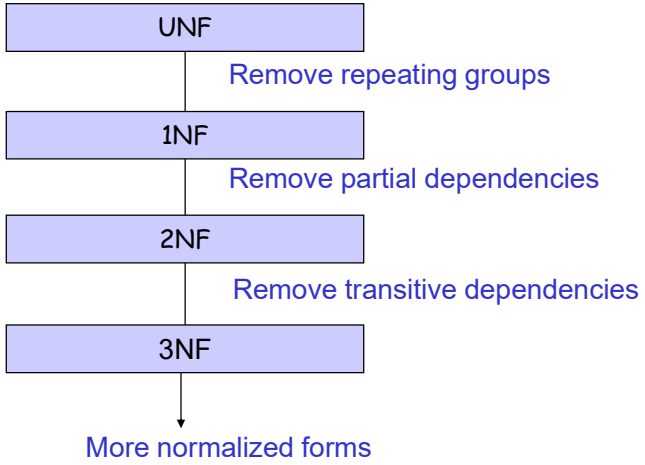
- Normalization complete ☺
- Any questions ???

Normalization (contd.)

- Denormalization…

Sometime for performance reasons, database designer may leave the relation in a lower normal form. This process is known as **denormalization**.

Normalization Flow



Your Turn!

Student Results Table

Course Code	Course Title	Student Code	Student Name	Date of Birth	Tutor Code	Tutor Name	Grade	Result
SYA	Systems Analysis	A2345 A7423 B3472 A3472 B9843	Smith Barker Green Harris Green	20/08/69 03/04/59 23/02/70 17/07/69 10/11/68	1746 1746 1330 1746 1330	Jones Jones Jarvis Jones Jarvis	A C D F B	Dist. Pass Pass Fail Merit
COB	COBOL	A7423 A4217 B8238	Barker Morris Carter	03/04/69 17/01/68 09/12/69	1520 1520 1520	Hooper Hooper Hooper	E B C	Fail Merit Pass
PAS	Pascal	A4217 B9843 A3393 A4247	Morris Green White Cross	17/01/68 10/11/68 30/09/69 25/12/69	1520 1283 1283 1520	Hooper Trotter Trotter Hooper	A B E C	Dist. Merit Fail Pass

Questions?

Database Management Systems

SQL

Creating Tables

SQL Components: DDL, DCL, & DML

- SQL is a very large and powerful language, but every type of SQL statement falls within one of three main categories (or sub-languages):
  - **Data Definition Language** (DDL) for creating a DB
    e.g. CREATE, DROP, ALTER
  - **Data Control Language** (DCL) for administering a DB
    e.g. GRANT, DENY, USE
  - **Data Manipulation Language** (DML) to access a DB
    e.g. SELECT, INSERT, UPDATE, DELETE

In this lecture you will learn

- The basic concepts and principles of SQL
- The different components of SQL

- How to use SQL (DDL) to perform basic create operations
  - Create
  - Insert
  - Modify
  - Delete data
- How to use SQL(DML) to perform basic database queries

Relational Tables Can Answer Many Queries

Student

Sid	Name
S0123	Smith
S1192	Jones
S3359	Brown

Enrolment

Sid	Cid	Date
S0123	C92	1/10
S3359	C92	17/10
S1192	D22	28/09
S0123	D07	1/10
S0123	C77	1/10
S1192	C77	03/10

Course

Cid	Name
C77	Java
C92	PHP
D07	DBMS
D22	C++

- How many courses are there & what are their names?
- Which students are enrolled for Java?
- How many students take 3 or more courses?

SQL - Structured Query Language

- SQL was developed at IBM around 1975...
- Structured programming?
  No! - Structured English (from 'SEQUEL')
  (for Structured English QUEry Language)
- SQL is a *declarative language* - says *what* not *how*
- SQL is an *abstract* & *portable* interface to RDMBs
- **Warning**: different vendors have **dialects** & **extensions**

SQL Syntax

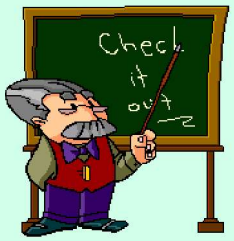
- SQL uses English keywords & user-defined names

```
CREATE TABLE Staff (
        StaffNo     INTEGER,
        Salary      FLOAT,
        Lname       CHAR(20)
);
INSERT INTO Staff VALUES (32, 25000.0, 'Smith');
```

- By convention, keywords are upper-case
- Text data is enclosed using single quotes (' ')
- Round brackets ('(') are used to group related items
- Commas (',') separate items in a list
- Statements are terminated with a semicolon (';')

Structured Query Language (contd.)

SQL is a comprehensive database language:



- Data Definition Language (DDL)
- Data Manipulation Language (DML)
- Facilities for security & authorization
- Facilities for transaction processing
- Facilities for embedding SQL in general purpose languages (Embedded SQL)

SQL Terminology

- SQL does not use formal relational terminology

Formal	Informal (SQL)
Relation	Table
Tuple	Row
Attribute	Column
Cardinality	No. of rows
Degree	No. of columns
Relationships	Foreign keys
Constraints	Assertions

Structured Query Language (contd.)

- Creating tables…
  - Tables can be created using **CREATE TABLE** statement
  - There are different data types available in SQL2

Structured Query Language (contd.)

Example…

```
CREATE TABLE Student (
        name        CHAR(20),
        address     CHAR(25),
        age         INTEGER,
        gpa         REAL
);
```

Structured Query Language (contd.)

- Integer (INT, INTEGER & SMALLINT)
- Real numbers (FLOAT, REAL, DOUBLE)
- Formatted numbers DECIMAL(i,j) or DEC(i,j) or NUMERIC(i,j)
- Character strings
  - Fixed length: CHAR(n) or CHARACTER(n)
  - Variable length: VARCHAR(n)
- Date and Time: DATE and TIME data types are supported in SQL2

Structured Query Language (contd.)

- Primary Key constraint
  - PRIMARY KEY

```
CREATE TABLE Student (
stdid       CHAR (10) PRIMARY KEY
name        CHAR(20),
address     CHAR(25),
age         INTEGER,
gpa         REAL
);
```

Structured Query Language (contd.)

- Alternative:

```
CREATE TABLE Student (
        stdid           CHAR (10)
        name            CHAR(20),
        address         CHAR(25),
        age             INTEGER,
        gpa             REAL,
        PRIMARY KEY     (stid)
);
```

Structured Query Language (contd.)

- Referential Integrity Constraints
  - FOREIGN KEY……REFERENCES
    - Student (stdId, name ,address , age , gpa)
    - Grade(subjectId, stdId , grade)

```
CREATE TABLE Grade
(
    subjectId       CHAR(4),
    stdId           CHAR(10),
    grade           CHAR(2),
    PRIMARY KEY(subjectId, stdId),
    FOREIGN KEY(stdId) REFERENCES Student
)
```

Structured Query Language (contd.)

- Other candidate keys…
  - UNIQUE

```
CREATE TABLE Student (
Stdid           CHAR (10)     PRIMARY KEY
Name            CHAR(20),
Address         CHAR(25),
Age             INTEGER,
Gpa             REAL,
NIC             CHAR(10)      UNIQUE);
```

Structured Query Language (contd.)

- Alternative
  - FOREIGN KEY

```
CREATE TABLE Grade (
        subjectId       CHAR(4),
        stdId           CHAR(10)        REFERENCES Student(stdId),
        grade           CHAR(2),
        PRIMARY KEY(subjectId,stdid)
    );
```

Structured Query Language (contd.)

- Alternative:

```
CREATE TABLE Grade (
        subjectId       CHAR(4),
        stdId           CHAR(10) ,
        grade           CHAR(2),
        PRIMARY KEY(subjectId,stdid),
        CONSTRAINT fk_Grade FOREIGN KEY(stdid)
                        REFERENCES Student(stdId)
    );
```

Structured Query Language (contd.)

- Specifying default values…
  - DEFAULT

```
CREATE TABLE Student (
Name            CHAR(20),
Address         CHAR(25)    DEFAULT 'Malabe',
Age             INTEGER,
Gpa             REAL
);
```

Structured Query Language (contd.)

- Specifying NOT NULL constraints…
  - NOT NULL

```
CREATE TABLE Student (
Name            CHAR(20)  NOT NULL,
Address         CHAR(25),
Age             INTEGER,
Gpa             REAL
);
```

Structured Query Language (contd.)

- Specifying valid values…
  - CHECK constraints

```
CREATE TABLE Emp (
….
age INT CHECK (age BETWEEN 0 AND 120),
….);
```

Structured Query Language (contd.)

- Referential Triggered Action:
  - Actions
    - SET NULL
    - CASCADE
    - SET DEFAULT
  - Operations
    - ON UPDATE
    - ON DELETE

Structured Query Language (contd.)

- The use of a constraint name allows identification of a constraint which can be dropped later (using the ALTER TABLE command)

Structured Query Language (contd.)

```
CREATE TABLE Employee (
    NIC          VARCHAR(10)    PRIMARY KEY,
    name         VARCHAR(50)    UNIQUE NOT NULL,
    address      VARCHAR(25)    DEFAULT 'Colombo',
    works_in     INTEGER,
    CONSTRAINT fk_EmpDept       FOREIGN KEY (works_in)
                 REFERENCES Dept ON DELETE CASCADE
                                 ON UPDATE NO ACTION
        );
```

Structured Query Language (contd.)

- Dropping tables

  DROP TABLE Employee [RESTRICT|CASCADE]

  – RESTRICT drops if no other constraints (such as foreign keys or views exist)
  – CASCADE drops all constraints & views that reference it

  * SQL Server 2000 has only RESTRICT option which is the default

Structured Query Language (contd.)

- Altering tables…
  - ALTER TABLE can be used to add or drop a column, change a column definition, and adding or dropping table constraints
- For example…

```
ALTER TABLE Employee ADD Job VARCHAR(12)
```

-- The new column has null values. Hence, NOT NULL cannot be used here

Structured Query Language (contd.)

- We can modify data using the following three commands: INSERT, DELETE and UPDATE
- INSERT statement:
  - Inserting a single row

```
INSERT INTO Dept VALUES (1, 'Sales', 'BoC Merchant Tower')
```

*The order of values must be the same as in the CREATE TABLE statement

Structured Query Language (contd.)

- Dropping a column…

```
ALTER TABLE Employee DROP COLUMN Job
```

- Changing column definition…

```
ALTER TABLE Employee ALTER COLUMN job vachar(50)
```

- Dropping a constraint

```
ALTER TABLE Employee DROP fk_EmpDept
```

Structured Query Language (contd.)

–Inserting to user-specified columns

```
INSERT INTO Employee (NIC, name, works_in)
VALUES ('781111111V', 'Ajith Perera', 1)
```

* Only the columns are specified are filled with values. Unspecified columns are filled with NULL if there is no default values.

Structured Query Language (contd.)

- Inserting multiple rows…

INSERT INTO Employees
<Select statement>*

*we'll learn the select statement in detail later

Structured Query Language (contd.)

- **Updating tuples in a table**

UPDATE <table>
SET <column> = <expression>
WHERE <selection condition>

Structured Query Language (contd.)

- Deleting tuples from tables…
  - Deleting all records

```
DELETE FROM Dept
```

  - Deleting only specified records

DELETE FROM <table>
WHERE <selection-condition>

```
DELETE FROM Dept WHERE dno = 2
```

  - Completely removing a table is a DDL operation

```
DROP TABLE Staff
```

Structured Query Language (contd.)

- Updating all tuples

```
UPDATE Employee SET works_in = 1
```

- Updating selected tuples

UPDATE Employee
SET works_in = 2
WHERE NIC = '781111111V'

Exercises…

- Consider the following schema:

  Student(StudentNo:Integer; name:varchar(50), major:char(4); GPA:float)

Write SQL statements to perform the following

1. Create the above table
2. Insert the following information:

StudentNo	Name	Major	GPA
1	Sampath	EE	3.5
2	Nishani	CSE	3.4

Summary

- SQL is the standard query language for RDBMS
- Three main categories of SQL
  - DDL, Data Definition Language
  - DCL, Data Control Language
  - DML, Data Manipulation Language
- CREATE belongs to DDL

Exercises… (contd.)

3. Update Sampath's GPA to 3.7.
4. Delete student table
5. Add a column address (i.e. address: varchar(50)) to the *Student* table.
6. Change the data type of address column into varchar(100).
7. Add a checking rule to GPA column, GPA values should between 0 and 4.
8. Remove the check constraint added to the GPA column.
9. Create table Major(majorId , description).
10. Change student table to reflect Primary Key.

Database Management Systems

Select Statements

Simple Queries Using SELECT

- The SELECT statement retrieves & formats data.
- SELECT is the most frequently used SQL statement.

SELECT * FROM Staff;

StaffNo	Fname	Lname	Position	Salary
731	Mark	Jones	Manager	20000
322	Andrew	Smith	Assistant	15000
261	Julie	Walters	Director	40000

In this lecture you will learn

- How to use SQL (DML) to perform basic database queries.
- How to sort and group query results.
- How to calculate aggregates and other derive data.
- The role of NULL values in databases.
- How to compose and use nested SELECT queries.

Structured Query Language

- SELECT clause in

```
SELECT      <attribute-list>
FROM        <table-list>
[WHERE      <condition>]
[GROUP BY   <group attribute(s)>]
[HAVING     <group condition>]
[ORDER BY   <attribute list>];
```

Simple Queries Using SELECT

```
SELECT * FROM Staff
```

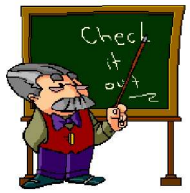
* - Acts as a 'wild card' - all columns.

•By default, SELECT outputs **all** the rows in the table.

•Use "SELECT DISTINCT target_list FROM Staff;" for avoiding duplicates.

Select

- Basic SELECT clause…




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

– <attribute-list> is a list of column names.
– <table-list> is a list of tables which the query accesses.
– <condition> is the condition that the output rows must satisfy
<Condition> (<, >, =, ≠, <=, >=) combined using AND, OR and NOT.

Use of DISTINCT

- List the property numbers of all properties that have been viewed.
  – Query1:
    SELECT propertyNo
    FROM Viewing;
  – Query2:
    SELECT DISTINCT propertyNo
    FROM Viewing;

Viewing

ClientNo	PropertyNo	ViewDate	Comment
CR56	PA14	24-May-01	too small
CR56	PG36	28-Apr-01	
CR56	PG4	26-May-01	
CR62	PA14	14-May-01	no dining room
CR76	PG4	20-Apr-01	too remote

Query1

propertyNo
PA14
PG36
PG4
PA14
PG4

Query2

propertyNo
PA14
PG36
PG4

Your Turn !

- Student (sid, name, address, dob, date of admission, GPA).
- List all students.

Selecting Specific Columns

•Specific columns can be output by giving their names:

SELECT Lname, Position, Salary FROM Staff;

Lname	Position	Salary
Jones	Manager	20000
Smith	Assistant	15000
Walters	Director	40000

•NB. must have a comma (',') between column names.

Selecting Specific Rows & Columns

•Specific rows can be selected with a WHERE clause:

```
SELECT Lname, Position, Salary
FROM Staff
WHERE Salary > 20000;
```

Lname	Position	Salary
Walters	Director	40000

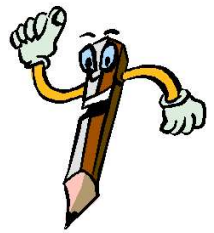
–The symbol '>' (greater than) is a **comparison operator.**
– Other comparison operators: <; =; <=; >=; ! =; <>.
– The condition 'Salary > 20000' is called a **predicate.**
– For each row, if predicate is true, row is output.

Your Turn !

- Student (sid, name, address, dob, date of admission, GPA)
- List student id, name and GPA.

Your Turn !

- Student (sid, name, address, dob, date of admission, GPA)
- List student id, name and address of all students who have a GPA of 3.5 or more.



Structured Query Language (contd.)

- IS NULL

  In SQL, NULLs are considered to be distinct from every other null.

Structured Query Language (contd.)

- IS NOT NULL…

  "List the employees who have managers"

```
SELECT   name
FROM     Emp
WHERE    manager    IS NOT NULL
```

Structured Query Language (contd.)

- Example…
- Emp (eid, name, address, dateJoined , manager)
  "List the employees who does not have a manager"

```
SELECT   name
FROM     Emp
WHERE    manager    IS NULL
```

Building Up Complex Predicates

- Predicates evaluate to either **true** or **false.**
- Predicates can be combined using AND, OR, and NOT.
- Use brackets to avoid ambiguity.
- The next two statements are different:

```
SELECT * FROM Staff WHERE
(Position = 'Manager') OR
(Position = 'Assistant' AND Salary > 10000);

SELECT * FROM Staff WHERE
(Position = 'Manager' OR Position = 'Assistant')
AND NOT (Salary <= 10000);
```

- In each case, whole WHERE clause is **true** or **false.**

Your Turn !

- Student (sid, name, address, dob, date of admission, GPA)
- List student id, name and address of all students who have a GPA of 3.5 or more and admitted in 2008 .

Structured Query Language (contd.)

- Example…

"List the company name's starting with A".

```
SELECT   CompanyName
FROM     Customers
WHERE    CompanyName    LIKE 'A%'
```

Other Types of Predicate

- Other predicates include BETWEEN, IN, and LIKE.
- But they still evaluate to either **true** or **false.**

```
SELECT * FROM Staff
WHERE
        (Salary BETWEEN 10000 AND 20000) AND
        (Position IN ('Manager', 'Assistant')) AND
        (Lname LIKE 'S%' OR Lname LIKE 'W____');
```

- '%' matches zero or more characters.
- '_' matches exactly one character.
-

Structured Query Language (contd.)

Example…

```
SELECT      CompanyName
FROM        Customers
WHERE       CompanyName     LIKE 'Londo_'
```

Your Turn!

- "List the student names starting with Hetti".

Sort - ORDER BY

- Produce a list of salaries for all staff, arranged in descending order of salary.
- Query1:

```
SELECT staffNo, fName,
                lName, salary
FROM Staff
ORDER BY salary DESC;
```

- Query2:

```
SELECT staffNo, fName,
                lName, salary
FROM Staff
ORDER BY 4 ASC;
```

Staff

StaffNo	Fname	Lname	Position	Sex	DOB	Salary	BranchNo
SA9	Mary	Howe	Assistant	F	19-Feb-70	9000	B007
SG14	David	Ford	Supervisor	M	24-Mar-58	18000	B003
SG37	Ann	Beech	Assistant	F	11-Oct-60	12000	B003
SG5	Susan	Brand	Manager	F	03-Jun-40	24000	B003
SL21	John	White	Manager	M	01-Oct-45	30000	B005
SL41	Julie	Lee	Assistant	F	13-Jun-65	90000	B005

Query1

staffNo	fName	lName	salary
SL41	Julie	Lee	90000
SL21	John	White	30000
SG5	Susan	Brand	24000
SG14	David	Ford	18000
SG37	Ann	Beech	12000
SA9	Mary	Howe	9000

Query2

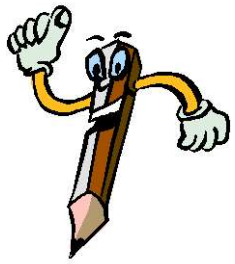
staffNo	fName	lName	salary
SA9	Mary	Howe	9000
SG37	Ann	Beech	12000
SG14	David	Ford	18000
SG5	Susan	Brand	24000
SL21	John	White	30000
SL41	Julie	Lee	90000

More Control Over SELECT

- What have we achieved so far using SELECT?
  - Retrieve data from all the rows and columns (whole table)
  - Retrieve data from all the rows and select columns
  - Retrieve data from select rows and columns
- Sometimes we want to re-format the output from SELECT: E.g. for reports...
- Examples of additional processing:
  - Eliminate duplicates
  - Sort and/or group the results
  - Rename column headings
  - Calculate totals, averages, etc. (often called aggregates)
  - Combine results from different tables

Your Turn !

- Student (sid, name, address, dob, date of admission, GPA)
- List student id, name and address of all students who have a GPA of 3.5 or more and admitted in 2008 in descending order of GPA.



Calculated Fields

- **Produce a list of monthly salaries for all staff, showing the staff number, the first and last names, and the salary details.**

Staff

StaffNo	Fname	Lname	Position	Sex	DOB	Salary	BranchNo
SA9	Mary	Howe	Assistant	F	19-Feb-70	9000	B007
SG14	David	Ford	Supervisor	M	24-Mar-58	18000	B003
SG37	Ann	Beech	Assistant	F	11-Oct-60	12000	B003
SG5	Susan	Brand	Manager	F	03-Jun-40	24000	B003
SL21	John	White	Manager	M	01-Oct-45	30000	B005
SL41	Julie	Lee	Assistant	F	13-Jun-65	90000	B005

- **Query1**

```
SELECT staffNo, fName,
       lName, salary/12
FROM Staff;
```

Query1

staffNo	fName	lName	Expr1003
SA9	Mary	Howe	750
SG14	David	Ford	1500
SG37	Ann	Beech	1000
SG5	Susan	Brand	2000
SL21	John	White	2500
SL41	Julie	Lee	7500

SQL Aggregate Functions

- We do not want to just retrieve data.
- We also want to summarise data.
- Aggregate functions compute summarization (or aggregation) of data.

- Aggregate functions
  - SUM
  - AVG
  - MIN
  - MAX
  - COUNT

Renaming Columns

- When new fields are calculated we can name them using 'AS'.

Staff

StaffNo	Fname	Lname	Position	Sex	DOB	Salary	BranchNo
SA9	Mary	Howe	Assistant	F	19-Feb-70	9000	B007
SG14	David	Ford	Supervisor	M	24-Mar-58	18000	B003
SG37	Ann	Beech	Assistant	F	11-Oct-60	12000	B003
SG5	Susan	Brand	Manager	F	03-Jun-40	24000	B003
SL21	John	White	Manager	M	01-Oct-45	30000	B005
SL41	Julie	Lee	Assistant	F	13-Jun-65	90000	B005

- **Query1:**

```
SELECT staffNo, fName, lName,
       salary/12 AS monthlySalary
FROM Staff;
```

Query1

staffNo	fName	lName	monthlySalary
SA9	Mary	Howe	750
SG14	David	Ford	1500
SG37	Ann	Beech	1000
SG5	Susan	Brand	2000
SL21	John	White	2500
SL41	Julie	Lee	7500

SUM, MIN, MAX, AVG

- Find the minimum, maximum, average and sum of staff salary.

Staff

StaffNo	Fname	Lname	Position	Sex	DOB	Salary	BranchNo
SA9	Mary	Howe	Assistant	F	19-Feb-70	9000	B007
SG14	David	Ford	Supervisor	M	24-Mar-58	18000	B003
SG37	Ann	Beech	Assistant	F	11-Oct-60	12000	B003
SG5	Susan	Brand	Manager	F	03-Jun-40	24000	B003
SL21	John	White	Manager	M	01-Oct-45	30000	B005
SL41	Julie	Lee	Assistant	F	13-Jun-65	90000	B005

- Query1

```
SELECT
    MIN(salary)  AS myMin,
    MAX(salary)  AS myMax,
    AVG(salary)  AS myAvg,
    SUM(salary) AS mySum
FROM Staff;
```

Query1

myMin	myMax	myAvg	mySum
9000	90000	30500	183000

Your Turn!

- Student (sid, name, address, dob, date of admission, GPA)
- Calculate the maximum, minimum and the average GPA.

Structured Query Language (contd.)

Purchase

Product	Date	Price	Quantity
Apple	10/21	35	15
Banana	10/22	10.50	7
Apple	10/20	40	20
Banana	10/19	11	17

Structured Query Language (contd.)

Purchase (product, date, price, quantity)

Example : find total sales for the entire database

```
SELECT  SUM(price * quantity)
FROM    Purchase
```

Example : find total sales of 'Banana'

```
SELECT  SUM(price * quantity)
FROM    Purchase
WHERE   product = 'Banana'
```

Your Turn !

- Student (sid, name, address, dob, date of admission, GPA)
- Find the minimum, maximum and average of GPA.

COUNT(*)

- Counts the number of rows in a table.
  - Including the rows that have duplicates and nulls.

Staff

StaffNo	Fname	Lname	Position	Sex		Salary	BranchNo
SA9	Mary	Howe	Assistant	F	19-Feb-70	9000	B007
SG14	David	Ford	Supervisor	M	24-Mar-58	18000	B003
SG37	Ann	Beech	Assistant	F	11-Oct-60	12000	B003
SG5	Susan	Brand	Manager	F	03-Jun-40	24000	B003
SL21	John	White	Manager	M	01-Oct-45	30000	B005
SL41	Julie	Lee	Assistant	F	13-Jun-65	90000	B005

```
SELECT    Count(*) as WomenStaff
FROM      Staff
WHERE     Sex='F';
```

Query1

WomenStaff
4

- SELECT list (target list) cannot refer to any other column.

SQL

- DISTINCT – Find the distinct values…
- Example…
  "List the different salary values of employees"

```
SELECT    DISTINCT Salary
FROM      EMP
```

Your Turn!

- Student (sid, name, address, dob, date of admission, GPA)
- Count the number of students.

SQL

- COUNT(DISTINCT …) – Count the distinct values

"Count the different salary grades in the company"

```
SELECT    COUNT(DISTINCT salary)
FROM      EMP
```

SQL

- COUNT applies to duplicates, unless otherwise stated:

```
SELECT    COUNT (category)
FROM      Product
WHERE      year>1995
```

- Better :

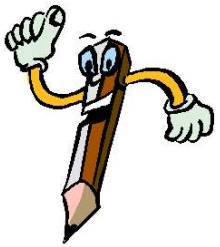
```
SELECT    COUNT (DISTINCT category)
FROM      Product
WHERE     year>1995
```

GROUP BY

- Aggregate functions help us to summarise the whole column(s) of data into one row.
- Sometimes we want to group data before applying aggregate functions.
  - This gives us 'subtotals' rather than 'overall total'.
- GROUP BY is used to achieve that.
- Produce one tuple for every group by applying aggregation.

Your Turn !

- Student (sid, name, address, dob, date of admission, GPA)
- Count the number of students with
  - GPA >= 3.5



Structured Query Language (contd.)

- Compute the FROM and WHERE clauses.
- Group By the attributes in the GROUP BY.
- Produce one tuple for every group by applying aggregation.

Structured Query Language (contd.)

```
SELECT      S
FROM        R1,…,Rn
WHERE       C1
GROUP BY    a1,…,ak
HAVING      C2
```

S = may contain attributes a1,…,ak and/or any aggregates but NO OTHER ATTRIBUTES

C1 = is any condition on the attributes in R1,…,Rn

C2 = is any condition on aggregate expressions

GROUP BY - Example

- Find the number of staff working in each branch and the sum of their salaries.
- Query1:

Staff

StaffNo	Fname	Lname	Position	Sex	DOB	Salary	BranchNo
SA9	Mary	Howe	Assistant	F	19-Feb-70	9000	B007
SG14	David	Ford	Supervisor	M	24-Mar-58	18000	B003
SG37	Ann	Beech	Assistant	F	11-Oct-60	12000	B003
SG5	Susan	Brand	Manager	F	03-Jun-40	24000	B003
SL21	John	White	Manager	M	01-Oct-45	30000	B005
SL41	Julie	Lee	Assistant	F	13-Jun-65	90000	B005

```
SELECT branchNo,
Count(staffNo) AS myCount,
       SUM(salary) AS mySum
FROM Staff
GROUP BY branchNo;
```

Attributes in group by column

Query1

branchNo	myCount	mySum
B003	3	54000
B005	2	120000
B007	1	9000

Structured Query Language (contd.)

```
SELECT      S
FROM        R1,…,Rn
WHERE       C1
GROUP BY    a1,…,ak
HAVING      C2
```

Evaluation steps:

1. Compute the FROM-WHERE part, obtain a table with all attributes in R1,…,Rn
2. Group by the attributes a1,…,ak
3. Compute the aggregates in C2 and keep only groups satisfying C2
4. Compute aggregates in S and return the result

Your Turn !

- Student (sid, name, address, dob, year_admission, GPA)
- Find the student intake for each year.

HAVING

- Query1:

```
SELECT branchNo,
 Count(staffNo) AS myCount,
 SUM(salary) AS mySum
FROM Staff
GROUP BY branchNo
HAVING COUNT(staffNo)>1;
```

Staff

StaffNo	Fname	Lname	Position	Sex	DOB	Salary	BranchNo
SA9	Mary	Howe	Assistant	F	19-Feb-70	9000	B007
SG14	David	Ford	Supervisor	M	24-Mar-58	18000	B003
SG37	Ann	Beech	Assistant	F	11-Oct-60	12000	B003
SG5	Susan	Brand	Manager	F	03-Jun-40	24000	B003
SL21	John	White	Manager	M	01-Oct-45	30000	B005
SL41	Julie	Lee	Assistant	F	13-Jun-65	90000	B005

Query1

branchNo	myCount	mySum
B003	3	54000
B005	2	120000

Structured Query Language (contd.)

- Find all authors who have a vocabulary over 10000 words:

```
SELECT      a.name
FROM        Author a, Wrote w, Mentions m
WHERE       a.login=w.login AND w.url=m.url
GROUP BY    a.name
HAVING      COUNT(DISTINCT m.word)>10000
```

Structured Query Language (contd.)

Author (login ,name)
Document (url , title)
Wrote (login ,url)
Mentions (url ,word)

- Find all authors who wrote at least 10 documents

```
SELECT      a.name
FROM        Author a, Wrote w
WHERE       a.login=w.login
GROUP BY a.login, a.name
HAVING      count(w.url) >= 10
```

Your Turn!

- Student (sid, name, address, dob, date of admission, GPA)
- Find student intake for each year.
- List the student intake for every year, where there have been an intake of less than 800.

Why Do DB Systems Allow NULLs ?

- A NULL value can be used to represent several situations:
  - Don't care; Don't know; Don't know yet; Used to know!
  - SQL has special rules and logic to handle NULLs:
  - SELECT * FROM Staff WHERE Fname IS NULL;
- NB. WHERE Colname = 'NULL' does not work !
- Can also say: WHERE Colname IS NOT NULL.
  - NULLs can be useful, difficult, or dangerous.
- Use NULLs wisely!

Set Operations in SQL

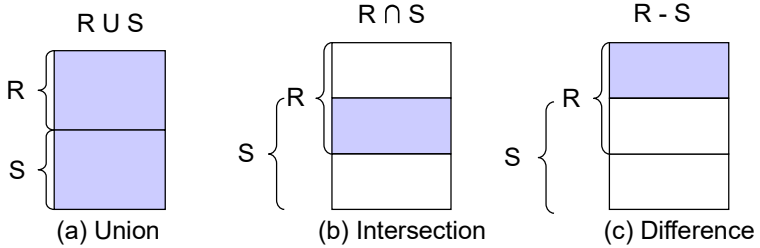
- Syntax:

```
(SELECT ...) UNION        (SELECT ...)
(SELECT ...) INTERSECT    (SELECT ...)
(SELECT ...) EXCEPT       (SELECT ...)
```

- Some DBMSs use MINUS instead of EXCEPT.
- For set operations, the tables must be union-compatible
  - i.e. have the same number and types of columns.

Combining Results Tables

- Sometimes its useful to be able to combine query results using set operations:



UNION **INTERSECT** **EXCEPT**

Set Operation Example

- List the cities with both a branch office & a property for rent:

```
(SELECT City FROM Branch)
INTERSECT
(SELECT City FROM PropertyForRent);
```

Branch	
BrNo	City
B003	Glasgow
B004	Bristol
B002	London

PropertyForRent	
PropNo	City
PA14	Aberdeen
PL94	London
PG4	Glasgow

→

Result
City
London
Glasgow

Cities with branch office OR a property for rent: UNION
Cities with branch office but NO props for rent: EXCEPT

Set Operation Example

- List the cities with both a branch office OR property for rent:

```
(SELECT City FROM Branch)
UNION
(SELECT City FROM PropertyForRent);
```

Branch	
BrNo	City
B003	Glasgow
B004	Bristol
B002	London

PropertyForRent	
PropNo	City
PA14	Aberdeen
PL94	London
PG4	Glasgow

→

Result
Glasgow
Bristol
London
Aberdeen

Exercises…

Employee(eid, name, salary, dept, address)

Dept(deptNo,dname, building, mgr)

- Print all employee names.
- Print names of employees working for 'Administration' (i.e. dname) department.
- Print names of employees working for 'Administration' (i.e. dname) department and getting a salary > Rs. 50,000.

Set Operation Example

- List the cities with both a branch office but no property for rent:

```
(SELECT City FROM Branch
EXCEPT
(SELECT City FROM PropertyForRent);
```

Branch	
BrNo	City
B003	Glasgow
B004	Bristol
B002	London

PropertyForRent	
PropNo	City
PA14	Aberdeen
PL94	London
PG4	Glasgow

→

Result
Bristol

Exercises… (contd.)

- Print the employee's name, his/her department and his/her manager's name.
- Print names of employees who are managers. If an employee is managing more than one department, print his/her name only once.
- Print names of employees managing 'Administration' and 'Sales' department. If the same employee is managing both departments, repeat his/her name twice.

Exercises… (contd.)

- Print the names of employees whose names have a letter ‘A’ or ‘a’ in them.
- Print names of employees whose address contains the following string ‘_olombo’ where _ is any character.
- Print names, deptNos, and salaries of employees, Order firstly in ascending order by deptNos and then salary by descending order .
- Print the names of employees who are not working for any department.
- Print the names of employees who are working for some department.

Exercises

- Count the number of employees working for department 5.
- Print the maximum and minimum salaries for department 5.
- Find the number of employees obtaining a salary greater than the Average salary of all.
- Print the total number of hours worked by employees for project 5.

Exercises

- Count the number of employees working for department 5.
- Print the maximum and minimum salaries for department 5.
- Find the number of employees obtaining a salary greater than the Average salary of all.
- Print the total number of hours worked by employees for project 5.

Summary So Far...

- SQL is a powerful but “quirky" query language.
- SQL is not like other programming languages (no variables)
- You can answer many kinds of question...

Database Management Systems

SQL-3

Querying Multiple Tables

- How do we list all the properties that a given client has viewed?
- Could start with an example - e.g. client CR56 ...

PropertyForRent

PropertyNo	Street	City	Postcode	Type	Rooms	Rent	OwnerNo	StaffNo	BranchNo
PA14	16 Holhead	Aberdeen	AB7 5SU	House	6	650	CO46	SA9	B007
PG16	5 Novar Dr	Glasgow	G12 9AX	Flat	4	450	CO93	SG14	B003
PG21	18 Dale Rd	Glasgow	G12	House	5	600	CO87	SG37	B003
PG36	2 Manor Rd	Glasgow	G32 4QX	Flat	3	375	CO93	SG37	B003
PG4	6 Lawrence St	Glasgow	G11 9QX	Flat	3	350	CO40		B003
PL94	6 Argyll St	London	NW2	Flat	4	400	CO87	SL41	B005

Client

ClientNo	Fname	Lname	TelNo	PrefType	MaxRent
CR56	Aline	Stewart	0141-848-1825	Flat	350
CR62	Mary	Tregear	01224-196720	Flat	600
CR74	Mike	Ritchie	01475-392178	House	750
CR76	John	Kay	0207-774-5632	Flat	425

Viewing

ClientNo	PropertyNo	ViewDate	Comment
CR56	PA14	24-May-01	too small
CR56	PG36	28-Apr-01	
CR56	PG4	26-May-01	
CR62	PA14	14-May-01	no dining room
CR76	PG4	20-Apr-01	too remote

In this lecture you will learn

- The concept of joining tables
- Why joins are central to relational database systems
- How to specify joins in SQL
- The different ways of joining tables
- Using table aliases & full column names in queries

Property Query - First Attempt

- First attempt: List the property numbers viewed by client number 'CR56':

```
SELECT    PropertyNo
FROM      Viewing
WHERE     ClientNo = 'CR56';
```

PropertyNo
PG36
PG4
PA14

- But we'd like to see the client name & property details.
- So we'll need to access Client and PropertyForRent for names etc...

Property Query - Second Version

```
SELECT      Viewing.PropertyNo, Street, City, ViewDate
FROM        Viewing, PropertyForRent
WHERE       ClientNo = 'CR56'
   AND Viewing.PropertyNo = PropertyForRent.PropertyNo;
```

PropertyNo	Street	City	ViewDate
PG36	2 Manor Rd	Glasgow	28-Apr-01
PG4	6 Lawrence St	Glasgow	26-May-01
PA14	16 Holhead	Aberdeen	24-May-01

- We now have two table names in the FROM clause.
- Note use of “Table.ColumnName" to avoid ambiguity in column names.

Property Query - Third Version

```
SELECT      Fname, Lname, Street, City, ViewDate
FROM        Viewing, PropertyForRent, Client
WHERE       Viewing.ClientNo = 'CR56'
     AND Viewing.PropertyNo = PropertyForRent.PropertyNo
     AND Viewing.ClientNo = Client.ClientNo;
```

Fname	Lname	Street	City	ViewDate
Aline	Stewart	2 Manor Rd	Glasgow	28-Apr-01
Aline	Stewart	6 Lawrence St	Glasgow	26-May-01
Aline	Stewart	16 Holhead	Aberdeen	24-May-01

- The two “AND" clauses are called join criteria.

Property Query - Fourth Version

- Users shouldn't have to know about internal keys...

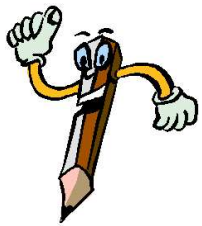
```
SELECT      Fname, Lname, Street, City, ViewDate
FROM        Viewing, PropertyForRent, Client
WHERE       Fname = 'Aline' AND Lname = 'Stewart'
  AND Viewing.PropertyNo = PropertyForRent.PropertyNo;
  AND Viewing.ClientNo = Client.ClientNo;
```

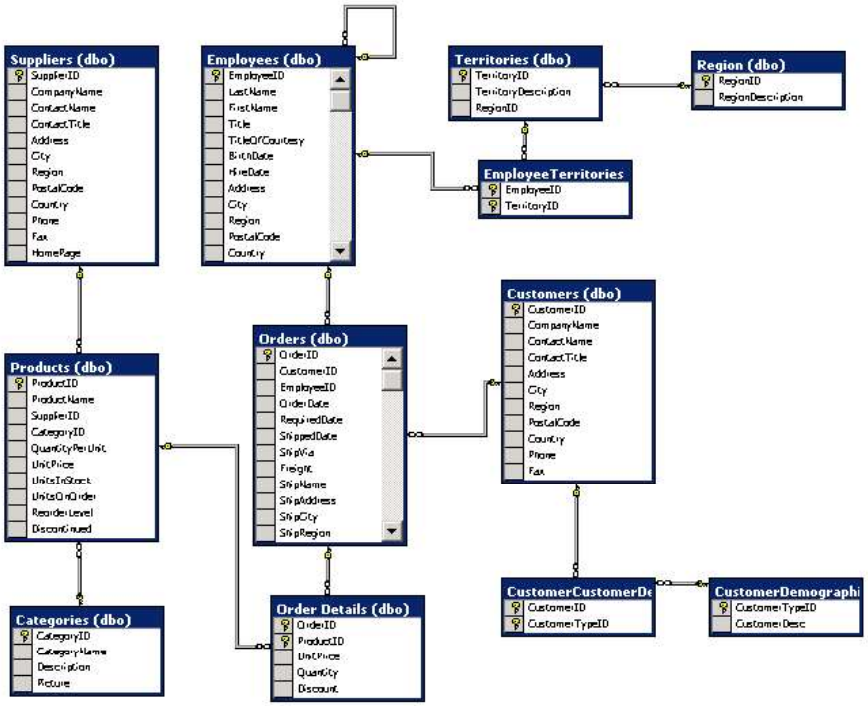
Fname	Lname	Street	City	ViewDate
Aline	Stewart	2 Manor Rd	Glasgow	28-Apr-01
Aline	Stewart	6 Lawrence St	Glasgow	26-May-01
Aline	Stewart	16 Holhead	Aberdeen	24-May-01

Structured Query Language (contd.)

- Example… (JOINS)

“List all products and their suppliers who are based in Tokyo”

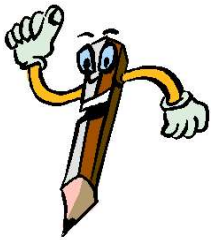




Structured Query Language (contd.)

- Example… (multiple joins)

"For each order, list the CustomerID and product names that the order contains"



Structured Query Language (contd.)

```
SELECT    Products.ProductName, Suppliers.CompanyName
FROM      Products, Suppliers
WHERE     (Products.SupplierID = Suppliers.SupplierID)
          AND (Suppliers.City = 'Tokyo')
```

Structured Query Language (contd.)

```
SELECT    Orders.CustomerID, Products.ProductName
FROM      Orders, [Order Details], Products
WHERE     (Orders.OrderID = [Order Details].OrderID)
          AND ([Order Details].ProductID =
          Products.ProductID)
```

Structured Query Language (contd.)

Renaming & Aliasing…

- Sometimes the same attribute name appears in multiple tables of the FROM clause. Then we need to qualify the table name.
- Example

```
SELECT   O.CustomerID AS CustomerName,
         P.ProductName AS PName
FROM     Orders AS O, [Order Details] AS OD, Products P
WHERE    (O.OrderID = OD.OrderID) AND
           (OD.ProductID = P.ProductID)
```

Structured Query Language (contd.)

- You can use aliases without AS keyword.
- Also, you can use it for convenience.
- Example…

```
SELECT   E.name, M.name
FROM     Emp E, Emp M
WHERE    E.manager = M.ID
```

Using Table Aliases

- Table aliases can help reduce amount of typing.
- The following is identical to the previous query:

```
SELECT   C.Fname, C.Lname, P.Street, P.City, V.ViewDate
FROM     Viewing V, PropertyForRent P, Client C
WHERE    C.Fname = 'Aline' AND C.Lname = 'Stewart'
         AND V.PropertyNo = P.PropertyNo
         AND V.ClientNo = C.ClientNo
```

- Table aliases help reduce the risk of typing mistakes.
- But shouldn't the DBMS know how to match keys for us?

Structured Query Language (contd.)

Product (pname, price, category, manufacturer)
Purchase (buyer, seller, store, product)
Person(persname, phoneNumber, city)

Find names of people living in 'Kurunegala' that bought some product in the 'Gadgets' category, and the names of the stores.

```
SELECT   DISTINCT persname, store
FROM     Person, Purchase, Product
WHERE     persname=buyer AND product = pname AND
         city='Kurunegala'  AND category='Gadgets'
```

Structured Query Language (contd.)

- We can nest queries within queries
  - Nested queries (subqueries)
- Nested queries…
  - Inner query
  - Outer query
- Operators
  - IN, NOT IN
  - <operator> ALL
  - <operator> SOME | ANY
  - where <operator> $\in$ { =, ≤, ≥, ≠, <, >}

Structured Query Language (contd.)

- Print names of employees working for 'Administration' (i.e. dname) department and getting a salary > Rs. 50,000

```
SELECT      e.name
FROM        EMP
WHERE       salary > 50000 and dno =(
        SELECT      dno
        FROM        Dept
        WHERE       dname = 'Administrative')
```

***If subquery returns more, it's a run-time error.

Structured Query Language (contd.)

- Let us consider the following schema:

Emp (nic, name, sex, addr, salary, dno)

Dept (dno,dname,mgr)

Project(pnum, pname, plocation, dno)

Works_On (enic, pnum, hours)

Dependent(enic, name, gender, bdate, relationship)

Structured Query Language (contd.)

- Example

  Find nic and names of employees who work for both project 1 (i.e. pnum) and project 3.

```
SELECT      e.nic, e.name
FROM        Emp e, Works_On w
WHERE       e.nic = w.enic AND
            w.pnum = 3 AND

            e.nic IN (
            SELECT          e1.nic
             FROM           Emp e1, Works_On w1
             WHERE          e1.nic = w1.enic AND
                    w1.pnum = 1)
```

Structured Query Language (contd.)

- Example
  - Find nic and names of employees who do not work for project 5 (i.e. pnum)

```
SELECT    e.nic, e.name
FROM      Emp e, Works_On w
WHERE     e.nic = w.enic AND
          e.nic NOT IN
                (SELECT     enic
                 FROM       Works_On
                 WHERE      pnum = 5)
```

Structured Query Language (contd.)

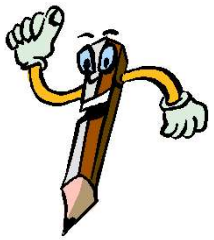
- Example…

```
SELECT    NAME
FROM      EMP
WHERE     SALARY > ALL
          (SELECT     SALARY
           FROM       EMP
           WHERE      DNO = 5)
```

Structured Query Language (contd.)

- Example…

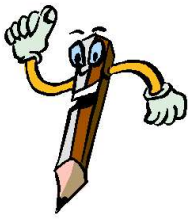
"List the employee's name who get salaries more than all employees of department 5"



Structured Query Language (contd.)

- Example…

"List the employee's name who get salaries more than any employee of department 5"



Structured Query Language (contd.)

Answer 1:

```
SELECT      NAME
FROM        EMP
WHERE       SALARY > ANY
        (SELECT     SALARY
        FROM        EMP
        WHERE       DNO = 5)
```

Answer 2:

```
SELECT      NAME
FROM        EMP
WHERE       SALARY > SOME
        (SELECT     SALARY
        FROM        EMP
        WHERE       DNO = 5)
```

Structured Query Language (contd.)

```
SELECT      E. NAME
FROM        EMP E
WHERE       E.NIC IN (
            SELECT      ENIC
            FROM        DEPENDENT
            WHERE       GENDER = E.GENDER)
```

Structured Query Language (contd.)

- Correlated Nested Queries…
  Nested queries referencing tables of outer queries.
- Example…
  "List the employee who has at least one dependent who has the same gender as the employee"

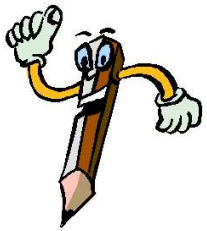
Structured Query Language (contd.)

- **EXISTS…**
  Can be used to check the existence of set…
- Previous example…

```
SELECT  E.Name
FROM  EMP E
WHERE  EXISTS (
                SELECT          *
                FROM    DEPENDENT
                WHERE ENIC = E.NIC AND
                        GENDER = E.GENDER)
```

Structured Query Language (contd.)

- NOT EXISTS…
- Example…
  "List the names of employees who do not have any dependents"



Your Turn!!

- Print the names of the manager whose salary is greater than the salary of manager of dept 5.
- Print names of departments that do not have any projects.
- Print the names of employees who work for project 5 but not for project 7.
- List the Employees' NIC who worked within the same project and same number of hours as employee '783456789V' (i.e. nic).

Structured Query Language (contd.)

```
SELECT  E.NAME
FROM        EMP  AS E
WHERE  NOT EXISTS (
                    SELECT          *
                    FROM    DEPENDENT
                    WHERE  ENIC = E.NIC)
```

Your Turn!!

Employee(<u>eid</u>, name, salary, dept, address)

Dept(<u>deptNo</u>,dname, building, mgr)

- Print all employee names.
- Print names of employees working for 'Administration' (i.e. dname) department.
- Print names of employees working for 'Administration' (i.e. dname) department and getting a salary > Rs. 50,000.

Your Turn!! … (contd.)

- Print the employee's name, his/her department and his/her manager's name.
- Print names of employees who are managers. If an employee is managing more than one department, print his/her name only once.
- Print names of employees managing 'Administration' and 'Sales' department. If the same employee is managing both departments, repeat his/her name twice.

Your Turn!!

- Count the number of employees working for department 5.
- Print the maximum and minimum salaries for department 5.
- Find the number of employees obtaining a salary greater than the Average salary of all.
- Print the total number of hours worked by employees for project 5.

Your Turn!! … (contd.)

- Print the names of employees whose names have a letter 'A' or 'a' in them.
- Print names of employees whose address contains the following string '_olombo' where _ is any character.
- Print names, deptNos, and salaries of employees, Order firstly in ascending order by deptNos and then salary by descending order.
- Print the names of employees who are not working for any department.
- Print the names of employees who are working for some department.

Summary

- You can build up complex queries from simpler sub-queries.
- It often requires thought to compose complex queries.
- Approach:
  - Break problem into sub-parts,
  - and then build up final query...

Database Management Systems

Database Security

What is Database security?

- **The mechanism that protect the database against intentional or accidental threats.**
- **The db. security encompasses H/W, S/W, people and data.**
- **DB security is concerned with avoiding the following situation:**
  - **Theft & fraud**
  - **Loss of confidentiality (secrecy)**
  - **Loss of privacy**
  - **Loss of integrity**
  - **Loss of availability**

Database Security

In this chapter you will learn:

- **The scope of database security**
- **Why database security is a serious concern for an organization**
- **The types of threats that can affect a database system**
- **How to protect a computer system using computer based controls**
- **The security measures provided by MS Access**
- **Approaches for securing a DBMS on the web**

- Threat : any situation or event, whether intentional or accidental, that may adversely affect a system and consequently the organization.
- DB security aims to minimize losses caused by anticipated events in a cost effective manner without unduly constraints the users.

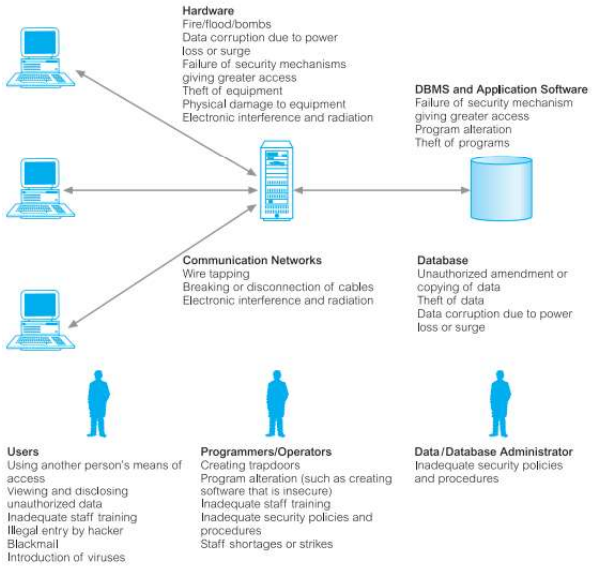
examples of various types of threat:

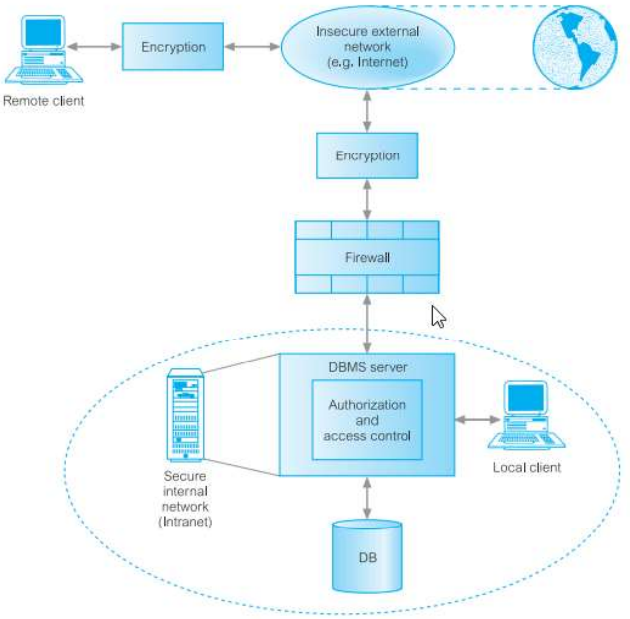
Threat	Theft and fraud	Loss of confidentiality	Loss of privacy	Loss of integrity	Loss of availability
Using another person's means of access	✓	✓	✓		
Unauthorized amendment or copying of data	✓			✓	
Program alteration	✓			✓	✓
Inadequate policies and procedures that allow a mix of confidential and normal output	✓	✓	✓		
Wire tapping	✓	✓	✓		
Illegal entry by hacker	✓	✓	✓		
Blackmail	✓	✓	✓		
Creating 'trapdoor' into system	✓	✓	✓		
Theft of data, programs, and equipment	✓	✓	✓		✓
Failure of security mechanisms, giving greater access than normal		✓	✓	✓	
Staff shortages or strikes				✓	✓
Inadequate staff training		✓	✓	✓	✓
Viewing and disclosing unauthorized data	✓	✓	✓		
Electronic interference and radiation				✓	✓
Data corruption owing to power loss or surge				✓	✓
Fire (electrical fault, lightning strike, arson), flood, bomb				✓	✓
Physical damage to equipment				✓	✓
Breaking cables or disconnection of cables				✓	✓
Introduction of viruses				✓	✓

Countermeasures – Computer-Based Controls

- Computer-based security controls for the multi-user environment include:
  - authorization
  - access controls
  - views
  - backup and recovery
  - integrity
  - encryption
  - RAID technology.

A summary of the potential threats to computer systems is represented





Representation of a typical multi-user computer environment

Countermeasures – Computer-Based Controls

- **Authorization:** The granting of a right or privilege that enables a subject to have legitimate access to a system or a system's object.
- **Authentication:** A mechanism that determines whether a user is who he or she claims to be.
- **Access control :** The typical way to provide access controls for a database system is based on the granting and revoking of privileges.
- There are two types of Access controls

1. **Discretionary Access Control (DAC)**
2. **Mandatory Access Control (MAC)**

View

- A view is the dynamic result of one or more relational operations operating on the base relations to produce another relation.
- A view is a virtual relation that does not actually exist in the database, but is produced upon request by a particular user, at the time of request.
- The view mechanism provides a powerful and flexible security mechanism by hiding parts of the database from certain users.

1. Discretionary Access Control (DAC)

- Most commercial DBMSs provide an approach called Discretionary Access Control (DAC),which manages privileges using SQL.
- The SQL standard supports DAC through the GRANT and REVOKE commands.

2. Mandatory Access Control (MAC)

- Some commercial DBMSs also provide an approach to access control called Mandatory Access Control (MAC), which is based on system-wide policies that cannot be changed by individual users.
- In this approach each database object is assigned a security class and each user is assigned a clearance for a security class, and rules are imposed on reading and writing of database objects by users.
- The SQL standard does not include support for MAC.

Backup

- The process of periodically taking a copy of the database and log file (and possibly programs) on to offline storage media.
- A DBMS should provide backup facilities to assist with the recovery of a database following failure.
- It is advisable to make backup copies of the database and log file at regular intervals and to ensure that the copies are in a secure location.
- In the event of a failure, the backup copy and the details captured in the log file are used to restore the database to the latest possible consistent state.
- **Journaling :**The process of keeping and maintaining a log file (or journal) of all changes made to the database to enable recovery to be undertaken effectively in the event of a failure.

Integrity

- Integrity constraints also contribute to maintaining a secure database system by preventing data from becoming invalid, and hence giving misleading or incorrect results.

RAID (Redundant Array of Independent Disks)

- Disk drives are the most vulnerable components with the shortest times between failure of any of the hardware components.
- One solution is the use of Redundant Array of Independent Disks (RAID) technology.
- RAID originally stood for Redundant Array of Inexpensive Disks, but more recently the 'I' in RAID has come to stand for Independent.
- RAID works on having a large disk array comprising an arrangement of several independent disks that are organized to improve reliability and at the same time increase performance.

Encryption

- The encoding of the data by a special algorithm that renders the data unreadable by any program without the decryption key.

- There are a number of different disk configurations with RAID, termed RAID levels.
  - RAID 0
  - RAID 1
  - RAID 2
  - RAID 3
  - RAID 4
  - RAID 5
  - RAID 6

The security measures provided by MS Access

- Refer access.docx

Approaches for securing a DBMS on the web

- The security measures associated with DBMSs on the Web include:
  - proxy servers
  - firewalls
  - message digest algorithms and digital signatures
  - digital certificates
  - kerberos
  - Secure Sockets Layer (SSL) and Secure HTTP (S-HTTP)
  - Secure Electronic Transactions (SET) and Secure Transaction Technology (SST)
  - Java security
  - ActiveX security
- Briefly explain the above approaches.

Approaches for securing a DBMS on the web

- The challenge is to transmit and receive information over the Internet while ensuring that:
- It is inaccessible to anyone but the sender and receiver (privacy)
- It has not been changed during transmission (integrity); n the receiver can be sure it came from the sender (authenticity)
- The sender can be sure the receiver is genuine (non-fabrication)
- The sender cannot deny he or she sent it (non-repudiation).

References

- Connolly, T. M. & Begg, C. E. (2005). Database Systems A Practical Approach to Design, Implementation, and Management (4th Edition)