

Lectures / Notes

Subject: Files & Databases

For BCS (5th Semester)

Prepared by: Sir Muneer A. Shaikh

Chapter#1**Introduction To Database Systems****Lecture#1****Introduction to Data, Database & Database Sytem****Data:**

- Raw facts such as an employee's name and number of hours worked in a week or sales orders.
- Data is the coded representation of information for use in a computer.
- Data has attributes, such as type and length. Data may be:
 - Numeric
 - Alphabetic
 - Alpha-numeric
 - Graphic

Database:**The "Heart and Soul" of Today's Business Systems**

- “A set of information held in a computer” (Oxford English Dictionary)
- “One or more large structured sets of persistent data, usually associated with software to update and query the data” (Free On-Line Dictionary of Computing)
- “A collection of data arranged for ease and speed of search and retrieval” (Dictionary.com)

Examples of databases:

- Library catalogues
- Medical records
- Bank accounts
- Telephone directories
- Airline bookings
- Student records
- Customer histories
- Stock market prices
- and so on

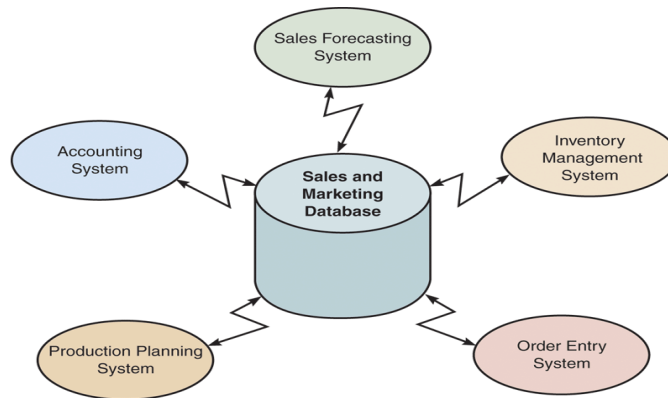
- A database is a collection of information that is organized so that it can easily be accessed, managed, and updated.
- One possible definition is that a database is a collection of records stored in a computer in a systematic way, such that a computer program can consult it to answer questions. For better retrieval and sorting, each record is usually organized as a set of data elements (facts).
- Strictly speaking, the term database refers to the collection of related information, accessed and managed by its DBMS.
- Many professionals would consider a collection of data to constitute a database only if it has certain properties: for example, if the data is managed to ensure its integrity and quality, if it allows shared access by a community of users, if it has a schema, or if it supports a query language.
- A database may be generated and maintained manually or it may be computerized. The library card catalog is an example of a database that may be created and maintained manually. A computerized database may be created and maintained either by a group of application programs written specially for that task or by a database management system(DBMS)

Database System

- A database system can be thought of as a computerized record-keeping system. Such a system involves:
 - Stored Data (Database)
 - Hardware
 - Software (in particular the DBMS)
 - Users

Why Computerized Database:

- **Compactness:** There is no need for possibly voluminous paper files.
- **Speed:** The machine can retrieve and update data far faster than a human can.
- **Currency:** Accurate, up-to date information is available on demand at any time.

Overview of Database Systems**Database Applications:**

- Banking: all transactions
- Airlines: reservations, schedules
- Universities: registration, grades
- Sales: customers, products, purchases
- Online retailers: order tracking, customized recommendations
- Manufacturing: production, inventory, orders, supply chain
- Human resources: employee records, salaries, tax deductions

Advantages

- Scalability
- Better support for client/server systems
- Economy of scale
- Flexible data sharing
- Enterprise-wide application – database administrator (DBA)
- Stronger standards
- Controlled redundancy
- Better security

Types of Database Systems

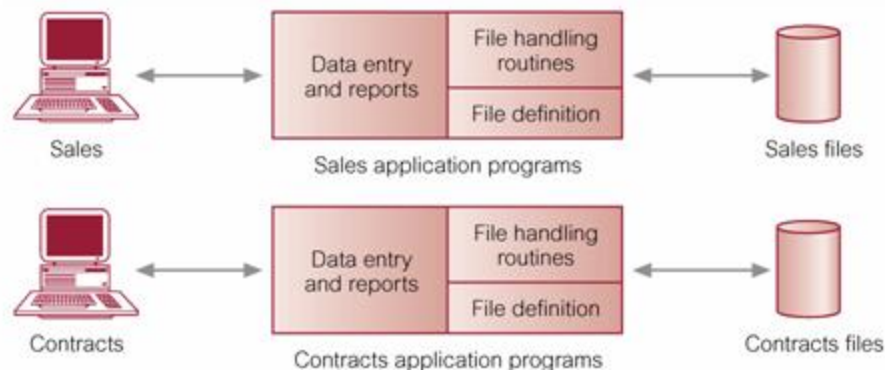
- **Number of Users**
 - Single-user
 - Desktop database
 - Multi-user
 - Workgroup database
 - Enterprise database
- **Scope**
 - Desktop
 - Workgroup
 - Enterprise
- **Location**
 - Centralized
 - Distributed
- **Use**
 - Transactional (Production)
 - Decision support
 - Data warehouse

Lecture#2**Introduction to File Based Systems and File Processing Environment****File Based Systems:**

- Before the advent of database systems, computer-readable data was usually kept in files stored on magnetic tapes or disks.
- Each file has a specific format.
- Programs that use these files depend on knowledge about that format.
- Each program defines and manages its own data.

The Traditional File Processing Environment

In a typical file processing system, each department has its own set of applications and its own files, designed specially for those applications. The department itself, working with data processing staff, sets policies or standards for the format and maintenance of its files. The file is owned by a particular department, and others who need some of the same information must either store it in their own files or obtain a printout by the permission of the owner. Having multiple copies of the same data (redundancy) leads to inconsistency, since different departments have different data standards and update policies.

**Drawbacks of using file based systems to store data:**

- Separation and isolation of data
 - Each program maintains its own set of data.
 - Users of one program may be unaware of potentially useful data held by other programs.
- Duplication of data
 - Same data is held by different programs.
 - Wasted space and potentially different values and/or different formats for the same item.

- Data dependence
 - File structure is defined in the program code.
- Incompatible file formats
 - Programs are written in different languages, and so cannot easily access each others files.
- Fixed Queries
 - Programs are written to satisfy particular functions.
- Difficulty in accessing data
 - Need to write a new program to carry out each new task
- Integrity problems
 - Integrity constraints (e.g. account balance > 0) become part of program code
 - Hard to add new constraints or change existing ones
- Security problems
 - Hard to provide user access to some, but not all, data

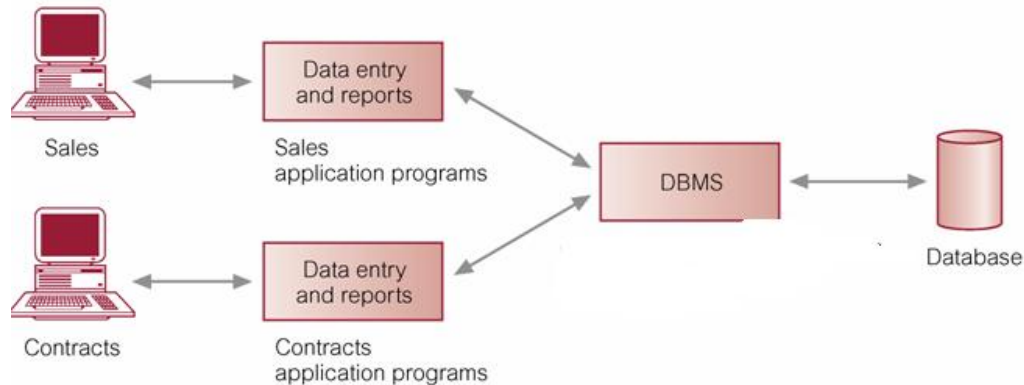
Advantages:

- Easy Recovery
- No need for Database administrator
- Not required DBMS

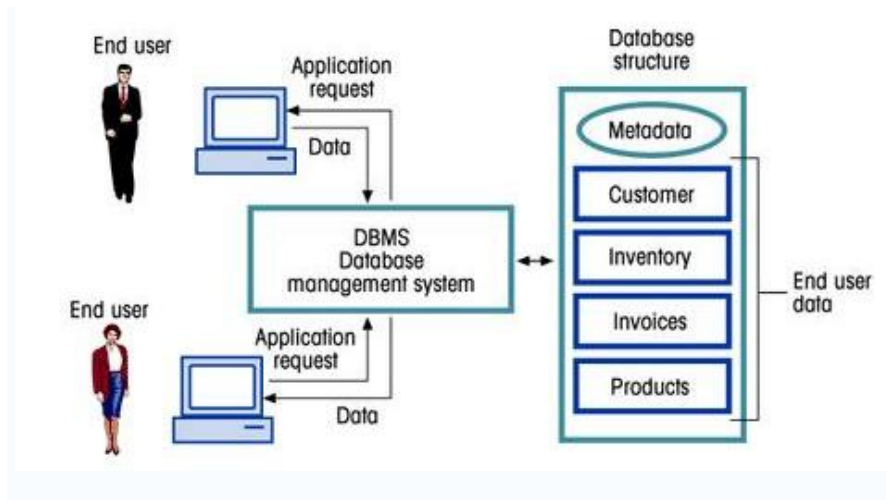
Database Management System (DBMS)

- A collection of programs that enables you to store, modify, and extract information from a database (query a database).
- A software package such as Oracle or MS-Access.
- Manages data and relationships in the database.
- Creates a Data Dictionary to store Metadata – data about data.
- Manages all day-to-day transactions.
- Provides user with data independence at application level.
- Transforms logical data requests to match physical data structures.

- Secures access through passwords, restricted user access, and encryption.
- Provides backup and recovery mechanisms.
- Provides export and import utilities.
- Allows sharing of data with locking capabilities.



- Manages the Interaction between the End User and the Database



OBJECTIVE OF DATABASE APPROACH

- The **basic objective** of a database is to provide **speedy up-to-date information** for the effective control of business operations and for the making of **decisions based on facts** rather than assumptions.

- It also provides efficient *searching techniques* which save the valuable time of executives and professional accountants etc.

Importance of DBMS

- It helps make data management more efficient and effective.
- Its query language allows quick answers to *ad hoc* queries.
- It provides end users better access to more and better-managed data.
- It promotes an integrated view of organization's operations -- "big picture."
- It reduces the probability of inconsistent data.

Common Database Brands

- Corel Paradox
- DB2
- Informix
- MS Access
- MS SQL Server
- MySQL
- Oracle

Lecture#3**Advantages and Disadvantages of the Relational / Integrated database Design Approach**

Databases are used in thousands of organizations, ranging from government agencies to small businesses. Databases often replace earlier file processing systems, in which departments have control over their own data. Some of the advantages of the integrated database approach are:

- **Sharing of data:** Many users can be authorized to access the same piece of information.
- **Control of redundancy:** Information is integrated so that several copies of the same data are not stored.
- **Data Consistency:** If a data item appears only once, any update to its value needs to be performed only once, and all users have immediate access to the new value.
- **Improved Data standards:** DBA can define and enforce organization-wide standards for representation of data in the database.
- **Better data security:** All authorized access to the database is through the DBMS, which can require that users go through security procedures and additional passwords to gain access to data.
- **Improved data integrity:** DBMSs allow the DBA to define integrity constraints or consistency rules that the database must obey.
- **Faster development of new Applications:** A well-designed database provides an accurate model of the operations of the organization.
- **Better data accessibility:** DBMS provides Query Language that permits users to ask questions and obtain the required information.
- **Availability:** Availability means that authorized users can access and change data as needed to support the business. Increasingly, businesses are coming to expect their data to be available at all times ("24x7", or 24 hours a day, 7 days a week,).
- **Economy of scale:** The portion of the budget that would ordinarily be allocated to various departments for their data design, storage and maintenance costs can be pooled, possibly resulting in a lower total cost.
- **More control over concurrency:** The goal in a 'concurrent' DBMS is to allow multiple users to access the database simultaneously without interfering with each other. Most DBMSs have subsystems to control concurrency so that transactions are not lost or performed incorrectly.

- **Better back-up and recovery procedures:** When ever the database is modified, a log entry is made. If the system fails, the tape and log are used to bring the database to the state it was in just prior to the failure.

Some of the disadvantages of the integrated / Relational database approach are:

- High cost of the DBMS
- Higher Hardware costs
- Higher programming cost
- Require Specialized IS roles
- More difficult recovery

DBMS Functions

1. Data Dictionary Management
2. Data Storage Management
3. Data Transformation and Presentation
4. Security Management
5. Multi-User Access Control
6. Backup and Recovery Management
7. Data Integrity Management
8. Database Access Languages (DDL and DML) and Application Programming Interfaces
9. Database Communication Interfaces

DBMS Interfaces

■ **Menu-based interfaces:**

- They present the user with lists of menus to formulate his request.
- The menus generate the request and send it to the DBMS.

■ **Form-based interfaces:**

- They display forms to users to help them in entering new data or retrieving required data.

■ **Graphical user interfaces:**

- They display the schema in diagrammatic form where users can specify their queries by manipulating these diagrams.

Components of a Database Management System (DBMS)

A DBMS may have a number of components including:

1. Query Language:

- A query language is an easy-to-use computer language for making queries to a database and for retrieving information from a database.
- There are several different query languages; one of the most popular is *Structured Query Language, or SQL*.

2. Data Dictionary:

- The dictionary or catalog stores information about the database itself
- This is data about data or ‘metadata’
- Almost every aspect of the DBMS uses the dictionary
- The dictionary holds
 - Descriptions of database objects (tables, users, rules, views, indexes,...)
 - Information about who is using which data (locks)
 - Schemas and mappings

3. Utilities:

- The DBMS utilities are programs that enables users to maintain the database.

4. Report Generator:

- The report generator aspect of DBMS software simplifies the process of generating an on-screen or printed-out report.

5. Access Security:

- Access security is a feature that allow database administrators (DBA) to specify different access privileges for **different** users of a DBMS.

6. System Recovery:

- Some advanced database management systems have a system recovery feature, which enables the DBA to recover contents of the database in the event of a hardware or software failure.

History of Database Systems

- First-generation
 - Hierarchical and Network
- Second generation
 - Relational
- Third generation
 - Object Relational
 - Object-Oriented

Lecture#4**Types of DBMS**

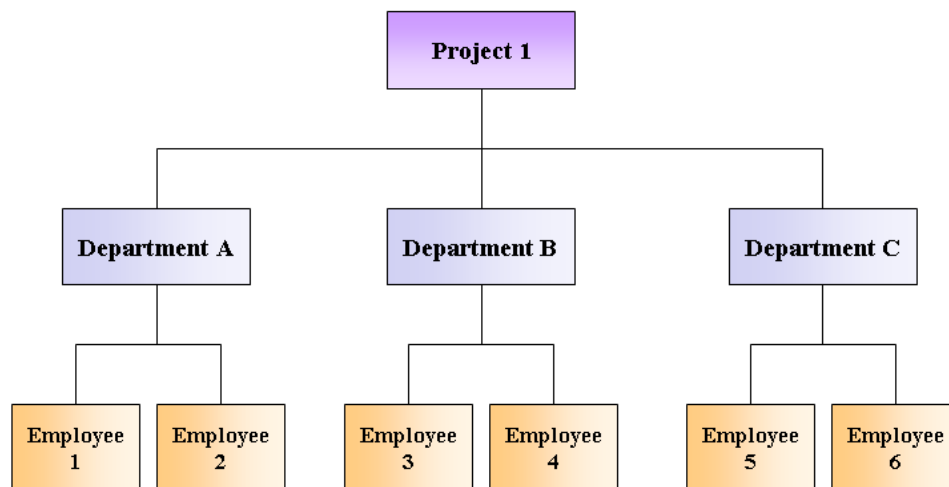
Several different types of DBMSs have been developed to support these requirements. These systems can broadly be classified in the following classes:

Hierarchical Database Model

- A data model in which data are organized in a top-down, or inverted tree structure

Basic Structure

- Collection of records logically organized to conform to the upside-down tree (hierarchical) structure.
- The top layer is perceived as the parent of the segment directly beneath it.
- The segments below other segments are the children of the segment above them.
- A tree structure is represented as a hierarchical path on the computer's storage media.



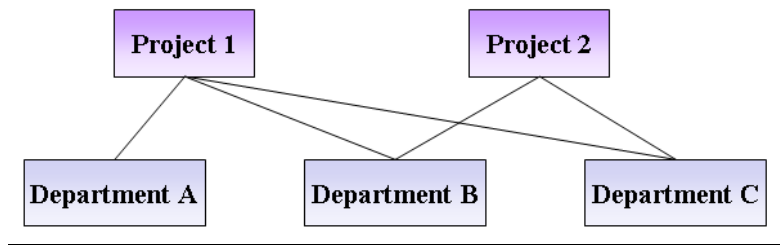
A hierarchical database consists of the following:

- It contains nodes connected by branches.
- The top node is called the root.
- One parent may have many children

Network Database Model

- An expansion of the hierarchical database model with an owner-member relationship in which a member may have many owners.

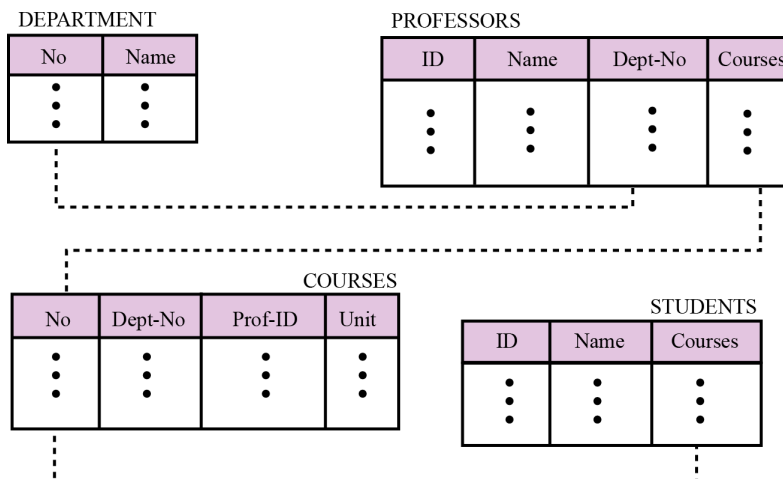
- In the network model, the entities are organized in a graph, in which some entities can be accessed through several paths



Relational Database Model

- In the relational model, data is organized in two-dimensional tables called relations. The tables or relations are, however, related to each other

An example of the relational model representing a university



Basic Structure

- RDBMS allows operations in a human logical environment.
- The relational database is perceived as a collection of tables.
- Each table consists of a series of row/column intersections.
- Tables (or relations) are related to each other by sharing a common entity characteristic.
- The relationship type is often shown in a relational schema.
- A table yields complete data and structural independence.

Object-oriented DBMS: These systems can handle objects such as videos, images, pictures, and so on.

Database Objects

Following objects / components are included in a database:

1. Tables:

- The table is a basic unit of storage in a relational database management system.
- It is an object in a database that represents entities & stores data as a collection of rows and columns.
- They consist of *rows* (also known as *tuples* or *instances* in the world of relational theory and modeling, respectively) and *columns* (*attributes*).

2. Forms:

- A form is a window that displays a collection of controls, such as, text boxes, check boxes and list for viewing, entering and editing the information in database fields.
- A table stores the actual database data while a form is merely a tool for viewing or modifying the data.

3. Reports:

- Reports are used for printing information from the database.
- A report can combine data from more than one table. Reports are designed by adding visual objects.

4. Queries:

- Queries are used to gather selected information from database and organize it either for use in reports or for viewing on screen.
- Each query consists of one or more criteria that we use to create a pattern or rule for selecting matching records.
- The data in each record is compared to the query criteria and if the information in the record matches the criteria, the record is included in the query's result.

User Types

When considering users of a Database system, there are three broad classes to consider:

1. **The Application Programmer**, responsible for writing programs in some high-level language such as COBOL, C++, etc.

2. **The End-User**, who accesses the database via a query language
3. **The Database Administrator (DBA)**, who controls all operations on the database

Metadata

- Metadata is data about data. .
- The data dictionary (also called system catalog) stores metadata
- For example:
 - Information about relations
 - names of relations
 - names and types of attributes of each relation
 - names and definitions of views
 - integrity constraints
 - User and account information, including passwords
 - Statistical and descriptive data
 - number of tuples in each relation

It is possible to create meta-meta-...-metadata. Since, according to the common definition, metadata itself is data, it is possible to create metadata about metadata, metadata about metadata about metadata and so on.

Lecture#5**Database Administrator (DBA)**

The database Administrator (DBA) is the person who makes the strategic and policy decisions regarding the data of the enterprise, The DBA is responsible for the design, operation and management of the database. He or she must be technically competent, a good manager, a skilled diplomat and possess excellent communication skills.

- Management skills are required to plan, coordinate and carry out a multitude of tasks during all phases of the database project and to supervise a staff.
- Technical skills are needed because the DBA has to be able to understand the complex hardware and software issues involved and to work with system and application experts in solving problem.
- Diplomatic skills are used to communicate with users and determine their needs, to negotiate agreements on data definitions and database access rights, to ensure agreements on changes to the database structure or operations that affect users and to mediate between users with conflicting requirements.
- Excellent communication skills are required for all of these activities.

Database administrator's duties include:

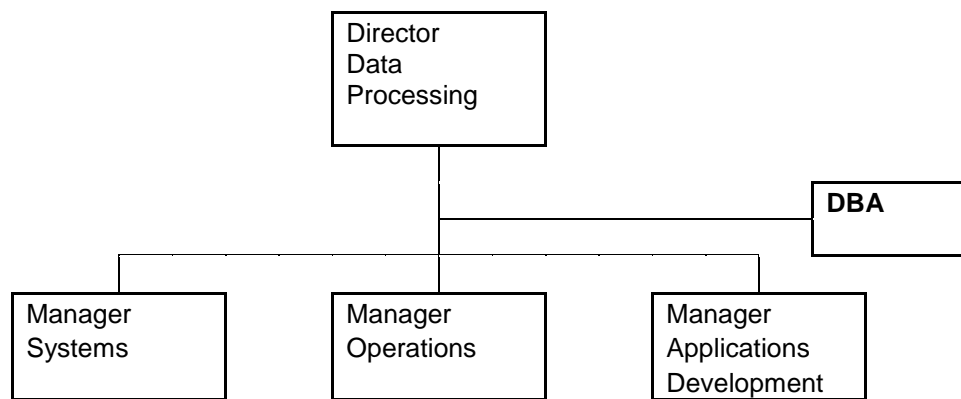
- Storage structure and access method definition
- Schema and physical organization modification
- Granting users authority to access the database
- Backing up data
- Monitoring performance and responding to changes
 - ▶ Database tuning

DBA Staff

The size of the staff depends on the phase of the database project, the size of the organization, the importance of the database within the organization, the sophistication of its data processing operations. In large organization with sophisticated data processing needs several people may be assigned to each function. In a small organization, a single DBA may be responsible for all of these tasks.

Placement of the DBA

The placement of the DBA also varies but it often a staff position reporting directly to the director of data processing. Large organizations with many databases may have a **Data Administrator**, who responsible for the entire information resources, as well as one or more database administrators.



Overview of Relational databases

A **relational database** is based on the relational model as introduced by Edgar F. Codd. A relational database stores all its data inside tables, and nothing more. All operations on data are done on the tables themselves or produce other tables as the result. You never see anything except for tables.

<i>customer_id</i>	<i>customer_name</i>	<i>customer_street</i>	<i>customer_city</i>	<i>account_number</i>
192-83-7465	Johnson	12 Alma St.	Palo Alto	A-101
192-83-7465	Johnson	12 Alma St.	Palo Alto	A-201
677-89-9011	Hayes	3 Main St.	Harrison	A-102
182-73-6091	Turner	123 Putnam St.	Stamford	A-305
321-12-3123	Jones	100 Main St.	Harrison	A-217
336-66-9999	Lindsay	175 Park Ave.	Pittsfield	A-222
019-28-3746	Smith	72 North St.	Rye	A-201

A table is a set of rows and columns. Each row is a set of columns with only one value for each. All rows from the same table have the same set of columns, although some columns may have NULL values, i.e. the values for that rows was not initialized. Note that a NULL value for a string column is different from an empty string. You should think about a NULL value as an "unknown" value.

Properly managed relational databases minimize the need for application programs to contain information about the physical storage of the data they access. To maximize the isolation of programs from data structures, relational DBMSs restrict data access to the messaging protocol SQL, a nonprocedural language that limits the programmer to specify desired results.

Codd's Relational Database

Codd's idea for an RDB uses the mathematical concepts of relational algebra to break down data into sets and related common subsets. Because information can naturally be grouped into distinct sets, Dr. Codd organized his database system around this concept. Under the relational model, data is separated into sets that resemble a table structure. This table structure consists of individual data elements called columns or fields. A single set of a group of fields is known as a record or row. Through the mathematical concepts of join and union, relational databases can quickly retrieve pieces of data from different sets (tables) and return them to the user or program as one "joined" collection of data.

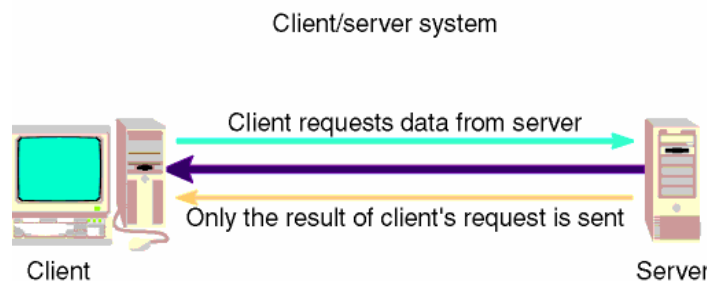
Dr. Codd s 12 Rules for a Relational Database Model

The most popular data storage model is the relational database, which grew from the seminal paper "A Relational Model of Data for Large Shared Data Banks," written by Dr. E. F. Codd in 1970. SQL evolved to service the concepts of the relational database model. Dr. Codd defined 13 rules, oddly enough referred to as Codd's 12 Rules, for the relational model:

0. A relational DBMS must be able to manage databases entirely through its relational capabilities.
1. Information rule-- All information in a relational database (including table and column names) is represented explicitly as values in tables.
2. Guaranteed access--Every value in a relational database is guaranteed to be accessible by using a combination of the table name, primary key value, and column name.
3. Systematic null value support--The DBMS provides systematic support for the treatment of null values (unknown or inapplicable data), distinct from default values, and independent of any domain.
4. Active, online relational catalog--The description of the database and its contents is represented at the logical level as tables and can therefore be queried using the database language.
5. Comprehensive data sublanguage--At least one supported language must have a well-defined syntax and be comprehensive. It must support data definition, manipulation, integrity rules, authorization, and transactions..
6. View updating rule--All views that are theoretically updatable can be updated through the system.
7. Set-level insertion, update, and deletion--The DBMS supports not only set-level retrievals but also set-level inserts, updates, and deletes.
8. Physical data independence--Application programs and ad hoc programs are logically unaffected when physical access methods or storage structures are altered.
9. Logical data independence--Application programs and ad hoc programs are logically unaffected, to the extent possible, when changes are made to the table structures.
10. Integrity independence--The database language must be capable of defining integrity rules. They must be stored in the online catalog, and they cannot be bypassed.
11. Distribution independence--Application programs and ad hoc requests are logically unaffected when data is first distributed or when it is redistributed.
12. Nonsubversion--It must not be possible to bypass the integrity rules defined through the database language by using lower-level languages.

Lecture#6**Client-Server and Multi-Tier Architecture of Database Management System**

Client/server is a system of computing in which two or more computers share processing across a network. In this system, a single application is partitioned between multiple processors (front-end and back-end) to complete the processing as a single unified task. Using SQL and a network connection, the application can interface to a database residing on a remote server. The increased power of personal computer hardware enables critical database information to be stored on a relatively inexpensive standalone server.



- The server is just the DBMS itself. It supports all of the basic DBMS functions, data definitions, data manipulation, data security and integrity, and so on. The server portion receives and processes SQL and PL/SQL statements originating from client applications. The computer that manages the server portion must be optimized for its duties.

Characteristics of a server:

- Passive (slave)
 - Waits for requests
 - Upon receipt of requests, processes them and then serves replies
- The clients are the various applications that run on top of the DBMS – both user-written applications and built-in applications i.e. Software applications that request the services, data, or processing of another application or computer (the "server"). In a two-task environment, the client is the user process. In a network environment, the client is the local user process and the server may be local or remote.

Characteristics of a client:

- Active (master)
- Sends requests
- Waits for and receives server replies

Client/server advantages

- Client/server systems are scalable, powerful, and flexible
- Businesses can size their systems easily to a changing environment
- Communication is possible across multiple platforms

Multi-tier Architecture

In software engineering, multi-tier architecture (often referred to as *n-tier architecture*) is a client-server architecture in which an application is executed by more than one distinct software agent. Generic client/server architecture has two types of nodes on the network: clients and servers. As a result, these generic architectures are sometimes referred to as "two-tier" architectures.

Some networks will consist of three different kinds of nodes: server, application servers which process data for the clients and database servers which store data for the application servers. This is called a three-tier architecture.

The advantage of an n-tier architecture compared with a two-tier architecture (or a three-tier with a two-tier) is that it separates out the processing that occurs to better balance the load on the different servers; it is more scalable. The disadvantages of n-tier architectures are:

1. It puts a more load on the network.
2. It is much more difficult to program and test software than in two-tier architecture because more devices have to communicate to complete a user's transaction.

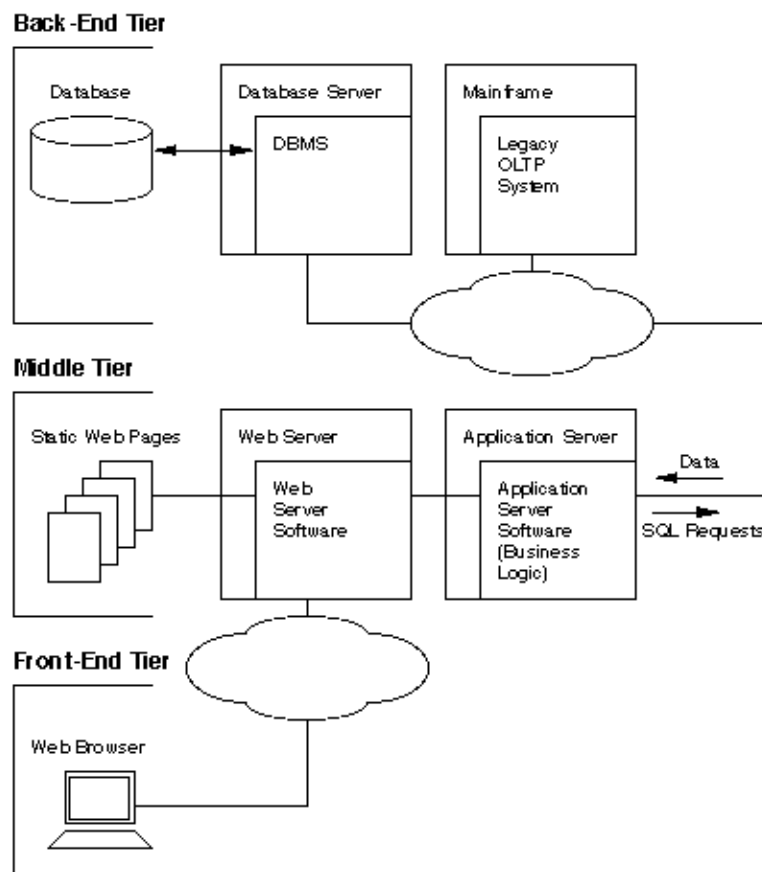
The most widespread use of "multi-tier architecture" refers to three-tier architecture.

Three-tier Architecture

At first, the Web was used to access static documents. Now companies want to use web browsers as a way to provide access to corporate databases. This requires linking the web server to the database system. The methods used to link web servers and DBMS systems have evolved into the three-tier network architecture shown below. In the "front" tier, is the user interface and a web browser running on a PC or "thin client".

The front tier communicates with a web server in the "middle tier." When the user request is for something more complex than a simple web page, the web server passes the request to an application server. The application server role is to handle the business logic required to process the request. Often, the request will involve access to a corporate database. The database systems run in the "back" tier of the architecture. SQL is the standard database language for communication between the application server and back-end database. All of the packaged application server products provide a SQL-based API for database access.

Database Management in a Three-tier Internet Architecture



Lecture#7**DBMS Languages**

- SQL (Structured Query Language) is a relational DB language includes a combination of DDL and DML and also statements for schema updating.

- **DDL (data definition language)**

- It is used by the DBA and the DB designers to define their schemas.
- Permits specification of data types, structures and any data constraints.
- All specifications are stored in the database.
- DDL SQL commands include the following:
 - Create - To make a new database, table, index, or stored query.
 - Drop - To destroy an existing database, table, index, or view.
 - Alter table – To change or inserts a column in a table
 - Rename table – To change the name of a table, view or sequence
 - Truncate table – To remove all rows from a table

- **DML (data manipulation language)**

- It is used in retrieving, inserting, deleting and manipulating data.

Overview of query processing

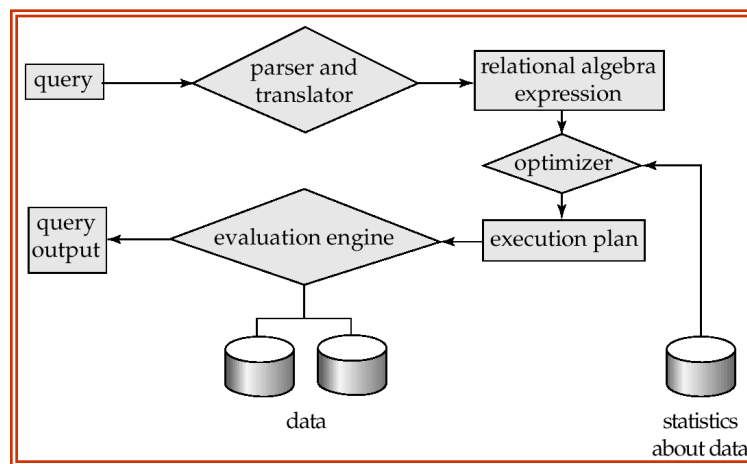
SQL is the de facto standard language used to manipulate and retrieve data from the relational databases. SQL enables a programmer or database administrator to do the following:

- Modify a database's structure
- Change system security settings
- Add user permissions on databases or tables
- Query a database for information
- Update the contents of a database

A SQL command (specifically, a SQL SELECT command) that retrieves information from one or more tables, in any combination, expression, or order. Queries are read-only operations; they do not change any data, they only retrieve data. Queries are often considered to be Data Retrieval statements. The data retrieved by a query are called **Query Results**.

Query Processing Steps:

1. Parsing and translation
2. Optimization
3. Evaluation



Data retrieval

The most frequently used operation in transactional databases is the data retrieval operation. The syntax of query is:

```
SELECT    column(s)
FROM      table(s)
WHERE     condition(s)
GROUP BY  column
HAVING    group_condition
ORDER BY  column;
```

- **SELECT** is used to retrieve zero or more rows from one or more tables in a database. In most applications, SELECT is the most commonly used DML command. Commonly available keywords related to SELECT include:
- **FROM** is used to indicate from which tables the data is to be taken, as well as how the tables join to each other.
- **WHERE** is used to identify which rows to be retrieved, or applied to GROUP BY.
- **GROUP BY** is used to combine rows with related values into elements of a smaller set of rows.

- **HAVING** is used to identify which of the "combined rows" (combined rows are produced when the query has a GROUP BY keyword or when the SELECT part contains aggregates), are to be retrieved.
- **ORDER BY** is used to identify which columns are used to sort the resulting data.

Transaction Processing

- A logical unit of work or a set of one or more commands issued against the database that comprises one or more SQL statements executed by a single user.
- After work is performed in a transaction, two outcomes are possible:
 - **Commit** - Any changes made during the transaction are committed to the database.
 - **Abort/Roll back** - All the changes made during the transaction are not made to the database.
- A transaction is atomic, consistent, isolated, and durable.

Data Redundancy

- In non-database systems each application has its own private files. This can often lead to redundancy in stored data, with resultant waste in storage space. In a database the data is integrated.
- Redundancy is
 - direct if a value is a copy of another
 - indirect if the value can be derived from other values:
- Data redundancy can lead to inconsistency in the database unless controlled. The system should be aware of any data duplication - the system is responsible for ensuring updates are carried out correctly.

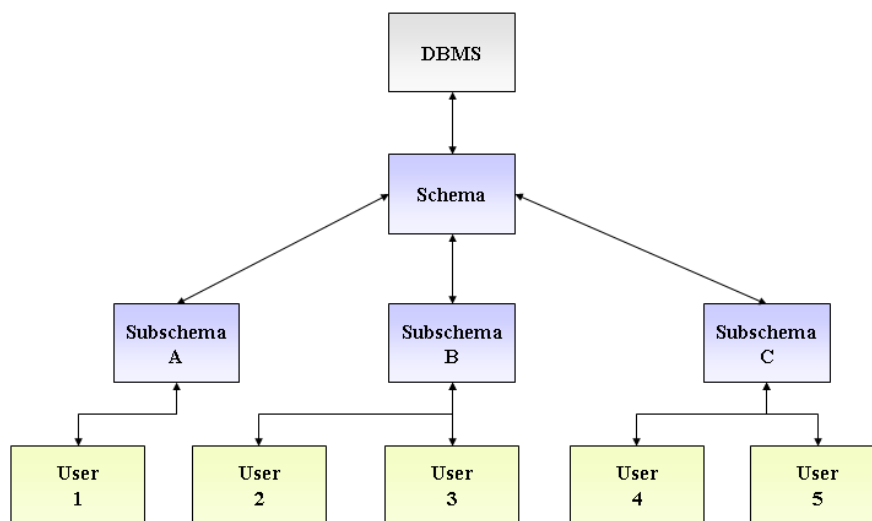
Data Integrity

- **Data integrity** is the accuracy and reliability of data. It is important in both single-user and multi-user environments.
- This describes the problem of ensuring that the data in the database is accurate... Integrity is important in a database system - an application running without validation procedures can produce erroneous data, which can then affect other applications using that data.
- Centralized control of the database helps maintain integrity, and permits the DBA to define validation procedures to be carried out whenever any update operation is attempted.
- Integrity checks on data items can be divided into 4 groups:
 1. Type Checks
 2. Redundancy Checks
 3. Range Checks
 4. Comparison Checks

Lecture#8

Schema

- The word **schema** comes from the Greek word "σχῆμα" (skhēma, pronounced *skee-ma*), which means *shape* or more generally *plan*. In English, both *schemas* and *schemata* are used.
- The description of a database is called the DB Schema.
- In database, schema is a description of the structure of a database generated by the data definition language (DDL) of the database management system (DBMS).
- In a relational database, the schema defines the tables, the fields in each table, and the relationships between fields and tables.
- It is specified during the DB design
- It is not expected to be changed frequently
- It is usually represented by diagrams (Schema diagram)
- Each object of the schema is called a schema construct
- It specifies some aspect (names of records and data items) and some other aspects as data type and relationships are not specified.
- The schema is also called the “intension”.
- The stored description of the schema is called the “meta-data” & generally stored in a data dictionary.



- A single schema has three kinds of schema: logical schema, storage schema and user schema.

1. **Logical schema:** It defines what the data is, rather than how it may be stored and accessed. For any given database system there will be one logical schema. It is defined in a formal language that can be processed by a machine. That language is sometimes referred to as the **logical schema data definition language** (or **logical schema DDL**).
2. **User schema:** It is a definition of the logical properties of the data required by some user, or group of users, of the database, to be referenced by a user process for that user. Since there will typically be many users of a database, there will be many user schemas, each based on the same single logical schema. The only way a user may access the database is via a user schema. A user process may use only one user schema.
3. **Storage schema:** It is a definition of how the data defined in a logical schema is stored and accessed. It is defined in a language sometimes referred to as the **storage schema data definition language** (or **storage schema DDL**).

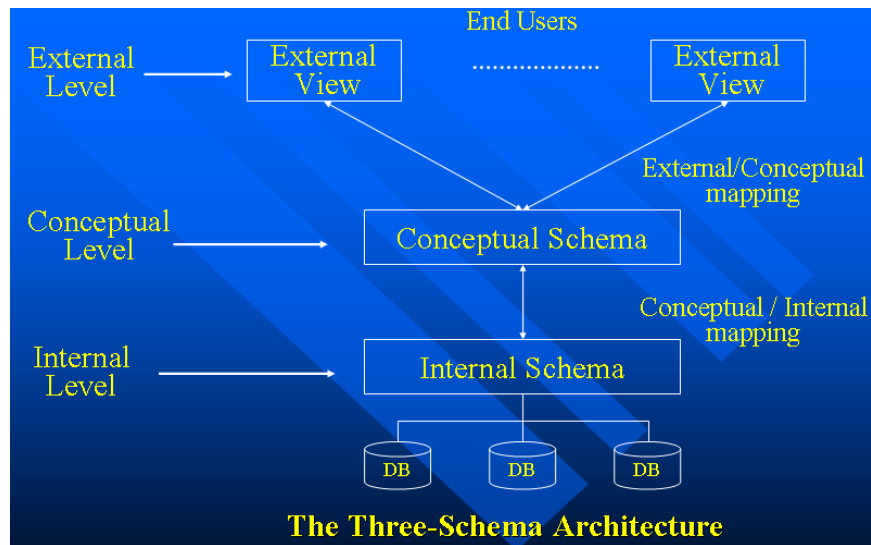
Data Independence

- It is the capacity to change the schema at one level of a database without having to change the schema at the next level.
- There are two types of Data Independence:
 1. **Logical data independence:**
 - Logical data independence is the ability of changing the conceptual Schema without having to change the external schema or the application programs.
 - Conceptual schema may be changed to expand the DB as adding new data items or reducing it by removing data items.
 - The applications involved in the modified data items can only be changed without any effect on the remaining applications.
 2. **Physical data independence:**
 - Physical data independence is the ability of changing the internal Schema without having to change the conceptual schema.
 - Internal schema may be changed due to some file reorganization to meet certain system performance.
 - In general, the interfaces between the various levels and components should be well defined so that changes in some parts do not seriously influence others.

DBS Architecture and Data Independence

The architecture for DBMSs is divided into three general levels:

- external
- conceptual
- internal



1. The Internal Level:

- has an internal schema which describes the physical storage of the DB.
- Internal schema has a Physical Data Model.

2. The conceptual Level:

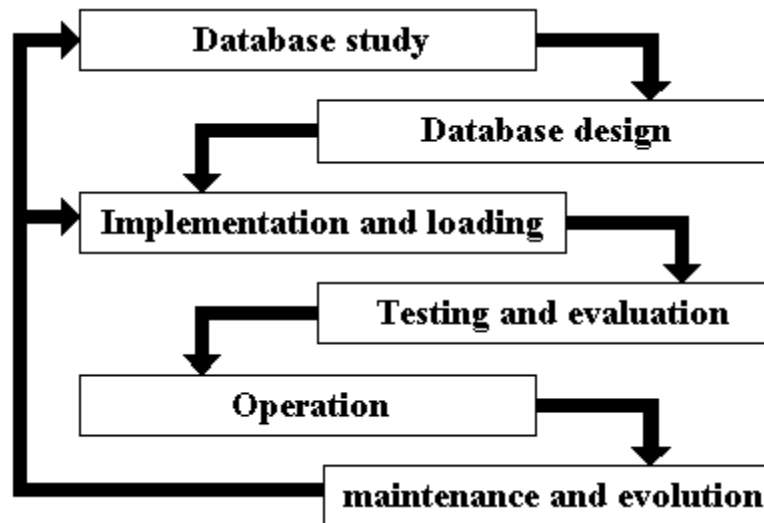
- has a conceptual schema which describes the structure of the DB
- hides the details of the physical storage structure.
- describes the entities, data types, relationships, constraints and user operations.
- high-level conceptual data model or implementation data model can be used at this level.

3. The External or View Level:

- has a number of external schemas or user views.
- provides each group of users by a view which describes a part of the DB which is interested to that group.
- also high-level conceptual data model or implementation data model can be used at this level.

Database Analysis Life Cycle

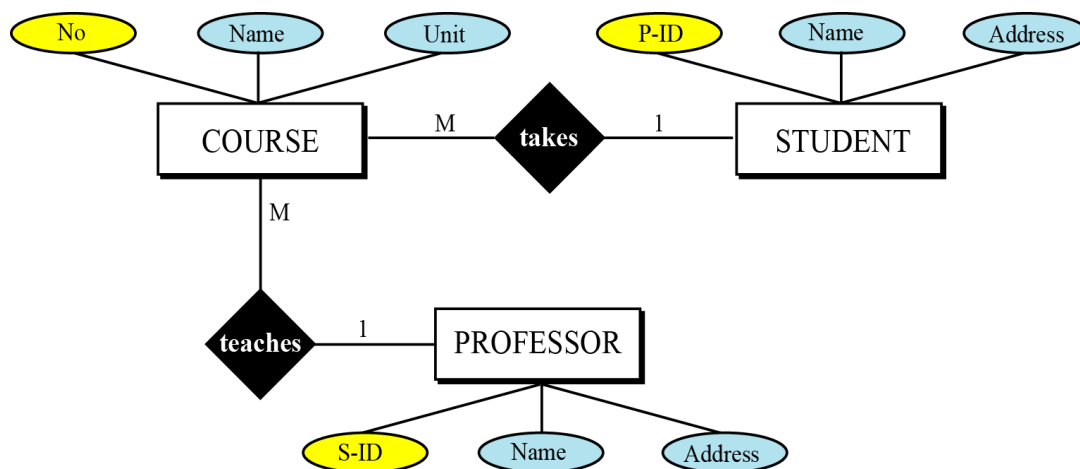
When a database designer is approaching the problem of constructing a database system, the logical steps followed is that of the database analysis life cycle:



- **Database study** - here the designer creates a written specification in words for the database system to be built. This involves:
 - Analyzing the company situation
 - Define problems and constraints
 - Define objectives
 - define scope and boundaries
- **Database Design** - conceptual, logical, and physical design steps in taking specifications to physical implementable designs. This is looked at more closely in a moment.
- **Implementation and loading** - Once a DBMS has been installed, the database itself must be created within the DBMS. Finally, not all databases start completely empty, and thus must be loaded with the initial data set (such as the current inventory, current staff names, current customer details, etc).
- **Testing and evaluation** - the database, once implemented, must be tested against the specification supplied by the client.
- **Operation** - this step is where the system is actually in real usage by the company.
- **Maintenance and evolution** - designers rarely get everything perfect first time, and it may be the case that the company requests changes to fix problems with the system or to recommend enhancements or new requirements.


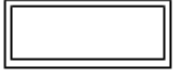







Chapter # 2**Entity – Relationship Model****Lecture# 9****Entity Relationship Diagram / Model**

- ER model was proposed by Peter Chen in 1976.
- ER model has become the standard tool for *conceptual schema* design
- It is one of the most widely accepted graphical data modeling tools.
- ER model consists of three basic constructs: entities, attributes and relationships.
- It graphically represents data as entities and their relationships in a database structure.
- **Example:** Figure shows a very simple E-R diagram with three entity sets, their attributes and the relationship between the entity sets

**E-R Diagram Components**

- **Rectangles** represent entity sets.
- **Ellipses** represent attributes.
- **Diamonds** represent relationship sets.
- **Lines** link attributes to entity sets and entity sets to relationship sets.
- **Double ellipses** represent multivalued attributes.
- **Dashed ellipses** denote derived attributes.
- Primary key attributes are underlined.

NOTATION FOR ER SCHEMAS

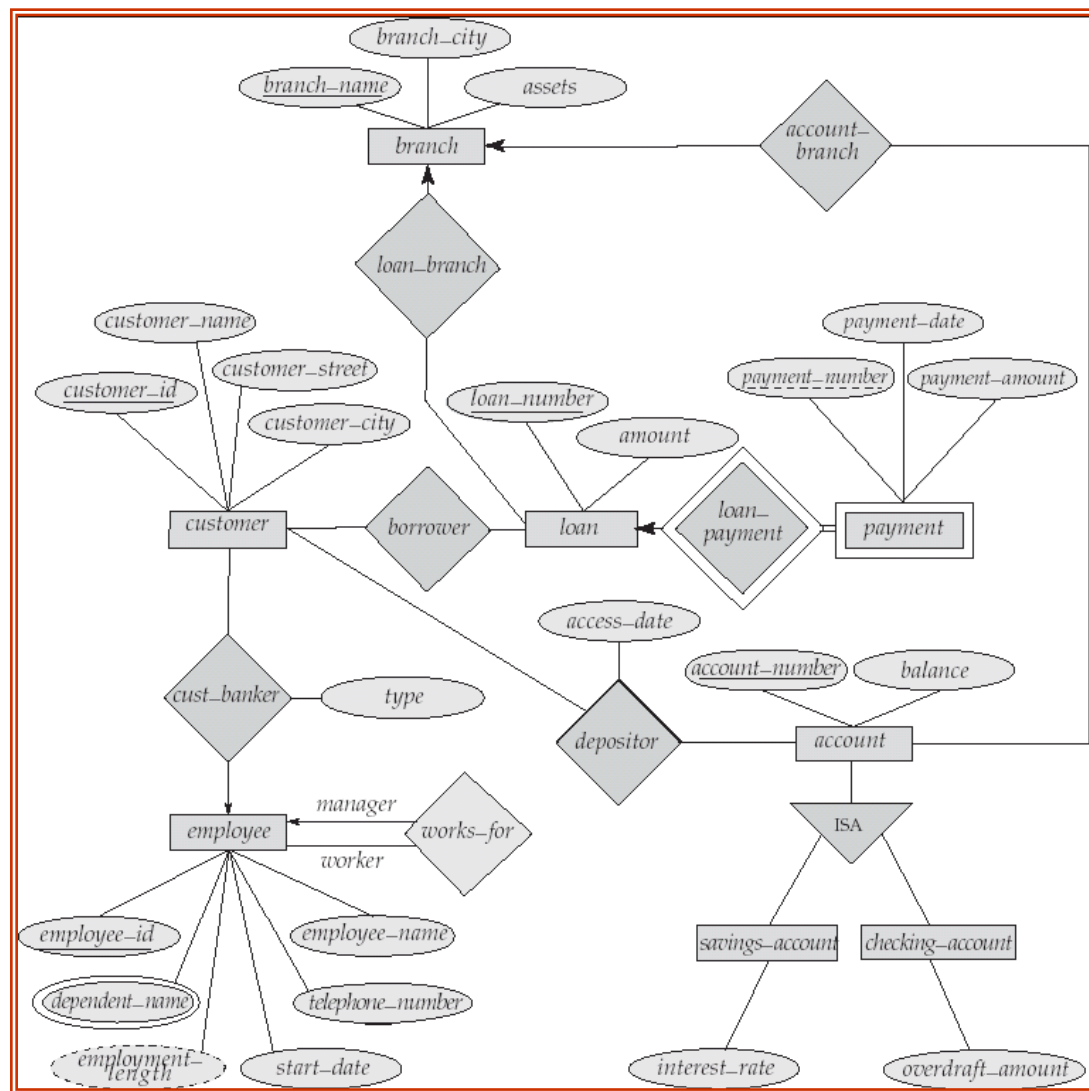
<u>Symbol</u>	<u>Meaning</u>
	ENTITY TYPE
	WEAK ENTITY TYPE
	RELATIONSHIP TYPE
	IDENTIFYING RELATIONSHIP TYPE
	ATTRIBUTE
	KEY ATTRIBUTE
	MULTIVALUED ATTRIBUTE
	COMPOSITE ATTRIBUTE
	DERIVED ATTRIBUTE

Entity-Relationship Data Model**Advantages**

- Visual representation
- Effective communication tool
- Integrated with the relational database model

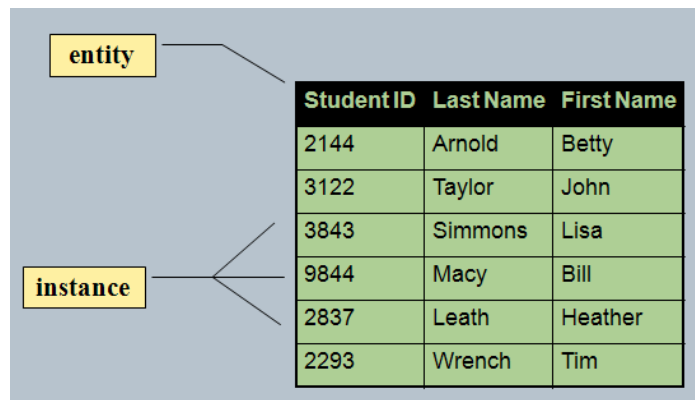
Disadvantages

- Limited constraint representation
- Limited relationship representation
- No data manipulation language
- Loss of information content

Example: E-R Diagram for a Banking Enterprise

Lecture# 10**Entity**

- An entity is a “thing” in the real world with an independent existence.
- An entity is analogous to a table in the relational model
- It may be an object with physical existence (e.g. person, car, house, employee), or it may be an object with conceptual existence (company, job, university, course).
- **Two types of entities:**
 - **Strong entity:** can exist independently (or can uniquely identify itself)
 - **Weak entity:** existence depends on the existence of other (strong) entity or entities
- **Examples:**
 - An employee is a strong entity but the dependents of the employee could be weak entities
 - An account in a bank is a strong entity but a transaction could be a weak entity
- An entity occurrence (also called an instance) is an individual occurrence of an entity. An occurrence is analogous to a row in the relational table.

**Relationships**

- A Relationship represents an association between two or more entities.
- An example of a relationship would be:
 - Employees are assigned to projects
 - Projects have subtasks
 - Departments manage one or more projects
- Relationships are classified in terms of degree, connectivity, cardinality, and existence.

Attributes

- An entity is represented by a set of attributes, that is, descriptive properties possessed by all members of an entity set.

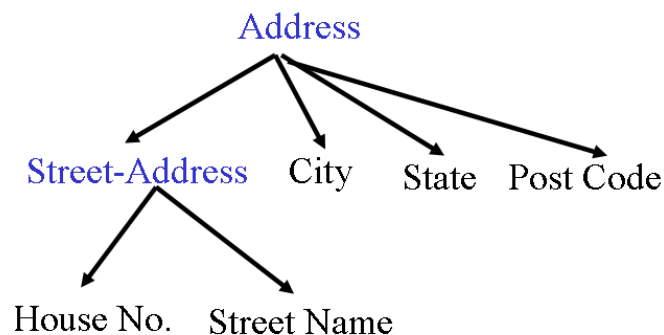
Examples:

customer = (customer-name, NIC#, customer-street, customer-city)

account = (account-number, balance)

- *Domain* -- the set of permitted values for each attribute
- An entity will have a value for each of its attributes
- **Attribute types:**
 - Simple and composite attributes.
 - Single-valued and multi-valued attributes.
 - Null attributes
 - Derived attributes
- **Simple** (or *atomic*) attribute is a one which cannot be divided into smaller parts.
 - Examples:
 - NIC#, GPA, Salary.
- **Composite** attribute is an attribute which can be divided into smaller subparts, these subparts represent more basic attributes with independent meanings of their own
 - Examples:
 - Name: First_Name, Middle_Name, Last_Name
 - Address: Street_Address, City, State, Post code

An Example of a composite attribute



- A single-valued attribute is a one which has one (single) value for a particular entity.

- ❑ Example: Age, **BirthDate**
- A multi-valued attribute is a one which may have one or more values for the same entity.
 - ❑ **College Degrees for Person: 0, 1, 2, 3, ...**
 - ❑ Color for a Car: 1, 2,
 - ❑ Author of Books
 - ❑ Phone Number
- Derived-attributes: whose values are computed from other attributes.
 - ❑ Age from Birthdate
 - ❑ Annual Salary from Monthly Salary
 - ❑ NoOfEmployees ==> Count number of employees in the Employee table.
- Attributes can be classified as identifiers or descriptors. Identifiers, more commonly called keys, uniquely identify an instance of an entity. A descriptor describes a non-unique characteristic of an entity instance.

Classifying Relationships

- Relationships are classified by their degree, connectivity, cardinality, direction, type, and existence.

Degree of a Relationship Set

- Refers to number of entity sets that participate in a relationship set.
- Relationship sets that involve two entity sets are *binary* (or degree two). Generally , most relationship sets in a database system are binary
- Relationship sets may involve more than two entity sets. The entity sets *customer*, *Product* and *Loan* may be linked by the ternary (degree three) relationship set.

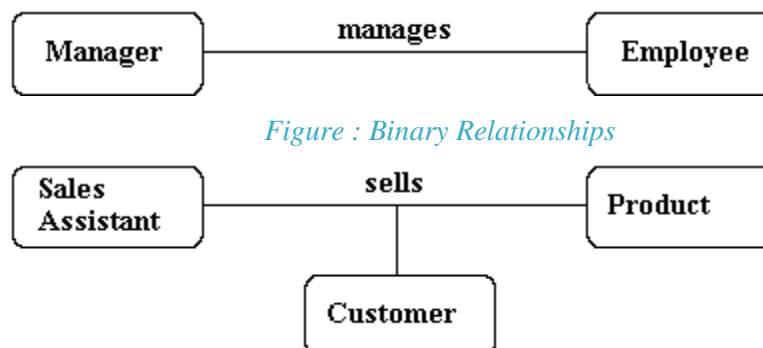


Figure : Binary Relationships

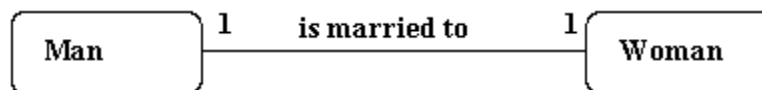
Figure : Ternary relationship

Cardinality of Relationships

- The cardinality of a relationship is the actual number of related occurrences for each of the two entities. The basic types of connectivity for relations are:

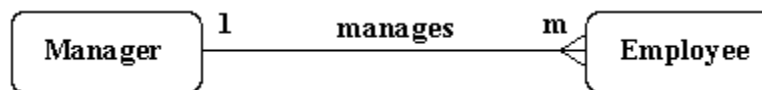
■ **One-to-One**

- Each entity in the relationship will have exactly one related entity
- A department has one manager and a manager manages one department
- a man can only marry one woman, and a woman can only marry one man, so it is a one to one (1:1) relationship



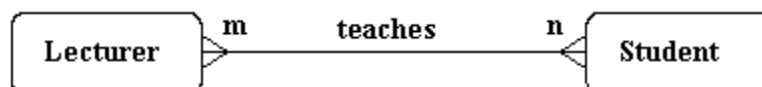
■ **One-to-Many**

- An entity on one side of the relationship can have many related entities, but an entity on the other side will have a maximum of one related entity
- An employee works for one company, and a company has many employees working for it
- one manager manages many employees, but each employee only has one manager, so it is a one to many (1:m) relationship



■ **Many-to-Many**

- Entities on both sides of the relationship can have many related entities on the other side
- An employee works on many projects, and a project has many employees working on it
- One lecturer teaches many students and a student is taught by many lecturers, so it is a many to many (m:n) relationship



Lecture# 11**Entity and Entity Set**

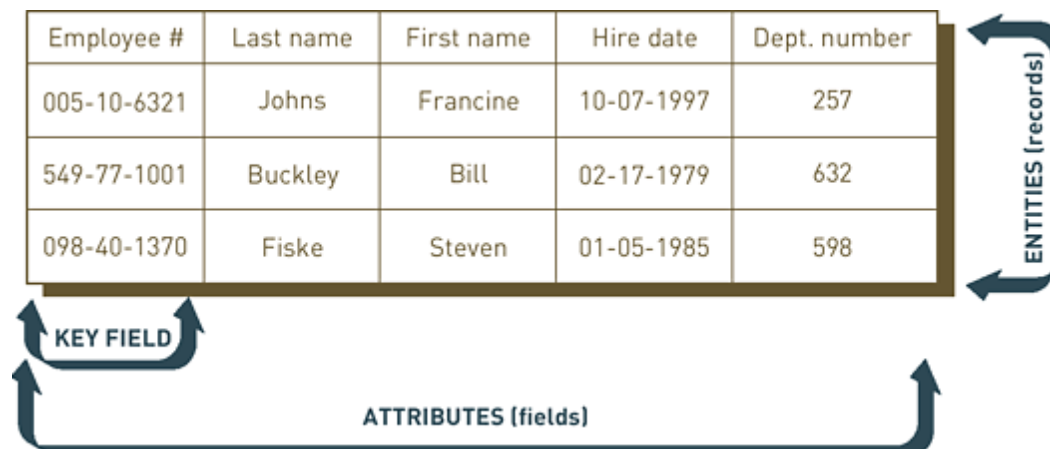
- A *database* can be modeled as:
 - a collection of entities,
 - Relationships among entities.
- An *entity set* is a set of entities of the same type that share the same properties.
Example: set of all persons, companies etc.
- An entity type defines a set of entities that have the same attributes.

- **STUDENT** is an **entity type (Schema)**

- Examples:

- **Rema, Ali, Amal, Samer, Rana** are entity set of an entity type **STUDENT**

Employee #	Last name	First name	Hire date	Dept. number
005-10-6321	Johns	Francine	10-07-1997	257
549-77-1001	Buckley	Bill	02-17-1979	632
098-40-1370	Fiske	Steven	01-05-1985	598



- Formally, an attribute is a **function** which maps an entity set into a domain.
 - Every entity is described by a set of (attribute, data value) pairs.
 - There is one pair for each attribute of the entity set.

- E.g. a particular employee entity is described by the set {(Employee #, 005-10-6321), (Last name, Johns), (First name, Francine), (Hire date, 10-07-1997), (Dept_number, 257)}.
- Sometimes it is useful to try out various examples of entities from an ER model. One reason for this is to confirm the correct cardinality and optionality of a relationship. We use an 'entity set diagram' to show entity examples graphically. Consider the example of 'course is_studied_by student'.

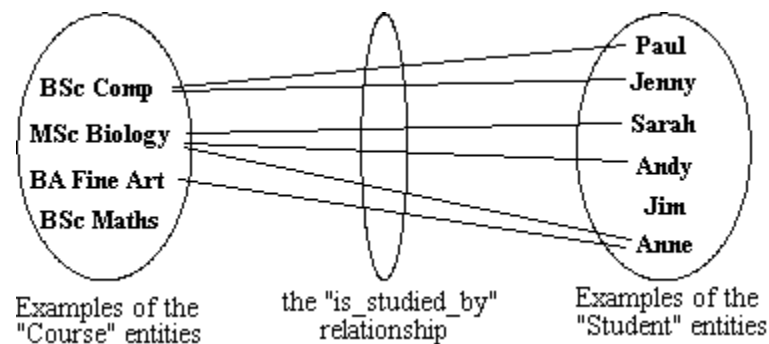


Figure : Entity set example

Confirming Correctness

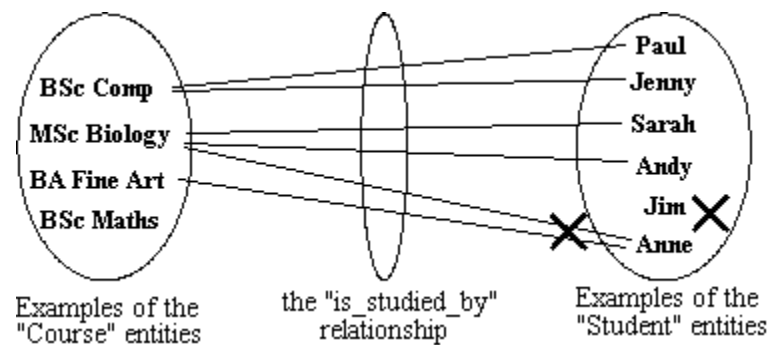


Figure : Entity set confirming errors

- Use the diagram to show all possible relationship scenarios.
- Go back to the requirements specification and check to see if they are allowed.
- If not, then put a cross through the forbidden relationships
- This allows you to show the cardinality and optionality of the relationship

Keys:

- A key is a data item that uniquely identifies a row, & defines the relationship between two tables.

Super keys

- A *super key* of an entity set is a set of one or more attributes whose values uniquely determine each entity
 - For example, in the entity set student, *student-name* and *StuID* is a super key.
 - Note that *student-name* alone is not, as two students could have the same name.
 - A super key may contain extraneous attributes, and we are often interested in the smallest super key.

Candidate Key

- It is a unique identifier for a row within a database table. A candidate key can be made up of one or more columns. In a normalized database, every table must have at least one candidate key.
- A candidate key is a super key that does not contain extra attributes. A *candidate key* of an entity set is a minimal super key
- There may be several candidate keys for an entity set.
- Note that a candidate key may consist of a single attribute, as *StuID* or it may be a combination of attributes, for example, the combination {name, address }
- When a key consists of more than one attribute, we call it a **Composite Key**.
- However, it is possible for a table to have more than one candidate key

Primary Key:

- A **primary key** is a candidate key, chosen by the DB designer to identify uniquely entities in an entity set.
- Although several candidate keys may exist, one of the candidate keys is selected to be the primary key.
- Often other keys become **alternate key**; the term secondary key is also sometimes used.

- An important characteristic of a primary key is that none of its attributes may have null values.
- An entity set that does not possess sufficient attributes to form a primary key is called a **weak entity set**. One that does have a primary key is called a **strong entity set**.

Foreign Key

- The column or combination of columns whose values match the primary key (PK) or unique key in the same or another table is called foreign key.
- A foreign key does not have to be unique.
- When a foreign key appears, it is a signal that there is a relationship between the entities involved.
- A foreign key is often in a many-to-one relationship with a primary key.
- Foreign key values should be copies of the primary key values; no value in the foreign key except NULL should ever exist unless the same value exists in the primary key.

Common Key:

- A key created to make explicit a logical relationship between two tables in a database is called common key.

Referential Integrity

- In a relational database, referential integrity means that a foreign key value cannot be entered in one table unless it matches an existing primary key in another table.
- Ensures that vital data in a database, such as the unique identifier for a given piece of data, remains accurate and usable as the database changes.

Weak Entity Sets

- An entity set that does not have a primary key is referred to as a *weak entity set*.
- The existence of a weak entity set depends on the existence of a strong entity set; it must relate to the strong set via a one-to-many relationship set.
- The *discriminator* (or *partial key*) of a weak entity set is the set of attributes that distinguishes among all the entities of a weak entity set.
- The primary key of a weak entity set is formed by the primary key of the strong entity set on which the weak entity set is existence dependent, plus the weak entity set's discriminator.
- We depict a weak entity set by double rectangles.
- We underline the discriminator of a weak entity set with a dashed line.

Weak Entity Set Examples

- In a university, a *course* is a strong entity and a *course_offering* can be modeled as a weak entity
- The discriminator of *course_offering* would be *semester* (including year) and *section_number* (if there is more than one section)

- If we model *course_offering* as a strong entity we would model *course_number* as an attribute.
- Then the relationship with *course* would be implicit in the *course_number* attribute

Lecture# 12**Constructing an ER model****Steps:**

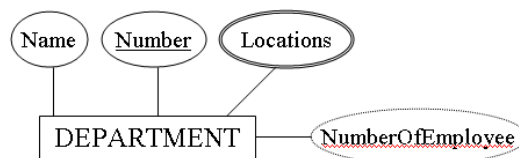
1. Identify entities
2. Remove duplicate entities
3. List the attributes of each entity
4. Mark the primary keys
5. Define the relationships

An Example

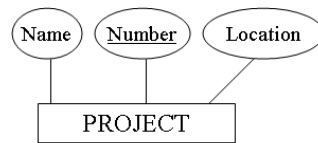
- The Company database keeps track of a company's
 - Employees, Departments, Projects
- The following are the requirements and specifications
- The company is organized into departments.
- Each department has a:
 - unique name, unique number
 - particular employee who manages the department
- We keep track of the start date when that employee began managing the department
- A department may have several locations
- A department controls a number of projects, each of which has a
 - unique name, unique number, and single location
- We store each employee's
 - name, social security number, address, salary, sex, 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 number of hours per week that an employee works on each project
- We keep track of the direct supervisor of each 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

ER diagram for the company database

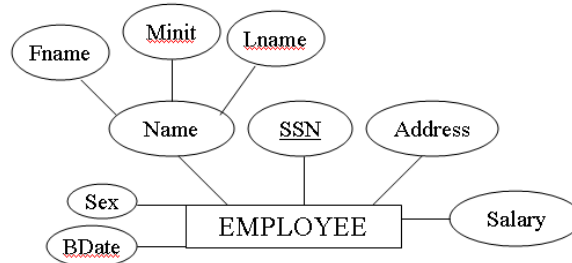
Each department has a unique name, a unique number, particular employee who manages the department. A department may have several locations.



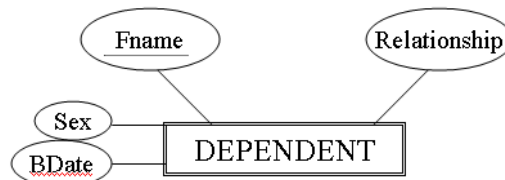
A department controls a number of projects, each of which has a unique name, unique number, and single location



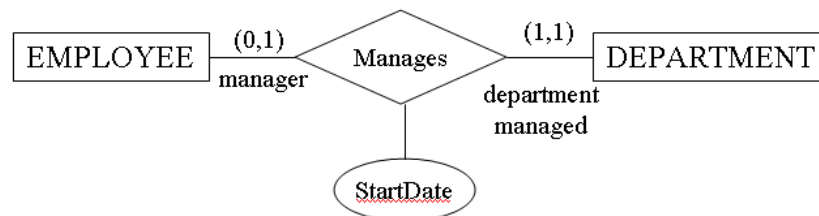
We store each employee's name, social security number, address, salary, sex, and birth date.



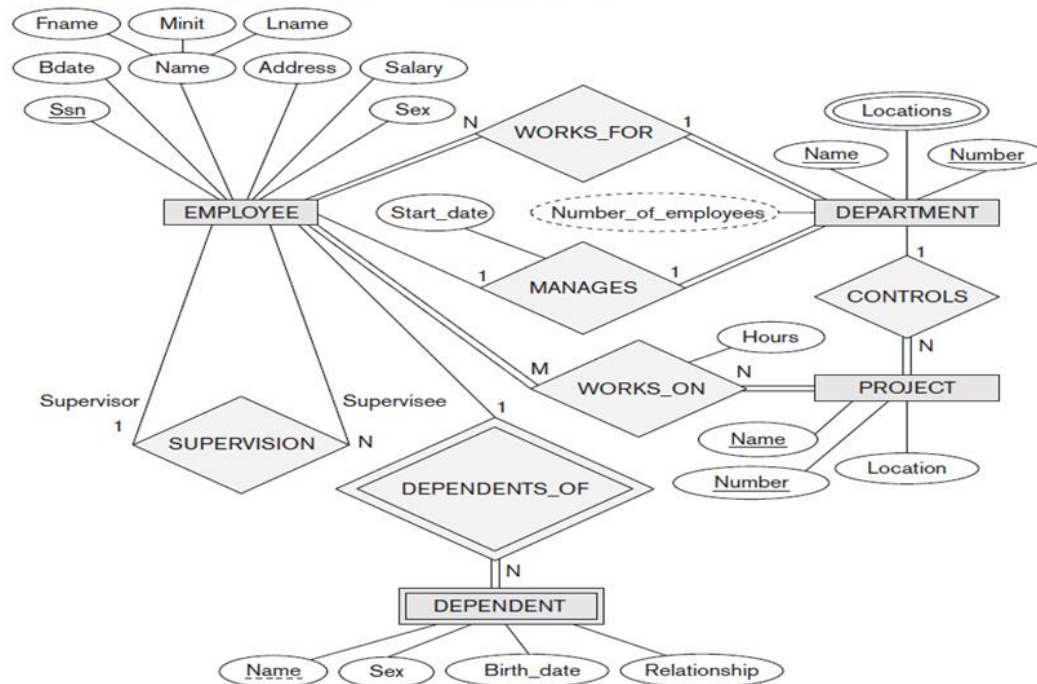
We want to keep track of the dependents of each employee for insurance purposes. We keep each dependent's first name, sex, birth date, and relationship to the employee



Each department has a particular employee who manages the department. We keep track of the start date when that employee began managing the department



The ER conceptual schema diagram for the COMPANY database.



Integrity Constraints

- Constraints enforce rules at the table level whenever a row is inserted, updated or deleted from that table.
- Constraints prevent the deletion of a table if there are dependencies from other tables.

Classification of Constraints:

The Constraint types are:

1. **Not Null :** Specifies that this column may not contain a null value.
2. **Unique :** Specifies a column or combination of columns whose values must be unique for all rows in the table.
3. **Primary Key:** Uniquely identifies each row of the table
4. **Foreign Key:** Establishes and enforces a foreign key relationship between the column and a column of the referenced table.
5. **Check:** Specifies a condition that must be true.

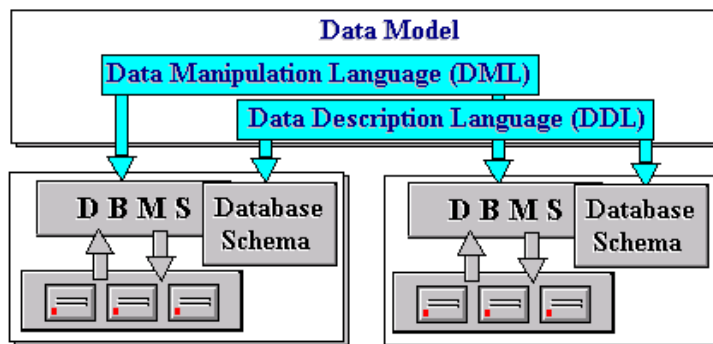
Chapter # 3

Relational Data Model

Lecture# 13

Data Models

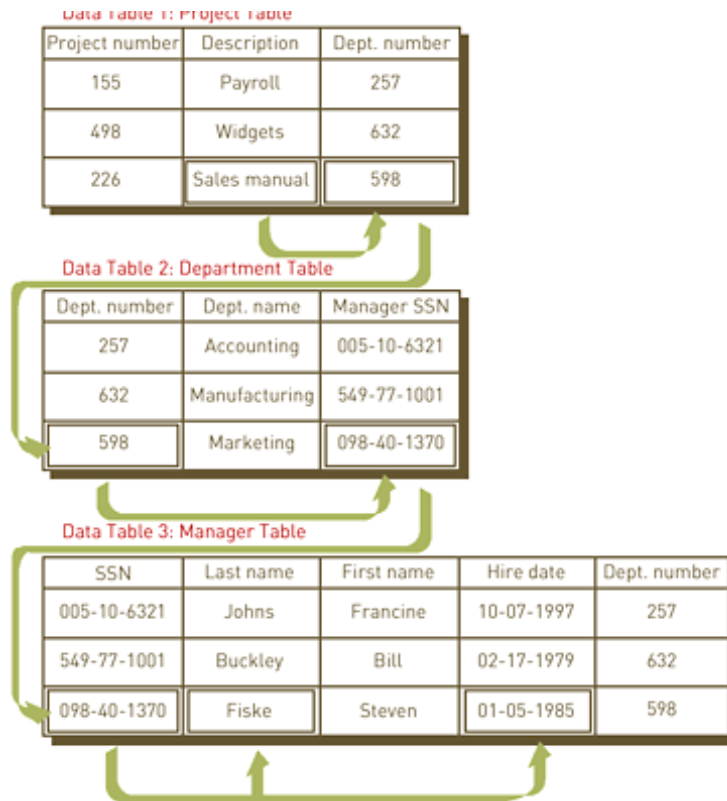
- Data Model is a collection of concepts that can be used to describe the structure of a data.
- A data model is a collection of **high-level data description constructs** that hide many low-level storage details.
- More formally, a Data Model is a combination of at least three components:
 - 1) A collection of data structure types
 - 2) A collection of operators or rules of inference,
 - 3) A collection of general integrity rules (constraints).
- In short, Data Model = DDL + DML
- Data Model may include a set of basic operations (update and retrieve)
- Most database management systems today are based on the **Relational data model**.



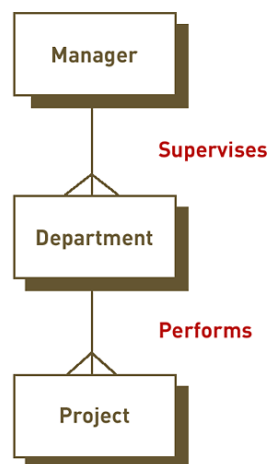
Relational Data Model

- **Relational model:** describes data in which all data elements are placed in two-dimensional tables, called relations, that are the logical equivalent of files
 - Each row of a table represents a data entity
 - Columns of the table represent attributes
 - **Domain:** allowable values for data attributes
- A relation may be expressed using the notation $R(\underline{A}, B, C, \dots)$ where:
 - R = the name of the relation.
 - $(\underline{A}, B, C, \dots)$ = the attributes within the relation.
 - \underline{A} = the attribute(s) which form the primary key.

- The model is based on the mathematical theory of relations, a part of set theory.
- Example: A relational data model with three relations (tables) is shown below.



- A Simplified ER Diagram Showing the Relationship between the Manager, Department, and Project Tables is:



Relational Model Concepts

- The relational Model of Data is based on the concept of a Relation.
- A Relation is a mathematical concept based on the ideas of sets.
- The strength of the relational approach to data management comes from the formal foundation provided by the theory of relations.

Relation: A table of values

- A relation may be thought of as a **set of rows**.
- A relation may alternately be thought of as a **set of columns**.
- Each row represents a fact that corresponds to a real-world **entity** or **relationship**.
- Each row has a value of an item or set of items that uniquely identifies that row in the table.
- Sometimes row-ids or sequential numbers are assigned to identify the rows in the table.
- Each column typically is called by its column name or column header or attribute name.
- Each value is derived from an appropriate domain
- A **Relation** may be defined in multiple ways.
- The **Schema** of a Relation: $R(\underline{A1}, A2, \dots, An)$
 - Relation schema R is defined over **attributes** $A1, A2, \dots, An$. The underlined attribute is Primary key.
- For Example -
CUSTOMER (C#, Cname, Ccity, Cphone)

Customer				
	C#	Cname	Ccity	Cphone
	1	Codd	London	2263035
Row	2	Martin	Paris	5555910
	3	Deen	London	2234391
	Column			

- A relation has a unique name and represents a particular entity. Figure illustrates a relation called 'Customer', intended to represent the set of persons who are customers of some enterprise.

DEFINITION SUMMARY

<u>Informal Terms</u>	<u>Formal Terms</u>
Table	Relation

Column	Attribute
Row	Tuple
Values in a column	Domain
Table Definition	Schema of a Relation

Lecture# 14

Characteristics of Relations:

- A relation with N columns and M rows (tuples) is said to be of degree N and cardinality M.

Customer			
C#	Cname	Ccity	Cphone
1	Codd	London	2263035
2	Martin	Paris	5555910
3	Deen	London	2234391

Degree 4

Cardinality 3

- 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 $R(A_1, A_2, \dots, A_n)$ and the values in $t = \langle v_1, v_2, \dots, v_n \rangle$ to be *ordered*.
- Values in a tuple:** All values are considered *atomic* (indivisible). A special **null** value is used to represent values that are unknown or inapplicable to certain tuples.

Relational Integrity Constraints

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

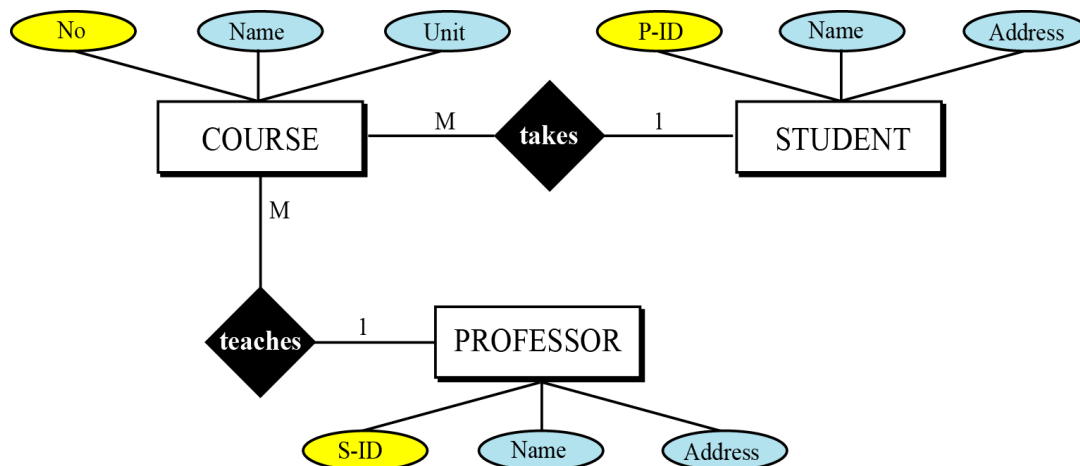
The Scope of Relational Data Model

- The relational data model permits the designer to create a consistent logical model of information, to be refined through database normalization.

- The basic principle of the relational model is the Information Principle: all information is represented by data values in relations.
- Relationships are not explicit, but rather implied by values in specific fields, foreign keys in one table (e.g., departments) that match those of records in a second table (e.g., employees). Many-to-many relationships typically require an intermediate table that contains just the relationships.

From E-R diagrams to Relational Databases:

- **Example:** Figure shows a very simple E-R diagram with three entity sets, their attributes and the relationship between the entity sets



- There are two relationship sets in Figure: teaches and takes, each connected to two entity sets. The relations for these relationship sets are added to the previous relations for the entity set and shown in Figure.

COURSE		
No	Name	Unit
⋮	⋮	⋮

STUDENT		
S-ID	Name	Address
⋮	⋮	⋮

PROFESSOR		
P-ID	Name	Address
⋮	⋮	⋮

TEACHES	
P-ID	No.
⋮	⋮

TAKES	
S-ID	No.
⋮	⋮

Chapter # 4
Lecture# 15

Relational Algebra

Introduction to Relational Algebra

- The basic set of operations for the relational model is known as the relational algebra. These operations enable a user to specify basic retrieval requests.
- operands are tables
- operators like “choose the rows that satisfy ...”
- The result of retrieval is a new relation, which may have been formed from one or more relations. The **algebra operations** thus produce new relations, which can be further manipulated using operations of the same algebra.
- A sequence of relational algebra operations forms a **relational algebra expression**,

Operators - Retrieval

- There are two groups of operations:
 - Special database operations:
 - select: σ
 - project: Π
 - Rename: ρ
 - JOIN
 - Mathematical set theory based operations:
 - union: \cup
 - Set intersection: \cap
 - set difference: $-$
 - Cartesian product: \times

SELECT Operation

- SELECT operation is used to select a *subset* of the tuples from a relation that satisfy a selection condition.
- It is a filter that keeps only those tuples that satisfy a qualifying condition – those satisfying the condition are selected while others are discarded.

- In general, the select operation is denoted by σ <selection condition>(R) where the symbol σ (sigma) is used to denote the select operator, and the selection condition is a Boolean expression specified on the attributes of relation R
- Example#1: To select the EMPLOYEE tuples whose department number is 10

$$\sigma_{\text{Dept_NO} = 10}(\text{EMP})$$

- Example#2: To select the EMPLOYEE tuples those whose salary is greater than \$3,000

$$\sigma_{\text{Sal} > 3,000}(\text{EMP})$$

Select Operation – Example

■ Relation r

A	B	C	D
α	α	1	7
α	β	5	7
β	β	12	3
β	β	23	10

■ $\sigma_{A=B \wedge D > 5}(r)$

A	B	C	D
α	α	1	7
β	β	23	10

PROJECT Operation

- This operation selects certain *columns* from the table and discards the other columns.
- The PROJECT creates a vertical partitioning – one with the needed columns (attributes) containing results of the operation and other containing the discarded Columns.
- The general form of the project operation is π <attribute list>(R) where π (pi) is the symbol used to represent the project operation and <attribute list> is the desired list of attributes from the attributes of relation R.
- Example: To list each employee's name and salary, the following is used:

$$\pi_{\text{ename, sal}}(\text{EMP})$$

Project Operation – Example

Relation r:

A	B	C
α	10	1
α	20	1
β	30	1
β	40	2

$\Pi_{A,C}(r)$

A	C
α	1
α	1
β	1
β	2

=

A	C
α	1
β	1
β	2

Rename Operation

- We may want to apply several relational algebra operations one after the other. Either we can write the operations as a single relational algebra expression by nesting the operations, or we can apply one operation at a time and create intermediate result relations. In the latter case, we must give names to the relations that hold the intermediate results.
- Example: To retrieve the name & salary of all employees who work in department number 10, we must apply a select and a project operation. We can write a single relational algebra expression as follows:

$$\pi_{\text{ename, sal}}(\sigma_{\text{dept_No}=10}(\text{EMP}))$$

- OR We can explicitly show the sequence of operations, giving a name to each intermediate relation:

$$\text{DEP10_EMPS} \leftarrow \sigma_{\text{Dept_No}=10}(\text{EMP})$$

$$\text{RESULT} \leftarrow \pi_{\text{ename, sal}}(\text{DEP10_EMPS})$$

- The rename operator is ρ , The general Rename operation can be expressed by any of the following forms:
 $\rho_S(B_1, B_2, \dots, B_n)(R)$ is a renamed relation S based on R with column names B_1, B_2, \dots, B_n .

Lecture# 16**Union Operation**

- The result of this operation, denoted by $R \cup S$, is a relation that includes all tuples that are either in R or in S or in both R and S. Duplicate tuples are eliminated.
- Example: To retrieve the social security numbers of all employees who either work in department 10 or directly supervise an employee who works in department 10, we can use the union operation as follows:

$$\text{DEP10_EMPS} \leftarrow \sigma_{\text{Dept_No}=10}(\text{EMP})$$

$$\text{RESULT1} \leftarrow \pi_{\text{SSN}}(\text{DEP10_EMPS})$$

$$\text{RESULT2}(\text{SSN}) \leftarrow \pi_{\text{SUPERSSN}}(\text{DEP10_EMPS})$$

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

- The union operation produces the tuples that are in either RESULT1 or RESULT2 or both. The two operands must be “type compatible”.
- **Type Compatibility**
 - The operand relations $R1(A1, A2, \dots, An)$ and $R2(B1, B2, \dots, Bn)$ must have the same number of attributes, and the domains of corresponding attributes must be compatible.

Union Operation – Example

■ Relations r, s :

A	B
α	1
α	2
β	1

r

A	B
α	2
β	3

s

■ $r \cup s$:

A	B
α	1
α	2
β	1
β	3

Intersection Operation

- The result of this operation, denoted by $R \cap S$, is a relation that includes all tuples that are in both R and S. The two operands must be "type compatible"

Intersection Operation – Example

■ Relation r, s:

A	B
α	1
α	2
β	1

r

A	B
α	2
β	3

s

■ $r \cap s$

A	B
α	2

Difference (or MINUS) Operation

- The result of this operation, also denoted by $R - S$, is a relation that includes all tuples that are in R but not in S. The two operands must be "type compatible".

Set Difference Operation – Example

A	1
B	2
D	3
F	4
E	5

B	2
F	4
E	5

A	1
C	2
D	3
E	4

C	2
E	4

Note:

- Notice that both union and intersection are *commutative operations*; that is

$$R \cup S = S \cup R, \text{ and } R \cap S = S \cap R$$

- Both union and intersection can be treated as n-ary operations applicable to any number of relations as both are *associative operations*; that is

$$R \cup (S \cup T) = (R \cup S) \cup T, \text{ and } (R \cap S) \cap T = R \cap (S \cap T)$$

- The minus operation is *not commutative*; that is, in general

$$R - S \neq S - R$$

Cartesian / Cross Product Operation

- This operation is used to combine tuples from two relations in a combinatorial fashion.
- In general, the result of $R(A_1, A_2, \dots, A_n) \times S(B_1, B_2, \dots, B_m)$ is a relation Q with degree $n + m$ attributes $Q(A_1, A_2, \dots, A_n, B_1, B_2, \dots, B_m)$, in that order. The resulting relation Q has one tuple for each combination of tuples—one from R and one from S .
- Hence, if R has nR tuples (denoted as $|R| = nR$), and S has nS tuples, then
 $|R \times S|$ will have $nR * nS$ tuples.
- The two operands do NOT have to be "type compatible"

Cartesian-Product Operation – Example

■ Relations r, s :

A	B
α	1
β	2

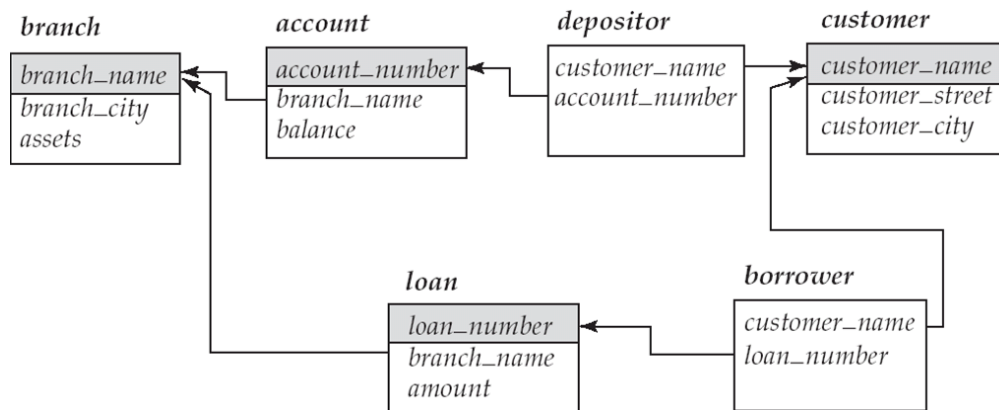
r

C	D	E
α	10	a
β	10	a
β	20	b
γ	10	b

s

■ $r \times s$:

A	B	C	D	E
α	1	α	10	a
α	1	β	10	a
α	1	β	20	b
α	1	γ	10	b
β	2	α	10	a
β	2	β	10	a
β	2	β	20	b
β	2	γ	10	b

Banking Example**branch** (branch_name, branch_city, assets)**customer** (customer_name, customer_street, customer_city)**account** (account_number, branch_name, balance)**loan** (loan_number, branch_name, amount)**depositor** (customer_name, account_number)**borrower** (customer_name, loan_number)

- Find all loans of over \$1200
 $\sigma_{amount > 1200} (loan)$
- Find the loan number for each loan of an amount greater than \$1200
 $\Pi_{loan_number} (\sigma_{amount > 1200} (loan))$
- Find the names of all customers who have a loan, an account, or both, from the bank
 $\Pi_{customer_name} (borrower) \cup \Pi_{customer_name} (depositor)$
- Find the names of all customers who have a loan at the Perryridge branch.
 $\Pi_{customer_name} (\sigma_{branch_name = "Perryridge"} (\sigma_{borrower.loan_number = loan.loan_number} (borrower \times loan)))$
- Find the names of all customers who have a loan at the Perryridge branch but do not have an account at any branch of the bank
 $\Pi_{customer_name} (\sigma_{branch_name = "Perryridge"} (\sigma_{borrower.loan_number = loan.loan_number} (borrower \times loan))) - \Pi_{customer_name} (depositor)$
- Find the names of all customers who have a loan at the Perryridge branch.
 $\Pi_{customer_name} (\sigma_{branch_name = "Perryridge"} (\sigma_{borrower.loan_number = loan.loan_number} (borrower \times loan)))$

Lecture# 17**JOIN Operation**

- This operation is very important for any relational database with more than a single relation, because it allows us to process relationships among relations.
- The general form of a join operation on two relations $R(A_1, A_2, \dots, A_n)$ and $S(B_1, B_2, \dots, B_m)$ is:
 - $R \bowtie_{\langle \text{join condition} \rangle} S$
 - Where R and S can be any relations that result from general *relational algebra expressions*.

Natural Join:

- The most common use of join involves join conditions with equality comparisons only.
- Such a join, where the only comparison operator used is $=$, is called an EQUIJOIN.
- Natural join is written as $R \bowtie S$ where R and S are relations. The result of the natural join is the set of all combinations of tuples in R and S that are equal on their common attribute names. For an example consider the tables Employee and Dept and their natural join:

Employee			Dept		Employee \bowtie Dept			
<u>Name</u>	<u>EmpId</u>	<u>DeptName</u>	<u>DeptName</u>	<u>Manager</u>	<u>Name</u>	<u>EmpId</u>	<u>DeptName</u>	<u>Manager</u>
Harry	3415	Finance	Finance	George	Harry	3415	Finance	George
Sally	2241	Sales	Sales	Harriet	Sally	2241	Sales	Harriet
George	3401	Finance	Production	Charles	George	3401	Finance	George
Harriet	2202	Sales			Harriet	2202	Sales	Harriet

In particular, natural join allows the combination of relations that are associated by a foreign key. For example, in the above example a foreign key probably holds from Employee.

Theta (θ) Join:

- A join based on a comparison of scalar values ($=$, $>$, $>=$, $<$, $<=$, $<>$).

$$R \bowtie S$$

- The θ -join is written as $a \theta b$ where a and b are attribute names, θ is a binary relation in the set $\{<, \leq, =, >, \geq\}$.
- The result of this operation consists of all combinations of tuples in R and S that satisfy the relation θ .
- In case the operator θ is the equality operator ($=$) then this join is also called an **equijoin**.

Semi join:

- The semijoin is joining similar to the natural join and written as $R \ltimes S$ where R and S are relations.
- The result of the semi join is only the set of all tuples in R for which there is a tuple in S that is equal on their common attribute names.
- For an example consider the tables Employee and Dept and their semi join:

Employee			Dept		Employee \ltimes Dept		
<u>Name</u>	<u>EmpId</u>	<u>DeptName</u>	<u>DeptName Manager</u>		<u>Name</u>	<u>EmpId</u>	<u>DeptName</u>
Harry	3415	Finance	Sales	Harriet	Sally	2241	Sales
Sally	2241	Sales	Production	Charles	Harriet	2202	Sales
George	3401	Finance					
Harriet	2202	Sales					

Antijoin:

- The antijoin, written as $R \Join S$ where R and S are relations, is similar to the natural join, but the result of an antijoin is only those tuples in R for which there is NOT a tuple in S that is equal on their common attribute names.
- For an example consider the tables Employee and Dept and their antijoin:

Employee

Name EmpId DeptName

Harry 3415 Finance

Sally 2241 Sales

George 3401 Finance

Harriet 2202 Sales

Dept

DeptName Manager

Sales Harriet

Production Charles

Employee ▷ Dept

Name EmpId DeptName

Harry 3415 Finance

George 3401 Finance

Outer Join:

- A join that includes all rows from the joined tables regardless of whether there is a matching row between the joined tables.
- There are three forms of the outer join, depending on which data is to be kept.
 - LEFT OUTER JOIN - keep data from the left-hand table
 - RIGHT OUTER JOIN - keep data from the right-hand table
 - FULL OUTER JOIN - keep data from both tables

OUTER JOIN example

R ColA ColB

A	1
B	2
D	3
F	4
E	5

R LEFT OUTER JOIN R.ColA = S.SColA S

A	1	A	1
D	3	D	3
E	5	E	4
B	2	-	-
F	4	-	-

S SColA SColB

A	1
C	2
D	3
E	4

R RIGHT OUTER JOIN R.ColA = S.SColA S

A	1	A	1
D	3	D	3
E	5	E	4
-	-	C	2

R FULL OUTER JOIN R.ColA = S.SColA S

A	1	A	1
D	3	D	3
E	5	E	4
B	2	-	-
F	4	-	-
-	-	C	2

Recap of Relational Algebra Operations

Operation	Purpose	Notation
SELECT	Selects all tuples that satisfy the selection condition from a relation R .	$\sigma_{\langle \text{SELECTION CONDITION} \rangle}(R)$
PROJECT	Produces a new relation with only some of the attributes of R , and removes duplicate tuples.	$\pi_{\langle \text{ATTRIBUTE LIST} \rangle}(R)$
THETA JOIN	Produces all combinations of tuples from R_1 and R_2 that satisfy the join condition.	$R_1 \bowtie_{\langle \text{JOIN CONDITION} \rangle} R_2$
EQUIJOIN	Produces all the combinations of tuples from R_1 and R_2 that satisfy a join condition with only equality comparisons.	$R_1 \bowtie_{\langle \text{JOIN CONDITION} \rangle} R_2$, OR $R_1 \bowtie_{(\langle \text{JOIN ATTRIBUTES 1} \rangle, \langle \text{JOIN ATTRIBUTES 2} \rangle)} R_2$
NATURAL JOIN	Same as EQUIJOIN except that the join attributes of R_2 are not included in the resulting relation; if the join attributes have the same names, they do not have to be specified at all.	$R_1 \star_{\langle \text{JOIN CONDITION} \rangle} R_2$, OR $R_1 \star_{(\langle \text{JOIN ATTRIBUTES 1} \rangle, \langle \text{JOIN ATTRIBUTES 2} \rangle)} R_2$ OR $R_1 \star R_2$
UNION	Produces a relation that includes all the tuples in R_1 or R_2 or both R_1 and R_2 ; R_1 and R_2 must be union compatible.	$R_1 \cup R_2$
INTERSECTION	Produces a relation that includes all the tuples in both R_1 and R_2 ; R_1 and R_2 must be union compatible.	$R_1 \cap R_2$
DIFFERENCE	Produces a relation that includes all the tuples in R_1 that are not in R_2 ; R_1 and R_2 must be union compatible.	$R_1 - R_2$
CARTESIAN PRODUCT	Produces a relation that has the attributes of R_1 and R_2 and includes as tuples all possible combinations of tuples from R_1 and R_2 .	$R_1 \times R_2$

Examples of Queries in Relational Algebra

- Question**

- Given following database schema, write following queries in relational algebra.

patients (*pnum*, *pname*, *age*)
doctors (*dnum*, *dname*, *rank*, *phone*)
visits (*pnum*, *dnum*, *date*, *diagnosis*)

- Q1: Who are the patients 10 years old or younger?
 $\sigma_{age \leq 10}(patients)$

- Q2: Who are the surgeons?
 $\sigma_{rank=surgeon}(doctors)$

- Q3: What are the phone numbers of doctors?
 $\pi_{dname, phonenum}(doctors)$

- Q4: What are the phone numbers of surgeons?
 $\pi_{dname, phonenum}(\sigma_{rank=surgeon}(doctors))$

order matters

Q5: Retrieve the name and address of all employees who work for the ‘Research’ department.

RESEARCH_DEPT $\leftarrow \sigma_{DNAME='Research'}(DEPT)$

RESULT $\leftarrow \pi_{FNAME, LNAME, ADDRESS}(RESEARCH_DEPT)$

Q6: Real databases use SQL, so why study relational algebra?

- Relational algebra is at the core of SQL
- The first thing a DBMS does with an SQL query is convert it into RA (or something very similar)
- query execution planning and query optimization are based on RA

- Question**

patients (*pnum*, *pname*, *age*)
doctors (*dnum*, *dname*, *rank*)
visits (*pnum*, *dnum*, *date*, *diagnosis*)

- Q7: Find all the patients who saw doctor 801 but not 802 (i.e. $dnum=801, dnum \neq 802$)

- The way of thinking:**

- find the set A of patients who saw doctor 801
- find the set B of patients who saw doctor 802;
- take $A - B$.

- **Answer:** $\pi_{pnum} \sigma_{dnum=801}(visits) - \pi_{pnum} \sigma_{dnum=802}(visits)$

Chapter # 5**Functional dependencies & Normalization****Lecture# 18****Functional Dependencies:**

- A functional dependency can be described as follows:
 - An attribute is functionally dependent if its value is determined by another attribute which is a key.
 - That is, if we know the value of one (or several) data items, then we can find the value of another (or several).
- If A and B are attributes or sets of attributes of relation R, we say that B is functional dependent on A if each value of A in R has associated with it exactly one value of B in R.
- We write this as $A \rightarrow B$, read as “A functionally determines B” or “A determines B”.
- It simply means that if we know the value of A and we examine the table of relation R, we will find only one value of B.
- However, for a given B value, there may be several different A values. When a functional dependency exists, the attribute or set of attributes on the left side of the arrow is called a determinant.
- To illustrate functional dependency, let us consider the following relation:
STUDENT (Seat#, Name, Major, Credits, Status, Reg#)
- Here, we will assume that every student has a unique Seat# and reg# and that each student has at most one major. We will assume that names are not unique, so that two different students may have the same name. Credits means the number of credits completed, and Status refers to the year the student is in – fresh, junior or senior.
- Figures shows as instance of the table:

Seat#	Name	Major	Credits	Status	Reg#
S1001	Smith, Tom	History	90	Sen	10025487
S1003	Jones, Mary	Math	95	Sen	10025471
S1006	Lee, Pamela	Computer	15	Fresh	12457812
S1010	Burns, Edward	Art	63	Jun	15478154
S1060	Jones, Mary	Computer	25	Fresh	12478963

- Examining the table, we see that if we are given a specific value of Seat#, there is only one value of Name associated with that particular Seat#.

- For example, for Seat# S1006, the associated Name is Lee, Pamela. So Name is functionally depends on Seat#, we can write: **Seat# → Name**
- However, for a given value of Name, there may be more than one Seat#, we note that for the Name Jones, Mary there are two Seat# values, S1003 and S1006.
- Therefore we cannot turn the functional dependency around and write Name → Seat#.
- For each of the other attributes, there is only one value associated with a particular value of Seat#, so all the attributes are functionally dependent on Seat#. We write

Seat# → Name, Major, Credits, Status, Reg#

- To indicate that each of the attributes on the right is functionally dependent on Seat#. Similarly, we have

Reg# → Seat#, Name, Major, Credits, Status,

- Also note that Status is functionally dependent on Credits. For example, if the value of Credits is known to be 15, the Status is automatically fresh. We write

Credits → Status

Example: Find All FDs from the following table??

Student	Dept	Course	Room
Ali	CS	C++	020
Badar	CS	C++	020
Ali	ES	HW	040
Kamran	CS	DB	045
Danish	CS	Java	050
Elisha	CS	DB	045
Frahan	ES	Circuits	020

Answer

Course → Dept, Room

Dept, Room → Course

Student, Dept → Course, Room

Student, Course → Dept, Room

Student, Room → Dept, Course

Use of Functional Dependencies

➤ We use functional dependencies to:

- test relations to see if they are legal under a given set of functional dependencies.
 - If a relation r is legal under a set F of functional dependencies, we say that r satisfies F .
- specify constraints on the set of legal relations
 - We say that F holds on R if all legal relations on R satisfy the set of functional dependencies F .

Data Anomalies

- When a database is poorly designed, then data is repeated
- Dependencies between attributes cause redundancy
- Eg. All addresses in the same town have the same zip code

<i>SSN</i>	<i>Name</i>	<i>Town</i>	<i>Zip</i>
1234	Joe	Stony Brook	11790
4321	Mary	Stony Brook	11790
5454	Tom	Stony Brook	11790
.....			

Redundancy

ER Model

<i>SSN</i>	<i>Name</i>	<i>Address</i>	<i>Hobby</i>
1111	Joe	123 Main	{biking, hiking}

Relational Model

<i>SSN</i>	<i>Name</i>	<i>Address</i>	<i>Hobby</i>
1111	Joe	123 Main	biking
1111	Joe	123 Main	hiking
.....			

Redundancy

■ Redundancy leads to anomalies:

- **Update anomaly:** A change in *Address* must be made in several places
- **Deletion anomaly:** Suppose a person gives up all hobbies. Do we:
 - Set *Hobby* attribute to null? No, since *Hobby* is part of key
 - Delete the entire row? No, since we lose other information in the row

- **Insertion anomaly:** *Hobby* value must be supplied for any inserted row since *Hobby* is part of key
- Solution: use two relations to store Person information
 - Personal (*SSN, Name, Address*)
 - Hobbies (*SSN, Hobby*)
- The decomposition is more general: people with hobbies can now be described
- No update anomalies:
 - Name and address stored once
 - A hobby can be separately supplied or deleted

Example: Persons with several phones

Name	SSN	PhoneNumber	City
Jawed	123-45-6789	021-355-1234	Karachi
Jawed	123-45-6789	021-355-5654	Karachi
Khan	987-65-4321	022-555-2121	Hyderabad

Anomalies:

- Redundancy = repeat data
- Update anomalies = jawed moves to “Sukkur”
- Deletion anomalies = Khan deletes his phone number: what is his city ?

Lecture# 19**Data Normalization / Normalization of Relations**

- Data normalization is a set of rules and techniques concerned with:
 - Identifying relationships among attributes.
 - Combining attributes to form relations.
 - Combining relations to form a database.
- Normalization is the process of decomposing unsatisfactory "bad" relations by breaking up their attributes into smaller relations
- Normalization is needed to allow any relation in the database to be represented, to allow a language like SQL to use powerful retrieval operations composed of atomic operations, to remove anomalies in insertion, deletion, and updating, and reduce the need for restructuring the database as new data types are added.
- A normalized relational database provides several benefits:
 - Elimination of redundant data storage.
 - Close modeling of real world entities, processes, and their relationships.
 - Structuring of data so that the model is flexible.
- The normalization process defines a set of hierarchical normal forms (NFs). Several normal forms have been proposed, including 1NF, 2NF, 3NF, BCNF (Boyce-Codd Normal Form), 4NF, PJNF (Projection/Joint Normal Form), 5NF and so on.
- A lot of database designers often find it unnecessary to go beyond 3rd Normal Form. This does not mean that those higher forms are unimportant.
- However, all database designers should be aware of all the forms of normalization so that they may be in a better position to detect when a particular rule of normalization is broken and then decide if it is necessary to take appropriate action.

First Normal Form

- A table is in first normal form (1NF) if it does not contain a repeating group
 - To convert, you must expand the table's primary key to include the primary key of the repeating group
- When we transform entities or relationships into tabular relations, there may be some relations in which there are more values in the intersection of a row or column.
 - **An example of 1NF**

TEACHES

ID	No.
• • •	• • •
8256	CIS15 CIS18 CIS21
• • •	• • •

a. Not in 1NF

Three values
in one intersection

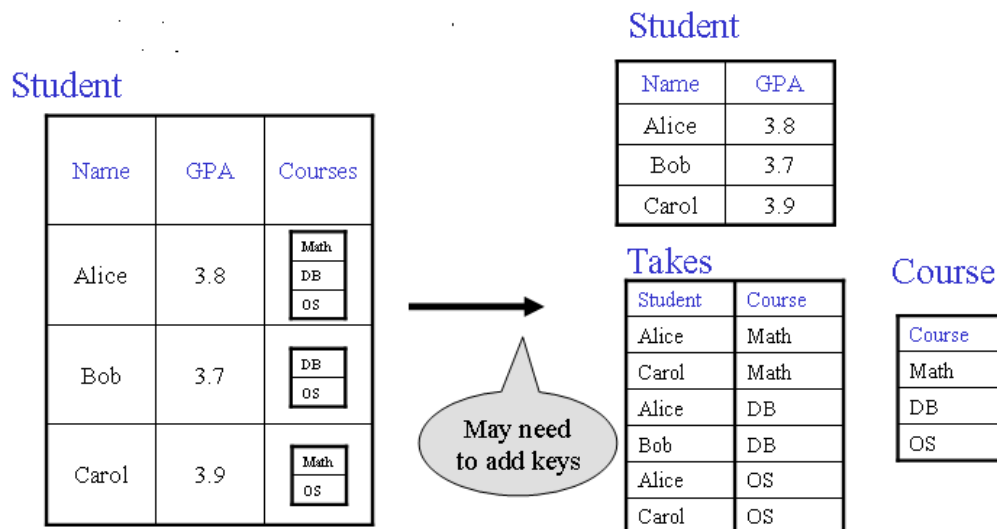
TEACHES

ID	No.
• • •	• • •
8256	CIS15
8256	CIS18
8256	CIS21
• • •	• • •

b. In 1NF

Only one value
in each intersection

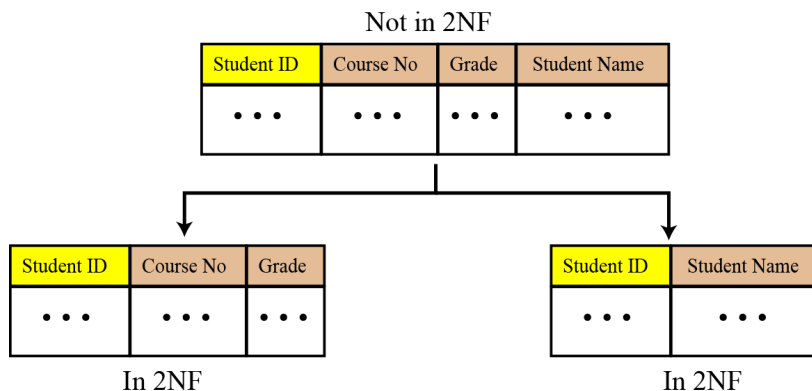
- A database schema is in First Normal Form if all tables are flat, For example:



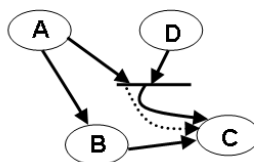
- First normal form deals with the "shape" of a record type.
- Under first normal form, all occurrences of a record type must contain the same number of fields.
- First normal form excludes variable repeating fields and groups

Second Normal Form

- A relation is said in second normal form (2NF), if and only if it is 1NF and that every column non key depends elementarily on the primary key
 - A standard process exists for converting a table from 1NF to 2NF
 - Create and name a separate table for each field in the existing primary key
 - Create a new table for each possible combination of the original primary key fields
 - Study the three tables and place each field with its appropriate primary key
 - Four kinds of problems are found with 1NF description that do not exist with 2NF
 - Consider the work necessary to change a particular product's description
 - 1NF tables can contain inconsistent data
 - Adding a new product is a problem
 - Deleting a product is a problem
- For example, if the ID of a student is given, it should be possible to find the student's name.



- Example: Let's consider the relation $R(A, B, C, D)$ with the following functional dependencies:
 $A \rightarrow B$; $A, D \rightarrow C$; $B \rightarrow C$
- The graph of the functional dependencies is



The relation is not 2NF because B that doesn't belong at the key depends on a part of the key that is A, D

Decomposition in 3 relations 2NF



$R1(A, B) : A \rightarrow B$
 $R2(A, D, C) : A, D \rightarrow C$
 $R3(B, C) : B \rightarrow C$

Third Normal Form

- A relation is said in third normal form (3NF), if and only if it is 2NF and that none of its attributes non keys depends on another attribute non key
 - 3NF design avoids redundancy and data integrity problems that still can exist in 2NF designs
 - To convert the table to 3NF, you must remove all fields from the 2NF table that depend on another nonkey field and place them in a new table that uses the nonkey field as a primary key

in 2NF, the nonkey field SALES-REP-NAME is functionally dependent on another nonkey field SALES-REP-NUM

CUSTOMER IN 2NF

RECORD#	CUSTOMER- NUM	CUSTOMER- NAME	ADDRESS	SALES-REP- NUM	SALES-REP- NAME
1	108	Benedict, Louise	San Diego, CA	41	Kaplan, James
2	233	Corelli, Helen	Nashua, NH	22	McBride, Jon
3	254	Gomez, J.P.	Butte, MT	38	Stein, Ellen
4	431	Lee, M.	Snow Camp, NC	74	Roman, Harold
5	779	Paulski, Diane	Lead, SD	38	Stein, Ellen
6	800	Zuider, Z.	Greer, SC	74	Roman, Harold

In order to express the same facts without violating 3NF, it is necessary to split the table into two:

Customer

RECORD#	CUSTOMER- NUM	CUSTOMER- NAME	ADDRESS	SALES-REP- NUM
1	108	Benedict, Louise	San Diego, CA	41
2	233	Corelli, Helen	Nashua, NH	22
3	254	Gomez, J.P.	Butte, MT	38
4	431	Lee, M.	Snow Camp, NC	74
5	779	Paulski, Diane	Lead, SD	38
6	800	Zuider, Z.	Greer, SC	74

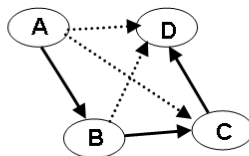
Sales-Rep

SALES-REP- NUM	SALES-REP- NAME
41	Kaplan, James
22	McBride, Jon
38	Stein, Ellen
74	Roman, Harold
38	Stein, Ellen
74	Roman, Harold

- Example: Let's consider the relation R(A, B, C, D) with the following functional dependencies:

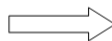
$$A \rightarrow B; B \rightarrow C; C \rightarrow D$$

- Relation in third normal form



The relation is not 3NF because C depends on an attribute non key (D also)

Decomposition in
3 relations 3NF



R1(A, B) : $A \rightarrow B$
 R2(B, C) : $B \rightarrow C$
 R3(C, D) : $C \rightarrow D$

Boyce-Codd Normal Form

- A relation is in BCNF if and only if the only elementary functional dependencies are those in which a key determines an attribute

- A simple condition for removing anomalies from relations:

A relation R is in BCNF if:

If $A_1, \dots, A_n \rightarrow B$ is a non-trivial dependency in R, then $\{A_1, \dots, A_n\}$ is a key for R

- Let's consider relation products(Raw, Country, Region) with the supposes functional dependences:

Region \rightarrow Country

(Raw, Country) \rightarrow Region

- This relation is well in third normal form because no attribute non key doesn't depends on a part of the key or on an attribute non key. However, one finds numerous redundancies

Raw	Country	Region
Chenas	France	Beaujolais
Julienas	France	Beaujolais
Morgon	France	Beaujolais
Brouilly	France	Beaujolais
Chablis	United-States	California

- Relation Products will be able to be decomposed in two relations :

Raw (Raw, Region)

Regions (Region, Country)

Chapter # 6**SQL****Lecture# 20****SQL Introduction**

- Abbreviation of structured query language, and pronounced either *see-kwell* or as separate letters.
- SQL is a standardized query language for requesting information from a database.
- Historically, SQL has been the favorite query language for database management systems running on minicomputers and mainframes.
- Increasingly, however, SQL is being supported by PC database systems because it supports distributed databases (databases that are spread out over several computer systems). This enables several users on a local-area network to access the same database simultaneously.
- It is a ***declarative*** rather than ***procedural*** language, which means that users declare what they want without having to write a step-by-step procedure.
- The SQL language was first implemented by the Oracle Corporation in 1979, with various versions of SQL being released since then.
- ANSI and ISO standard SQL:
 - SQL-86
 - SQL-89
 - SQL-92
 - SQL:1999
 - SQL:2003
- Commercial systems offer most, if not all, SQL-92 features, plus varying feature sets from later standards and special proprietary features.
- SQL is the most popular computer language used to create, modify and retrieve data from relational database management systems. The data retrieved by a query are called **Query Results**.
- A SQL command (specifically, a SQL SELECT command) that retrieves information from one or more tables, in any combination, expression, or order. Queries are read-only operations; they do not change any data, they only retrieve data. Queries are often considered to be Data Retrieval statements.

Simple queries in SQL

The most frequently used operation in transactional databases is the data retrieval operation. The syntax of query is:

```
SELECT    column(s)
FROM      table(s)
WHERE     condition(s)
Order by  column(s)
```

COMPARISION OPERATORS :

<u>OPERATOR</u>	<u>MEANING</u>
=	Equal to
>	Greater than
>=	Greater than or equal to
<	Less than
<=	less than or equal to
<>	Not equal to

OTHER COMPARISION OPERATORS :-

```
BETWEEN- - - - - AND- - - -
IN (List)
LIKE      _ , %
IS NULL
```

LOGICAL OPERATORS :

- AND
- OR
- NOT

ORDER BY CLAUSE :

Sorts rows with order by clause

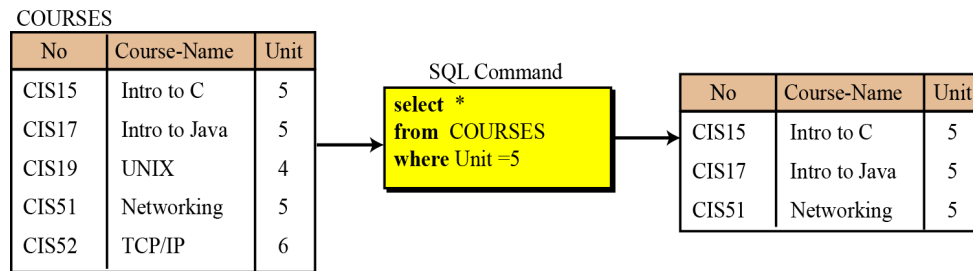
ASC	→	Ascending order (by default)
DESC	→	Descending order

Select

The select operation is a unary operation. The tuples (rows) in the resulting relation are a subset of the tuples in the original relation.

```
select  *
from    RELATION-NAME
where   criteria
```

An example of a select operation

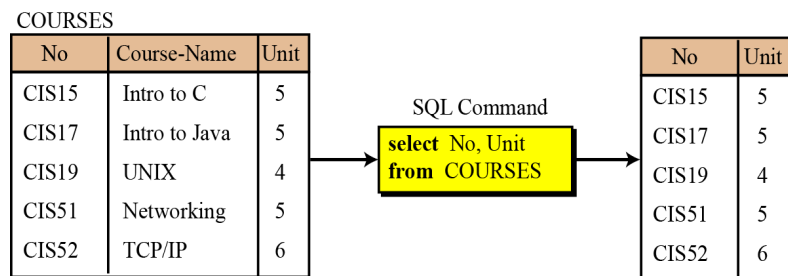


Project

The project operation is also a unary operation and creates another relation. The attributes (columns) in the resulting relation are a subset of the attributes in the original relation.

```
select attribute-list  
from RELATION-NAME
```

An example of a project operation



Lecture# 21**Simple SQL Queries****Selecting All rows and all columns from a table:**

You can display all columns of data in a table by following the SELECT keyword with an asterisk (*). For example:

```
SQL> select *
      2 from emp;
```

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7369	SMITH	CLERK	7902	17-DEC-80	800		20
7499	ALLEN	SALESMAN	7698	20-FEB-81	1600	300	30
7521	WARD	SALESMAN	7698	22-FEB-81	1250	500	30
7566	JONES	MANAGER	7839	02-APR-81	2975		20
7654	MARTIN	SALESMAN	7698	28-SEP-81	1250	1400	30
7698	BLAKE	MANAGER	7839	01-MAY-81	2850		30
7782	CLARK	MANAGER	7839	09-JUN-81	2450		10
7788	SCOTT	ANALYST	7566	19-APR-87	3000		20
7839	KING	PRESIDENT		17-NOV-81	5000		10
7844	TURNER	SALESMAN	7698	08-SEP-81	1500	0	30
7876	ADAMS	CLERK	7788	23-MAY-87	1100		20
7900	JAMES	CLERK	7698	03-DEC-81	950		30
7902	FORD	ANALYST	7566	03-DEC-81	3000		20
7934	MILLER	CLERK	7782	23-JAN-82	1300		10

14 rows selected.

Selecting Specific Columns:

In the SELECT clause, specify the columns that you want to see, in the order in which you want them to appear in the output. For Example:

```
SQL> select deptno, loc
      2 from dept;
```

DEPTNO	LOC
10	NEW YORK
20	DALLAS
30	CHICAGO
40	BOSTON

Arithmetic Expressions:

Create expressions on NUMBER and DATA data by using arithmetic operators.

Operator	Description
+	Add
-	Subtract
*	Multiply
/	Divide

You can use arithmetic operators in any clause of a SQL statement except the FROM clause.

Example:

```
SQL> select ename, sal, sal+300
```

2 from emp;

ENAME	SAL	SAL+300
SMITH	800	1100
ALLEN	1600	1900
WARD	1250	1550
JONES	2975	3275
MARTIN	1250	1550
BLAKE	2850	3150
CLARK	2450	2750
SCOTT	3000	3300
KING	5000	5300
TURNER	1500	1800
ADAMS	1100	1400
JAMES	950	1250
FORD	3000	3300
MILLER	1300	1600

14 rows selected.

Defining a NULL Value:

- A null is a value that is unavailable, unassigned.
- A null is not the same as zero or a blank space.

Example:

```
SQL> select ename, job, COMM
2 FROM EMP;
```

ENAME	JOB	COMM
SMITH	CLERK	
ALLEN	SALESMAN	300
WARD	SALESMAN	500
JONES	MANAGER	
MARTIN	SALESMAN	1400
BLAKE	MANAGER	
CLARK	MANAGER	
SCOTT	ANALYST	
KING	PRESIDENT	
TURNER	SALESMAN	0
ADAMS	CLERK	
JAMES	CLERK	
FORD	ANALYST	
MILLER	CLERK	

14 rows selected.

Null values in Arithmetic Expressions:

Arithmetic expressions containing a null value evaluate to null.

Example:

```
SQL> Select ename, 12*sal+comm
      2  from emp;
```

ENAME	12*SAL+COMM
SMITH	
ALLEN	19500
WARD	15500
JONES	
MARTIN	16400
BLAKE	
CLARK	
SCOTT	
KING	
TURNER	18000
ADAMS	
JAMES	
FORD	
MILLER	

14 rows selected.

Duplicate Rows:

The default display of queries is all rows, including duplicates rows.

Example:

```
SQL> Select deptno
      2  from emp;
```

DEPTNO
20
30
30
20
30
30
10
20
10
30
20
30
20
10

14 rows selected.

Eliminating Duplicate Rows:

To eliminate duplicate rows in the result, include DISTINCT key word in the SELECT clause, immediately after the SELECT keyword.

Example:

```
SQL> Select Distinct deptno
      2  from emp;
```

DEPTNO

10
20
30

```
SQL> Select distinct job, deptno
      2  from emp;
```

JOB	DEPTNO
-----	-----
ANALYST	20
CLERK	10
CLERK	20
CLERK	30
MANAGER	10
MANAGER	20
MANAGER	30
PRESIDENT	10
SALESMAN	30

9 rows selected.

Lecture# 22

Joins in SQL

The join operation is a binary operation that combines two relations on common attributes.

```
select attribute-list
from RELATION1, RELATION2
where criteria
```

COURSES

No	Course-Name	Unit
CIS15	Intro to C	5
CIS17	Intro to Java	5
CIS19	UNIX	4
CIS51	Networking	5
CIS52	TCP/IP	6

SQL Command

```
select No, Course-name, Unit, Professor
from COURSES, TAUGHT-BY
where COURSE.No = TAUGHT-BY.No;
```

TAUGHT-BY

No	Professor
CIS15	Lee
CIS17	Lu
CIS19	Walter
CIS51	Lu
CIS52	Lee

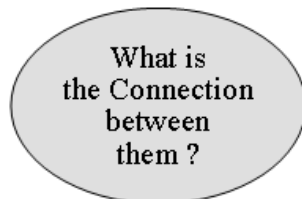
No	Course-Name	Unit	Professor
CIS15	Intro to C	5	Lee
CIS17	Intro to Java	5	Lu
CIS19	UNIX	4	Walter
CIS51	Networking	5	Lu
CIS52	TCP/IP	6	Lee

- Connect two or more tables

Product

<u>PName</u>	<u>Price</u>	<u>Category</u>	<u>Manufacturer</u>
Gizmo	\$19.99	Gadgets	GizmoWorks
Powergizmo	\$29.99	Gadgets	GizmoWorks
SingleTouch	\$149.99	Photography	Canon
MultiTouch	\$203.99	Household	Hitachi

Company



<u>CName</u>	<u>StockPrice</u>	<u>Country</u>
GizmoWorks	25	USA
Canon	65	Japan
Hitachi	15	Japan

Product (pname, price, category, manufacturer)

Company (cname, stockPrice, country)

Find all products under \$200 manufactured in Japan; return their names and prices.

```
SELECT PName, Price
FROM Product, Company
WHERE Manufacturer=CName AND Country='Japan'
AND Price <= 200
```

Join
between Product
and Company

Joins in SQL

Product

PName	Price	Category	Manufacturer
Gizmo	\$19.99	Gadgets	GizmoWorks
Powergizmo	\$29.99	Gadgets	GizmoWorks
SingleTouch	\$149.99	Photography	Canon
MultiTouch	\$203.99	Household	Hitachi

Company

CName	StockPrice	Country
GizmoWorks	25	USA
Canon	65	Japan
Hitachi	15	Japan

```
SELECT PName, Price
FROM Product, Company
WHERE Manufacturer=CName AND Country='Japan'
AND Price <= 200
```



PName	Price
SingleTouch	\$149.99

Find all countries that manufacture some product in the 'Gadgets' category.

```
SELECT Country
FROM Product, Company
WHERE Manufacturer=CName AND Category='Gadgets'
```

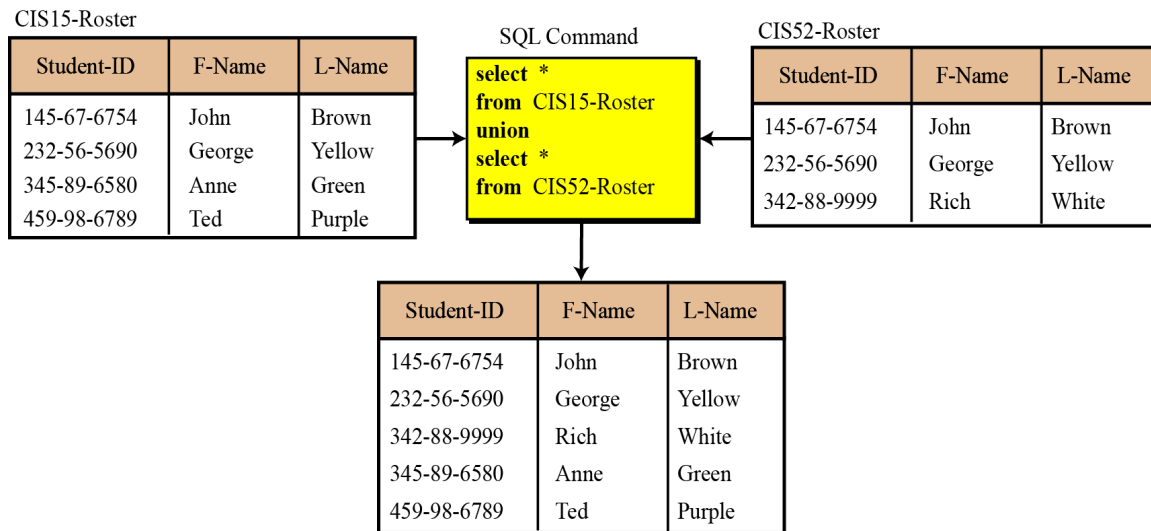
Country
??
??

Union

The union operation takes two relations with the same set of attributes.

```
select *  
from RELATION1  
union  
select *  
from RELATION2
```

An example of a union operation

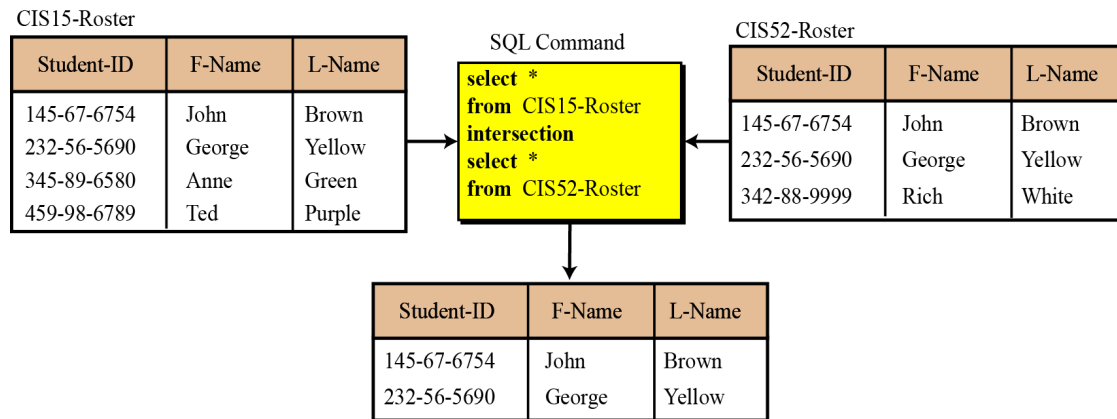


Intersection

The intersection operation takes two relations and creates a new relation, which is the intersection of the

```
select *  
from RELATION1  
intersection  
select *  
from RELATION2
```

An example of an intersection operation

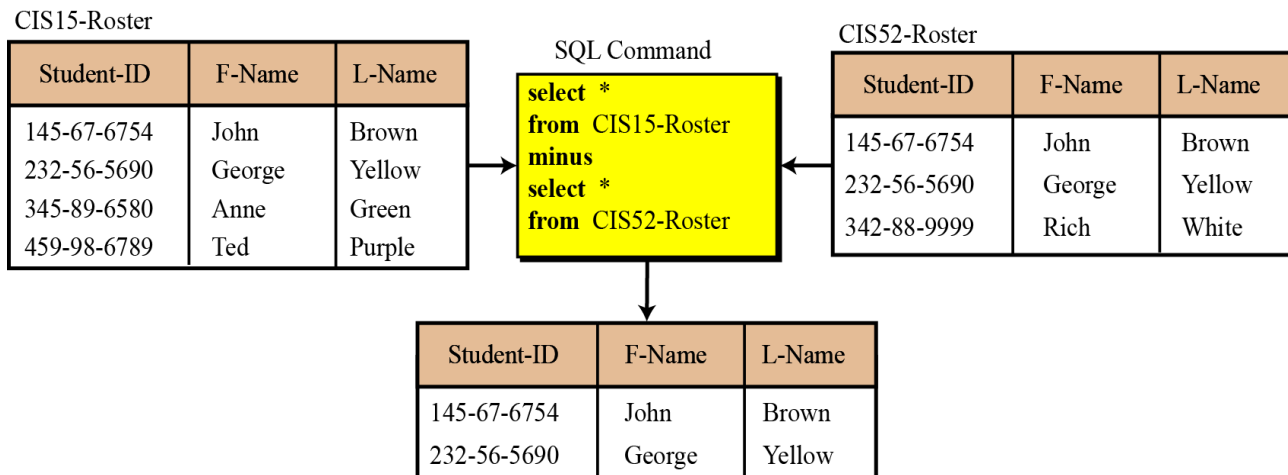


Difference

The difference operation is applied to two relations with the same attributes. The tuples in the resulting relation are those that are in the first relation but not the second.

```
select *
from RELATION1
minus
select *
from RELATION2
```

An example of a difference operation

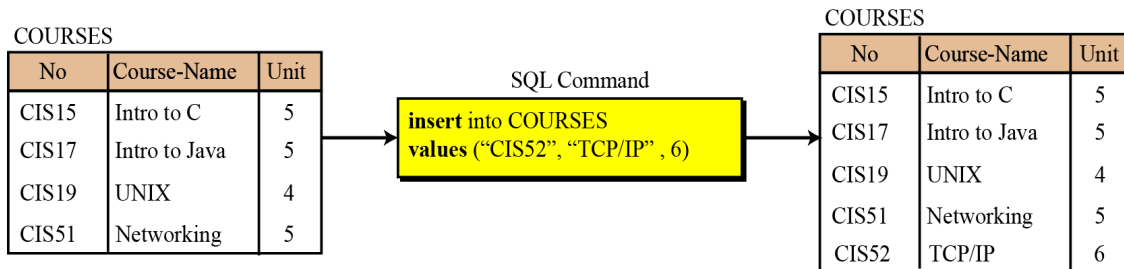


Insert

The **insert operation** is a unary operation—that is, it is applied to a single relation. The operation inserts a new tuple into the relation. The insert operation uses the following format:

```
insert into  RELATION-NAME
values  (... , ... , ...)
```

An example of an insert operation



Delete

The delete operation is also a unary operation. The operation deletes a tuple defined by a criterion from the relation. The delete operation uses the following format:

```
delete from RELATION-NAME
where criteria
```

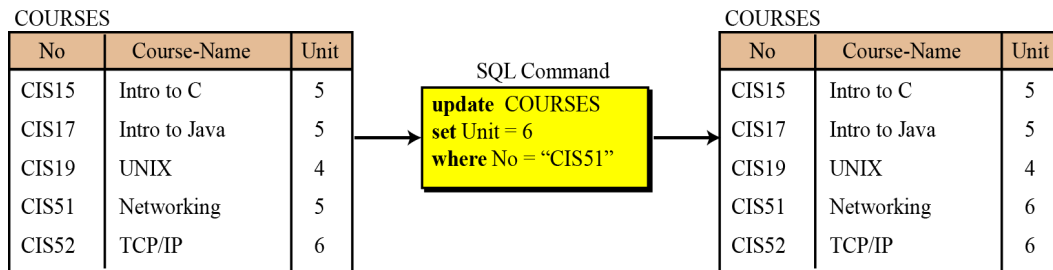
An example of a delete operation



Update

The update operation is also a unary operation that is applied to a single relation. The operation changes the value of some attributes of a tuple. The update operation uses the following format:

```
update  RELATION-NAME
set attribute1 = value1, attribute2 = value2, ...
where  criteria
```

An example of an update operation

Lecture# 23**Aggregating Data using Group Functions:**

Group Functions: Group functions operate on sets of rows to give one result per group. These sets may be the whole table or the table spilt into groups.

- **Avg** The `AVG` function computes the average of a column.
- **Count** The function `COUNT` returns the number of rows that satisfy the condition in the `WHERE` clause.
- **Max** The `MAX` function finds the largest value in a column.
- **Min** The `MIN` function finds the lowest value in a column

Using Group Functions: The clause **GROUP BY** is used for group function. As:

GROUP BY <col> [, <col> ...]

`GROUP BY` statement groups all the rows with the same column value.

Example:

```
SQL> Select Avg(sal), Max(sal), Min(sal), Sum(sal)
2  from emp
3  WHERE job like 'SALES%';
```

AVG(SAL)	MAX(SAL)	MIN(SAL)	SUM(SAL)
1400	1600	1250	5600

You can use the `MIN` and `MAX` functions for any datatype.

Example:

```
SQL> Select Min(hiredate), Max(hiredate)
2  from emp;
```

MIN(HIRED)	MAX(HIRED)
17-DEC-80	23-MAY-87

Example 2:

```
SQL> Select Min(ename), Max(ename)
2  from emp;
```

MIN(ENAME)	MAX(ENAME)
ADAMS	WARD

Using the Count Function: Count(*) returns the number of rows in a table including duplicate rows and rows containing Null values in any of the column.

Example:

```
SQL> Select Count(*)
      2  from emp
      3  Where deptno = 30;

COUNT(*)
-----
        6
```

Count(Expr) returns the number of non-null rows in the column identified by expr.

Example:

```
SQL> Select Count(Comm)
      2  from emp
      3  Where deptno = 30;

COUNT(COMM)
-----
         4
```

Example 2:

```
SQL> Select Count(deptno)
      2  from emp;

COUNT(DEPTNO)
-----
        14

SQL> Select Count(Distinct(deptno))
      2  from emp;

COUNT(DISTINCT(DEPTNO))
-----
         3
```

Creating Groups of Data:

Group By clause divides rows in a table into smaller groups.

Example:

```
SQL> Select Deptno, Avg(sal)
2  from emp
3  Group by deptno;
```

DEPTNO	AUG(SAL)
10	2916.66667
20	2175
30	1566.66667

The Group By column does not have to be in the Select list.

```
SQL> Select Avg(sal)
2  from emp
3  Group by deptno;
```

AUG(SAL)
2916.66667
2175
1566.66667

You can use the group function in the Order By clause.

```
SQL> Select Deptno, Avg(sal)
2  from emp
3  Group by deptno
4  Order by Avg(sal);
```

DEPTNO	AUG(SAL)
30	1566.66667
20	2175
10	2916.66667

Using the Group By clause on multiple columns.

Example:

```
SQL> Select Deptno, job, sum(sal)
2  from emp
3  group by deptno, job;
```

DEPTNO	JOB	SUM(SAL)
10	CLERK	1300
10	MANAGER	2450
10	PRESIDENT	5000
20	ANALYST	6000
20	CLERK	1900
20	MANAGER	2975
30	CLERK	950
30	MANAGER	2850
30	SALESMAN	5600

You use the **HAVING** clause to restrict groups.

HAVING <search_cond>

HAVING is valid only with **GROUP BY** and limits the selection of groups to those that satisfy the search condition.

```
SQL> Select Deptno, Avg(sal)
2   from emp
3   group by deptno
4   Having Avg(sal) > 2000;
```

DEPTNO	Avg(SAL)
10	2916.66667
20	2175

Sub-queries

A subquery is a query whose results are passed as the argument for another query. Subqueries enable you to bind several queries together. Simply, a subquery lets you tie the result set of one query to another. The general syntax is as follows:

```
SELECT *
FROM TABLE1
WHERE TABLE1.SOMECOLUMN =
(SELECT SOMEOTHERCOLUMN
FROM TABLE2
WHERE SOMEOTHERCOLUMN = SOMEVALUE)
```

The inner query or the sub-query returns a value that is used by the outer query or the main query. Using a subquery is equivalent to performing two sequential queries and using the result of the first query as the search value in the second query.

Example:

```
SQL> Select ename
2   from emp
3   where sal >
4         (select sal
5           from emp
6           where empno = 7566);

ENAME
-----
SCOTT
KING
FORD
```

Guidelines for using Sub-queries:

- Enclose subqueries in parentheses.
- Place subqueries on the right side of the comparison operator.
- Don't add an ORDER BY clause to a subquery.

Example 1:

```
SQL> Select ename, job
2  from emp
3  where job =
4          (select job
5           from emp
6           where empno = 7369);
```

ENAME	JOB
SMITH	CLERK
ADAMS	CLERK
JAMES	CLERK
MILLER	CLERK

Example 2:

```
SQL> Select ename, job
2  from emp
3  where job =
4          (select job
5           from emp
6           where empno = 7369)
7  AND
8          (select sal
9           from emp
10          where empno = 7876);
```

ENAME	JOB
MILLER	CLERK

Chapter # 7**Views****Lecture# 24****View**

- Relation that is not of the conceptual model but is made visible to a user as a “virtual relation” is called a **view**.
- A view is a “virtual” table that is derived from other tables
- A view is essentially some subset of the database.
- A view is defined using the **create view** statement which has the form
 - **create view v as** < query expression >
- where <query expression> is any legal SQL expression. The view name is represented by v.
- Once a view is defined, the view name can be used to refer to the virtual relation that the view generates.
- Allows each user to have his or her own view of the database.
- Allows for limited update operations (since the table may not physically be stored)
- Allows full query operations

Benefits of views:

- Reduce complexity;
- Provide a level of security;
- Provide a mechanism to customize the appearance of the database;
- Present a consistent, unchanging picture of the structure of the database, even if the underlying database is changed.

Example

- A view consisting of branches and their customers

```
create view all_customer as
(select branch_name, customer_name
from depositor, account
  where depositor.account_number =
        account.account_number )
union
(select branch_name, customer_name
from borrower, loan
  where borrower.loan_number = loan.loan_number )
```
- Find all customers of the Perryridge branch

```
select customer_name
from all_customer
where branch_name = 'Perryridge'
```

Uses of Views

- Hiding some information from some users
 - Consider a user who needs to know a customer's name, loan number and branch name, but has no need to see the loan amount.
 - Define a view

```
(create view cust_loan_data as
select customer_name, borrower.loan_number, branch_name
from borrower, loan
where borrower.loan_number = loan.loan_number )
```
 - Grant the user permission to read *cust_loan_data*, but not *borrower* or *loan*
- Predefined queries to make writing of other queries easier
 - Common example: Aggregate queries used for statistical analysis of data

Processing of Views

- When a view is created
 - the query expression is stored in the database along with the view name
 - the expression is substituted into any query using the view
- Views definitions containing views
 - One view may be used in the expression defining another view
 - A view relation *v1* is said to *depend directly* on a view relation *v2* if *v2* is used in the expression defining *v1*
 - A view relation *v1* is said to *depend on* view relation *v2* if either *v1* depends directly to *v2* or there is a path of dependencies from *v1* to *v2*
 - A view relation *v* is said to be *recursive* if it depends on itself.

View Update

- Update on a single view without aggregate operations: update may map to an update on the underlying base table
- Views involving joins: an update *may* map to an update on the underlying base relations
 - not always possible

Chapter # 8**Transaction****Lecture# 25****Introduction to Transaction**

- A **transaction** is a unit of program execution that accesses and possibly updates various data items
- Logical unit of work that must be either entirely completed or aborted
- Successful transaction changes database from one **consistent state** to another
 - One in which all data integrity constraints are satisfied
- When the transaction is *committed*, the database must be consistent
- Two main issues to deal with:
 - Failures, e.g. hardware failures and system crashes
 - Concurrency, for simultaneous execution of multiple transactions

Transaction Properties (ACID properties)

A transaction is a unit of program execution that accesses and possibly updates various data items. To preserve integrity of data, the database system must ensure:

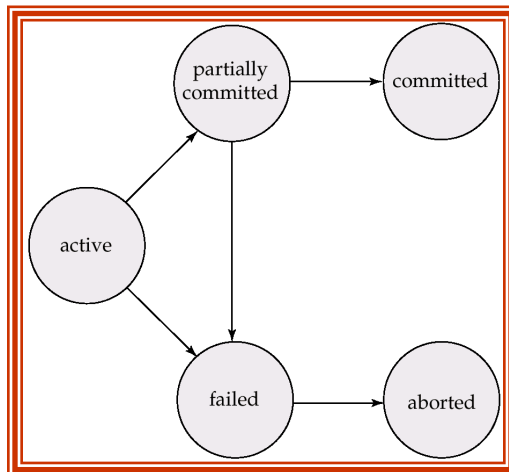
- **Atomicity:** Either all operations of the transaction are properly reflected in the database or none are
- **Consistency:** Execution of a transaction in isolation preserves the consistency of the database
- **Isolation:** Although multiple transactions may execute concurrently, each transaction must be unaware of other concurrently executing transactions; intermediate transaction results must be hidden from other concurrently executed transactions
- **Durability:** After a transaction completes successfully, they cannot be undone

Transaction Management with SQL

- Isolation (+ Consistency) => Concurrency Control
- Atomicity + Durability => Recovery
- Transaction support is provided by two SQL statements: COMMIT and ROLLBACK
- Transaction sequence must continue until:
 - COMMIT statement is reached
 - ROLLBACK statement is reached
 - End of program is reached
 - Program is abnormally terminated

Transaction State

- **Active**, the initial state; the transaction stays in this state while it is executing
- **Partially committed**, after the final statement has been executed.
- **Committed**, after *successful completion*
- **Failed**, after the discovery that normal execution can no longer proceed.
- **Aborted**, after the transaction has been rolled back and the database restored to its state prior to the start of the transaction.



Recovery:

- Recovery is the process of restoring of database to normal operation after failure.
- Recovery should protect the database and associated users from unnecessary problems and avoid or reduce the possibility to duplicate work manually.
- Automatic recovery protects your database if there is a system failure.

Serializability in SQL transactions:

- **Basic Assumption** – Each transaction preserves database consistency.
- It ensures that a schedule for executing concurrent transactions is equivalent to one that executes the transactions serially in some order.
- Thus serial execution of a set of transactions preserves database consistency.
- A (possibly concurrent) schedule is serializable if it is equivalent to a serial schedule. Different forms of schedule equivalence give rise to the notions of:
 1. **Conflict Serializability**
 2. **View Serializability**

➤ *Simplified view of transactions*

- We ignore operations other than **read** and **write** instructions
- We assume that transactions may perform arbitrary computations on data in local buffers in between reads and writes.
- Our simplified schedules consist of only **read** and **write** instructions.

Locking Transaction:

- Guarantees exclusive use of a data item to a current transaction
- Required to prevent another transaction from reading inconsistent data
- A lock is a mechanism to control concurrent access to a data item
- Data items can be locked in two modes :
 1. *exclusive (X) mode*. Data item can be both read as well as written. X-lock is requested using **lock-X** instruction.
 2. *shared (S) mode*. Data item can only be read. S-lock is requested using **lock-S** instruction.

Lock-compatibility matrix

		lock requested		
		-	S	X
lock held	-	Ok	Ok	Ok
	S	Ok	Ok	
	X	Ok		
		compatibility of locks		

- A transaction may be granted a lock on an item if the requested lock is compatible with locks already held on the item by other transactions
- Any number of transactions can hold shared locks on an item,
 - but if any transaction holds an exclusive on the item no other transaction may hold any lock on the item.
- If a lock cannot be granted, the requesting transaction is made to wait till all incompatible locks held by other transactions have been released. The lock is then granted.

Timestamp-Based Protocols

- Each transaction is issued a timestamp when it enters the system. If an old transaction T_i has time-stamp $TS(T_i)$, a new transaction T_j is assigned time-stamp $TS(T_j)$ such that $TS(T_i) < TS(T_j)$.

- The protocol manages concurrent execution such that the time-stamps determine the serializability order.
- In order to assure such behavior, the protocol maintains for each data Q two timestamp values:
 - **W-timestamp**(Q) is the largest time-stamp of any transaction that executed **write**(Q) successfully.
 - **R-timestamp**(Q) is the largest time-stamp of any transaction that executed **read**(Q) successfully.
- The timestamp ordering protocol ensures that any conflicting **read** and **write** operations are executed in timestamp order.
- Suppose a transaction T_i issues a **read**(Q)
 - If $TS(T_i) \leq \mathbf{W}\text{-timestamp}(Q)$, then T_i needs to read a value of Q that was already overwritten.
 - Hence, the **read** operation is rejected, and T_i is rolled back.
 - If $TS(T_i) \geq \mathbf{W}\text{-timestamp}(Q)$, then the **read** operation is executed, and $\mathbf{R}\text{-timestamp}(Q)$ is set to $\mathbf{max}(\mathbf{R}\text{-timestamp}(Q), TS(T_i))$.
- Suppose that transaction T_i issues **write**(Q).
 - If $TS(T_i) < \mathbf{R}\text{-timestamp}(Q)$, then the value of Q that T_i is producing was needed previously, and the system assumed that that value would never be produced.
 - Hence, the **write** operation is rejected, and T_i is rolled back.
 - If $TS(T_i) < \mathbf{W}\text{-timestamp}(Q)$, then T_i is attempting to write an obsolete value of Q .
 - Hence, this **write** operation is rejected, and T_i is rolled back.
 - Otherwise, the **write** operation is executed, and $\mathbf{W}\text{-timestamp}(Q)$ is set to $TS(T_i)$.

Chapter # 9**Concurrency Control****Lecture# 26****Introduction to Concurrency:**

- Coordination of simultaneous transaction execution in a multiprocessing database system
- Objective is to ensure serializability of transactions in a multiuser database environment

Purpose of Concurrency Control

- To enforce Isolation (through mutual exclusion) among conflicting transactions.
- To preserve database consistency through consistency preserving execution of transactions.
- To resolve read-write and write-write conflicts.

Example: In concurrent execution environment if T1 conflicts with T2 over a data item A, then the existing concurrency control decides if T1 or T2 should get the A and if the other transaction is rolled-back or waits.

Related Issues**Lost update problem:**

- Two concurrent transactions update same data element
- One of the updates is lost
 - Overwritten by the other transaction

Uncommitted data phenomenon:

- Two transactions executed concurrently
- First transaction rolled back after second already accessed uncommitted data

Inconsistent retrievals:

- First transaction accesses data
- Second transaction alters the data
- First transaction accesses the data again
- Transaction might read some data before they are changed and other data after changed
- Yields inconsistent results

Example of Fund Transfer

- transaction to transfer \$50 from account A to account B:

1. **read(A)**
2. **$A := A - 50$**
3. **write(A)**
4. **read(B)**
5. **$B := B + 50$**
6. **write(B)**

- Two main issues to deal with:

- Failures of various kinds, such as hardware failures and system crashes
- Concurrent execution of multiple transactions

- **Atomicity requirement**

- if the transaction fails after step 3 and before step 6, money will be “lost” leading to an inconsistent database state
 - Failure could be due to software or hardware
- the system should ensure that updates of a partially executed transaction are not reflected in the database

- **Durability requirement**

- Once the user has been notified that the transaction has completed (i.e., the transfer of the \$50 has taken place), the updates to the database by the transaction must persist even if there are software or hardware failures.

- **Consistency requirement**

- in above example, the sum of A and B is unchanged by the execution of the transaction
- In general, consistency requirements include
 - Explicitly specified integrity constraints such as primary keys and foreign keys
 - Implicit integrity constraints
 - e.g. sum of balances of all accounts, minus sum of loan amounts must equal value of cash-in-hand
- A transaction must see a consistent database.
- During transaction execution the database may be temporarily inconsistent.
- When the transaction completes successfully the database must be consistent
 - Erroneous transaction logic can lead to inconsistency

Isolation requirement

- if between steps 3 and 6, another transaction T_2 is allowed to access the partially updated database, it will see an inconsistent database (the sum $A + B$ will be less than it should be).

Conflicts in concurrency Controls**Concurrent Execution and Schedules**

- **Concurrent execution:** executing transactions simultaneously has the following advantages:
 - increased processor and disk utilization, leading to better throughput
 - one transaction can be using the CPU while another is reading from or writing to the disk
 - reduced average response time for transactions: short transactions need not wait behind long ones
- **Concurrency control schemes:** these are mechanisms to achieve isolation
 - to control the interaction among the concurrent transactions in order to prevent them from destroying the consistency of the database
- **Schedules:** sequences that indicate the chronological order in which instructions of concurrent transactions are executed
 - a schedule for a set of transactions must consist of all instructions of those transactions
 - must preserve the order in which the instructions appear in each individual transaction

Example Schedules

Let T_1 transfer \$50 from A to B , and T_2 transfer 10% of the balance from A to B . The following is a serial schedule (Schedule 1 in the text), in which T_1 is followed by T_2 .

Schedule 1

- Let T_1 transfer \$50 from A to B , and T_2 transfer 10% of the balance from A to B .
- A serial schedule in which T_1 is followed by T_2 :

T_1	T_2
<code>read(A)</code> <code>A := A - 50</code> <code>write(A)</code> <code>read(B)</code> <code>B := B + 50</code> <code>write(B)</code>	<code>read(A)</code> <code>temp := A * 0.1</code> <code>A := A - temp</code> <code>write(A)</code> <code>read(B)</code> <code>B := B + temp</code> <code>write(B)</code>

Schedule 2

- A serial schedule where T_2 is followed by T_1

T_1	T_2
<code>read(A)</code> <code>A := A - 50</code> <code>write(A)</code> <code>read(B)</code> <code>B := B + 50</code> <code>write(B)</code>	<code>read(A)</code> <code>temp := A * 0.1</code> <code>A := A - temp</code> <code>write(A)</code> <code>read(B)</code> <code>B := B + temp</code> <code>write(B)</code>

Schedule 3

Let T_1 and T_2 be the transactions defined previously. The following schedule is not a serial schedule, but it is *equivalent* to Schedule 1.

T_1	T_2
<code>read(A)</code> <code>A := A - 50</code> <code>write(A)</code>	<code>read(A)</code> <code>temp := A * 0.1</code> <code>A := A - temp</code> <code>write(A)</code>
<code>read(B)</code> <code>B := B + 50</code> <code>write(B)</code>	<code>read(B)</code> <code>B := B + temp</code> <code>write(B)</code>

In Schedules 1, 2 and 3, the sum $A + B$ is preserved.

Schedule 4

The following concurrent schedule does not preserve the value of $(A + B)$.

T_1	T_2
read(A) $A := A - 50$	read(A) $temp := A * 0.1$ $A := A - temp$ write(A) read(B)
write(A) read(B) $B := B + 50$ write(B)	 $B := B + temp$ write(B)

Chapter # 10**Database security****Lecture# 27****Database Security**

- Database Security refers to protection from malicious access.
- To protect a database, we must take security measures at several levels; Absolute protection is impossible
- We must also ensure that unauthorized users are prevented from accessing or modifying our database.
- To do this, we implement authorization rules for users called privileges.
- A DBMS typically includes a database security and authorization subsystem that is responsible for ensuring the security portions of a database against unauthorized access.

Malicious Access

- Some forms of malicious access:
 - Unauthorized reading (theft) of data
 - Unauthorized modification of data
 - Unauthorized destruction of data

Two types of database security mechanisms:

- Discretionary security mechanisms
- Mandatory security mechanisms

Security Levels

- Database System: Since some users may modify data while some may only query, it is the job of the system to enforce authorization rules.
- Operating System: No matter how secure the database system is, the operating system may serve as another means of unauthorized access.
- Network: Since most databases allow remote access, hardware and software security is crucial.
- Physical: Sites with computer systems must be physically secured against entry by intruders or **terrorists**.
- Human: Users must be authorized carefully to reduce the chance of a user giving access to an intruder

Threats to databases

- Loss of integrity
- Loss of availability
- Loss of confidentiality

Database Security Mechanisms:

- The security mechanism of a DBMS must include provisions for restricting access to the database as a whole; this function is called **access control** and is handled by creating user accounts and passwords to control login process by the DBMS.
- Another security is that of **flow control**, which prevents information from flowing in such a way that it reaches unauthorized users.
- A final security issue is **data encryption**, which is used to protect sensitive data (such as credit card numbers) that is being transmitted via some type communication network. The data is encoded using some coding algorithm. An unauthorized user who access encoded data will have difficulty deciphering it, but authorized users are given decoding or decrypting algorithms (or keys) to decipher data.

Database Security Measures:**1. Authentication & Authorization:**

- Authentication means verifying the identity of someone (a user, device or entity) who wants to use database.
- Each user must enter password along with user login name.
- Database stores user's password in the data dictionary in an encrypted format to prevent unauthorized alteration.
- After authentication, the authorization is the operation that verifies the permissions and access rights granted to a user. Authorization primarily includes two processes:
 - Permitting only certain users to access, process or alter data
 - Applying varying limitations on user's access or actions.

2. Encryption

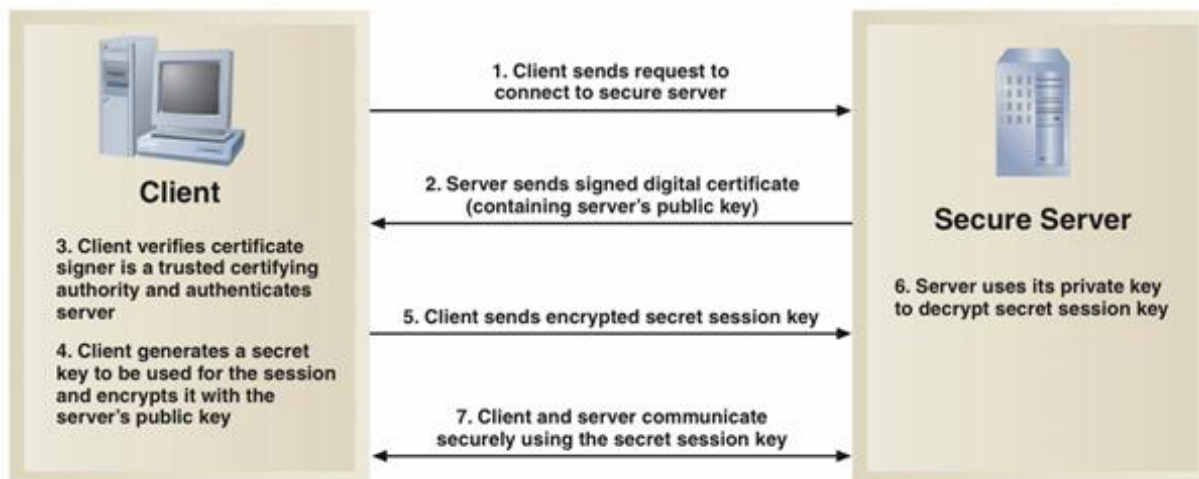
- Encryption is a means of maintaining secure data in an insecure environment.

- Encryption consists of applying an **encryption algorithm** to data using some prespecified **encryption key**. The resulting data has to be **decrypted** using a **decryption key** to recover the original data

3. Digital Signatures and Certificates

- A digital signature is an example of using encryption techniques to provide authentication services in e-commerce applications.
- Encryption of messages enables secure exchange of information between two entities with appropriate keys
- Digital signature encrypts document with private key to verify document author
- Digital certificate is institution's name and public key that is encrypted and certified by third party

Using a Digital Certificate



Designing Security Controls

- Security controls protect assets of organization from all threats
 - External threats such as hackers, viruses, worms, and message overload attacks
- Security control objectives
 - Maintain stable, functioning operating environment for users and application systems (24 x 7)
 - Protect information and transactions during transmission outside organization (public carriers)

Security for Access to Systems

- Used to control access to any resource managed by operating system or network
- User categories
 - Unauthorized user – no authorization to access
 - Registered user – authorized to access system
 - Privileged user – authorized to administrate system
- Organized so that all resources can be accessed with same unique ID/password combination

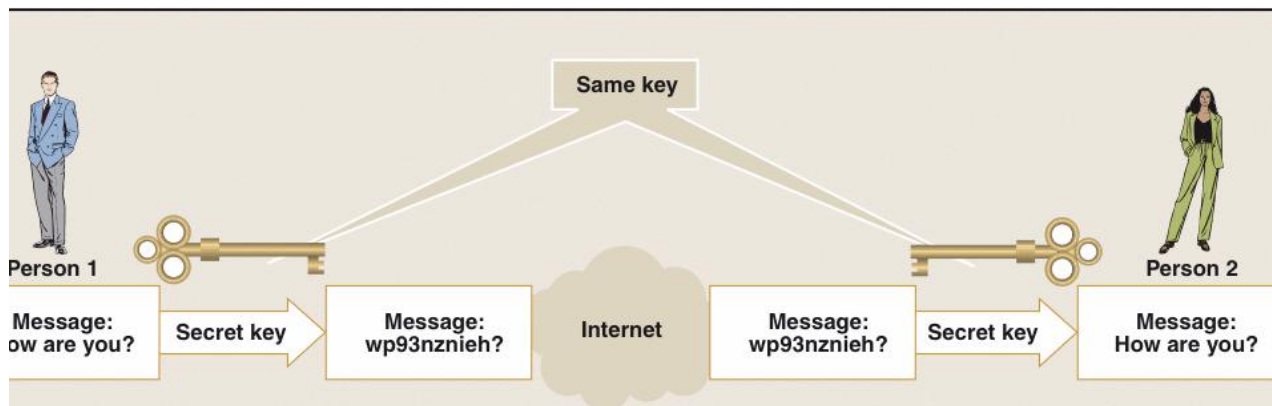
Managing User Access

- Most common technique is user ID / password
- Authorization – Is user permitted to access?
- Access control list – users with rights to access
- Authentication – Is user who they claim to be?
- Smart card – computer-readable plastic card with embedded security information
- Biometric devices – keystroke patterns, fingerprinting, retinal scans, voice characteristics

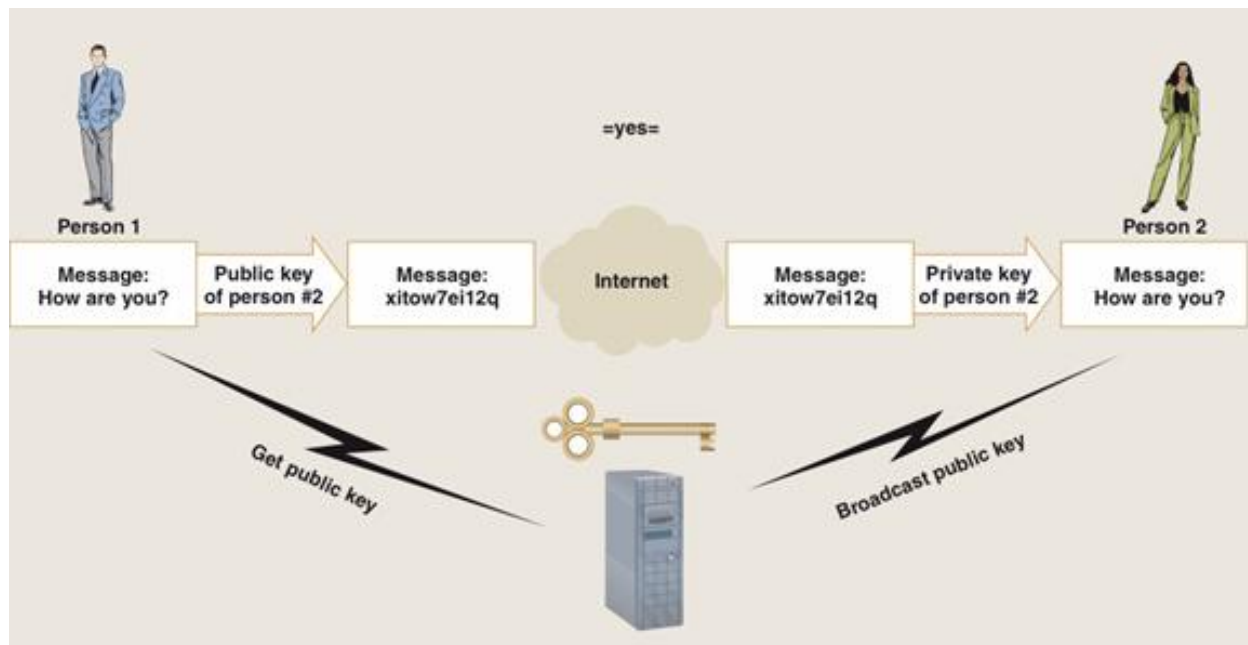
Data Security

- Data and files themselves must be secure
- Encryption – primary security method
 - Altering data so unauthorized users cannot view
- Decryption
 - Altering encrypted data back to its original state
- Symmetric key – same key encrypts and decrypts
- Asymmetric key – different key decrypts
- Public key – public encrypts; private decrypts

Symmetric Key Encryption



Asymmetric Key Encryption



**Department of Computer Science
University of Karachi**

BCS Program (5th Semester)

2302-Files & Databases

Terminal Examination Date: 14-July-2016

Max. Time: 2 Hours Max. Marks: 60

Seat Number:	Name:
Enrollment Number:	Institute/College:

Instructions:

- 1) Attempt all questions. All questions carry equal marks.
- 2) Do not write on question paper except Name, Seat Number and Enrollment Number.
- 3) Please return the question paper along with your answer script.

Question # 1

- a) Differentiate between the "Data Administrator" and "Database Administrator". Write down any three major responsibilities of a "data administrator".
- b) Why normalization is considered to be an important and integral part of the Database design? What would be the drawbacks if normalization is NOT performed with raw relational schema design?
- c) How the internal structure of database management system is constituted? Explain with the help of block diagram
- d) Describe the features of Embedded SQL and Dynamic SQL. Give suitable examples.

Question # 2

- a) Explain the Boyce-Codd normal form with an example. Also state how it differs from that of 3NF.
- b) How concurrency is performed? Explain the protocol that is used to maintain the concurrency concept.
- c) What do we mean by the term Data models? Differentiate between records based and object based data models.
- d) Describe the concept of serializability (w.r.t DB transactions) with suitable example.

Question # 3

- a) Describe the main characteristics of the Relational Data Model, including the properties of relations and the rules for relational integrity.
- b) What are Views and list any three uses of views?
- c) Do as Directed (read the rules and answer the questions mentioned below in *italic*)

Rule 12

If a relational system has a low-level language, that low level cannot be used to subvert or bypass the integrity rules and constraints expressed in the higher-level relational language

Here what is meant by low-level Language and Low Level access? What could be the effect of bypassing the integrity rules/constraints?

Rule 10

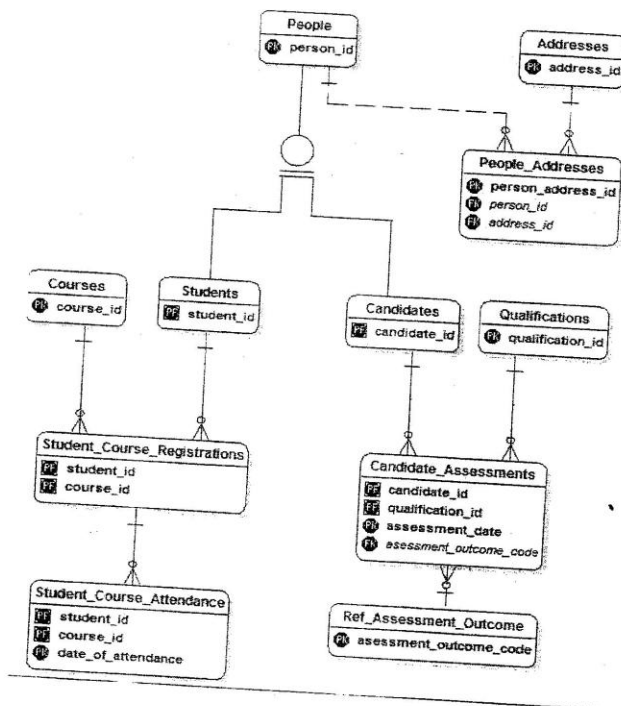
Integrity constraints specific to a particular relational database must be definable in the relational data sublanguage and storable in the catalog, not in the application programs.

Here what is being referenced w.r.t definable in relational data sublanguage and storable in the catalog? What alternative we have? What could be the consequences of failure?

Rule 11

The data manipulation sublanguage of a relational DBMS must enable application programs and inquiries to remain logically the same whether and whenever data are physically centralized or distributed.

In the rule mentioned above if the Central database is fragmented and distributed then how the application programs and inquiries remain the same? Suggest some way out.

Question # 4

Drawn above is the student assessment ER-diagram. You are required to

- Check the above ER for "Fan-Trap" and "Chasm-Trap" (explain each of them first).
- After attempting part a) further check the ER that it validates the First 4 Normal forms, IF the ER is NOT normalized then Normalize the ER-Schema up-to 4th Normal form (Mention the definition of Each Normal form before performing normalization over the ER-Schema)
- Convert the ER to the proper relation schema.

Department of Computer Science
University of Karachi

BCS Program (5th Semester)**2302-Files & Databases**

Terminal Examination

Date: 22-June-2015

Max. Time: 2½ Hours

Max. Marks: 60

Seat Number:	Name:
Enrollment Number:	Institute/College:

Instructions:

- **SWITCH OFF MOBILE PHONES**
- **RETURN THE QUESTION PAPER ALONG WITH YOUR ANSWER SCRIPTS**
- **ATTEMPT ANY SIX**

QUESTION 1

- a. Describe the three-schema architecture. Why do we need mappings between schema levels? How do different schema definition languages support this architecture?
- b. Give examples of systems in which it may make sense to use traditional file processing instead of database approach.

QUESTION 2

- a. If you were designing a Web-based system to make airline reservations and sell airline tickets, which DBMS architecture (data model) would you choose? Why? Why would the other architectures not be a good choice?
- b. What are the strengths of relational data model? List any six rules defined by E.F. Codd for any relational database product.

QUESTION 3

- a. Discuss insertion, deletion, and modification anomalies. Why are they considered adverse for databases? Illustrate with examples.
- b. Consider the following relation schema and the functional dependencies:
 CAR-SALE (Car#, Date-sold, Salesman#, Commission, Discount)
 Primary key: Car#, Salesman#
 Functional dependencies: (fd1) Salesman# → Commission
 (fd2) Date-sold → Discount
 1. Is the given relation schema in 1NF? Explain & justify your answer.
 2. Is the given relation schema in 2NF? Explain & justify your answer.
 3. Is the given relation schema in 3NF? Explain & justify your answer.
 4. If in your opinion the given relation schema is not in 3NF then transform it into a set of relation schemas such that each resulting relation schema is in 3NF.

QUESTION 4

- a. Why does a DBMS store data on external storage? Why are I/O costs important in a DBMS?
- b. Consider the following relations containing airline flight information: write SQL expression for the following queries.
 Flights(fldno: integer, from: string, to: string, distance: integer, departs: time, arrives: time)
 Aircraft(aid: integer, aname: string, cruisingrange: integer)
 Certified(eid: integer, aid: integer)
 Employees(eid: integer, ename: string, salary: integer)
 1. Find the "eid(s)" of pilots certified for some Boeing aircraft.
 2. Find the "names" of pilots certified for some Boeing aircraft
 3. Find the "eid(s)" of employees who make the highest salary
 4. Identify the flights that can be piloted by every pilot whose salary is more than \$100,000.

QUESTION 5

Consider a CONFERENCE_REVIEW database in which researchers submit their research papers for consideration. Reviews by reviewers are recorded for use in the paper selection process. The database system caters primarily to reviewers who record answers to evaluation questions for each paper they review and make recommendations regarding whether to accept or reject the paper. The data requirements are summarized as follows: Design an Entity-Relationship diagram for the CONFERENCE_REVIEW database

- Authors of papers are uniquely identified by e-mail id. First and last names are also recorded.
- Each paper is assigned a unique identifier by the system and is described by a title, abstract, and the name of the electronic file containing the paper.
- A paper may have multiple authors, but one of the authors is designated as the contact author.
- Reviewers of papers are uniquely identified by e-mail address. Each reviewer's first name, last name, phone number, affiliation, and topics of interest are also recorded.
- Each paper is assigned between two and four reviewers. A reviewer rates each paper assigned to him or her on a scale of 1 to 10 in four categories: technical merit, readability, originality, and relevance to the conference. Finally, each reviewer provides an overall recommendation regarding each paper.
- Each review contains two types of written comments: one to be seen by the review committee only and the other as feedback to the author(s).

QUESTION 6

- a. Why should NULLs in a relation be avoided as much as possible? Discuss the problem of spurious tuples and how we may prevent it.

- b. Consider the following schema:
Suppliers(sid: integer, sname: string, address: string)
Parts(pid: integer, pname: string, color: string)
Catalog(sid: integer, pid: integer, cost: real)

The key fields are underlined, and the domain of each field is listed after the field name. Therefore sid is the key for Suppliers, pid is the key for Parts, and sid and pid together form the key for Catalog. The Catalog relation lists the prices charged for parts by Suppliers. Write the following queries in relational algebra.

1. Find the sids of suppliers who supply some red or green part.
2. Find the sids of suppliers who supply every part.
3. Find the sids of suppliers who supply every red part.
4. Find the pids of parts supplied by at least two different suppliers.

QUESTION 7

- a. Define the following terms:
1. Database catalog
 2. Program-data independence
 3. Transaction management
- b. Discuss the advantages and disadvantages of using (a) an unordered file, (b) an ordered file, and (c) a static hash file with buckets and chaining.
- c. When is the concept of a weak entity used in data modeling? Define the terms owner entity type, weak entity type, identifying relationship type, and partial key.