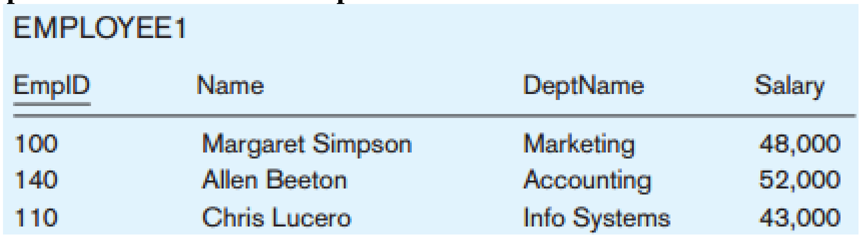
UNIT – I DATABASE MODELLING, MANAGEMENT AND DEVELOPMENT
PART- A

1. **Define Database design.**
   Database design is the process of producing a detailed data model of a database. This data model contains all the needed logical and physical design choices and physical storage parameters needed to generate a design in a data definition language, which can then be used to create a database. The activities of database design transform the requirements for data storage developed during database analysis into specifications to guide database implementation.
2. **What are the two forms of database design specification?**
   There are two forms of specifications:
   1) Logical specifications, which map the conceptual requirements into the data model associated with a specific database management system
   2) Physical specifications, which indicate all the parameters for data storage that are then input to database implementation, during which a database is actually defined using a data definition language
3. **What do you mean by Conceptual data modeling and logical database design?**
   - **Conceptual data modeling** is about understanding the organization—getting the right requirements.
   - **Logical database design** is about creating stable database structures—correctly expressing the requirements in a technical language.
   - Both are important steps that must be performed carefully and concentrated on each important part of database development.
4. **Define Relational Model.**
   The relational data model represents data in the form of tables. The relational model is based on mathematical theory and therefore has a solid theoretical foundation. However, we need only a few simple concepts to describe the relational model.
5. **What are the three components of the Relational Data Model?**
   The relational data model consists of the following three components:
   1. **Data structure** Data are organized in the form of tables, with rows and columns.
   2. **Data manipulation** Powerful operations (using the SQL language) are used to manipulate data stored in the relations.
   3. **Data integrity** The model includes mechanisms to specify business rules that maintain the Integrity of data when they are manipulated.
6. **Define Relational Data Structure.**
   A relation is a named, two-dimensional table of data. Each relation (or table) consists of a set of named columns and an arbitrary number of unnamed rows. An attribute is a named column of a relation. Each row of a relation corresponds to a record that contains data (attribute) values for a single entity.
7. **Give an example for a relation with sample data and its shorthand notation.**



EMPLOYEE1

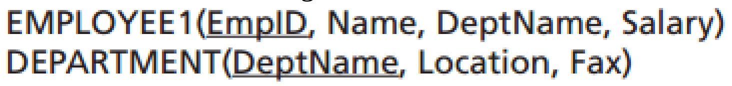
EmpID	Name	DeptName	Salary
100	Margaret Simpson	Marketing	48,000
140	Allen Beeton	Accounting	52,000
110	Chris Lucero	Info Systems	43,000

The above table shows an example of a relation named EMPLOYEE1. This relation contains the following attributes describing employees: EmpID, Name, DeptName, and Salary.

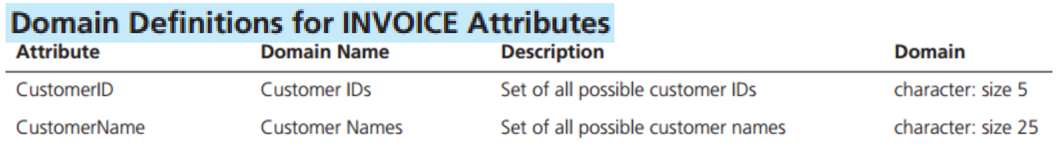
EMPLOYEE1(EmpID, Name, DeptName, Salary)

The name of the relation is followed (in parentheses) by the names of the attributes in that relation

8. **Define Primary Key.**
   A primary key is an attribute or a combination of attributes that uniquely identifies each row in a relation. A primary key is designated by underlining the attribute name(s). Example, the primary key for the relation EMPLOYEE1 is EmpID.
9. **Define Composite Key.**
   A composite key is a primary key that consists of more than one attribute. For example, the primary key for a relation DEPENDENT would likely consist of the combination EmpID and DependentName.
10. **Define Foreign Key.**
    A foreign key is an attribute (possibly composite) in a relation that serves as the primary key of another relation. For example, consider the relations EMPLOYEE1 and DEPARTMENT. The attribute DeptName is a foreign key in EMPLOYEE1. It allows a user to associate any employee with the department to which he or she is assigned.


EMPLOYEE1(EmpID, Name, DeptName, Salary)
DEPARTMENT(DeptName, Location, Fax)

11. **Mention few properties of Relation.**
    1) Each relation (or table) in a database has a unique name.
    2) An entry at the intersection of each row and column is atomic (or single valued). There can be only one value associated with each attribute on a specific row of a table; no multivalued attributes are allowed in a relation.
    3) Each row is unique; no two rows in a relation can be identical.
    4) Each attribute (or column) within a table has a unique name.
12. **Define Integrity Constraints and give their types.**
    The relational data model includes several types of constraints, or rules limiting acceptable values and actions, whose purpose is to facilitate maintaining the accuracy and integrity of data in the database. The major types of integrity constraints are domain constraints, entity integrity, and referential integrity.
13. **Define Domain Constraints.**
    All of the values that appear in a column of a relation must be from the same domain. A domain is the set of values that may be assigned to an attribute. A domain definition usually consists of the following components: domain name, meaning, data type, size (or length), and allowable values or allowable range
    Ex:


Domain Definitions for INVOICE Attributes

Attribute	Domain Name	Description	Domain
CustomerID	Customer IDs	Set of all possible customer IDs	character: size 5
CustomerName	Customer Names	Set of all possible customer names	character: size 25

14. **Define Entity Integrity.**
    The entity integrity rule is designed to ensure that every relation has a primary key and that the data values for that primary key are all valid. In particular, it guarantees that every primary key attribute is non-null.
15. **Define Referential Integrity.**
    A referential integrity constraint is a rule that maintains consistency among the rows of two relations. The rule states that if there is a foreign key in one relation, either each foreign key value must match a primary key value in another relation or the foreign key value must be null.
16. **Define Well Structured Relation.**
    A relation that contains minimal redundancy and allows users to insert, modify, and delete the rows in a table without errors or inconsistencies.
17. **Define Anomaly.**
    An error or inconsistency that may result when a user attempts to update a table that contains redundant data. The three types of anomalies are insertion, deletion, and modification anomalies.

18. Define Normalization.

The process of decomposing relations with anomalies to produce smaller, well-structured relations.

19. Define Functional Dependency.

A functional dependency is a constraint between two attributes or two sets of attributes. For any relation R, attribute B is functionally dependent on attribute A if, for every valid instance of A, that value of A uniquely determines the value of B.The functional dependency of B on A is represented by an arrow, as follows: A → B. A functional dependency is not a mathematical dependency: B cannot be computed from A.

Ex:

EmpID, CourseTitle → DateCompleted

The comma between EmpID and CourseTitle stands for the logical AND operator, because DateCompleted is functionally dependent on EmpID and CourseTitle in combination.

20. Define Candidate key.

A candidate key is an attribute, or combination of attributes, that uniquely identifies a row in a relation. A candidate key must satisfy the following properties (Dutka and Hanson, 1989), which are a subset of the six properties of a relation previously listed:

21. What are the four problems that arise in view integration?

Synonyms, Homonyms, Transitive dependencies, and Supertype/ Subtype relationships.

22. Define E-R diagram or ERD.

An entity-relationship model (E-R model) is a detailed, logical representation of the data for an organization or for a business area. The E-R model is expressed in terms of entities in the business environment, the relationships (or associations) among those entities, and the attributes (or properties) of both the entities and their relationships. An E-R model is normally expressed as an entity-relationship diagram (E-R diagram, or ERD), which is a graphical representation of an E-R model.

23. Define Business Rules.

A business rule is "a statement that defines or constrains some aspect of the business. It is intended to assert business structure or to control or influence the behavior of the business ...rules prevent, cause, or suggest things to happen"

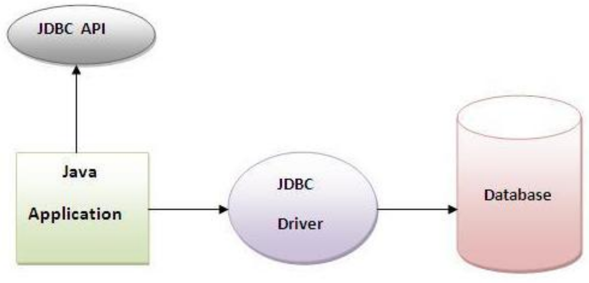
Ex: *"A student may register for a section of a course only if he or she has successfully completed the prerequisites for that course."*

24. What are the Characteristics of good business rule?

- Declarative
- Precise
- Atomic
- Consistent
- Expressible
- Distinct
- Business-oriented

25. What is meant by Java Database Connectivity?

Java Database Connectivity (JDBC) is an application programming interface (API) for the programming language Java, which defines how a client may access a database. It provides methods to query and update data in a database, and is oriented towards relational databases. A JDBC-to-ODBC bridge enables connections to any ODBC-accessible data source in the Java virtual machine (JVM) host environment.



26. What are the steps involved in connecting to the database in Java?

5 Steps to connect to the database in java

1) Register the driver class
2) Create the connection object
3) Create the Statement object
4) Execute the query
5) Close the connection object

27. Define Database Connection Manager.

Database Connection Manager is a useful tool, which allows you to connect to a local or remote MySQL database for Database Generation, Database Modification, Diagram Synchronization, or running SQL scripts. Use Connect ( ) item on Database tab of the Ribbon to open Database Connection Manager. Database Connection Manager uses connection profiles, which allow you to set all the connection parameters for a specific database only once, and quickly connect to this database later.

28. Define Stored Procedures and give its advantages.

- Stored procedures are modules of code that implement application logic and are included on the database server.
- Stored procedures have the following advantages:
    - Performance improves for compiled SQL statements.
    - Network traffic decreases as processing moves from the client to the server.
    - Security improves if the stored procedure rather than the data is accessed and code is moved to the server, away from direct end-user access.

29. What is Big data?

Every day, we create 2.5 quintillion bytes of data — so much that 90% of the data in the world today has been created in the last two years alone. This data comes from everywhere: sensors used to gather climate information, posts to social media sites, digital pictures and videos, purchase transaction records, and cell phone GPS signals to name a few. This data is big data.

30. What Comes Under Big Data?

- **Black Box Data:** It is a component of helicopter, airplanes, and jets, etc. It captures voices of the flight crew, recordings of microphones and earphones, and the performance information of the aircraft.
- **Social Media Data:** Social media such as Facebook and Twitter hold information and the views posted by millions of people across the globe.
- **Stock Exchange Data:** The stock exchange data holds information about the 'buy' and 'sell' decisions made on a share of different companies made by the customers.
- **Search Engine Data:** Search engines retrieve lots of data from different databases.

31. What are the three V's that defines the Big Data?

Volume: Enterprises are flooded with ever-growing data of all types, easily amassing terabytes—even petabytes—of information.

- **Ex:** Convert 350 billion annual meter readings to better predict power consumption

Velocity: Sometimes 2 minutes is too late. For time-sensitive processes such as catching fraud, big data must be used as it streams into your enterprise in order to maximize its value.

- **Ex:** Scrutinize 5 million trade events created each day to identify potential fraud

Variety: Big data is any type of data - structured and unstructured data such as text, sensor data, audio, video, click streams, log files and more.

- **Ex:** Monitor 100's of live video feeds from surveillance cameras to target points of interest

32. What are the four core capabilities of Big Data Platform?

The core capabilities are:

1) **Hadoop-based analytics:** Processes and analyzes any data type across commodity server clusters.
2) **Stream Computing:** Drives continuous analysis of massive volumes of streaming data with sub-millisecond response times.
3) **Data Warehousing:** Delivers deep operational insight with advanced in-database analytics.
4) **Information Integration and Governance:** Allows you to understand, cleanse, transform, govern and deliver trusted information to your critical business initiatives.

33. Define Operational Big Data.

- This include systems like MongoDB that provide operational capabilities for real-time, interactive workloads where data is primarily captured and stored.
- NoSQL Big Data systems are designed to take advantage of new cloud computing architectures that have emerged over the past decade to allow massive computations to be run inexpensively and efficiently. This makes operational big data workloads much easier to manage, cheaper, and faster to implement.

34. Define Analytical Big Data.

- This includes systems like Massively Parallel Processing (MPP) database systems and MapReduce that provide analytical capabilities for showing and complex analysis that may touch most or all of the data.
- MapReduce provides a new method of analyzing data that is complementary to the capabilities provided by SQL, and a system based on MapReduce that can be scaled up from single servers to thousands of high and low end machines.

35. Define NoSQL.

NoSQL database, also called Not Only SQL, is an approach to data management and database design that's useful for very large sets of distributed data.It's high performance with high availability, offers rich query language and easy scalability. NoSQL databases are structured in a key-value pair, graph database, document-oriented or column-oriented structure.

Ex: Hadoop, MangoDB and MapReduce

36. What is Hadoop HDFS?

Hadoop is an open-source framework that allows to store and process big data in a distributed environment across clusters of computers using simple programming models.Hadoop File System was developed using distributed file system design (HDFS). It is run on commodity hardware. These files are stored in redundant fashion to rescue the system from possible data losses in case of failure. HDFS also makes applications available to parallel processing.

37. Define MapReduce.

MapReduce is a processing technique and a program model for distributed computing based on java. The MapReduce algorithm contains two important tasks, namely Map and Reduce. Map takes a set of data and converts it into another set of data, where individual elements are broken down into tuples (key/value pairs). Secondly, reduce task, which takes the output from a map as an input and combines those data tuples into a smaller set of tuples.

38. Define Hive.

Hive is a data warehouse infrastructure tool to process structured data in Hadoop. It resides on top of Hadoop to summarize Big Data, and makes querying and analyzing easy. Initially Hive was developed by Facebook, later the Apache Software Foundation took it up and developed it further as an open source under the name Apache Hive.

PART- B

1. <u>Explain about the Database Modelling with a sample ER Diagram.</u>

DATABASE DESIGN AND MODELLING

- Relatively simple representations, usually graphical, of complex real-world data structures
- Facilitate interaction among the designer, the applications programmer, and the end user
- End-users have different views and needs for data

- Data model organizes data for various users

Basic Building blocks of data models

- Entity - anything about which data are to be collected and stored
- Attribute - a characteristic of an entity
- Relationship - describes an association among entities
  - One-to-many (1:M) relationship
  - Many-to-many (M:N or M:M) relationship
  - One-to-one (1:1) relationship
- Constraint - a restriction placed on the data
  {*each Person was born on at most one Date*}

BUSINESS RULE

- These are often implemented as 'Constraints' in the Database.
- These Business Rules are important because they define the conditions that the Database must meet.
- ***Example***
  - *Every Order must be associated with a valid Product.*
  - *A SHIP DATE cannot be prior to an ORDER DATE for any given order.*
- This prevents invalid Orders being entered into the Database.
- Business rules, the foundation of data models, are derived from policies, procedures, events,
  functions, and other business objects, and they state constraints(limit) on the organization.
- Business rules represent the language and fundamental structure of an organization.
- Business rules formalize the understanding of the organization by organization owners, managers, and leaders with that of information systems architects.
- Examples of basic business rules are data names and definitions.
- In terms of conceptual data modeling, names and definitions must be provided for the main data objects:
  - entity types (e.g., Customer)
  - attributes (Customer Name)
  - relationships (Customer Places Orders).
- These constraints can be captured in a data model, such as an entity-relationship diagram, and associated documentation.
- Additional business rules govern the people, places, events, processes, networks, and objectives of the organization, which are all linked to the data requirements through other system documentation.
- E-R model remains the mainstream approach for conceptual data modeling.
- The E-R model is most used as a tool for communications between database designers and end users during the analysis phase of database development
- The E-R model is used to construct a conceptual data model
- A representation of the structure and constraints of a database that is independent of software (such as a database management system).
- Some authors introduce terms and concepts peculiar to the relational data model
- The relational data model is the basis for most database management systems in use today.
- In today's database environment, the database may be implemented with object-oriented technology or with a mixture of object-oriented and relational technology.

Many systems developers believe that data modeling is the most important part of the systems
development process for the following reasons

1. The characteristics of data captured during data modeling are crucial in the design of databases, programs, and other system components.

The facts and rules captured during the process of data modeling are essential in assuring data integrity in an information system.

2. Data rather than processes are the most complex aspect of many modern information systems and hence require a central role in structuring system requirements.
Often the goal is to provide a rich data resource that might support any type of information inquiry (investigation), analysis, and summary.

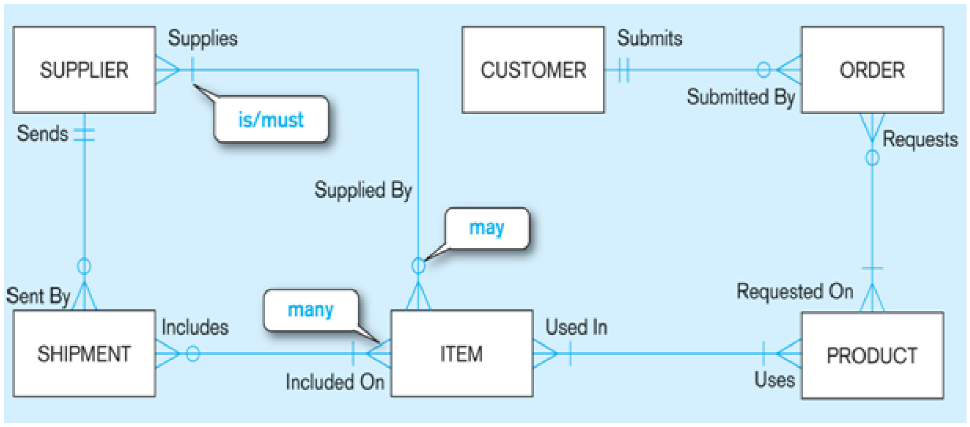
3. Data tend to be more stable than the business processes that use that data.
Thus, an information system design that is based on a data orientation should have a longer useful life than one based on a process orientation.
Business rules are important in data modeling because they govern how data are handled and
stored.

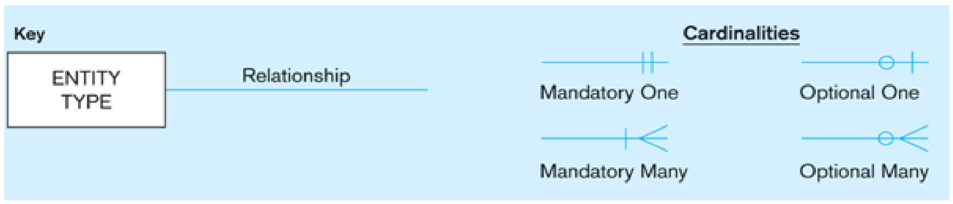
THE ER-MODEL: AN OVERVIEW

- An entity-relationship model (E-R model) is a detailed, logical representation of the data for an organization or for a business area.
- The E-R model is expressed in terms of
  - entities in the business environment
  - the relationships (or associations) among those entities
  - the attributes (or properties) of both the entities and their relationships.
- An E-R model is normally expressed as an entity-relationship diagram (E-R diagram, or ERD), which is a graphical representation of an E-R model.

SAMPLE ER MODEL

- o A simplified E-R diagram for a small furniture manufacturing company, Pine Valley Furniture
Company.
- o A number of suppliers supply and ship different items to Pine Valley Furniture.
- o The items are assembled into products that are sold to customers who order the products.
- o Each customer order may include one or more lines corresponding to the products appearing
on that order.





- o The diagram in the above Figure shows the entities and relationships for this company.
- o Attributes are omitted to simplify the diagram for now.

- Entities (the objects of the organization) are represented by the rectangle symbol
- Relationships between entities are represented by lines connecting the related entities.

The entities in the Figure are:

- **CUSTOMER** A person or an organization that has ordered or might order products.
- **PRODUCT** A type of furniture made by Pine Valley Furniture that may be ordered by customers.
- **ORDER** The transaction associated with the sale of one or more products to a customer and identified by a transaction number from sales or accounting.
- **ITEM** A type of component that goes into making one or more products and can be supplied by one or more suppliers.
- **SUPPLIER** Another company that may provide items to Pine Valley Furniture.
- **SHIPMENT** The transaction associated with items received in the same package by Pine Valley Furniture from a supplier.
- It is important to clearly define, as metadata, each entity.

Example

- It is important to know that the CUSTOMER entity includes persons or organizations that have not yet purchased products from Pine Valley Furniture.
- It is common for different departments (Accounting, marketing) in an organization to have different meanings for the same term

---

2. **Discuss in detail about the Business rules.**

OVERVIEW OF BUSINESS RULES

A business rule is "a statement that defines or constraints some aspect of the business".

- It is intended to assert business structure or to control or influence the behavior of the business
- Rules prevent, cause, or suggest things to happen

Example

- "A student may register for a section of a course only if he or she has successfully completed the prerequisites for that course."
- "A preferred customer qualifies for a 10 percent discount, unless he has an overdue account balance."

THE BUSINESS RULE PARADIGM

- The concept of business rules has been used in information systems for some time.
- There are many software products that help organizations manage their business rules (for example, JRules from ILOG, an IBM company).
- In the database world, it has been more common to use the related **term *integrity constraint*** when referring to such rules maintaining valid data values and relationships in the database

A business rules approach is based on the following premises:

- Business rules are a core concept in an enterprise (project) because they are an expression of business policy and guide individual and aggregate behavior.
    - Well-structured business rules can be stated in natural language for end users and in a data model for systems developers.
- Business rules can be expressed in terms that are familiar to end users.
    - Users can define and then maintain their own rules.
- Business rules are highly maintainable.
    - They are stored in a central repository, and each rule is expressed only once, and then shared throughout the organization.

    - Each rule is discovered and documented only once, to be applied in all systems development projects.
- Enforcement of business rules can be automated through the use of software that can interpret the rules and enforce them using the integrity mechanisms of the database management system.
- Automatic generation and maintenance of systems will not only simplify the systems development process but also will improve the quality of systems.

SCOPE OF BUSINESS RULES

- Most organizations have a host of rules and/or policies that fall outside this definition.
    - For example, the rule "Friday is business casual dress day" may be an important policy statement, but it has no immediate impact on databases.
    - In contrast, the rule "A student may register for a section of a course only if he or she has successfully completed the prerequisites for that course" is within our scope because it constrains the transactions that may be processed against the database.
- In particular, it causes any transaction to be rejected that attempts to register a student who does not have the necessary prerequisites.
- Some business rules cannot be represented in common data modeling notation; those rules that cannot be represented in a variation of an entity-relationship diagram are stated in natural language, and some can be represented in the relational data model

GOOD BUSINESS RULES

- Whether stated in natural language, a structured data model, or other information systems documentation, a business rule will have certain characteristics if it is to be consistent with the premises.
- These characteristics are summarized in the below Table.
- These characteristics will have a better chance of being satisfied if a business rule is defined, approved, and owned by business, not technical, people.
- Business people become stewards of the business rules.
- You, as the database analyst, assist the developing of the rules and the transformation of ill-stated rules into ones that satisfy the desired characteristics.

Characteristics	**Explanation**
Declarative	A business rule is a statement of policy, not how policy is enforced or conducted; the rule does not describe a process or implementation, but rather describes what a process validates.
Precise	With the related organization, the rule must have only one interpretation among all interested people, and its meaning must be clear.
Atomic	A business rule marks one statement, not several; no part of the rule can stand on its own as a rule (that is, the rule is indivisible, yet sufficient).
Consistent	A business rule must be internally consistent (that is, not contain conflicting statements) and must be consistent with (and not contradict) other rules.

Expressible	A business rule must be able to be stated in natural language, but it will be stated in a structured natural language so that there is no misinterpretation.
Distinct	Business rules are not redundant, but a business rule may refer to other rules (especially to definitions).
Business-oriented	A business rule is stated in terms businesspeople can understand, and because it is a statement of business policy, only business people can modify or invalidate a rule; thus, a business rule is owned by the business.

GATHERING BUSINESS RULES

- Business rules appear (possibly implicitly) in descriptions of business functions, events, policies, units, stakeholders, and other objects.
- These descriptions can be found in interview notes from individual and group information systems requirements collection sessions, organizational documents (e.g., personnel manuals, policies, contracts, marketing brochures, and technical instructions), and other sources.
- Rules are identified by asking questions about the who, what, when, where, why, and how of the organization.
- Usually, a data analyst has to be persistent in clarifying initial statements of rules because initial statements may be vague or imprecise (what some people have called "business ramblings").
- Thus, precise rules are formulated from an iterative inquiry process.

---

3. **Explain modeling entities and attributes with suitable examples.**

MODELING ENTITIES AND ATTRIBUTES

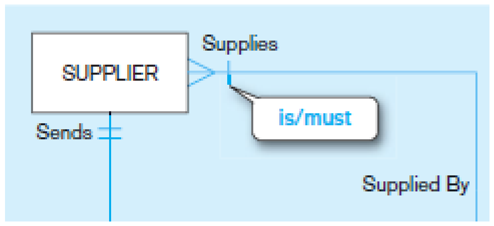
- The basic constructs of the E-R model are entities, relationships, and attributes.
- The richness of the E-R model allows designers to model real-world situations accurately and expressively, which helps account for the popularity of the model.

Entities

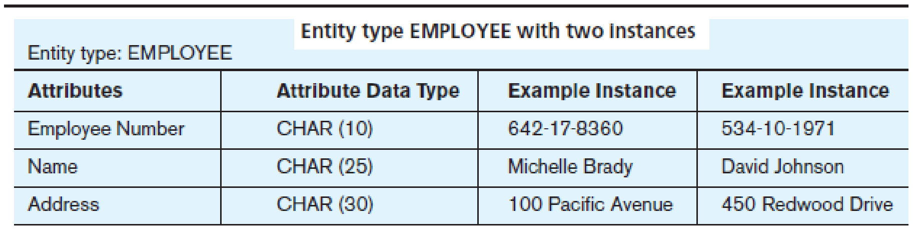
- An **entity** is a person, a place, an object, an event, or a concept in the user environment about which the organization wishes to maintain data.
- Thus, an entity has a noun name.
- Some examples of each of these *kinds* of entities follow:
- *Person:* EMPLOYEE, STUDENT, PATIENT
- *Place:* STORE, WAREHOUSE, STATE
- *Object:* MACHINE, BUILDING, AUTOMOBILE
- *Event:* SALE, REGISTRATION, RENEWAL
- *Concept:* ACCOUNT, COURSE, WORK CENTER

ENTITY TYPE VERSUS ENTITY INSTANCE

- An **entity type** is a collection of entities that share common properties or characteristics.
- Each entity type in an E-R model is given a name.
- Because the name represents a collection (or set) of items, it is always singular.
- We use capital letters for names of entity type(s).
- In an E-R diagram, the entity name is placed inside the box representing the entity type



- An **entity instance** is a single occurrence of an entity type.
- An entity type is described just once (using metadata) in a database, whereas many instances of that entity type may be represented by data stored in the database.
- For example, there is one EMPLOYEE entity type in most organizations, but there may be hundreds (or even thousands) of instances of this entity type stored in the database.



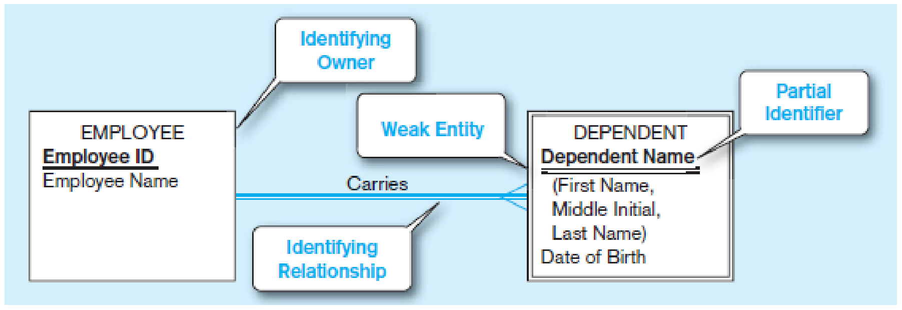
Entity type EMPLOYEE with two instances

Entity type: EMPLOYEE

Attributes	Attribute Data Type	Example Instance	Example Instance
Employee Number	CHAR (10)	642-17-8360	534-10-1971
Name	CHAR (25)	Michelle Brady	David Johnson
Address	CHAR (30)	100 Pacific Avenue	450 Redwood Drive

STRONG VERSUS WEAK ENTITY TYPES

- Most of the basic entity types to identify in an organization are classified as strong entity types.
- A **strong entity type** is one that exists independently of other entity types. (Some data modeling software, in fact, use the term *independent entity*.)
- **Examples** include STUDENT, EMPLOYEE, AUTOMOBILE, and COURSE.
- Instances of a strong entity type always have a unique characteristic (called an *identifier*)—that is, an attribute or a combination of attributes that uniquely distinguish each occurrence of that entity.
- In contrast, a **weak entity type** is an entity type whose existence depends on some other entity type.
  (Some data modeling software, in fact, use the term *dependent entity*.)
- A weak entity type has no business meaning in an E-R diagram without the entity on which it depends. The entity type on which the weak entity type depends is called the **identifying owner** (or simply *owner* for short).
- EMPLOYEE is a strong entity type with identifier Employee ID.
- DEPENDENT is a weak entity type, as indicated by the double-lined rectangle.



- The relationship between a weak entity type and its owner is called an **identifying relationship.**
- The attribute Dependent Name serves as a *partial* identifier.

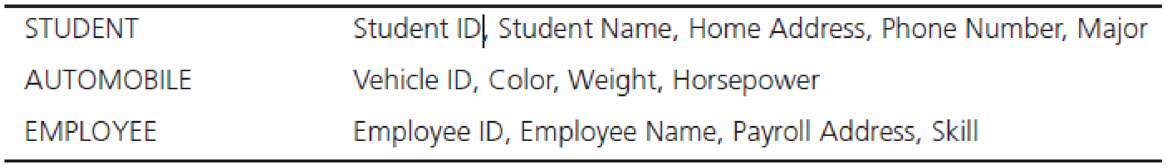
NAMING AND DEFINING ENTITY TYPES

There are a few special guidelines for *naming* entity types, which follow:

- An entity type name is a singular noun (such as CUSTOMER, STUDENT, or AUTOMOBILE)
- An entity type name should be specific to the organization.
- An entity type name should be concise, using as few words as possible.
- An abbreviation, or a short name, should be specified for each entity type name
- The name used for the same entity type should be the same on all E-R diagrams on which the entity type appears.

Attributes

- Each entity type has a set of attributes associated with it.
- An **attribute** is a property or characteristic of an entity type that is of interest to the organization.
- An attribute has a noun name.
- Following are some typical entity types and their associated attributes:

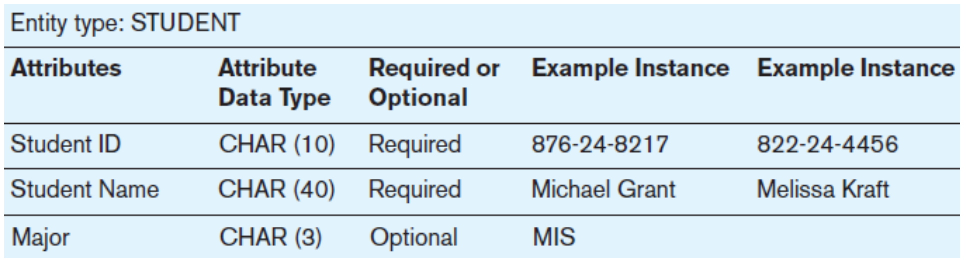


STUDENT	Student ID, Student Name, Home Address, Phone Number, Major
AUTOMOBILE	Vehicle ID, Color, Weight, Horsepower
EMPLOYEE	Employee ID, Employee Name, Payroll Address, Skill

- In naming attributes, we use an initial capital letter followed by lowercase letters.
- If an attribute name consists of more than one words, we use a space between the words and we start each word with a capital letter.
- An attribute is associated with exactly one entity or relationship.

REQUIRED VERSUS OPTIONAL ATTRIBUTES

- Each entity (or instance of an entity type) potentially has a value associated with each of the attributes of that entity type.
- An attribute that must be present for each entity instance is called a **required attribute**, whereas an attribute that may not have a value is called an **optional attribute**.



Entity type: STUDENT

Attributes	Attribute Data Type	Required or Optional	Example Instance	Example Instance
Student ID	CHAR (10)	Required	876-24-8217	822-24-4456
Student Name	CHAR (40)	Required	Michael Grant	Melissa Kraft
Major	CHAR (3)	Optional	MIS	

SIMPLE VERSUS COMPOSITE ATTRIBUTES

- An attribute that cannot be broken down into smaller components that are meaningful to the organization called Simple Attributes.
- Some attributes can be broken down into meaningful component parts (detailed attributes) called Composite Attributes.
- Address, which can usually be broken down into the following component attributes: Street Address, City, State, and Postal Code.

- A **composite attribute** is an attribute, such as Address, that has meaningful component parts, which are more detailed attributes.

SINGLE-VALUED VERSUS MULTIVALUED ATTRIBUTES

- For each entity instance, each of the attributes in the diagram has one value.-Single valued
- It frequently happens that there is an attribute that may have more than one value for a given instance.-Multi valued
- **Example,** the EMPLOYEE entity has an attribute named Skill, whose values record the skill (or skills) for that employee.
- Some employees may have more than one skill, such as PHP Programmer and C++ Programmer.
- A **Multivalued attribute** is an attribute that may take on more than one value for a given entity (or relationship) instance.

STORED VERSUS DERIVED ATTRIBUTES

- Some attribute values that are of interest to users can be calculated or derived from other related attribute values that are stored in the database.
- For example, suppose that for an organization, the EMPLOYEE entity type has a Date Employed attribute.
- If users need to know how many years a person has been employed, that value can be calculated using Date Employed and today's date.
- A **derived attribute** is an attribute whose values can be calculated from related attribute values

IDENTIFIER ATTRIBUTE

- **IDENTIFIER ATTRIBUTE** An **identifier** is an attribute (or combination of attributes) whose value distinguishes individual instances of an entity type.
- No two instances of the entity type may have the same value for the identifier attribute.

NAMING AND DEFINING ATTRIBUTES

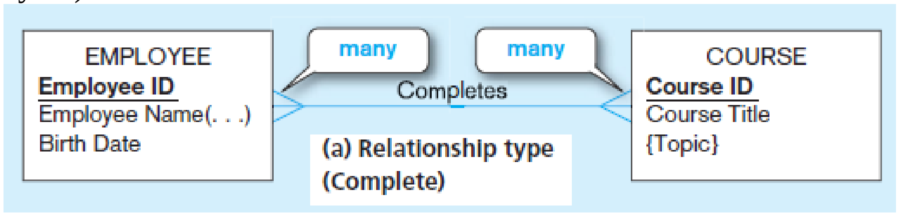
There are a few special guidelines for *naming* attributes, which follow:

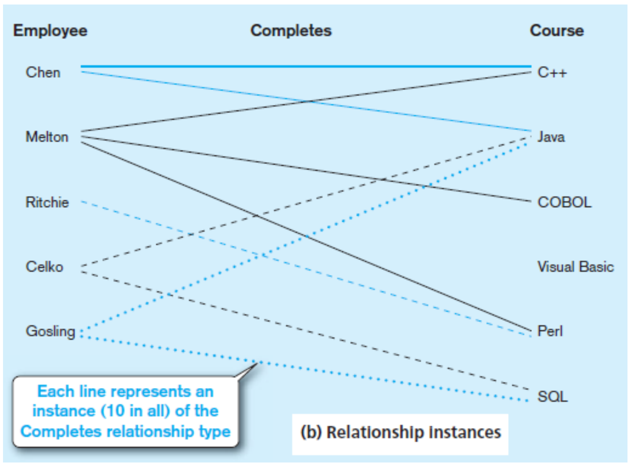
- An attribute name is a *singular noun or noun phrase* (such as Customer ID, Age, or Major).
- An attribute name should be *unique*. No two attributes of the same entity type may have the same name
- *Each attribute name should follow a standard format*

4. <u>Explain in detail modeling relationship and their basic concepts.</u>

- Relationships are the glue that holds together various components of an E-R model.
- A *relationship* is an association representing an interaction among the instances of one or more entity types that is of interest to the organization.
- Thus, a relationship has a verb phrase name.
- Relationships and their characteristics (degree and cardinality) represent business rules.
- To understand relationships more clearly, we must distinguish between relationship types and relationship instances.

- To illustrate, consider the entity types EMPLOYEE and COURSE, where COURSE represents training courses that may be taken by employees.
- To track courses that have been completed by particular employees, we define a relationship called Completes between the two entity types,
- This is a many-to-many relationship, because each employee may complete any number of courses (zero, one, or many courses), whereas a given course may be completed by any number of employees (nobody, one employee, many employees).




Employee
Completes
Course
Chen
Melton
Ritchie
Celko
Gosling
C++
Java
COBOL
Visual Basic
Perl
SQL
Each line represents an instance (10 in all) of the Completes relationship type
(b) Relationship instances

- In the below Figure the employee Melton has completed three courses (C++, COBOL, and Perl).
- The SQL course has been completed by two employees (Celko and Gosling), and the Visual Basic course has not been completed by anyone.

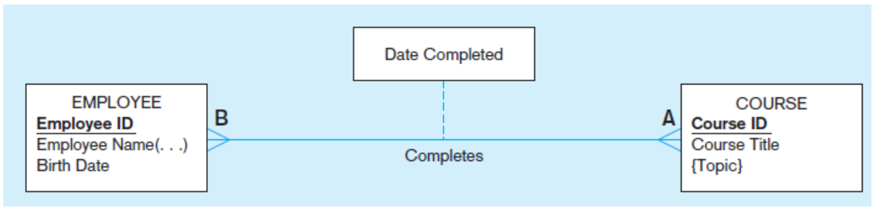
BASIC CONCEPTS AND DEFINITIONS IN RELATIONSHIPS

- A **relationship type** is a meaningful association between (or among) entity types.
- The phrase *meaningful association* implies that the relationship allows us to answer questions that could not be answered given only the entity types.
- A relationship type is denoted by a line labeled with the name of the relationship, as in the example shown in the above Figure.
- A **relationship instance** is an association between (or among) entity instances,
  where each relationship instance associates exactly one entity instance from each participating entity type.
- Example, in the above figure each of the 10 lines in the figure represents a relationship instance between one employee and one course, indicating that the employee has completed that course.
- For example, the line between Employee Ritchie to Course Perl is one relationship instance.

ATTRIBUTES ON RELATIONSHIPS

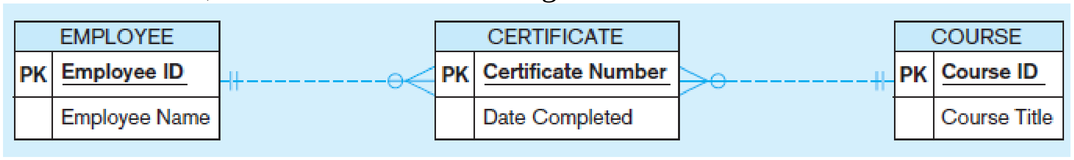
- Example, suppose the organization wishes to record the date (month andyear) when an employee completes each course.
- This attribute is named Date Completed.
  In the above diagram Date Completed has not been associated with either the EMPLOYEE or COURSE entity.
- That is because Date Completed is a property of the relationship Completes, rather than a property of either entity.
- In other words, for each instance of the relationship Completes, there is a value for Date Completed.
- One such instance (for example) shows that the employee named Melton completed the course titled C++ in 06/2009.
- A revised version of the ERD for this example is shown in the below Figure.

Attribute on a relationship



ASSOCIATIVE ENTITIES

- An **associative entity** is an entity type that associates the instances of one or more entity types and contains attributes that are peculiar to the relationship between those entity instances.
- The associative entity CERTIFICATE is represented with the rectangle with rounded corners, as shown in the below Figure



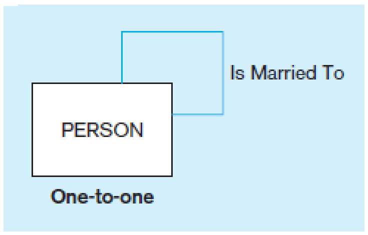
DEGREE OF A RELATIONSHIP

- The **degree** of a relationship is the number of entity types that participate in that relationship.
- The three most common relationship degrees in E-R models are unary (degree 1), binary (degree 2), and ternary (degree 3).

UNARY RELATIONSHIP

- A **unary relationship** is a relationship between the instances of a *single* entity type.
  (Unary relationships are also called *recursive relationships*.)
- **Example:** Is Married To is shown as a one-to-one relationship between instances of the PERSON entity type.

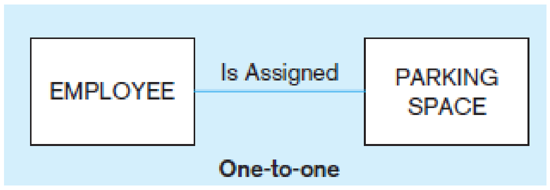
Unary relationships



BINARY RELATIONSHIP

- A **binary relationship** is a relationship between the instances of two entity types and is the most common type of relationship encountered in data modeling.
- **Example:** The first (one-to-one) indicates that an employee is assigned one parking place, and that each parking place is assigned to one employee.

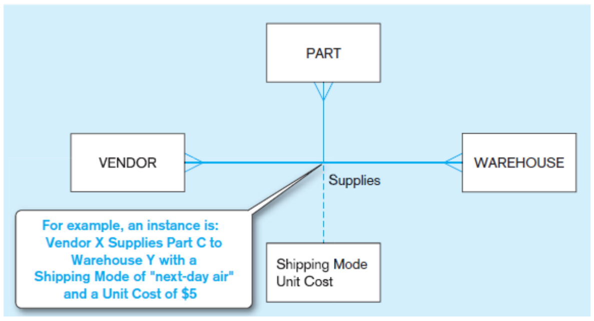
Binary relationships



TERNARY RELATIONSHIP

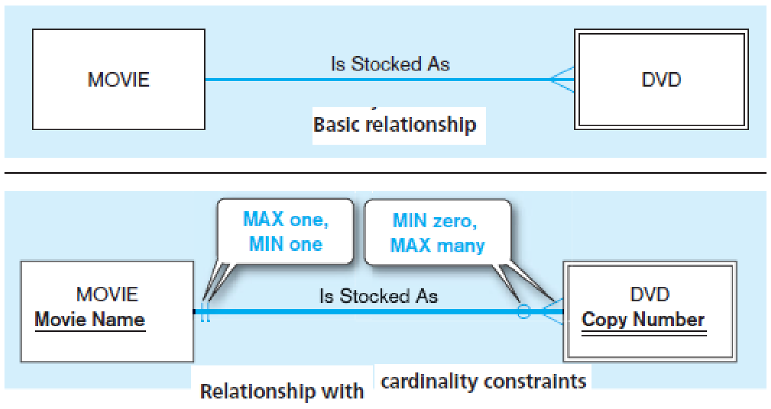
- A **ternary relationship** is a *simultaneous* relationship among the instances of three entity types.
- **Example:** vendors can supply various parts to warehouses.
- The relationship Supplies is used to record the specific parts that are supplied by a given vendor to a particular warehouse.
- Thus there are three entity types: VENDOR, PART, and WAREHOUSE.
- There are two attributes on the relationship Supplies: Shipping Mode and Unit Cost.
- One instance of Supplies might record the fact that vendor X can ship part C to warehouse Y, that the shipping mode is next-day air, and that the cost is $5 per unit.

Ternary relationship


PART
VENDOR
WAREHOUSE
Supplies
For example, an instance is: Vendor X Supplies Part C to Warehouse Y with a Shipping Mode of "next-day air" and a Unit Cost of $5
Shipping Mode
Unit Cost

CARDINALITY CONSTRAINTS

- There is one more important data modeling notation for representing common and
  important business rules.
- Suppose there are two entity types, A and B that are connected by a relationship.
- A **cardinality constraint** specifies the number of instances of entity B that can (or must) be associated with each instance of entity A.
  Example:
- Consider a video store that rents DVDs of movies.
- The store may stock more than one DVD for each movie, this is intuitively a one-to-many relationship
- It is also true that the store may not have any DVDs of a given movie in stock at a particular time (e.g., all copies may be checked out).



MINIMUM CARDINALITY

- The **minimum cardinality** of a relationship is the minimum number of instances of entity B that may be associated with each instance of entity A.

MAXIMUM CARDINALITY

- The **maximum cardinality** of a relationship is the maximum number of instances of entity B that may be associated with each instance of entity A.

---

5. <u>Explain in detail the Relational Data Model with suitable examples.</u>

RELATIONAL DATA MODEL

- The relational data model was first introduced in 1970 by E. F. Codd, then of IBM
- (Codd, 1970).
- Two early research projects were launched to prove the feasibility of the relational model and to develop prototype systems.
- The first of these, at IBM's San Jose Research Laboratory, led to the development of System R.
- The second, at the University of California at Berkeley, led to the development of an academically oriented RDBMS. Commercial
- RDBMS products from numerous vendors started to appear about 1980.
- Today RDBMSs have become the dominant technology for database management, and there are literally hundreds of RDBMS products for computers ranging from smart phones and personal computers to mainframes.

BASIC DEFINITIONS

- The relational data model represents data in the form of tables.
- The relational model is based on mathematical theory and therefore has a solid theoretical foundation.
- ***The relational data model consists of the following three components :***
  - ***Data structure*** Data are organized in the form of tables, with rows and columns.
  - ***Data manipulation*** Powerful operations (using the SQL language) are used to manipulate data stored in the relations.
  - ***Data integrity*** The model includes mechanisms to specify business rules that
    maintain the integrity of data when they are manipulated.

RELATIONAL DATA STRUCTURE

- A **relation** is a named, two-dimensional table of data.

- Each relation (or table) consists of a set of named columns and an arbitrary number of unnamed rows.
- An attribute, consistent with its definition is a named column of a relation.
- Each row of a relation corresponds to a record that contains data (attribute) values for a single entity.
- Below Figure shows an example of a relation named EMPLOYEE1 illustrate the structure of the EMPLOYEE1 relation
- This relation contains the following attributes describing employees: EmpID, Name, DeptName, and Salary.
- The five rows of the table correspond to five employees.
- We could delete all of the rows shown in Figure, and the EMPLOYEE1 relation would still exist.
- In other words, the below Figure is an instance of the EMPLOYEE1 relation.
- The structure of a relation can be expressed by using a shorthand notation in which the name of the relation is followed (in parentheses) by the names of the attributes in that relation.
- For EMPLOYEE1 we would have

EMPLOYEE1(EmpID, Name, DeptName, Salary)

RELATIONAL KEYS

- We must be able to store and retrieve a row of data in a relation, based on the data values stored in that row.
- To achieve this goal, every relation must have a primary key.
- A **primary key** is an attribute or a combination of attributes that uniquely identifies
- each row in a relation.
- We designate a primary key by underlining the attribute name(s).
- **Example:** The primary key for the relation EMPLOYEE1 is EmpID.
- That this attribute is underlined in the above Figure.
- A **composite key** is a primary key that consists of more than one attribute.
- Often we must represent the relationship between two tables or relations.
- This is accomplished through the use of foreign keys.
- A **foreign key** is an attribute (possibly composite) in a relation that serves as the primary key of another relation.
- **Example:** Consider the relations EMPLOYEE1 and DEPARTMENT:

EMPLOYEE1(EmpID, Name, DeptName, Salary)
DEPARTMENT(DeptName, Location, Fax)

PROPERTIES OF RELATIONS

- We have defined relations as two-dimensional tables of data.
- However, not all tables are relations.
- Relations have several properties that distinguish them from non-relational tables.

Properties:

- Each relation (or table) in a database has a unique name.
- An entry at the intersection of each row and column is atomic (or single valued).

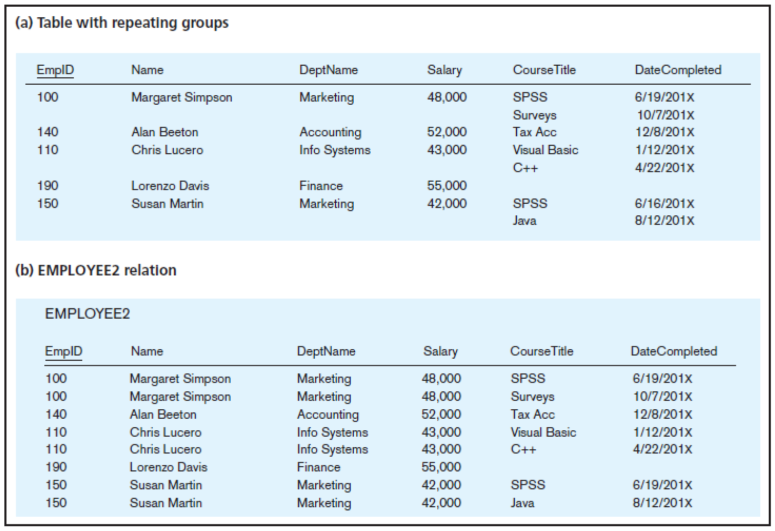
        - There can be only one value associated with each attribute on a specific row of a table; no multivalued attributes are allowed in a relation.
    - Each row is unique; no two rows in a relation can be identical.
    - Each attribute (or column) within a table has a unique name.
    - The sequence of columns (left to right) is insignificant.
        - The order of the columns in a relation can be changed without changing the meaning or use of the relation.
    - The sequence of rows (top to bottom) is insignificant.
        - As with columns, the order of the rows of a relation may be changed or stored in any sequence.

REMOVING MULTIVALUED ATTRIBUTES FROM TABLES

- No multivalued attributes are allowed in a relation.
- Thus, a table that contains one or more multivalued attributes is not a relation.
- For example, Figure below shows the employee data from the EMPLOYEE1 relation extended to include courses that may have been taken by those employees.
- Because a given employee may have taken more than one course, the attributes CourseTitle and DateCompleted are multivalued attributes.
- For example, the employee with EmpID 100 has taken two courses.
- If an employee has not taken any courses, the CourseTitle and DateCompleted attribute values are null.
- The multivalued attributes are eliminated in the below Figure by filling the relevant data values into the previously vacant cells of table.
- Table has only single-valued attributes
- The name EMPLOYEE2 is given to this relation to distinguish it from EMPLOYEE1.

SAMPLE DATABASE

Eliminating multivalued attributes



(a) Table with repeating groups

EmpID	Name	DeptName	Salary	CourseTitle	DateCompleted
100	Margaret Simpson	Marketing	48,000	SPSS	6/19/201X
				Surveys	10/7/201X
140	Alan Beeton	Accounting	52,000	Tax Acc	12/8/201X
110	Chris Lucero	Info Systems	43,000	Visual Basic	1/12/201X
				C++	4/22/201X
190	Lorenzo Davis	Finance	55,000		
150	Susan Martin	Marketing	42,000	SPSS	6/16/201X
				Java	8/12/201X

(b) EMPLOYEE2 relation

EMPLOYEE2

EmpID	Name	DeptName	Salary	CourseTitle	DateCompleted
100	Margaret Simpson	Marketing	48,000	SPSS	6/19/201X
100	Margaret Simpson	Marketing	48,000	Surveys	10/7/201X
140	Alan Beeton	Accounting	52,000	Tax Acc	12/8/201X
110	Chris Lucero	Info Systems	43,000	Visual Basic	1/12/201X
110	Chris Lucero	Info Systems	43,000	C++	4/22/201X
190	Lorenzo Davis	Finance	55,000		
150	Susan Martin	Marketing	42,000	SPSS	6/19/201X
150	Susan Martin	Marketing	42,000	Java	8/12/201X

6. <u>Discuss the following</u>

a. <u>Relational Data Model Constraints</u>

INTEGRITY CONSTRAINTS

- The relational data model includes several types of constraints, or rules limiting acceptable values and actions, whose purpose is to facilitate maintaining the accuracy and integrity of data in the database.

- The ***major types of integrity constraints are domain constraints, entity integrity, and referential integrity***.

DOMAIN CONSTRAINTS

- All of the values that appear in a column of a relation must be from the same domain.
- A domain is the set of values that may be assigned to an attribute.
- A domain definition usually consists of the following components: domain name, meaning, data type, size (or length), and allowable values or allowable range (if applicable).
- Table below shows domain definitions for the domains associated with the attributes

Domain Definitions for INVOICE Attributes

Attribute	Domain Name	Description	Domain
CustomerID	Customer IDs	Set of all possible customer IDs	character: size 5
CustomerName	Customer Names	Set of all possible customer names	character: size 25
CustomerAddress	Customer Addresses	Set of all possible customer addresses	character: size 30

ENTITY INTEGRITY

- The entity integrity rule is designed to ensure that every relation has a primary key and that the data values for that primary key are all valid.
- In particular, it guarantees that every primary key attribute is non-null.
- In some cases, a particular attribute cannot be assigned a data value.
- There are two situations in which this is likely to occur:
  - Either there is no applicable data value or the applicable data value is not known when values are assigned.
  - Example, that you fill out an employment form that has a space reserved for a fax number.
  - If you have no fax number, you leave this space empty because it does not apply to you.
- The relational data model allows us to assign a null value to an attribute in the just
  described situations.
- A **null** is a value that may be assigned to an attribute when no other value applies or when the applicable value is unknown.
- In reality, a null is not a value but rather it indicates the absence of a value.

REFERENTIAL INTEGRITY

- In the relational data model, associations between tables are defined through the use of foreign keys.
- A **referential integrity constraint** is a rule that maintains consistency among the rows of two relations.
- The rule states that if there is a foreign key in one relation, either each foreign key value must match a primary key value in another relation or the foreign key value must be null.

b. <u>STORED PROCEDURES</u>

STORED PROCEDURES

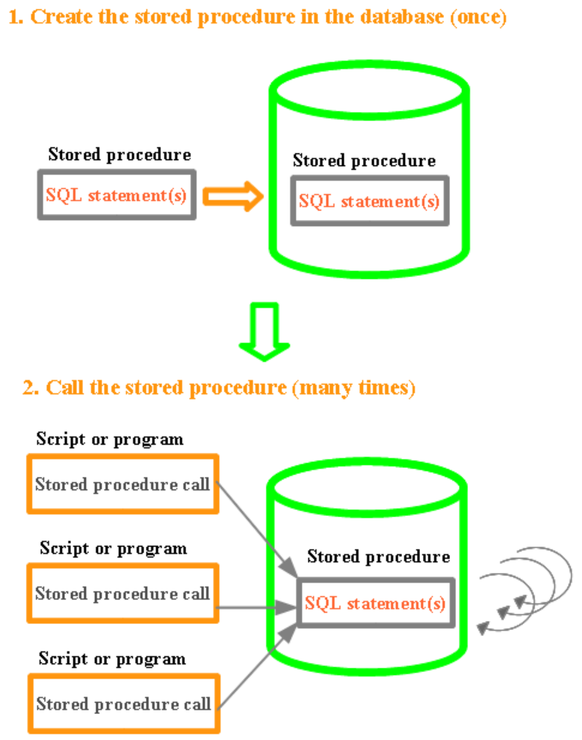
- Stored procedures are modules of code that implement application logic and are included on the database server.
- ***Stored procedures have the following advantages:***
  - Performance improves for compiled SQL statements.
  - Network traffic decreases as processing moves from the client to the server.

- Security improves if the stored procedure rather than the data is accessed and code is moved to the server, away from direct end-user access.
- Data integrity improves as multiple applications access the same stored procedure.
- Stored procedures result in a thinner client and a fatter database server.

- Writing stored procedures can also take more time than using Visual Basic or Java to create an application.
- Also, the proprietary nature of stored procedures reduces their portability and may make it difficult to change DBMSs without having to rewrite the stored procedures.
- However, using stored procedures appropriately, can lead to more efficient processing of database code.

USING STORED PROCEDURES WITH SQL SERVER

- A stored procedure executes one or more SQL statements.
- ***In order to use a stored procedure:***
  - Create a stored procedure containing the SQL statement(s). The DBMS will store it (them) in the database.
  - When a program needs to execute the SQL statement(s), it calls the stored procedure. The DBMS executes the corresponding actions in the database.
- **The diagram below illustrates these two steps:**

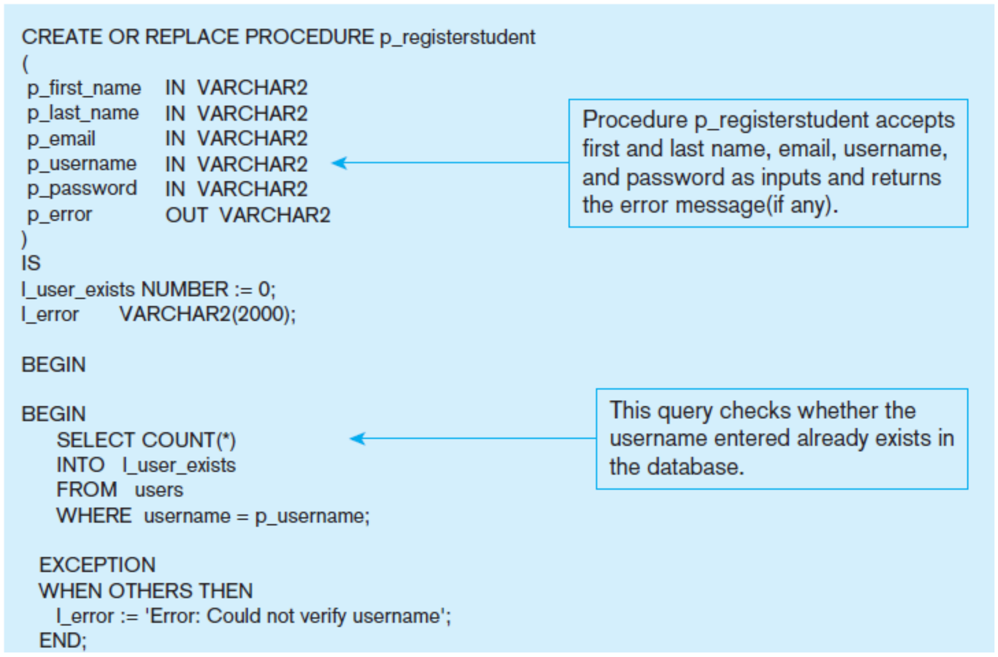


This technique commonly used in a professional environment:

- Creating a simple stored procedure.
- Stored procedure with input parameter validation.
- Stored procedure with output parameters.
- Granting rights to users to execute a stored procedure, deleting a stored procedure.

- Figure.1 below shows an example of a stored procedure written in Oracle's PL/SQL that is intended to check whether a user name already exists in the database.

Sample Oracle PL/SQL stored procedure

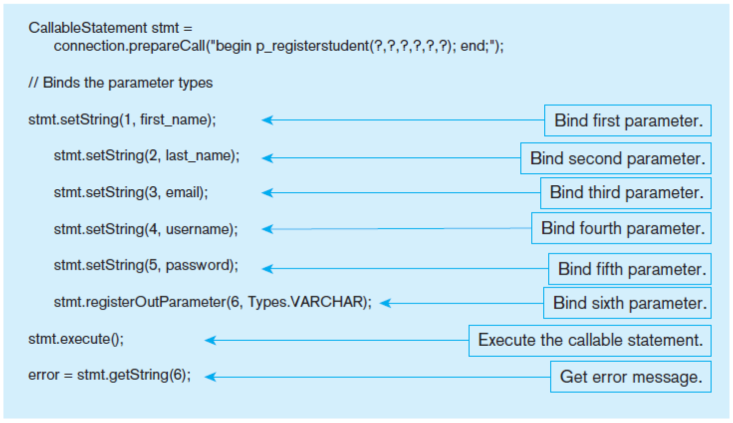


```
CREATE OR REPLACE PROCEDURE p_registerstudent
(
 p_first_name   IN  VARCHAR2
 p_last_name    IN  VARCHAR2
 p_email        IN  VARCHAR2
 p_username     IN  VARCHAR2
 p_password     IN  VARCHAR2
 p_error        OUT  VARCHAR2
)
IS
l_user_exists NUMBER := 0;
l_error      VARCHAR2(2000);

BEGIN

BEGIN
    SELECT COUNT(*)
    INTO  l_user_exists
    FROM  users
    WHERE  username = p_username;

  EXCEPTION
  WHEN OTHERS THEN
    l_error := 'Error: Could not verify username';
  END;
```

Procedure p_registerstudent accepts first and last name, email, username, and password as inputs and returns the error message(if any).

This query checks whether the username entered already exists in the database.

(b) Sample Java code for invoking the Oracle PL/SQL stored procedure



```
CallableStatement stmt =
    connection.prepareCall("begin p_registerstudent(?,?,?,?,?,?); end;");

// Binds the parameter types

stmt.setString(1, first_name);

    stmt.setString(2, last_name);

    stmt.setString(3, email);

    stmt.setString(4, username);

    stmt.setString(5, password);

    stmt.registerOutParameter(6, Types.VARCHAR);

stmt.execute();

error = stmt.getString(6);
```

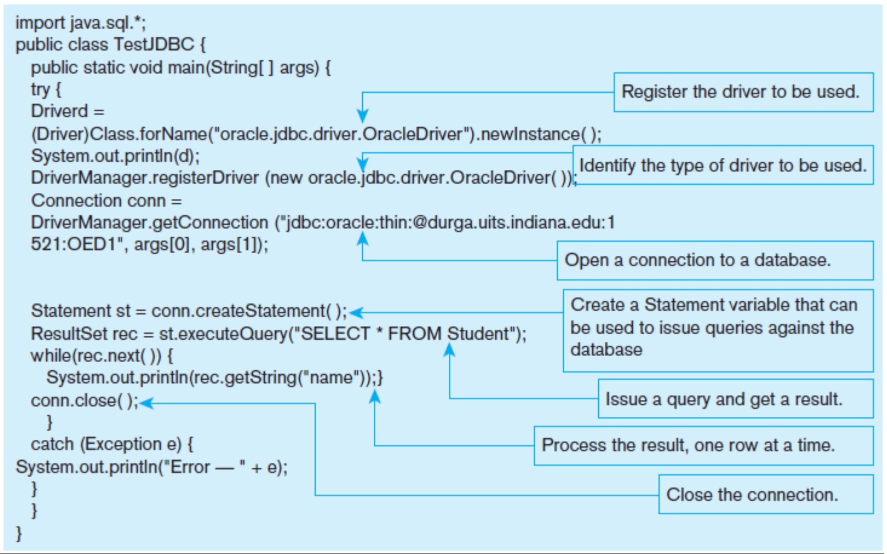
7. <u>With an example discuss how to connect to a database from a Java application.</u>

- Most two-tier applications are written in a programming language such as Java, VB.NET, or C#.
- Connecting an application written in a common programming language, such as Java, VB.NET, or C#, to a database is achieved through the use of special software called ***database-oriented middleware***.
- Middleware is often referred to as the glue that holds together client/server applications.
- It is a term that is commonly used to describe any software component between the PC client and the relational database in *n*-tier architectures.
- Simply put, **middleware** is any of several classes of software that allow an application to interoperate with other software without requiring the user to understand and code the low-level operations required to achieve interoperability.
- The database oriented middleware needed to connect an application to a database consists of two parts:

- An **application programming interface (API)** and a database driver to connect to a specific type database (e.g., SQL Server or Oracle).
    - The most common APIs are **Open Database Connectivity (ODBC)** and ADO.NET for the Microsoft platform (VB.NET and C#) and
- **Java Database Connectivity (JDBC)** for use with Java programs.
- ***The basic steps for accessing a database from an application:***
    1. Identify and register a database driver.
    2. Open a connection to a database.
    3. Execute a query against the database.
    4. Process the results of the query.
    5. Repeat steps 3–4 as necessary.
    6. Close the connection to the database.

A JAVA EXAMPLE

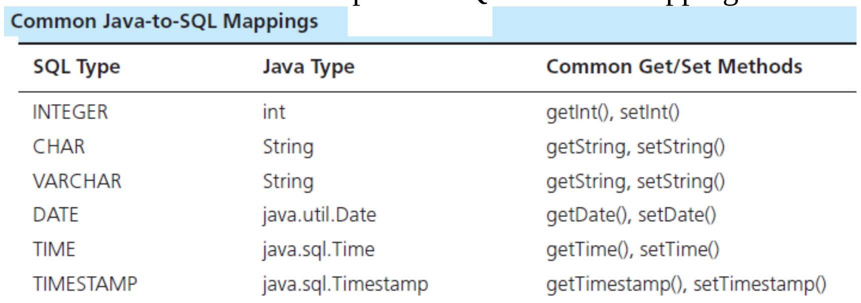
- An example of how to connect to a database from a Java application
- This Java application is actually connecting to the database
- Its purpose is to retrieve and print the names of all students in the Student table.
- In this example, the Java program is using the JDBC API and an Oracle thin driver to access the Oracle database.
- Running an SQL SELECT query requires us to capture the data inside an object that can appropriately handle the tabular data.
- **JDBC provides two key mechanisms for this:**
    - The ResultSet and RowSet objects.
- The ResultSet object has a mechanism, called the cursor that points to its current row of data.
- When the ResultSet object is first initialized, the cursor is positioned before the first row.
- This is why we need to first call the next() method before retrieving data.
- The ResultSet object is used to loop through and process each row of data and retrieve the column values that we want to access.
- In this case, we access the value in the name column using the rec.getString method, which is a part of the JDBC API.
- For each of the common database types, there is a corresponding *get* and *set* method that allows for retrieval and storage of data in the database.



```
import java.sql.*;
public class TestJDBC {
  public static void main(String[ ] args) {
  try {
  Driverd =
  (Driver)Class.forName("oracle.jdbc.driver.OracleDriver").newInstance( );
  System.out.println(d);
  DriverManager.registerDriver (new oracle.jdbc.driver.OracleDriver( ));
  Connection conn =
  DriverManager.getConnection ("jdbc:oracle:thin:@durga.uits.indiana.edu:1
  521:OED1", args[0], args[1]);


  Statement st = conn.createStatement( );
  ResultSet rec = st.executeQuery("SELECT * FROM Student");
  while(rec.next( )) {
    System.out.println(rec.getString("name"));}
  conn.close( );
    }
  catch (Exception e) {
System.out.println("Error — " + e);
  }
  }
}
```

Register the driver to be used.
Identify the type of driver to be used.
Open a connection to a database.
Create a Statement variable that can be used to issue queries against the database
Issue a query and get a result.
Process the result, one row at a time.
Close the connection.

- Table provides some common examples of SQL-to- Java mappings.



Common Java-to-SQL Mappings

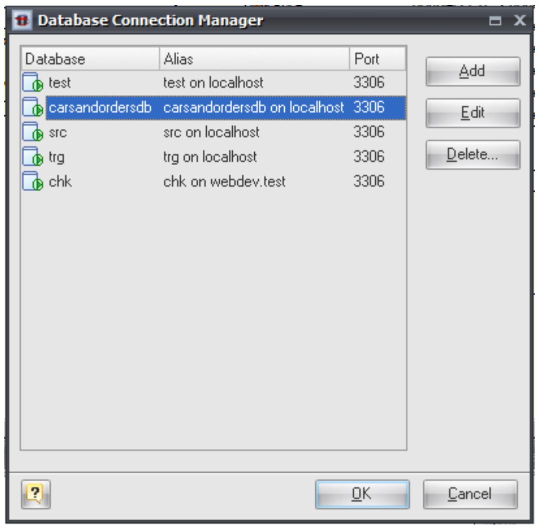
SQL Type	Java Type	Common Get/Set Methods
INTEGER	int	getInt(), setInt()
CHAR	String	getString, setString()
VARCHAR	String	getString, setString()
DATE	java.util.Date	getDate(), setDate()
TIME	java.sql.Time	getTime(), setTime()
TIMESTAMP	java.sql.Timestamp	getTimestamp(), setTimestamp()

- It is important to note that while the ResultSet object maintains an active connection to the database, depending on the size of the table, the entire table (i.e., the result of the query) may or may not actually be in memory on the client machine.
- How and when data are transferred between the database and client is handled by the Oracle driver.
- By default, a ResultSet object is read-only and can only be traversed in one direction (forward).
- However, advanced versions of the ResultSet object allow scrolling in both directions and can be updateable as well.

8. <u>Explain the following concepts</u>

a. <u>Database connection manager</u>

- Database Connection Manager is a useful tool, which allows you to connect to a local or remote MySQL database for Database Generation, Reverse Engineering, Database Modification, Diagram Synchronization, or running SQL scripts.
- Use Connect (connect) item on Database tab of the Ribbon to open Database Connection Manager.
- Database Connection Manager uses connection profiles, which allow you to set all the connection parameters for a specific database only once, and quickly connect to this database later.



Database Connection Manager

Database	Alias	Port
test	test on localhost	3306
carsandordersdb	carsandordersdb on localhost	3306
src	src on localhost	3306
trg	trg on localhost	3306
chk	chk on webdev.test	3306

- The dialog contains a list of available connection profiles. To connect to a database, select one of the defined profiles in the list, and click **OK**. Connection is assigned to your current diagram, so that each opened diagram could have its own connection.

- To add a profile to the list, click the **Add** button and enter the connection properties using **Connection Profile Editor** launched.
- Click **Edit** to change the profile connection properties.
- To remove a profile from the list, click the **Delete** button.

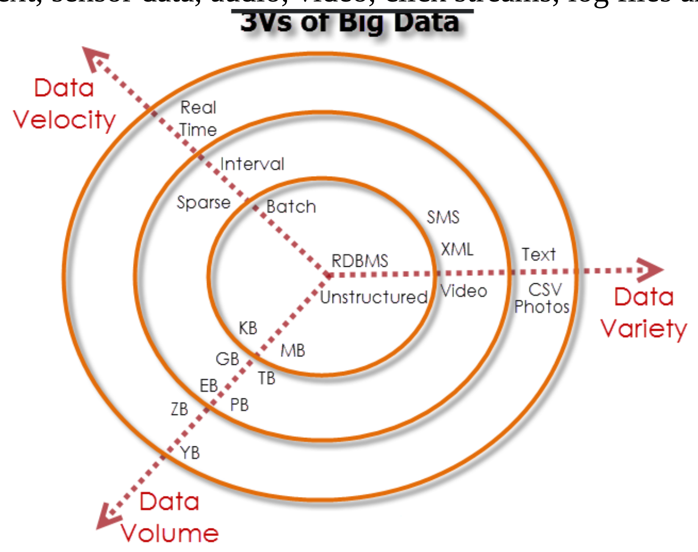
b. **Three V's and four core capabilities of big data.**

WHAT IS BIG DATA?

- Every day we create 2.5 quintillion bytes of data—in fact, 90 percent of the data in the world today has been created in the last two years alone.
- This data comes from a wide variety of sources: sensors used to gather climate information, posts to social media sites, digital pictures and videos, purchase transaction records and cell phone GPS signals, to name a few.

BIG DATA SPANS FOUR DIMENSIONS:

- **Volume:** Enterprises are awash with ever-growing data of all types, easily amassing terabytes and even petabytes of information.
- **Velocity:** For time-sensitive processes such as catching fraud, big data must be analyzed as it streams into the enterprise to maximize its business value.
- **Variety:** Big data extends beyond structured data to include unstructured data of all varieties: text, sensor data, audio, video, click streams, log files and more.



THE IBM BIG DATA PLATFORM

- IBM has developed a comprehensive, integrated and industrial strength big data platform that lets you address the full spectrum of big data business challenges.

The core capabilities of the platform include:

Hadoop-based analytics:

- Enables distributed processing of large data sets across commodity server clusters

Stream computing:

- Enables continuous analysis of massive volumes of streaming data with sub-millisecond response times

Data warehousing:

- Delivers deep operational insight with advanced in-database analytics

Information integration and governance:

- Allows you to understand, cleanse, transform, govern and deliver trusted information for critical business initiatives

Visualization and discovery:
- Helps users explore large, complex data sets

Application development:
- Streamlines the process of developing big data applications

Systems management:
- Monitors and manages big data systems for secure and optimized performance

Accelerators:
- Speed time-to-value with analytical and industry-specific modules

---

9. <u>Explain the following Big Data systems</u>

a. <u>NoSQL</u>

WHAT IS NOSQL?

- NoSQL is an approach to databases that represents a shift away from traditional relational database management systems (RDBMS).
- To define NoSQL, it is helpful to start by describing SQL, which is a query language used by RDBMS.
- Relational databases rely on tables, columns, rows, or schemas to organize and retrieve data.
- In contrast, NoSQL databases do not rely on these structures and use more flexible data models.
- NoSQL can mean "not SQL" or "not only SQL." As RDBMS have increasingly failed to meet the performance, scalability, and flexibility needs that next-generation, data-intensive applications require, NoSQL databases have been adopted by mainstream enterprises.
- NoSQL is particularly useful for storing unstructured data, which is growing far more rapidly than structured data and does not fit the relational schemas of RDBMS.
- Common types of unstructured data include: user and session data; chat, messaging, and log data; time series data such as IoT and device data; and large objects such as video and images.

TYPES OF NOSQL DATABASES

- Several different varieties of NoSQL databases have been created to support specific needs and use cases. These fall into four main categories:
- **Key-value data stores:** Key-value NoSQL databases emphasize simplicity and are very useful in accelerating an application to support high-speed read and write processing of non-transactional data. Stored values can be any type of binary object (text, video, JSON document, etc.) and are accessed via a key. The application has complete control over what is stored in the value, making this the most flexible NoSQL model. Data is partitioned and replicated across a cluster to get scalability and availability. For this reason, key value stores often do not support transactions. However, they are highly effective at scaling applications that deal with high-velocity, non-transactional data.
- **Document stores:** Document databases typically store self-describing JSON, XML, and BSON documents. They are similar to key-value stores, but in this case, a value is a single document that stores all data related to a specific key. Popular fields in the document can be indexed to provide fast retrieval without knowing the key. Each document can have the same or a different structure.
- **Wide-column stores:** Wide-column NoSQL databases store data in tables with rows and columns similar to RDBMS, but names and formats of columns can vary from row

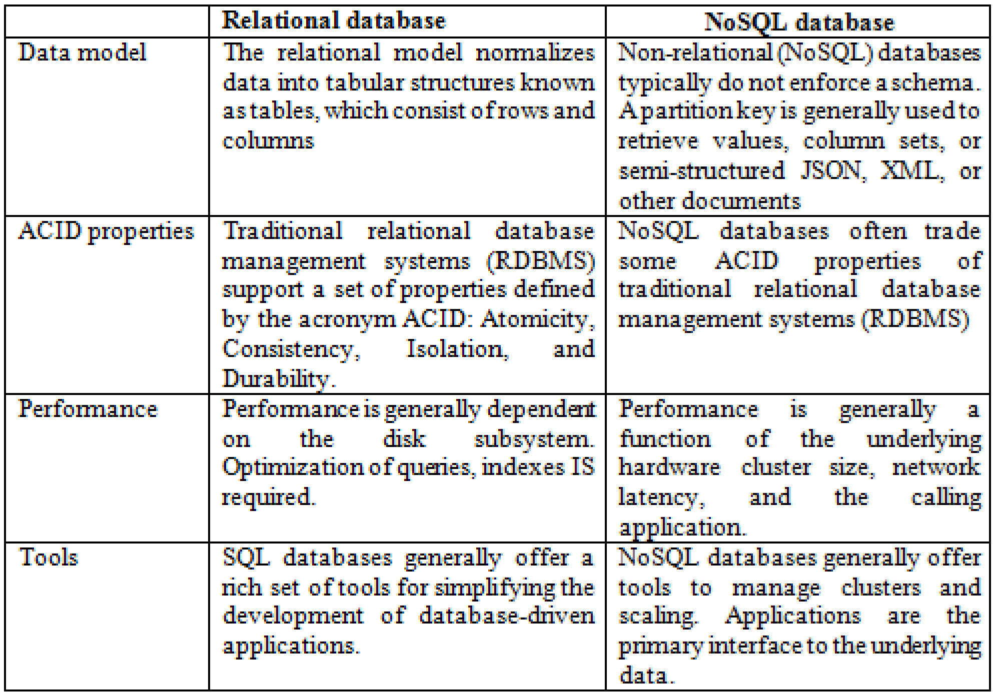
to row across the table. Wide-column databases group columns of related data together. A query can retrieve related data in a single operation because only the columns associated with the query are retrieved. In an RDBMS, the data would be in different rows stored in different places on disk, requiring multiple disk operations for retrieval.

- **Graph stores:** A graph database uses graph structures to store, map, and query relationships. They provide index-free adjacency, so that adjacent elements are linked together without using an index.
- Multi-modal databases leverage some combination of the four types described above and therefore can support a wider range of applications.

BENEFITS OF NOSQL

- NoSQL databases offer enterprises important advantages over traditional RDBMS, including:
- **Scalability:** NoSQL databases use a horizontal scale-out methodology that makes it easy to add or reduce capacity quickly and non-disruptively with commodity hardware.
- **Performance:** By simply adding commodity resources, enterprises can increase performance with NoSQL databases. This enables organizations to continue to deliver reliably fast user experiences with a predictable return on investment for adding resources—again, without the overhead associated with manual sharding.
- **High Availability:** NoSQL databases are generally designed to ensure high availability and avoid the complexity that comes with a typical RDBMS architecture that relies on primary and secondary nodes.
- **Global Availability:** By automatically replicating data across multiple servers, data centers, or cloud resources, distributed NoSQL databases can minimize latency and ensure a consistent application experience wherever users are located.
- **Flexible Data Modeling:** NoSQL offers the ability to implement flexible and fluid data models. Application developers can leverage the data types and query options that are the most natural fit to the specific application use case rather than those that fit the database schema.

COMPARISON OF RELATIONAL DATABASE AND NoSQL DATABASE



	Relational database	**NoSQL database**
Data model	The relational model normalizes data into tabular structures known as tables, which consist of rows and columns	Non-relational (NoSQL) databases typically do not enforce a schema. A partition key is generally used to retrieve values, column sets, or semi-structured JSON, XML, or other documents
ACID properties	Traditional relational database management systems (RDBMS) support a set of properties defined by the acronym ACID: Atomicity, Consistency, Isolation, and Durability.	NoSQL databases often trade some ACID properties of traditional relational database management systems (RDBMS)
Performance	Performance is generally dependent on the disk subsystem. Optimization of queries, indexes IS required.	Performance is generally a function of the underlying hardware cluster size, network latency, and the calling application.
Tools	SQL databases generally offer a rich set of tools for simplifying the development of database-driven applications.	NoSQL databases generally offer tools to manage clusters and scaling. Applications are the primary interface to the underlying data.

b. Hadoop HDFS

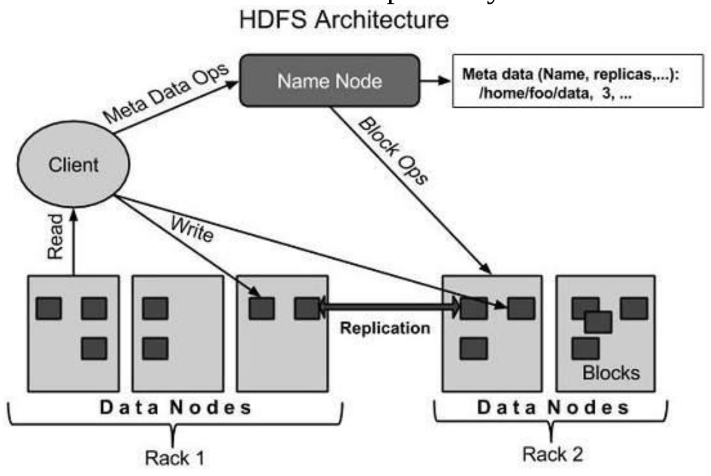
- Hadoop File System was developed using distributed file system design.
- It is run on commodity hardware.
- Unlike other distributed systems, HDFS is highly fault tolerant and designed using low-cost hardware.
- HDFS holds very large amount of data and provides easier access.
- To store such huge data, the files are stored across multiple machines.
- These files are stored in redundant fashion to rescue the system from possible data losses in case of failure.
- HDFS also makes applications available to parallel processing.

Features of HDFS

- It is suitable for the distributed storage and processing.
- Hadoop provides a command interface to interact with HDFS.
- The built-in servers of namenode and datanode help users to easily check the status of cluster.
- Streaming access to file system data.
- HDFS provides file permissions and authentication.

HDFS Architecture

Given below is the architecture of a Hadoop File System.



HDFS follows the master-slave architecture and it has the following elements.

Namenode

- The namenode is the commodity hardware that contains the GNU/Linux operating system and the namenode software. It is software that can be run on commodity hardware.
- The system having the namenode acts as the master server and it does the following tasks:
- Manages the file system namespace.
- Regulates client's access to files.
- It also executes file system operations such as renaming, closing, and opening files and directories.

Datanode

The datanode is a commodity hardware having the GNU/Linux operating system and datanode software. For every node (Commodity hardware/System) in a cluster, there will be a datanode. These nodes manage the data storage of their system.

- Datanodes perform read-write operations on the file systems, as per client request.

- They also perform operations such as block creation, deletion, and replication according to the instructions of the namenode.

Block

Generally the user data is stored in the files of HDFS. The file in a file system will be divided into one or more segments and/or stored in individual data nodes. These file segments are called as blocks. In other words, the minimum amount of data that HDFS can read or write is called a Block. The default block size is 64MB, but it can be increased as per the need to change in HDFS configuration.

Goals of HDFS

- **Fault detection and recovery**: Since HDFS includes a large number of commodity hardware, failure of components is frequent. Therefore HDFS should have mechanisms for quick and automatic fault detection and recovery.
- **Huge datasets**: HDFS should have hundreds of nodes per cluster to manage the applications having huge datasets.
- **Hardware at data**: A requested task can be done efficiently, when the computation takes place near the data. Especially where huge datasets are involved, it reduces the network traffic and increases the throughput.

HADOOPS HDFS OPERATIONS

Starting HDFS

Initially you have to format the configured HDFS file system, open namenode (HDFS server), and execute the following command.

```
$ hadoop namenode -format
```

After formatting the HDFS, start the distributed file system. The following command will start the namenode as well as the data nodes as cluster.

```
$ start-dfs.sh
```

Listing Files in HDFS

After loading the information in the server, we can find the list of files in a directory, status of a file, using ‘ls’. Given below is the syntax of ls that you can pass to a directory or a filename as an argument.

```
$ $HADOOP_HOME/bin/hadoop fs -ls <args>
```

Inserting Data into HDFS

Assume we have data in the file called file.txt in the local system which is ought to be saved

in the hdfs file system. Follow the steps given below to insert the required file in the Hadoop file system.

Step 1

You have to create an input directory.

```
$ $HADOOP_HOME/bin/hadoop fs -mkdir /user/input
```

Step 2

Transfer and store a data file from local systems to the Hadoop file system using the put command.

```
$ $HADOOP_HOME/bin/hadoop fs -put /home/file.txt /user/input
```

Step 3

You can verify the file using ls command.

```
$ $HADOOP_HOME/bin/hadoop fs -ls /user/input
```

Retrieving Data from HDFS

Assume we have a file in HDFS called outfile. Given below is a simple demonstration for retrieving the required file from the Hadoop file system.

Step 1

Initially, view the data from HDFS using cat command.

```
$ $HADOOP_HOME/bin/hadoop fs -cat /user/output/outfile
```

Step 2

Get the file from HDFS to the local file system using get command.

```
$ $HADOOP_HOME/bin/hadoop fs -get /user/output/ /home/hadoop_tp/
```

Shutting Down the HDFS

You can shut down the HDFS by using the following command.

```
$ stop-dfs.sh
```

---

10. <u>Discuss in detail the following to process the large amount of data.</u>

a. <u>MAPREDUCE</u>

- MapReduce is a framework using which we can write applications to process huge amounts of data, in parallel, on large clusters of commodity hardware in a reliable manner.

What is MapReduce?

- MapReduce is a processing technique and a program model for distributed computing based on java.
- The MapReduce algorithm contains two important tasks, namely Map and Reduce.
- Map takes a set of data and converts it into another set of data, where individual elements are broken down into tuples (key/value pairs).
- Secondly, reduce task, which takes the output from a map as an input and combines those data tuples into a smaller set of tuples.
- As the sequence of the name MapReduce implies, the reduce task is always performed after the map job.

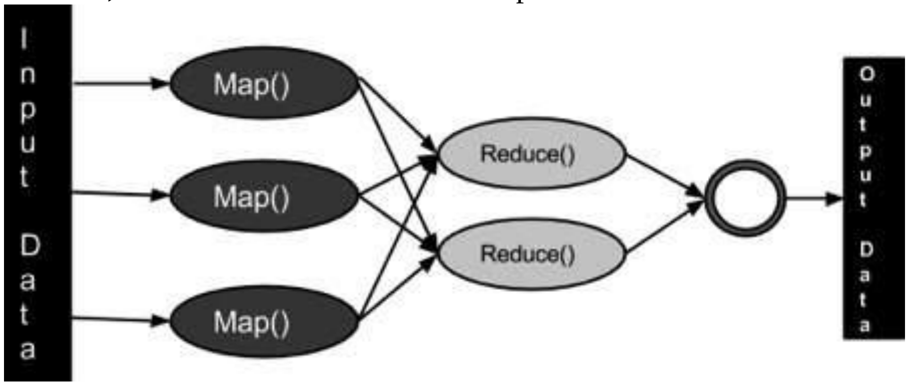
Advantage of MapReduce

- It is easy to scale data processing over multiple computing nodes.
- Under the MapReduce model, the data processing primitives are called mappers and reducers.
- Decomposing a data processing application into mappers and reducers is sometimes nontrivial.
- But, once we write an application in the MapReduce form, scaling the application to run over hundreds, thousands, or even tens of thousands of machines in a cluster is merely a configuration change.
- This simple scalability is what has attracted many programmers to use the MapReduce model.

The Algorithm

- Generally MapReduce paradigm is based on sending the computer to where the data resides!
- MapReduce program executes in three stages, namely map stage, shuffle stage, and reduce stage.
  - **Map stage** : The map or mapper's job is to process the input data. Generally the input data is in the form of file or directory and is stored in the Hadoop file system (HDFS). The input file is passed to the mapper function line by line. The mapper processes the data and creates several small chunks of data.
  - **Reduce stage** : This stage is the combination of the **Shuffle**stage and the **Reduce** stage. The Reducer's job is to process the data that comes from the mapper. After processing, it produces a new set of output, which will be stored in the HDFS.

- During a MapReduce job, Hadoop sends the Map and Reduce tasks to the appropriate servers in the cluster.
- The framework manages all the details of data-passing such as issuing tasks, verifying task completion, and copying data around the cluster between the nodes.
- Most of the computing takes place on nodes with data on local disks that reduces the network traffic.
- After completion of the given tasks, the cluster collects and reduces the data to form an appropriate result, and sends it back to the Hadoop server.



Inputs and Outputs (Java Perspective)

- The MapReduce framework operates on <key, value> pairs, that is, the framework views the input to the job as a set of <key, value> pairs and produces a set of <key, value> pairs as the output of the job, conceivably of different types.
- The key and the value classes should be in serialized manner by the framework and hence, need to implement the Writable interface. Additionally, the key classes have to implement the Writable-Comparable interface to facilitate sorting by the framework. Input and Output types of a MapReduce job: (Input) <k1, v1> -> map -> <k2, v2>-> reduce -> <k3, v3>(Output).

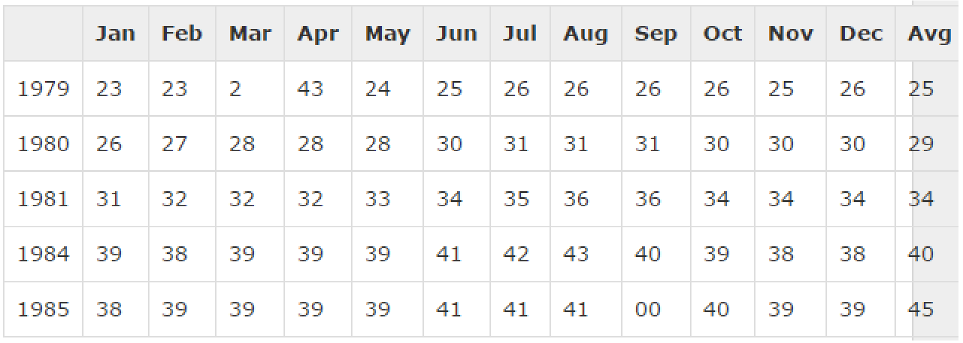
	Input	**Output**
Map	<k1, v1>	list (<k2, v2>)
Reduce	<k2, list(v2)>	list (<k3, v3>)

Terminology

- **PayLoad** - Applications implement the Map and the Reduce functions, and form the core of the job.
- **Mapper** - Mapper maps the input key/value pairs to a set of intermediate key/value pair.
- **NamedNode** - Node that manages the Hadoop Distributed File System (HDFS).
- **DataNode** - Node where data is presented in advance before any processing takes place.
- **MasterNode** - Node where JobTracker runs and which accepts job requests from clients.
- **SlaveNode** - Node where Map and Reduce program runs.
- **JobTracker** - Schedules jobs and tracks the assign jobs to Task tracker.
- **Task Tracker** - Tracks the task and reports status to JobTracker.
- **Job** - A program is an execution of a Mapper and Reducer across a dataset.
- **Task** - An execution of a Mapper or a Reducer on a slice of data.
- **Task Attempt** - A particular instance of an attempt to execute a task on a SlaveNode.

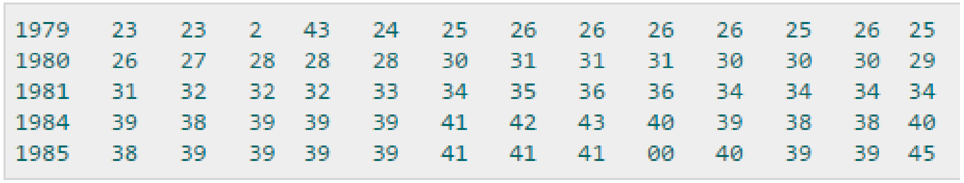
Example Scenario

- Given below is the data regarding the electrical consumption of an organization. It contains the monthly electrical consumption and the annual average for various years.



	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec	Avg
1979	23	23	2	43	24	25	26	26	26	26	25	26	25
1980	26	27	28	28	28	30	31	31	31	30	30	30	29
1981	31	32	32	32	33	34	35	36	36	34	34	34	34
1984	39	38	39	39	39	41	42	43	40	39	38	38	40
1985	38	39	39	39	39	41	41	41	00	40	39	39	45

- If the above data is given as input, we have to write applications to process it and produce results such as finding the year of maximum usage, year of minimum usage, and so on. This is a walkover for the programmers with finite number of records.
- They will simply write the logic to produce the required output, and pass the data to the application written.
- But, think of the data representing the electrical consumption of all the largescale industries of a particular state, since its formation.
- When we write applications to process such bulk data,
- They will take a lot of time to execute.
- There will be heavy network traffic when we move data from source to network server and so on.
- To solve these problems, we have the MapReduce framework.
- Input Data
- The above data is saved as **sample.txt**and given as input. The input file looks as shown below.



```
1979   23   23   2   43   24   25   26   26   26   26   25   26  25
1980   26   27   28  28   28   30   31   31   31   30   30   30  29
1981   31   32   32  32   33   34   35   36   36   34   34   34  34
1984   39   38   39  39   39   41   42   43   40   39   38   38  40
1985   38   39   39  39   39   41   41   41   00   40   39   39  45
```

b. <u>HIVE</u>

What is Hive

- Hive is a data warehouse infrastructure tool to process structured data in Hadoop. It resides on top of Hadoop to summarize Big Data, and makes querying and analyzing easy.
- Initially Hive was developed by Facebook, later the Apache Software Foundation took it up and developed it further as an open source under the name Apache Hive. It is used by different companies. For example, Amazon uses it in Amazon Elastic MapReduce.

Hive is not

- A relational database
- A design for OnLine Transaction Processing (OLTP)
- A language for real-time queries and row-level updates

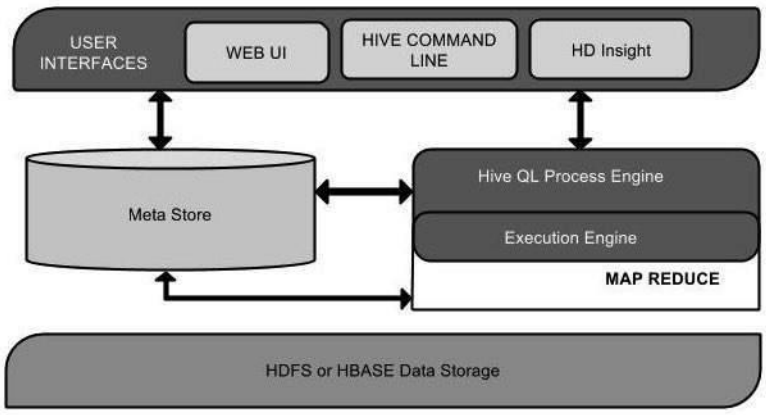
Features of Hive

- It stores schema in a database and processed data into HDFS.

- It is designed for OLAP.
- It provides SQL type language for querying called HiveQL or HQL.
- It is familiar, fast, scalable, and extensible.

Architecture of Hive

- The following component diagram depicts the architecture of Hive:



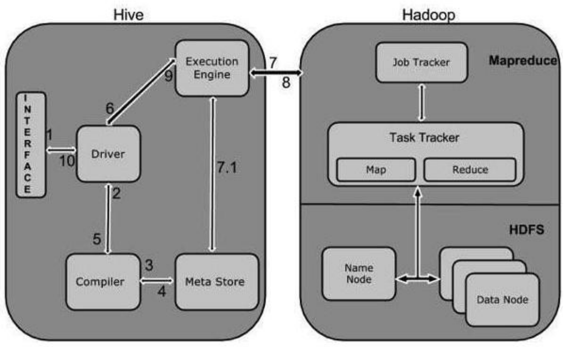
- This component diagram contains different units.

The following table describes each unit:

Unit Name	Operation
User Interface	Hive is a data warehouse infrastructure software t can create interaction between user and HDFS. T user interfaces that Hive supports are Hive Web Hive command line, and Hive HD Insight Windows server).
Meta Store	Hive chooses respective database servers to store schema or Metadata of tables, databases, column a table, their data types, and HDFS mapping.
HiveQL Process Engine	HiveQL is similar to SQL for querying on sche info on the Metastore. It is one of the replaceme of traditional approach for MapReduce progr Instead of writing MapReduce program in Java, can write a query for MapReduce job and process
Execution Engine	The conjunction part of HiveQL process Engine MapReduce is Hive Execution Engine. Execut engine processes the query and generates results same as MapReduce results. It uses the flavor MapReduce.
HDFS or HBASE	Hadoop distributed file system or HBASE are data storage techniques to store data into file syste

Working of Hive

- The following diagram depicts the workflow between Hive and Hadoop.



The following table defines how Hive interacts with Hadoop framework:

Step No.	Operation
1	**Execute Query** The Hive interface such as Command Line or Web UI sends query to Driver (any database driver such as JDBC, ODBC, etc.) to execute.
2	**Get Plan** The driver takes the help of query compiler that parses the query to check the syntax and query plan or the requirement of query.
3	**Get Metadata** The compiler sends metadata request to Metastore (any database).
4	**Send Metadata** Metastore sends metadata as a response to the compiler.
5	**Send Plan** The compiler checks the requirement and resends the plan to the driver. Up to here, the parsing and compiling of a query is complete.
6	**Execute Plan** The driver sends the execute plan to the execution engine.
7	**Execute Job** Internally, the process of execution job is a MapReduce job. The execution engine sends the job to JobTracker, which is in Name node and it assigns this job to TaskTracker, which is in Data node. Here, the query executes MapReduce job.
7.1	**Metadata Ops** Meanwhile in execution, the execution engine can execute metadata operations with Metastore.
8	**Fetch Result** The execution engine receives the results from Data nodes.
9	**Send Results** The execution engine sends those resultant values to the driver.
10	**Send Results** The driver sends the results to Hive Interfaces.

UNIT – II DATA SECURITY AND PRIVACY
PART -A

1. **What is a secure program?**

   Security implies some degree of trust that the program enforces expected confidentiality, integrity, and availability then we say the program is "secure".

2. **What is meant by bug, error and fault?**
   - A **bug** can be a mistake in interpreting a requirement, a syntax error in a piece of code, or the cause of a system crash.
   - When a human makes a mistake, called an **error**, in performing some software activity.
   - The error may lead to a **fault**, or an incorrect step, command, process, or data definition in a computer program.

3. **Define Failure.**

   A failure is a departure from the system's required behaviour. It can be discovered before or after system delivery, during testing, or during operation and maintenance. Thus, a fault is an inside view of the system, as seen by the eyes of the developers, whereas a failure is an outside view: a problem that the user sees.

4. **What is meant by program security flaws and its types?**

   A program security flaw is an undesired program behaviour caused by program vulnerability. There are two types namely malicious flaws (**Ex:** Buffer Overflow) and Non-malicious flaws (Ex: Virus and Worm).

5. **Define Cyber Attacks.**

   A cyber attack is intentional exploitation of computer systems, technology-dependent enterprises and networks. Cyber attacks use malicious code to alter computer code, logic or data, resulting in disruptive consequences that can compromise data and lead to cybercrimes, such as information and identity theft. Cyber attack (biggest security threat) also known as a computer network attack (CNA).

6. **How the program flaws are categorized?**

   Landwehr present a taxonomy of program flaws, dividing them first into intentional and inadvertent flaws. They further divide intentional flaws into malicious and non-malicious. The inadvertent flaws fall into six categories:
   - Validation error (incomplete or inconsistent): permission checks
   - Domain error: controlled access to data
   - Serialization and aliasing: program flow order
   - Inadequate identification and authentication: basis for authorization
   - Boundary condition violation: failure on first or last case
   - Other exploitable logic errors

7. **What a malicious code can do?**

   Malicious code can do anything any other program can, such as writing a message on a computer screen, stopping a running program, generating a sound, or erasing a stored file. Or malicious code can do nothing at all right now; it can be planted to lie dormant, undetected, until some event triggers the code to act.

8. **Define Transient virus.**

   A transient virus has a life that depends on the life of its host; the virus runs when its attached program executes and terminates when its attached program ends.

9. **Define Resident Virus.**

   A resident virus locates itself in memory; then it can remain active or be activated as a stand-alone program, even after its attached program ends.

10. **What is meant by zero day exploit?**

    A zero day exploit attack occurs on the same day a weakness is discovered in software. At that point, it's exploited before a fix becomes available from its creator. zero-day vulnerability, at its core, is a flaw.

11. **Define Trojan horse and Logic bomb.**

A Trojan horse is malicious code that, in addition to its primary effect, has a second, non-obvious malicious effect. As an example of a computer Trojan horse, a logic bomb is a class of malicious code that "detonates" or goes off when a specified condition occurs. A time bomb is a logic bomb whose trigger is a time or date.

12. Define Trapdoor or backdoor.

A trapdoor or backdoor is a feature in a program by which someone can access the program other than by the obvious, direct call, perhaps with special privileges. For instance, an automated bank teller program might allow anyone entering the number 990099 on the keypad to process the log of everyone's transactions at that machine. In this example, the trapdoor could be intentional, for maintenance purposes, or it could be an illicit way for the implementer to wipe out any record of a crime.

13. Define Worm and Rabbit.

A worm is a program that spreads copies of itself through a network. The primary difference between a worm and a virus is that a worm operates through networks, and a virus can spread through any medium. Worm spreads copies of itself as a stand-alone program, whereas the virus spreads copies of itself as a program that attaches to or embeds in other programs.

A rabbit is defined as a virus or worm that self-replicates without bound, with the intention of exhausting some computing resource. A rabbit might create copies of itself and store them on disk, in an effort to completely fill the disk

14. What is meant by Appended Viruses and Documented Viruses?

A program virus attaches itself to a program; then, whenever the program is run, the virus is activated it is called ***Appended Viruses***.

Document viruses are implemented within a formatted document, such as a written document, a database, a slide presentation, or a spreadsheet. The commands (such as formulas, formatting controls, links) are part of a rich programming language, including macros, variables and procedures and even system calls. The writer of a document virus uses any of the features of the programming language to perform malicious actions.

15. What are the qualities of Viruses?

- It is hard to detect.
- It is not easily destroyed or deactivated.
- It spreads infection widely.
- It is easy to create.
- It is machine independent and operating system independent.

16. What is meant by Virus Signatures?

A virus signature is the fingerprint of a virus. It is a set of unique data, or bits of code, that allow it to be identified. Anti-virus software uses a virus signature to find a virus in a computer file system, allowing detecting, quarantine and removing the virus. In the anti-virus software, the virus signature is referred to as a definition file or DAT file.

17. What are the different controls against threats?

Developmental, Operating system, and Administrative Controls

18. How the Operating System functions are categorized?

a. Access control
b. Identity and credential management
c. Information flow
d. Audit and integrity protection

19. Mention the commercially available operating systems.

There are many commercially available operating systems, but largely from two families are: the Microsoft Windows NT, 2000, XP, 2003 Server, and Vista operating systems (which we denote NT+) and Unix, Linux, and their derivatives (which we call Unix+). Other proprietary operating systems are in wide use, notably Apple's Mac OS X and IBM's z/OS but for security purposes, NT+ and Unix+ are the most widely known.

20. **Define Hazard Analysis and mention the techniques support the identification and management of potential hazards.**

    Hazard analysis is a set of systematic techniques intended to expose potentially hazardous system states. In particular, it can help us expose security concerns and then identify prevention or mitigation strategies to address them.

    Techniques support the identification and management of potential hazards:

    - Hazard and operability studies (HAZOP)
    - Failure modes and effects analysis (FMEA)
    - Fault tree analysis (FTA)

21. **What is meant by Executives?**

    The first operating systems were simple utilities, called executives, designed to assist individual programmers and to smooth the transition from one user to another. The early executives provided linkers and loaders for relocation, easy access to compilers and assemblers, and automatic loading of subprograms from libraries.

22. **What are the several ways the separation in an operating system can occur?**

    - Physical separation
    - Temporal separation
    - Logical separation
    - Cryptographic separation

23. **Define Fence and Fence register.**

    A fence is a method to confine users to one side of a boundary. The fence was a predefined memory address, enabling the operating system to reside on one side and the user to stay on the other. A hardware register, often called a fence register, containing the address of the end of the operating system.

24. **Define Base/Bounds register.**

    With two or more users, none can know in advance where a program will be loaded for execution. The relocation register solves the problem by providing a base or starting address. All addresses inside a program are offsets from that base address. A variable fence register is generally known as a base register.

    Fence registers provide a lower bound (a starting address) but not an upper one. To overcome this difficulty, a second register is often added, called a bounds register.

25. **Mention few kinds of objects for which protection are desirable.**

    - A file or data set on an auxiliary storage device
    - An executing program in memory
    - A directory of files
    - A hardware device
    - A data structure, such as a stack
    - A table of the operating system

26. **What is meant by Kerberos and mention the two systems required by it.**

    Kerberos implements both authentication and access authorization by means of capabilities, called tickets, secured with symmetric cryptography. Kerberos requires two systems, called the authentication server (AS) and the ticket-granting server (TGS), which are both part of the key distribution center (KDC).

27. **Define Firewalls.**

    A firewall is a device that filters all traffic between a protected or "inside" network and a less trustworthy or "outside" network. Usually a firewall runs on a dedicated device; because it is a single point through which traffic is channeled, performance is important, which means non-firewall functions should not be done on the same machine.

28. **What are the types of firewalls?**

    - Packet filtering gateways or screening routers
    - Stateful inspection firewalls
    - Application proxies

- Guards
- Personal firewalls

29. Define application proxy gateway.

An application proxy gateway, also called a bastion host, is a firewall that simulates the (proper) effects of an application so that the application receives only requests to act properly.

30. Define Intrusion detection system.

An intrusion detection system (IDS) is a device, typically another separate computer that monitors activity to identify malicious or suspicious events. An IDS is a sensor, like a smoke detector, that raises an alarm if specific things occur.

31. Mention the functions of IDS.

- Monitoring users and system activity
- Auditing system configuration for vulnerabilities and misconfigurations
- Assessing the integrity of critical system and data files
- Correcting system configuration errors
- Installing and operating traps to record information about intruders

32. What are the different types of IDS?

1) Signature-Based Intrusion Detection
2) Heuristic Intrusion Detection/anomaly based

Intrusion detection devices can be **network based or host based**. A network-based IDS is a stand-alone device attached to the network to monitor traffic throughout that network; a host-based IDS runs on a single workstation or client or host, to protect that one host.

33. What are the limitations of IDS?

- IDS limitation is its sensitivity, which is difficult to measure and adjust.
- IDSs will never be perfect, so finding the proper balance is critical.
- An IDS does not run itself; someone has to monitor its track record and respond to its alarms.

34. What is Privacy?

A good working definition is that privacy is the right to control who knows certain aspects about you, your communications, and your activities. In other words, you voluntarily choose who can know things about you and what those things are.

35. List all the eight dimensions of privacy.

- Information usage
- Information retention
- Information disclosure
- Information security
- Access control
- Monitoring
- Policy changes

36. Define Pseudonymity.

Multiple identities can also be convenient, for example, having a professional e-mail account and a social one. Similarly, disposable identities (that you use for a while and then stop using) can be convenient. When you sign up for something and you know your e-mail address will be sold many times, you might get a new e-mail address to use until the spam and other unsolicited e-mail are oppressive, and then you discard the address. These uses are called pseudonymity.

37. Define Statues.

Statutes are laws that state explicitly that certain actions are illegal. A statute is the result of a legislative process by which a governing body declares that the new law will be in force after a designated time. For example, the parliament may discuss issues related to taxing Internet transactions and pass a law about when relevant taxes must be paid.

38. Define Civil law.

Civil law is a different type of law, not requiring such a high standard of proof of guilt. In a civil case, an individual, organization, company, or group claims it has been harmed. The goal of a civil case is restitution (compensation): to make the victim "whole" again by repairing the harm.

39. What is Tort Law?

Tort law is the body of laws that enables people to seek compensation for wrongs committed against them. When someone’s actions cause some type of harm to another, whether it be physical harm to another person, or harm to someone's property or reputation, the harmed or injured person or entity may seek damages through the court. Fraud is a common example of tort law in which, basically, one person lies to another, causing harm.

PART- B

1. **Discuss in detail about how to secure the programs.**

PROGRAM SECURITY

- A program is "secure implies some degree of trust that the program enforces expected confidentiality, integrity, and availability.
- For example, one person may decide that code is secure because it takes too long to break through its security controls.
- And someone else may decide code is secure if it has run for a period of time with no apparent failures. But a third person may decide that *any* potential fault in meeting security requirements makes code insecure.
- An assessment of security can also be influenced by someone's general perspective on software quality.
- This security view played a role when a major computer manufacturer delivered all its machines with keyed locks, since a keyed lock was written in the requirements. But the machines were not secure, because all locks were configured to use the same key.
- For example, developers track the number of faults found in requirements, design, and code inspections and use them as indicators of the likely quality of the final product.

FIXING FAULTS

- One approach to judging quality in security has been fixing faults.
- A module in which 100 faults were discovered and fixed is better than another in which only 20 faults were discovered and fixed, suggesting that more rigorous analysis and testing had led to the finding of the larger number of faults.
- Early work in computer security was based on the paradigm of "penetrate and patch," in which analysts searched for and repaired faults.
- There are at least four reasons why.
  - The pressure to repair a specific problem encouraged a narrow focus on the fault itself and not on its context. In particular, the analysts paid attention to the immediate cause of the failure and not to the underlying design or requirements faults.
  - The fault often had no obvious side effects in places other than the immediate area of the fault.
  - Fixing one problem often caused a failure somewhere else, or the patch addressed the problem in only one place, not in other related places.
  - The fault could not be fixed properly because system functionality or performance would suffer as a consequence.

IEEE TERMINOLOGY FOR QUALITY

- A **bug** can be a mistake in interpreting a requirement, a syntax error in a piece of code, or the (as-yet-unknown) cause of a system crash.
- When a human makes a mistake, called an error, in performing some software activity, the error may lead to a **fault**, or an incorrect step, command, process, or data definition in a computer program.
- A **failure** is a departure from the system's required behavior. It can be discovered before or after system delivery, during testing, or during operation and maintenance.

UNEXPECTED BEHAVIOR

- To understand program security, examine programs to see whether they behave as their designers intended or users expected.
- Such unexpected behavior a program security flaw; it is inappropriate program behavior caused by a program vulnerability.

CONTINUING INCREASE IN CYBER ATTACKS

- Carnegie Mellon University's Computer Emergency Response Team (CERT) tracks the number and kinds of vulnerabilities and cyber attacks reported worldwide.
- Part of CERT's mission is to warn users and developers of new problems and also to provide information on ways to fix them.

TYPES OF FLAWS

- To aid our understanding of the problems and their prevention or correction, we can define categories that distinguish one kind of problem from another.
- Landwehr et al. present taxonomy of program flaws, dividing them first into intentional and inadvertent flaws.
- They further divide intentional flaws into malicious and non-malicious ones.
- In the taxonomy, the inadvertent flaws fall into six categories:
  - ***Validation* error** (incomplete or inconsistent): permission checks
  - ***Domain* error:** controlled access to data
  - ***Serialization* and *aliasing*:** program flow order
  - **Inadequate *identification and authentication*:** basis for authorization
  - ***Boundary condition violation:*** failure on first or last case
  - Other exploitable ***logic errors***

---

2. **Explain about the malicious code and its types.**

MALICIOUS CODE

- Malicious code is code causing damage to a computer or system. It is code not easily or solely controlled through the use of anti-virus tools.
- Malicious code can either activate itself or be like a virus requiring a user to perform an action, such as clicking on something or opening an email attachment.
- Viruses and worms are related classes of malicious code.

WHY WORRY ABOUT MALICIOUS CODE?

- When you last installed a major software package, such as a word processor, a statistical package, or a plug-in from the Internet, you ran one command, typically called INSTALL or SETUP.
- From there, the installation program took control, creating some files, writing in other files, deleting data and files, and perhaps renaming a few that it would change.
- Thousands or even millions of bytes of programs and data are transferred, and hundreds of modifications may be made to your existing files, all occurring without your explicit consent or knowledge.

MALICIOUS CODE CAN DO MUCH (HARM)

- Malicious code can do anything any other program can, such as writing a message on a computer screen, stopping a running program, generating a sound, or erasing a stored file.
- Malicious code can do nothing at all right now; it can be planted to lie dormant, undetected, until some event triggers the code to act.

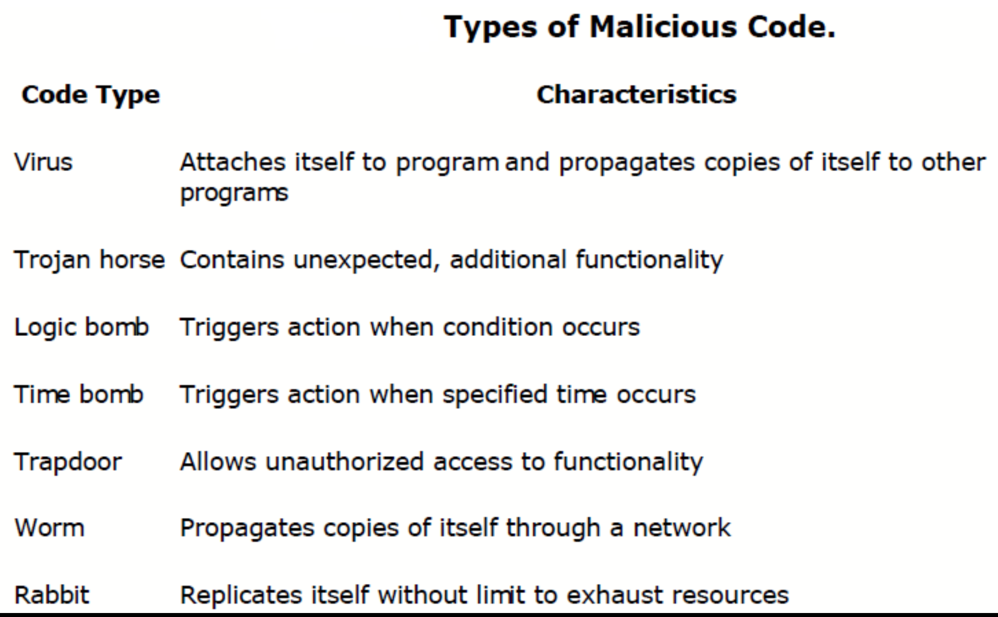
- The trigger can be a time or date, an interval (for example, after 30 minutes), an event (for example, when a particular program is executed), a condition (for example, when communication occurs on a network interface), a count (for example, the fifth time something happens), some combination of these, or a random situation.
- In fact, malicious code can do different things each time or nothing most of the time with something dramatic on occasion.
- Malicious code runs under the user's authority.
- Thus, malicious code can touch everything the user can touch, and in the same ways.
- Users typically have complete control over their own program code and data files; they can read, write, modify, append, and even delete them.
- Malicious code can do the same, without the user's permission or even knowledge.

KINDS OF MALICIOUS CODE

- Malicious code or rogue program is the general name for unanticipated or undesired effects in programs or program parts, caused by an agent intent on damage.
- This definition excludes unintentional errors, although they can also have a serious negative effect.
- The **agent** is the writer of the program or the person who causes its distribution.
- In fact, virus might actually have been a worm.
- The terminology of malicious code is sometimes used imprecisely.
- A **virus** is a program that can replicate itself and pass on malicious code to other non-malicious programs by modifying them.
- The term "virus" was coined because the affected program acts like a biological virus: It infects other healthy subjects by attaching itself to the program and either destroying it or coexisting with it.
- A virus can be either transient or resident.
- A **transient virus** has a life that depends on the life of its host; the virus runs when its attached program executes and terminates when its attached program ends. (During its execution, the transient virus may spread its infection to other programs.)
- A **resident virus** locates itself in memory; then it can remain active or be activated as a stand-alone program, even after its attached program ends.
- An attack before availability of the control is called a **zero day exploit**.
- Time between proof of concept and actual attack has been shrinking.
    - Code Red, one of the most virulent pieces of malicious code, in 2001 exploited vulnerabilities for which the patches had been distributed more than a month before the attack.
- A **Trojan horse** is malicious code that, in addition to its primary effect, has a second, non obvious malicious effect.
    - As an example of a computer Trojan horse, consider a login script that solicits a user's identification and password, passes the identification information on to the rest of the system for login processing, but also retains a copy of the information for later, malicious use.
    - In this example, the user sees only the login occurring as expected, so there is no evident reason to suspect that any other action took place.
- A **logic bomb** is a class of malicious code that "detonates" or goes off when a specified condition occurs.

  - A time bomb is a logic bomb whose trigger is a time or date.

- A **trapdoor** or backdoor is a feature in a program by which someone can access the program other than by the obvious, direct call, perhaps with special privileges.
- A **worm** is a program that spreads copies of itself through a network.
- The primary difference between a worm and a virus is that a worm operates through networks, and a virus can spread through any medium (but usually uses copied program or data files).
- The worm spreads copies of itself as a stand-alone program, whereas the virus spreads copies of itself as a program that attaches to or embeds in other programs.
- **Rabbit** as a virus or worm that self-replicates without bound, with the intention of exhausting some computing resource.
- A rabbit might create copies of itself and store them on disk in an effort to completely fill the disk.



Types of Malicious Code.

Code Type	Characteristics
Virus	Attaches itself to program and propagates copies of itself to other programs
Trojan horse	Contains unexpected, additional functionality
Logic bomb	Triggers action when condition occurs
Time bomb	Triggers action when specified time occurs
Trapdoor	Allows unauthorized access to functionality
Worm	Propagates copies of itself through a network
Rabbit	Replicates itself without limit to exhaust resources

3. Explain how the viruses get appended to the programs in detail.

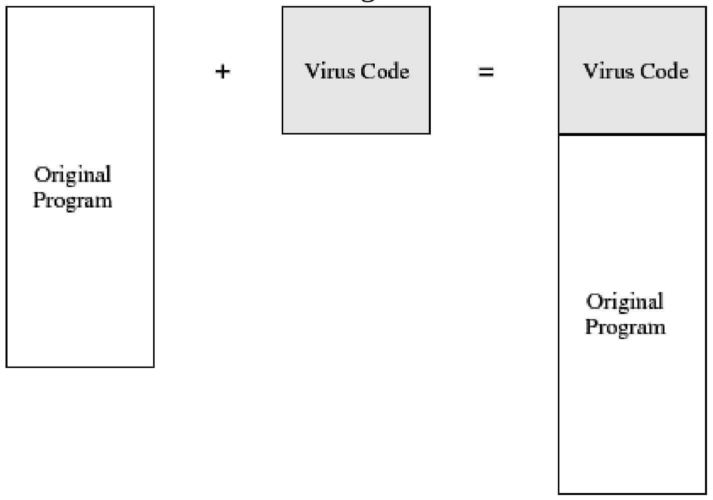
HOW VIRUSES ATTACH

- A printed copy of a virus does nothing and threatens no one.
- Even executable virus code sitting on a disk does nothing.
- For a virus to do its malicious work and spread itself, it must be activated by being executed.
- For example, recall the SETUP program that you initiate on your computer.
- It may call dozens or hundreds of other programs, some on the distribution medium, some already residing on the computer, some in memory.
- If any one of these programs contains a virus, the virus code could be activated. Suppose the virus code were in a program on the distribution medium, such as a CD; when executed, the virus could install itself on a permanent storage medium (typically, a hard disk), and also in any and all executing programs in memory.
- Human intervention is necessary to start the process; a human being puts the virus on the distribution medium, and perhaps another initiates the execution of the program to which the virus is attached.
- After that, no human intervention is needed; the virus can spread by itself.

- A more common means of virus activation is as an attachment to an e-mail message.
- In this attack, the virus writer tries to convince the victim (the recipient of an e-mail message) to open the attachment.
- Once the viral attachment is opened, the activated virus can do its work.
- The virus can be executable code embedded in an executable attachment, but other types of files are equally dangerous.
- For example, objects such as graphics or photo images can contain code to be executed by an editor, so they can be transmission agents for viruses.

APPENDED VIRUSES

- A program virus attaches itself to a program; then, whenever the program is run, the virus is activated. This kind of attachment is usually easy to program.
- In the simplest case, a virus inserts a copy of itself into the executable program file before the first executable instruction.
- Then, all the virus instructions execute first; after the last virus instruction, control flows naturally to what used to be the first program instruction.
- Such a situation is shown in the below figure.

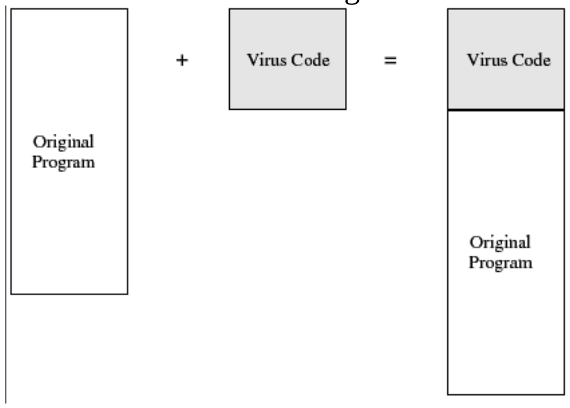


- This kind of attachment is simple and usually effective.
- The virus writer does not need to know anything about the program to which the virus will attach, and often the attached program simply serves as a carrier for the virus.
- The virus performs its task and then transfers to the original program.
- Typically, the user is unaware of the effect of the virus if the original program still does all that it used to. Most viruses attach in this manner.

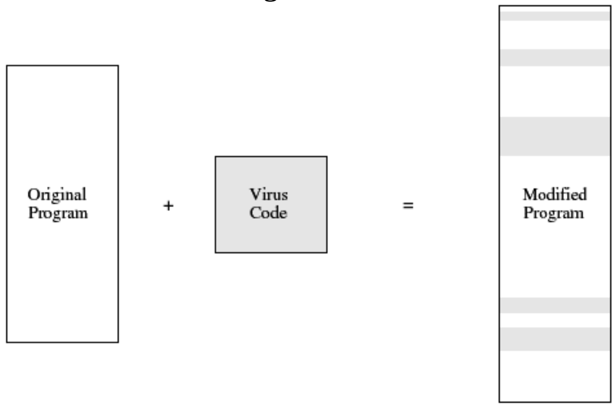
VIRUSES THAT SURROUND A PROGRAM

- An alternative to the attachment is a virus that runs the original program but has control before and after its execution.
- For example, a virus writer might want to prevent the virus from being detected.
- If the virus is stored on disk, its presence will be given away by its file name, or its size will affect the amount of space used on the disk.
- The virus writer might arrange for the virus to attach itself to the program that constructs the listing of files on the disk.

- If the virus regains control after the listing program has generated the listing but before the listing is displayed or printed, the virus could eliminate its entry from the listing and falsify space counts so that it appears not to exist.
- A surrounding virus is shown in the below figure



- A third situation occurs when the virus replaces some of its target, integrating itself into the original code of the target.
- Such a situation is shown in below figure.

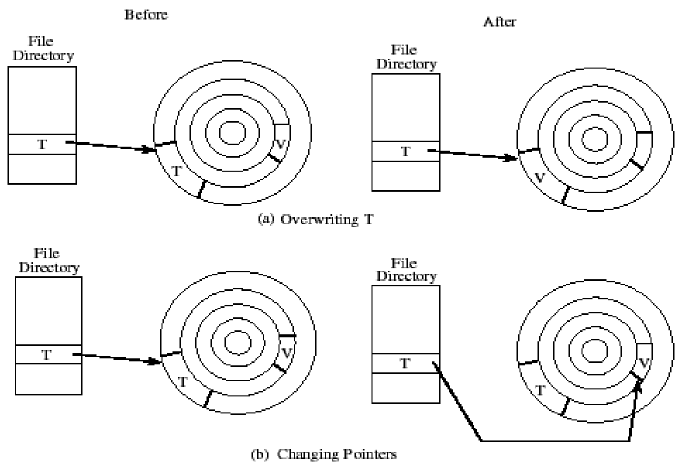


- Clearly, the virus writer has to know the exact structure of the original program to know where to insert which pieces of the virus.
- Finally, the virus can replace the entire target, either mimicking the effect of the target or ignoring the expected effect of the target and performing only the virus effect.
- In this case, the user is most likely to perceive the loss of the original program.

HOW VIRUSES GAIN CONTROL

- The virus (V) has to be invoked instead of the target (T).
- Essentially, the virus either has to seem to be T, saying effectively "I am T" (like some rock stars, where the target is the artiste formerly known as T) or the virus has to push T out of the way and become a substitute for T, saying effectively "Call me instead of T."
- A more blatant virus can simply say "invoke me [you fool]."
- The virus can assume T's name by replacing (or joining to) T's code in a file structure; this invocation technique is most appropriate for ordinary programs.
- The virus can overwrite T in storage (simply replacing the copy of T in storage, for example).
- Alternatively, the virus can change the pointers in the file table so that the virus is located instead of T whenever T is accessed through the file system.

- These two cases are shown in the below figure



- The virus can supplant T by altering the sequence that would have invoked T to now invoke the virus V; this invocation can be used to replace parts of the resident operating system by modifying pointers to those resident parts, such as the table of handlers for different kinds of interrupts.

HOMES FOR VIRUSES

The virus writer may find these qualities appealing in a virus:

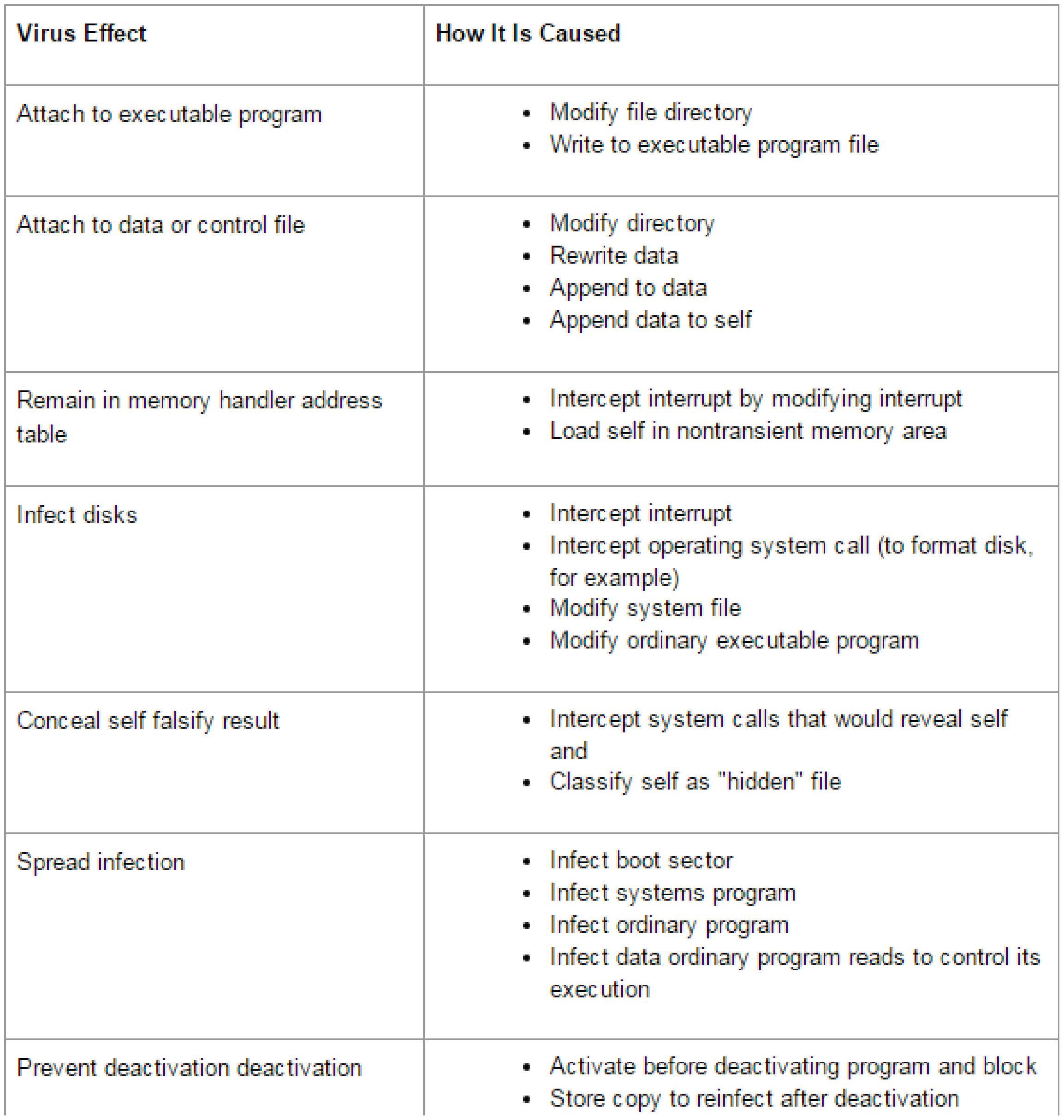
- It is hard to detect.
- It is not easily destroyed or deactivated.
- It spreads infection widely.
- It can reinfect its home program or other programs.
- It is easy to create.
- It is machine independent and operating system independent.

---

4. <u>Discuss in detail about the following</u>

a. <u>Virus effects and causes</u>

- A virus writer may want a virus to do several things at the same time, namely, spread infection, avoid detection, and cause harm.
- These goals are shown in below Table , along with ways each goal can be addressed.
- Many of these behaviors are perfectly normal and might otherwise go undetected.
- For instance, one goal is modifying the file directory; many normal programs create files, delete files, and write to storage media.
- Most virus writers seek to avoid detection for themselves and their creations.



Virus Effect	How It Is Caused
Attach to executable program	• Modify file directory • Write to executable program file
Attach to data or control file	• Modify directory • Rewrite data • Append to data • Append data to self
Remain in memory handler address table	• Intercept interrupt by modifying interrupt • Load self in nontransient memory area
Infect disks	• Intercept interrupt • Intercept operating system call (to format disk, for example) • Modify system file • Modify ordinary executable program
Conceal self falsify result	• Intercept system calls that would reveal self and • Classify self as "hidden" file
Spread infection	• Infect boot sector • Infect systems program • Infect ordinary program • Infect data ordinary program reads to control its execution
Prevent deactivation deactivation	• Activate before deactivating program and block • Store copy to reinfect after deactivation

- Because a disk's boot sector is not visible to normal operations (for example, the contents of the boot sector do not show on a directory listing), many virus writers hide their code there.
- A resident virus can monitor disk accesses and fake the result of a disk operation that would show the virus hidden in a boot sector by showing the data that *should* have been in the boot sector (which the virus has moved elsewhere).
- There are no limits to the harm a virus can cause.
- On the modest end, the virus might do nothing; some writers create viruses just to show they can do it.
- Or the virus can be relatively benign, displaying a message on the screen, sounding the buzzer, or playing music.
- One virus can erase files, another entire disk; one virus can prevent a computer from booting, and another can prevent writing to disk.
- The damage is bounded only by the creativity of the virus's author.

b. <u>Prevention of viruses</u>

PREVENTION OF VIRUS INFECTION

- The only way to prevent the infection of a virus is not to share executable code with an infected source.
- This philosophy used to be easy to follow because it was easy to tell if a file was executable or not.
- For example, on PCs, a *.exe* extension was a clear sign that the file was executable.
- However, as we have noted, today's files are more complex, and a seemingly non executable file may have some executable code buried deep within it.
- For example, a word processor may have commands within the document file; as we noted earlier, these commands, called macros, make it easy for the user to do complex or repetitive things.
- But they are really executable code embedded in the context of the document.
- Similarly, spreadsheets, presentation slides, and other office- or business-related files can contain code or scripts that can be executed in various ways—and thereby harbor viruses.
- Another approach virus writers have used is a little-known feature in the Microsoft file design.
- Although a file with a *.doc* extension is expected to be a Word document, in fact, the true document type is hidden in a field at the start of the file.
- This convenience ostensibly helps a user who inadvertently names a Word document with a *.ppt* (Power-Point) or any other extension.
- In some cases, the operating system will try to open the associated application but, if that fails, the system will switch to the application of the hidden file type.
- So, the virus writer creates an executable file, names it with an inappropriate extension, and sends it to the victim, describing it is as a picture or a necessary code add-in or something else desirable.
- The unwitting recipient opens the file and, without intending to, executes the malicious code.
- More recently, executable code has been hidden in files containing large data sets, such as pictures or read-only documents.
- These bits of viral code are not easily detected by virus scanners and certainly not by the human eye.
- For example, a file containing a photograph may be highly granular; if every sixteenth bit is part of a command string that can be executed, then the virus is very difficult to detect.

Nevertheless, there are several techniques for building a reasonably safe community for electronic contact, including the following:

- ***Use only commercial software acquired from reliable, well-established vendors.*** There is always a chance that you might receive a virus from a large manufacturer with a name everyone would recognize. However, such enterprises have significant reputations that could be seriously damaged by even one bad incident, so they go to some degree of trouble to keep their products virus-free and to patch any problem-causing code right away. Similarly, software distribution companies will be careful about products they handle.
- *Test all new software on an isolated computer.* If you must use software from a questionable source, test the software first on a computer with no hard disk, not connected to a network, and with the boot disk removed. Run the software and look

for unexpected behavior, even simple behavior such as unexplained figures on the screen. Test the computer with a copy of an up-to-date virus scanner, created before running the suspect program. Only if the program passes these tests should it be installed on a less isolated machine.

- ***Open attachments only when you know them to be safe.*** What constitutes "safe" is up to you, as you have probably already learned in this chapter. Certainly, an attachment from an unknown source is of questionable safety. You might also distrust an attachment from a known source but with a peculiar message.
- ***Make a recoverable system image and store it safely.*** If your system does become infected, this clean version will let you reboot securely because it overwrites the corrupted system files with clean copies. For this reason, you must keep the image write-protected during reboot. Prepare this image now, before infection; after infection it is too late. For safety, prepare an extra copy of the safe boot image.
- ***Make and retain backup copies of executable system files***. This way, in the event of a virus infection, you can remove infected files and reinstall from the clean backup copies (stored in a secure, offline location, of course).
- ***Use virus detectors (often called virus scanners) regularly and update them daily***. Many of the virus detectors available can both detect and eliminate infection from viruses. Several scanners are better than one, because one may detect the viruses that others miss. Because scanners search for virus signatures, they are constantly being revised as new viruses are discovered. New virus signature files, or new versions of scanners, are distributed frequently; often, you can request automatic downloads from the vendor's web site. Keep your detector's signature file up-to-date.

---

5. **Discuss any two examples of malicious Code in detail.**

1. THE BRAIN VIRUS

One of the earliest viruses is also one of the most intensively studied. The so-called Brain virus was given its name because it changes the label of any disk it attacks to the word "BRAIN." This particular virus, believed to have originated in Pakistan, attacks PCs running a Microsoft operating system. Numerous variants have been produced; because of the number of variants, people believe that the source code of the virus was released to the underground virus community.

What It Does

The Brain, like all viruses, seeks to pass on its infection. This virus first locates itself in upper memory and then executes a system call to reset the upper memory bound below itself, so that it is not disturbed as it works. It traps interrupt number 19 (disk read) by resetting the interrupt address table to point to it and then sets the address for interrupt number 6 (unused) to the former address of the interrupt 19. In this way, the virus screens disk read calls, handling any that would read the boot sector (passing back the original boot contents that were moved to one of the bad sectors); other disk calls go to the normal disk read handler, through interrupt 6.

The Brain virus appears to have no effect other than passing its infection, as if it were an experiment or a proof of concept. However, variants of the virus erase disks or destroy the file allocation table (the table that shows which files are where on a storage medium).

How It Spreads

The Brain virus positions itself in the boot sector and in six other sectors of the disk. One of the six sectors will contain the original boot code, moved there from the original boot sector, while two others contain the remaining code of the virus. The remaining three sectors contain a duplicate of the others. The virus marks these six sectors "faulty" so that the operating system will not try to use them. (With low-level calls, you can force the disk drive to read from what the operating system has marked as bad sectors.) The virus allows the boot process to continue.

Once established in memory, the virus intercepts disk read requests for the disk drive under attack. With each read, the virus reads the disk boot sector and inspects the fifth and sixth bytes for the hexadecimal value 1234 (its signature). If it finds that value, it concludes the disk is infected; if not, it infects the disk as described in the previous paragraph.

What Was Learned

This virus uses some of the standard tricks of viruses, such as hiding in the boot sector, and intercepting and screening interrupts. The virus is almost a prototype for later efforts. In fact, many other virus writers seem to have patterned their work on this basic virus. Thus, one could say it was a useful learning tool for the virus writer community.

Sadly, its infection did not raise public consciousness of viruses, other than a certain amount of fear and misunderstanding. Subsequent viruses, such as the Lehigh virus that swept through the computers of Lehigh University, the nVIR viruses that sprang from prototype code posted on bulletin boards, and the Scores virus that was first found at NASA in Washington D.C. circulated more widely and with greater effect. Fortunately, most viruses seen to date have a modest effect, such as displaying a message or emitting a sound. That is, however, a matter of luck, since the writers who could put together the simpler viruses obviously had all the talent and knowledge to make much more malevolent viruses.

There is no general cure for viruses. Virus scanners are effective against today's known viruses and general patterns of infection, but they cannot counter tomorrow's variant. The only sure prevention is complete isolation from outside contamination, which is not feasible; in fact, you may even get a virus from the software applications you buy from reputable vendors.

2. THE INTERNET WORM

On the evening of 2 November 1988, a worm was released to the Internet,[3] causing serious damage to the network. Not only were many systems infected, but when word of the problem spread, many more uninfected systems severed their network connections to prevent themselves from getting infected. Gene Spafford and his team at Purdue University [SPA89] and Mark Eichen and Jon Rochlis at M.I.T [EIC89] studied the worm extensively.

The perpetrator was Robert T. Morris, Jr., a graduate student at Cornell University who created and released the worm. He was convicted in 1990 of violating the 1986 Computer Fraud and Abuse Act, section 1030 of U.S. Code Title 18. He received a fine of $10,000, a three-year suspended jail sentence, and was required to perform 400 hours of community service.

What It Did

Judging from its code, Morris programmed the Internet worm to accomplish three main objectives:

1. determine to where it could spread

2. spread its infection
3. remain undiscovered and undiscoverable

What Effect It Had

The worm's primary effect was resource exhaustion. Its source code indicated that the worm was supposed to check whether a target host was already infected; if so, the worm would negotiate so that either the existing infection or the new infector would terminate. However, because of a supposed flaw in the code, many new copies did not terminate. As a result, an infected machine soon became burdened with many copies of the worm, all busily attempting to spread the infection. Thus, the primary observable effect was serious degradation in performance of affected machines.

A second-order effect was the disconnection of many systems from the Internet. System administrators tried to sever their connection with the Internet, either because their machines were already infected and the system administrators wanted to keep the worm's processes from looking for sites to which to spread or because their machines were not yet infected and the staff wanted to avoid having them become so.

The disconnection led to a third-order effect: isolation and inability to perform necessary work. Disconnected systems could not communicate with other systems to carry on the normal research, collaboration, business, or information exchange users expected. System administrators on disconnected systems could not use the network to exchange information with their counterparts at other installations, so status and containment or recovery information was unavailable.

The worm caused an estimated 6,000 installations to shut down or disconnect from the Internet. In total, several thousand systems were disconnected for several days, and several hundred of these systems were closed to users for a day or more while they were disconnected. Estimates of the cost of damage range from $100,000 to $97 million.

How It Worked

The worm exploited several known flaws and configuration failures of Berkeley version 4 of the Unix operating system. It accomplished—or had code that appeared to try to accomplish—its three objectives.

Where to spread.

The worm had three techniques for locating potential machines to victimize. It first tried to find user accounts to invade on the target machine. In parallel, the worm tried to exploit a bug in the *finger* program and then to use a trapdoor in the *sendmail* mail handler. All three of these security flaws were well known in the general Unix community.

Spread infection.

Having found a suitable target machine, the worm would use one of these three methods to send a bootstrap loader to the target machine. This loader consisted of 99 lines of C code to be compiled and executed on the target machine. The bootstrap loader would then fetch the rest of the worm from the sending host machine. There was an element of good computer security—or stealth—built into the exchange between the host and the target. When the target's bootstrap requested the rest of the worm, the worm supplied a one-time password back to the host. Without this password, the host would immediately break the connection to the target, presumably in an effort to ensure against "rogue" bootstraps (ones that a real administrator might develop to try to obtain a copy of the rest of the worm for subsequent analysis).

Remain undiscovered and undiscoverable.

The worm went to considerable lengths to prevent its discovery once established on a host. For instance, if a transmission error occurred while the rest of the worm was being fetched, the loader zeroed and then deleted all code already transferred and exited.

As soon as the worm received its full code, it brought the code into memory, encrypted it, and deleted the original copies from disk. Thus, no traces were left on disk, and even a memory dump would not readily expose the worm's code. The worm periodically changed its name and process identifier so that no single name would run up a large amount of computing time.

---

6. **<u>Explain in detail how to use controls during software development against the program threats.</u>**

CONTROLS AGAINST PROGRAM THREATS

- There are many ways a program can fail and many ways to turn the underlying faults into security failures.
- It is of course better to focus on prevention than cure; how do we use controls during **software development**—the specifying, designing, writing, and testing of the program—to find and eliminate the sorts of exposures have been discussed.
- In this three types of controls: developmental, operating system, and administrative discussed.

Developmental Controls

- Many controls can be applied during software development to ferret out and fix problems. Tasks involved in controls are specifying, designing, building, and testing software.

The Nature of Software Development

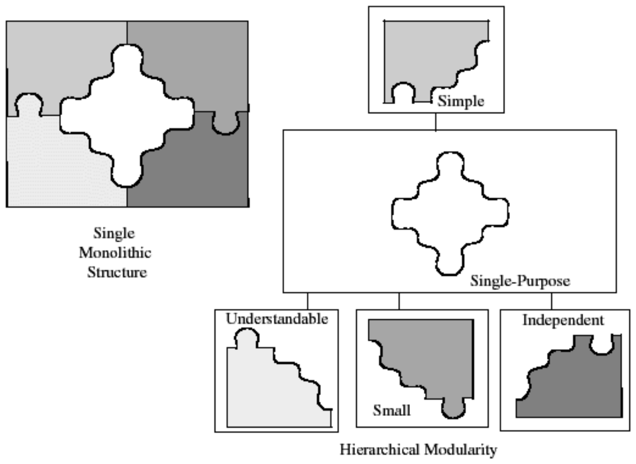
- Software development is often considered a solitary effort; a programmer sits with a specification or design and grinds out line after line of code. But in fact, software development is a collaborative effort, involving people with different skill sets who combine their expertise to produce a working product.
- Development requires people who can
  - *specify* the system, by capturing the requirements and building a model of how the system should work from the users' point of view
  - *design* the system, by proposing a solution to the problem described by the requirements and building a model of the solution
  - *implement* the system, by using the design as a blueprint for building a working solution
  - *test* the system, to ensure that it meets the requirements and implements the solution as called for in the design
  - *review* the system at various stages, to make sure that the end products are consistent with the specification and design models
  - *document* the system, so that users can be trained and supported
  - *manage* the system, to estimate what resources will be needed for development and to track when the system will be done
  - *maintain* the system, tracking problems found, changes needed, and changes made, and evaluating their effects on overall quality and functionality

Modularity, Encapsulation, and Information Hiding

- Code usually has a long shelf-life, and it is enhanced over time as needs change and faults are found and fixed. For this reason, a key principle of software engineering is to create a design or code in small, self-contained units, called **components** or **modules**; when a system is written this way, we say that it is **modular**. Modularity offers advantages for program development in general and security in particular.
- If a component is isolated from the effects of other components, then it is easier to trace a problem to the fault that caused it and to limit the damage the fault causes. It is also easier to maintain the system, since changes to an isolated component do not affect other components. And it is easier to see where vulnerabilities may lie if the component is isolated. This isolation is called **encapsulation**.
- **Information hiding** is another characteristic of modular software. When information is hidden, each component hides its precise implementation or some other design decision from the others. Thus, when a change is needed, the overall design can remain intact while only the necessary changes are made to particular components.

Modularity

- **Modularization** is the process of dividing a task into subtasks. This division is done on a logical or functional basis.
- Each component performs a separate, independent part of the task. **Modularity** is depicted in the below figure



There are several advantages to having small, independent components.

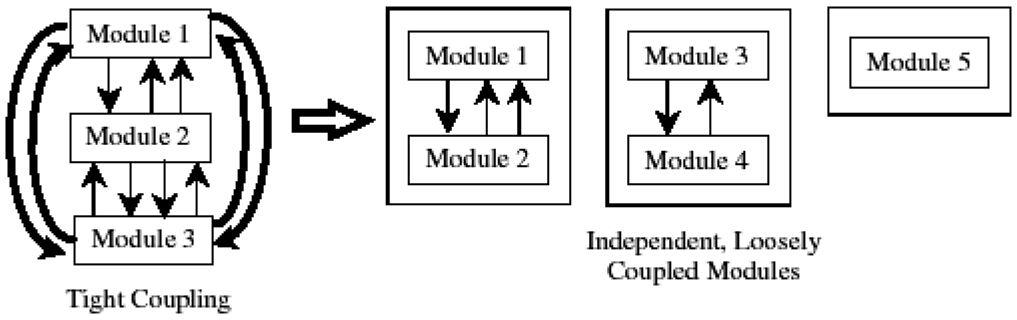
- ***Maintenance.*** If a component implements a single function, it can be replaced easily with a revised one if necessary. The new component may be needed because of a change in requirements, hardware, or environment.
- ***Understandability.*** A system composed of many small components is usually easier to comprehend than one large, unstructured block of code.
- ***Reuse.*** Components developed for one purpose can often be reused in other systems. Reuse of correct, existing design or code components can significantly reduce the difficulty of implementation and testing.
- ***Correctness.*** A failure can be quickly traced to its cause if the components perform only one task each.

- ***Testing.*** A single component with well-defined inputs, output, and function can be tested exhaustively by itself, without concern for its effects on other module.

- A modular component usually has high cohesion and low coupling.
- By **cohesion**, we mean that all the elements of a component have a logical and functional reason for being there; every aspect of the component is tied to the component's single purpose.
- A highly cohesive component has a high degree of focus on the purpose;
- A low degree of cohesion means that the component's contents are an unrelated jumble of actions, often put together because of time-dependencies or convenience.
- **Coupling** refers to the degree with which a component depends on other components in the system. Thus, low or loose coupling is better than high or tight coupling, because the loosely coupled components are free from unwitting interference from other components.

Encapsulation

- Encapsulation hides a component's implementation details, but it does not necessarily mean complete isolation.
- Many components must share information with other components, usually with good reason.
- However, this sharing is carefully documented so that a component is affected only in known ways by others in the system.
- Sharing is minimized so that the fewest interfaces possible are used.
- Limited interfaces reduce the number of covert channels that can be constructed.

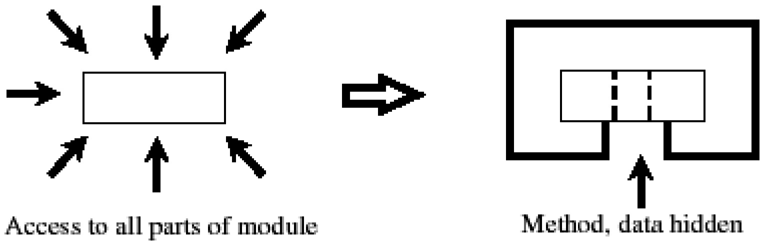


- Encapsulation is the "technique for packaging the information [inside a component] in such a way as to hide what should be hidden and make visible what is intended to be visible."

Information Hiding

- Developers who work where modularization is stressed can be sure that other components will have limited effect on the ones they write.
- Thus, we can think of a component as a kind of black box, with certain well-defined inputs and outputs and a well-defined function.
- Other components' designers do not need to know *how* the module completes its function; it is enough to be assured that the component performs its task in some correct manner.
- This concealment is the information hiding, depicted in the below figure.

- Information hiding is desirable, because developers cannot easily and maliciously alter the components of others if they do not know how the components work.



- These three characteristics—modularity, encapsulation, and information hiding—are fundamental principles of software engineering.
- They are also good security practices because they lead to modules that can be understood, analyzed, and trusted.

---

7. **Discuss about the Operating systems security methods and their role in computer security.**

An operating system has two goals:

- controlling shared access
- implementing an interface to allow that access
- Underneath those goals are support activities, including identification and authentication, naming, filing objects, scheduling, communication among processes, and reclaiming and reusing objects

Operating system functions can be categorized as

- access control
- identity and credential management
- information flow
- audit and integrity protection
- Each of these activities has security implications.

Protected Objects and Methods of Protection

Protected Objects

- The rise of multiprogramming meant that several aspects of a computing system required protection:
- memory
- sharable I/O devices, such as disks
- serially reusable I/O devices, such as printers and tape drives
- sharable programs and subprocedures
- networks
- sharable data

Security Methods of Operating Systems

- The basis of protection is separation: keeping one user's objects separate from other users.
- **physical separation**, in which different processes use different physical objects
- **temporal separation,** in which processes having different security requirements are executed at different times
- **logical separation,** in which users operate under the illusion that no other processes exist, as when an operating system constrains a program's accesses so that the program cannot access objects outside its permitted domain

- **cryptographic separation,** in which processes conceal their data and computations in such a way that they are unintelligible to outside processes
- An operating system can support separation and sharing in several ways, offering protection at any of several levels.
- **Do not protect.** Operating systems with no protection are appropriate when sensitive procedures are being run at separate times.
- **Isolate.** When an operating system provides isolation, different processes running concurrently are unaware of the presence of each other. Each process has its own address space, files, and other objects. The operating system must confine each process somehow so that the objects of the other processes are completely concealed.
- **Share all or share nothing.** The owner of an object declares it to be public or private. A public object is available to all users, whereas a private object is available only to its owner.
- **Share via access limitation.** With protection by access limitation, the operating system checks the allowability of each user's potential access to an object. That is, access control is implemented for a specific user and a specific object.
- **Share by capabilities.** An extension of limited access sharing, this form of protection allows dynamic creation of sharing rights for objects. The degree of sharing can depend on the owner or the subject, on the context of the computation, or on the object itself.
- **Limit use of an object.** Limits not just the access to an object but the use made of that object after it has been accessed. For example, a user may be allowed to view a sensitive document, but not to print a copy of it.

---

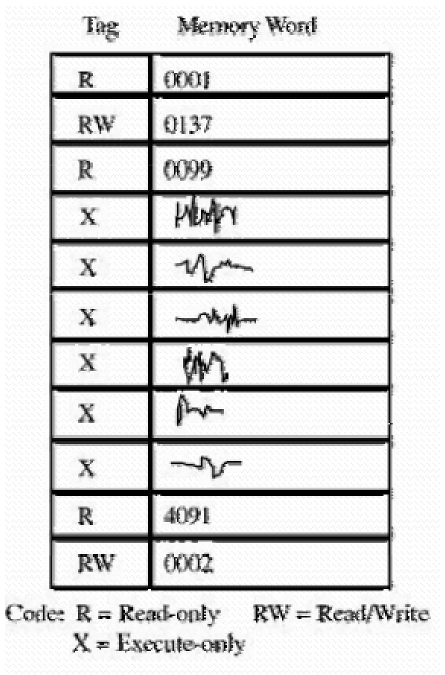
8. Explain the following

a. Tagged Architecture

TAGGED ARCHITECTURE

- Another problem with using base/bounds registers for protection or relocation is their contiguous nature.
- Each pair of registers confines accesses to a consecutive range of addresses.
- A compiler or loader can easily rearrange a program so that all code sections are adjacent and all data sections are adjacent.
- However, in some cases you may want to protect *some* data values but not *all*.
- For example, a personnel record may require protecting the field for salary but not office location and phone number.
- Moreover, a programmer may want to ensure the integrity of certain data values by allowing them to be written when the program is initialized but prohibiting the program from modifying them later.
- This scheme protects against errors in the programmer's own code.
- A programmer may also want to invoke a shared subprogram from a common library.
- These characteristics dictate that one program module must share with another module only the *minimum* amount of data necessary for both of them to do their work.
- Additional, operating-system-specific design features can help, too. Base/bounds registers create an all-or-nothing situation for sharing: Either a program makes all its data available to be accessed and modified or it prohibits access to all.

- Even if there were a third set of registers for shared data, all data would need to be located together.
- A procedure could not effectively share data items *A, B,* and *C* with one module, *A, C,* and *D* with a second, and *A, B,* and *D* with a third.
- The only way to accomplish the kind of sharing we want would be to move each appropriate set of data values to some contiguous space.
- However, this solution would not be acceptable if the data items were large records, arrays, or structures.

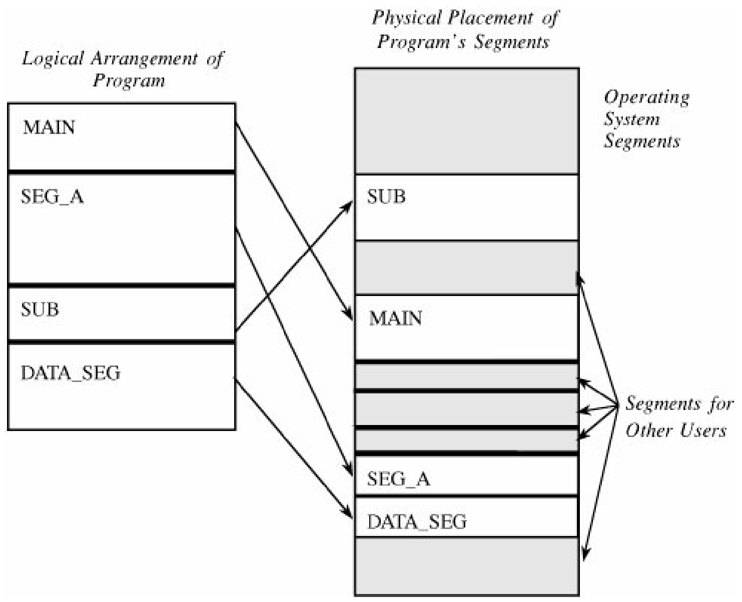


- An alternative is tagged architecture, in which every word of machine memory has one or more extra bits to identify the access rights to that word.
- These access bits can be set only by privileged (operating system) instructions.
- The bits are tested every time an instruction accesses that location.
- For example, as shown in the below figure, one memory location may be protected as execute-only (for example, the object code of instructions), whereas another is protected for fetch-only (for example, read) data access, and another accessible for modification (for example, write).
- In this way, two adjacent locations can have different access rights.
- Furthermore, with a few extra tag bits, different classes of data (numeric, character, address or pointer, and undefined) can be separated, and data fields can be protected for privileged (operating system) access only.
- This protection technique has been used on a few systems, although the number of tag bits has been rather small.
- The Burroughs B6500-7500 system used three tag bits to separate data words (three types), descriptors (pointers), and control words (stack pointers and addressing control words).
- The IBM System/38 used a tag to control both integrity and access.
- A variation used one tag that applied to a group of consecutive locations, such as 128 or 256 bytes.
- With one tag for a block of addresses, the added cost for implementing tags was not as high as with one tag per location.

- The Intel I960 extended architecture processor used a tagged architecture with a bit on each memory word that marked the word as a "capability," not as an ordinary location for data or instructions.
- A tagged architecture would require fundamental changes to substantially all the operating system code, a requirement that can be prohibitively expensive.
- But as the price of memory continues to fall, the implementation of a tagged architecture becomes more feasible.

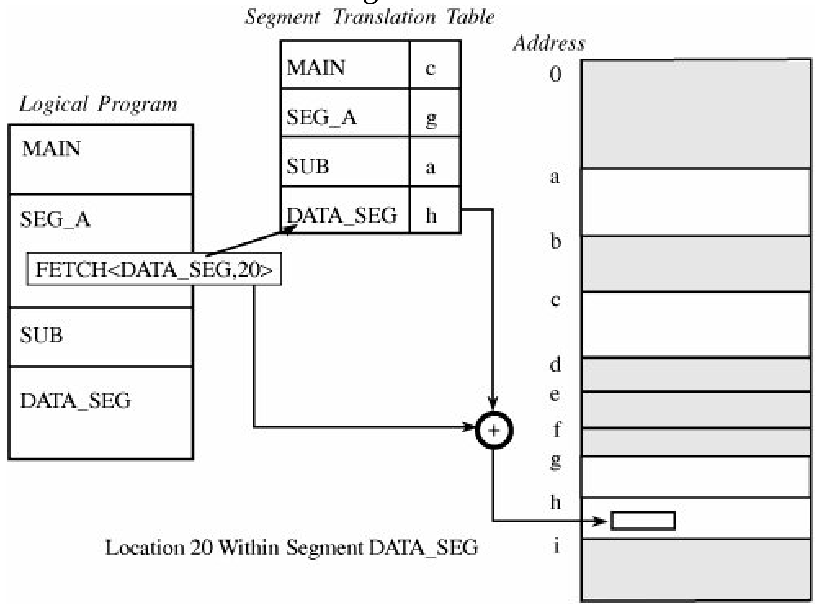
b. <u>Segmentation</u>

- Segmentation, involves the simple notion of dividing a program into separate pieces.
- Each piece has a logical unity, exhibiting a relationship among all of its code or data values.
- For example, a segment may be the code of a single procedure, the data of an array, or the collection of all local data values used by a particular module.
- Segmentation was developed as a feasible means to produce the effect of the equivalent of an unbounded number of base/bounds registers.
- In other words, segmentation allows a program to be divided into many pieces having different access rights.
- Each segment has a unique name.
- A code or data item within a segment is addressed as the pair <name, offset>, where name is the name of the segment containing the data item and offset is its location within the segment (that is, its distance from the start of the segment).
- Logically, the programmer pictures a program as a long collection of segments.
- Segments can be separately relocated, allowing any segment to be placed in any available memory locations.
- The relationship between a logical segment and its true memory position is shown in Figure


Logical Arrangement of Program
MAIN
SEG_A
SUB
DATA_SEG
Physical Placement of Program's Segments
SUB
MAIN
SEG_A
DATA_SEG
Operating System Segments
Segments for Other Users

- The operating system must maintain a table of segment names and their true addresses in memory.
- When a program generates an address of the form <name, offset>, the operating system looks up name in the segment directory and determines its real beginning memory address.

- To that address the operating system adds offset, giving the true memory address of the code or data item.
- This translation is shown in the below Figure.
- For efficiency there is usually one operating system segment address table for each process in execution.
- Two processes that need to share access to a single segment would have the same segment name and address in their segment tables.



- Thus, a user's program does not know what true memory addresses it uses.
- It has no way and no need to determine the actual address associated with a particular <name, offset>.
- The <name, offset> pair is adequate to access any data or instruction to which a program should have access.

This hiding of addresses has three advantages for the operating system.

- The operating system can place any segment at any location or move any segment to any location, even after the program begins to execute.
- A segment can be removed from main memory (and stored on an auxiliary device) if it is not being used currently.
- Every address reference passes through the operating system, so there is an opportunity to check each one for protection.

Segmentation offers these security benefits:

- Each address reference is checked for protection.
- Many different classes of data items can be assigned different levels of protection.
- Two or more users can share access to a segment, with potentially different access rights.
- A user cannot generate an address or access to an unpermitted segment.
- One protection difficulty inherent in segmentation concerns segment size.
- Each segment has a particular size.
- However, a program can generate a reference to a valid segment name, but with an offset beyond the end of the segment.
- For example, a segment might contain a dynamic data structure such as a stack.
- Therefore, secure implementation of segmentation requires checking a generated address to verify that it is not beyond the current end of the segment referenced.

- Although this checking results in extra expense (in terms of time and resources), segmentation systems must perform this check; the segmentation process must maintain the current segment length in the translation table and compare every address generated.
- Thus, we need to balance protection with efficiency, finding ways to keep segmentation as efficient as possible.

Efficient implementation of segmentation presents two problems:

- Segment names are inconvenient to encode in instructions
- The operating system's lookup of the name in a table can be slow.
- To overcome these difficulties, segment names are often converted to numbers by the compiler when a program is translated; the compiler also appends a linkage table matching numbers to true segment names.

---

9. **Discuss in detail about the Firewalls.**

What Is a Firewall?

- A firewall is a device that filters all traffic between a protected or "inside" network and a less trustworthy or "outside" network.
- Usually a firewall runs on a dedicated device; because it is a single point through which traffic is channeled, performance is important, which means nonfirewall functions should not be done on the same machine.
- Because a firewall is executable code, the attacker could compromise that code and execute from the firewall's device.
- Thus, the fewer pieces of code on the device, the fewer tools the attacker would have by compromising the firewall.
- Firewall code usually runs on a proprietary or carefully minimized operating system.

Purpose of a firewall

- The purpose of a firewall is to keep "bad" things outside a protected environment.
- To accomplish that, firewalls implement a security policy that is specifically designed to address what bad things might happen.
- For example, the policy might be to prevent any access from outside (while still allowing traffic to pass from the inside to the outside).
- Alternatively, the policy might permit accesses only from certain places, from certain users, or for certain activities.
- Part of the challenge of protecting a network with a firewall is determining which security policy meets the needs of the installation.
- People in the firewall community (users, developers, and security experts) disagree about how a firewall should work.
- In particular, the community is divided about a firewall's default behavior.
- The two schools of thought as "that which is not expressly forbidden is permitted" (default permit) and "that which is not expressly permitted is forbidden" (default deny) is described.

Design of Firewalls

- That a reference monitor must be always invoked
  - tamperproof
  - small and simple enough for rigorous analysis
- A firewall is a special form of reference monitor.
- By carefully positioning a firewall within a network, we can ensure that all network accesses that we want to control must pass through it.

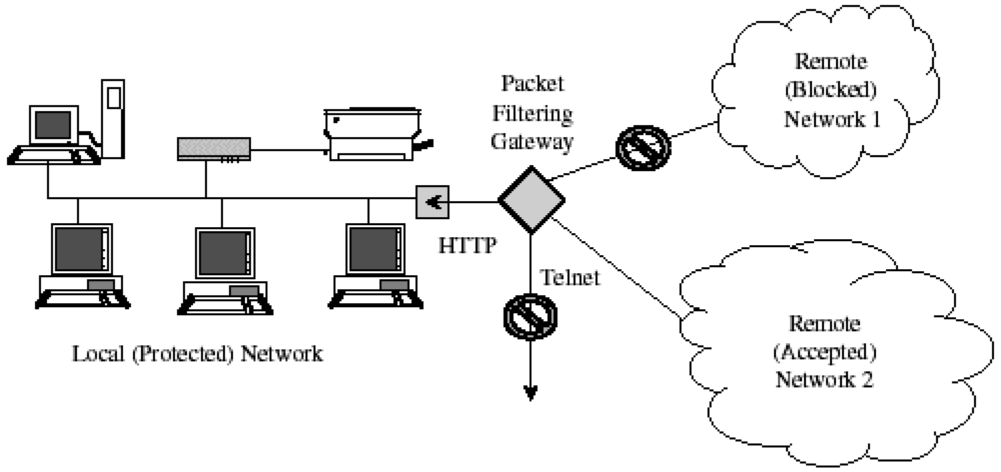
- This restriction meets the "always invoked" condition.
- A firewall is typically well isolated, making it highly immune to modification.
- Usually a firewall is implemented on a separate computer, with direct connections only to the outside and inside networks.
- This isolation is expected to meet the "tamperproof" requirement.
- And firewall designers strongly recommend keeping the functionality of the firewall simple.

Types of Firewalls

- Firewalls have a wide range of capabilities.
    - Packet filtering gateways or screening routers
    - Stateful inspection firewalls
    - Application proxies
    - Guards
    - Personal firewalls

Packet Filtering Gateway

- A packet filtering gateway or screening router is the simplest, and in some situations, the most effective type of firewall.
- A packet filtering gateway controls access to packets based on packet address (source or destination) or specific transport protocol type (such as HTTP web traffic).
- But a separate firewall behind (on the local side) of the router can screen traffic before it gets to the protected network.
- Figure below shows a packet filter that blocks access from (or to) addresses in one network; the filter allows HTTP traffic but blocks traffic using the Telnet protocol.


Packet Filtering Gateway
Remote (Blocked) Network 1
HTTP
Telnet
Local (Protected) Network
Remote (Accepted) Network 2

Stateful Inspection Firewall

- Filtering firewalls work on packets one at a time, accepting or rejecting each packet and moving on to the next.
- They have no concept of "state" or "context" from one packet to the next.
- A stateful inspection firewall maintains state information from one packet to another in the input stream.

Application Proxy

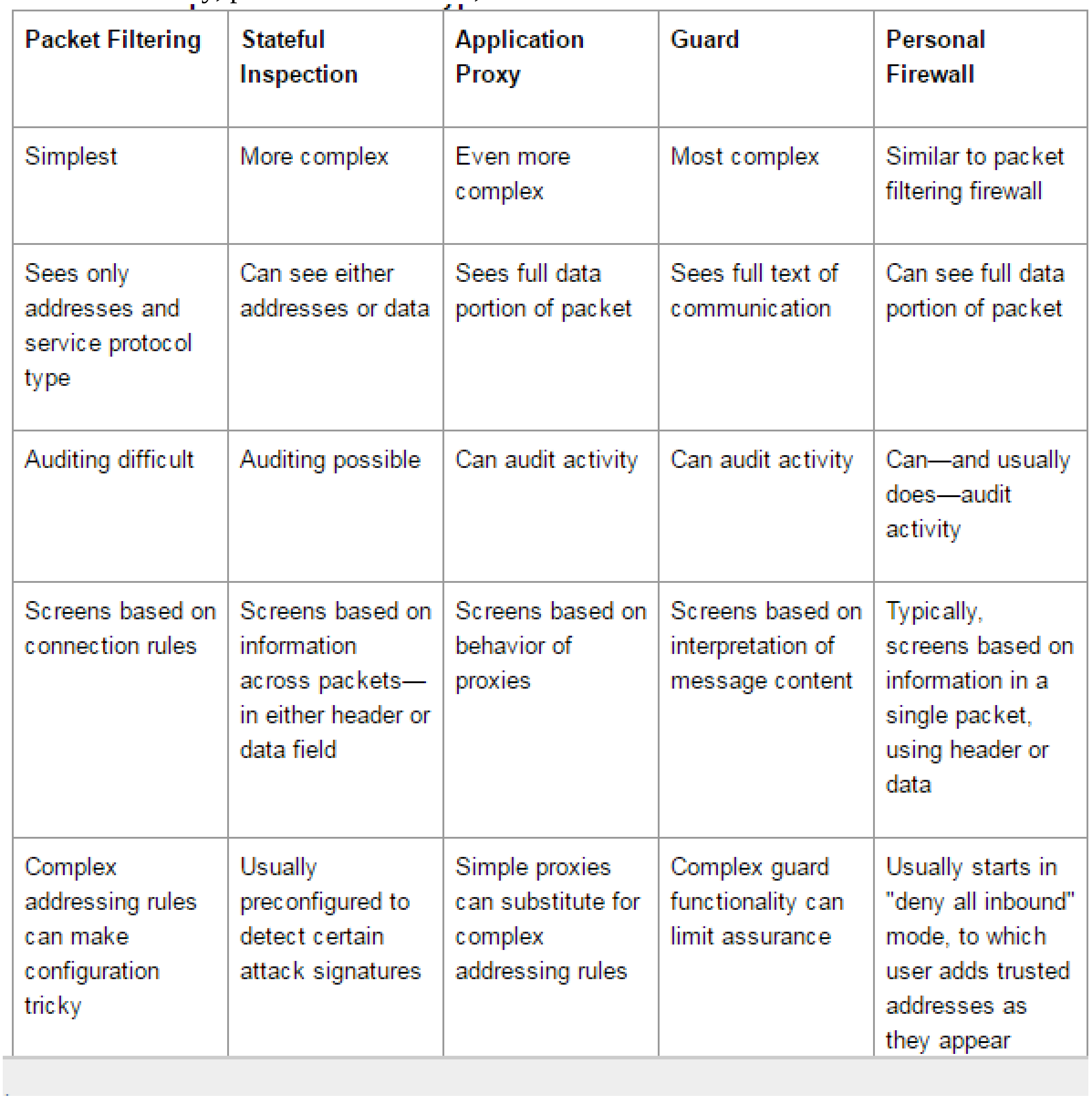
- An application proxy gateway, also called a bastion host, is a firewall that simulates the (proper) effects of an application so that the application will receive only requests to act properly.
- A proxy gateway is a two-headed device: It looks to the inside as if it is the outside (destination) connection, while to the outside it responds just as the insider would.

Real purpose of a proxy gateway example.

- A school wants to allow its students to retrieve any information from World Wide Web resources on the Internet. To help provide efficient service, it wants to know what sites have been visited, and what files from those sites have been fetched; particularly popular files will be cached locally.

<u>Guard</u>

- A guard is a sophisticated firewall.
- Like a proxy firewall, it receives protocol data units, interprets them, and passes through the same or different protocol data units that achieve either the same result or a modified result.
- The guard decides what services to perform on the user's behalf in accordance with its available knowledge, such as whatever it can reliably know of the (outside) user's identity, previous interactions, and so forth.

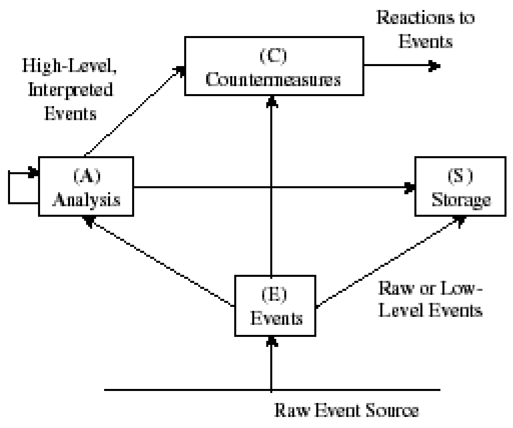


Packet Filtering	Stateful Inspection	Application Proxy	Guard	Personal Firewall
Simplest	More complex	Even more complex	Most complex	Similar to packet filtering firewall
Sees only addresses and service protocol type	Can see either addresses or data	Sees full data portion of packet	Sees full text of communication	Can see full data portion of packet
Auditing difficult	Auditing possible	Can audit activity	Can audit activity	Can—and usually does—audit activity
Screens based on connection rules	Screens based on information across packets—in either header or data field	Screens based on behavior of proxies	Screens based on interpretation of message content	Typically, screens based on information in a single packet, using header or data
Complex addressing rules can make configuration tricky	Usually preconfigured to detect certain attack signatures	Simple proxies can substitute for complex addressing rules	Complex guard functionality can limit assurance	Usually starts in "deny all inbound" mode, to which user adds trusted addresses as they appear

10. <u>Discuss in detail about the Intrusion detection Systems.</u>

Intrusion Detection Systems

- Prevention, although necessary, is not a complete computer security control; detection during an incident copes with harm that cannot be prevented in advance.
- An intrusion detection system (IDS) is a device, typically another separate computer that monitors activity to identify malicious or suspicious events.
- An IDS is a sensor, like a smoke detector, that raises an alarm if specific things occur.
- ***A model of IDS is shown in Figure.***



- The components in the figure are the four basic elements of an intrusion detection system, based on the Common Intrusion Detection Framework.
- An IDS receives raw inputs from sensors.
- It saves those inputs, analyzes them, and takes some controlling action.

IDSs perform a variety of functions:

- monitoring users and system activity
- auditing system configuration for vulnerabilities and misconfigurations
- assessing the integrity of critical system and data files
- recognizing known attack patterns in system activity
- identifying abnormal activity through statistical analysis
- managing audit trails and highlighting user violation of policy or normal activity
- correcting system configuration errors
- installing and operating traps to record information about intruders

- No one IDS performs all of these functions. Let us look more closely at the kinds of IDSs and their use in providing security.

Types of IDSs

- The two general types of intrusion detection systems are signature based and heuristic.
- **Signature-based intrusion detection systems** perform simple pattern-matching and report situations that match a pattern corresponding to a known attack type.
- **Heuristic intrusion detection systems**, also known as anomaly based, build a model of acceptable behavior and flag exceptions to that model; for the future, the administrator can mark a flagged behavior as acceptable so that the heuristic IDS will now treat that previously unclassified behavior as acceptable.
- Intrusion detection devices can be **network based or host based**. A network-based IDS is a stand-alone device attached to the network to monitor traffic throughout that network; a host-based IDS runs on a single workstation or client or host, to protect that one host.

Signature-Based Intrusion Detection

- A simple signature for a known attack type might describe a series of TCP SYN packets sent to many different ports in succession and at times close to one another, as would be the case for a port scan.
- An intrusion detection system would probably find nothing unusual in the first SYN, say, to port 80, and then another (from the same source address) to port 25.
- But as more and more ports receive SYN packets, especially ports that are not open, this pattern reflects a possible port scan.
- Similarly, some implementations of the protocol stack fail if they receive an ICMP packet with a data length of 65535 bytes, so such a packet would be a pattern for which to watch.
- The problem with signature-based detection is the signatures themselves.
- An attacker will try to modify a basic attack in such a way that it will not match the known signature of that attack.
- For example, the attacker may convert lowercase to uppercase letters or convert a symbol such as "blank space" to its character code equivalent %20.
- The IDS must necessarily work from a canonical form of the data stream in order to recognize that %20 matches a pattern with a blank space.
- The attacker may insert malformed packets that the IDS will see, to intentionally cause a pattern mismatch; the protocol handler stack will discard the packets because of the malformation.
- Each of these variations could be detected by an IDS, but more signatures require additional work for the IDS, which reduces performance.
- Of course, signature-based IDSs cannot detect a new attack for which a signature is not yet installed in the database.
- Every attack starts as a new attack at some time, and the IDS is helpless to warn of its existence.
- Ideally, signatures should match every instance of an attack, match subtle variations of the attack, but not match traffic that is not part of an attack.

IDS Strengths and Limitations

- Intrusion detection systems are evolving products.
- Research began in the mid-1980s and products had appeared by the mid-1990s.
- They are becoming cheaper and easier to administer.
- IDS limitation is its sensitivity, which is difficult to measure and adjust.
- IDSs will never be perfect, so finding the proper balance is critical.
- An IDS does not run itself; someone has to monitor its track record and respond to its alarms.

---

11. **<u>Explain in detail about the Data privacy principles.</u>**

Privacy Principles and Policies

- In the United States, interest in privacy and computer databases dates back at least to the early 1970s.
- Thus people in the United States were sensitive about privacy at that time.

Fair Information Policies

- In 1973 Willis Ware of the RAND Corporation chaired a committee to advise the Secretary of the U.S. Department of Human Services on privacy issues.

The report proposes a set of principles of fair information practice.

- **Collection limitation**. Data should be obtained lawfully and fairly.
- **Data quality**. Data should be relevant to their purposes, accurate, complete, and up-to-date.

- **Purpose specification**. The purposes for which data will be used should be identified and the data destroyed if no longer necessary to serve that purpose.
- **Use limitation**. Use for purposes other than those specified is authorized only with consent of the data subject or by authority of law.
- **Security safeguards**. Procedures to guard against loss, corruption, destruction, or misuse of data should be established.
- **Openness.** It should be possible to acquire information about the collection, storage, and use of personal data systems.
- **Individual participation**. The data subject normally has a right to access and to challenge data relating to her.
- **Accountability.** A data controller should be designated and accountable for complying with the measures to give effect to the principles.
- Turn and Ware consider protecting the data themselves, recognizing that collections of data will be attractive targets for unauthorized access attacks.

They suggest four ways to protect stored data:

- Reduce exposure by limiting the amount of data maintained, asking for only what is necessary and using random samples instead of complete surveys.
- Reduce data sensitivity by interchanging data items or adding subtle errors to the data (and warning recipients that the data have been altered).
- Anonymize the data by removing or modifying identifying data items.
- Encrypt the data.

U.S. Privacy Laws

- The Ware committee report led to the 1974 Privacy Act (5 USC 552a), which embodies most of these principles, although that law applies only to data maintained by the U.S. government.
- The Privacy Act is a broad law, covering all data collected by the government.
- It is the strongest U.S. privacy law because of its breadth: It applies to all personal data held anywhere in the government.
- The United States subsequently passed laws protecting data collected and held by other organizations, but these laws apply piecemeal, by individual data type.
- Consumer credit is addressed in the Fair Credit Reporting Act, healthcare information in the Health Insurance Portability and Accountability Act (HIPAA), financial service organizations in the GrammLeachBliley Act (GLBA), children's web access in the Children's Online Privacy Protection Act (COPPA), and student records in the Federal Educational Rights and Privacy Act.
- Laws and regulations do help in some aspects of privacy protection.

Controls on U.S. Government Web Sites

- The Federal Trade Commission (FTC) has jurisdiction over web sites, including those of the federal government, that solicit potentially private data.
- In 2000, the FTC set requirements for privacy policy for government web sites.
- Because government web sites are covered by the Privacy Act, it was easy for the FTC to require privacy protection.
- ***The FTC determined that in order to obey the Privacy Act, government web sites would have to address five privacy factors.***
- **Notice.** Data collectors must disclose their information practices before collecting personal information from consumers.
- **Choice.** Consumers must be given a choice as to whether and how personal information collected from them may be used.

- **Access.** Consumers should be able to view and contest the accuracy and completeness of data collected about them.
- **Security.** Data collectors must take reasonable steps to ensure that information collected from consumers is accurate and secure from unauthorized use.
- **Enforcement.** A reliable mechanism must be in place to impose sanctions for noncompliance with these fair information practices.

Controls on Commercial Web Sites

- FTC can take legal action in companies that engage in deceptive practices

Example

- 2005 CartManager International – runs web shopping cart software was sued by FTC because they sold customer data

Non-U.S. Privacy Principles

- 1981 Council of Europe adopted Convention 108 to protect individual data
- 1995 European Union adopted Directive 95/46/EC , also called European Privacy Directive
- Individual data should be:
- processed fairly and lawfully
- collected for specified, explicit and legitimate purposes
- adequate, relevant, and not excessive in relation to the purposes for which they are collected
- accurate
- kept in a form that permits identification of data subjects for no longer than is necessary

Identity Theft

- Taking another person's identity
- Credit card
- Drivers license

---

12. **<u>Discuss in detail about the laws and compliance for the information to be protected legally.</u>**

Protecting Information

- Clearly, current laws are inadequate for protecting the information itself and for protecting electronically based forms of commerce.
- Copyrights, patents, and trade secrets cover some, but not all, issues related to information.

Criminal and Civil Law

- Statutes are laws that state explicitly that certain actions are illegal.
- A statute is the result of a legislative process by which a governing body declares that the new law will be in force after a designated time.
- For example, the parliament may discuss issues related to taxing Internet transactions and pass a law about when relevant taxes must be paid.
- Often, a violation of a statute will result in a criminal trial, in which the government argues for punishment because an illegal act has harmed the desired nature of society.
- For example, the government will prosecute a murder case because murder violates a law passed by the government.
- In the United States, criminal transgressions are severe, and the law requires that the judge or jury find the accused guilty beyond reasonable doubt.
- For this reason, the evidence must be strong and compelling.

- The goal of a criminal case is to punish the criminal, usually by depriving him or her of rights in some way (such as putting the criminal in prison or assessing a fine).

Civil Law

- Civil law is a different type of law, not requiring such a high standard of proof of guilt.
- In a civil case, an individual, organization, company, or group claims it has been harmed. The goal of a civil case is restitution: to make the victim "whole" again by repairing the harm.
- For example, suppose Fred kills John. Because Fred has broken a law against murder, the government will prosecute Fred in criminal court for having broken the law and upsetting the order of society.

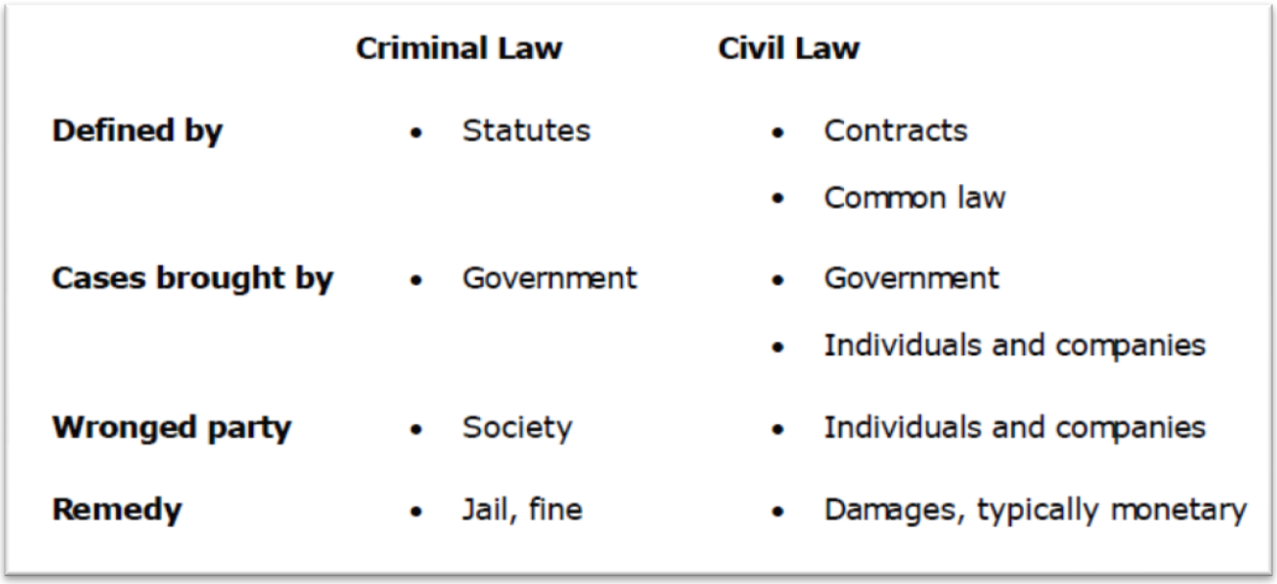
Tort Law

- Special legal language describes the wrongs treated in a civil case. T
- he language reflects whether a case is based on breaking a law or on violating precedents of behavior that have evolved over time. In other words, sometimes judges may make determinations based on what is reasonable and what has come before, rather than on what is written in legislation.
- A tort is harm not occurring from violation of a statute or from breach of a contract but instead from being counter to the accumulated body of precedents.
- Thus, statute law is written by legislators and is interpreted by the courts; tort law is unwritten but evolves through court decisions that become precedents for cases that follow.
- The basic test of a tort is what a reasonable person would do.
- Fraud is a common example of tort law in which, basically, one person lies to another, causing harm
- Computer information is perfectly suited to tort law.
- The court merely has to decide what is reasonable behavior, not whether a statute covers the activity.
- For example, taking information from someone without permission and selling it to someone else as your own is fraud.

Contract Law

- A third form of protection for computer objects is contracts.
- A contract is an agreement between two parties.
- A contract must involve three things:
    - an offer
    - an acceptance
    - a consideration
- One party offers something: "I will write this computer program for you for this amount of money."
- The second party can accept the offer, reject it, make a counter offer, or simply ignore it.
- In reaching agreement with a contract, only an acceptance is interesting; the rest is just the history of how agreement was reached.
- A contract must include consideration of money or other valuables.
- The basic idea is that two parties exchange things of value, such as time traded for money or technical knowledge for marketing skills.

The differences between criminal and civil law are summarized in



	Criminal Law	Civil Law
Defined by	• Statutes	• Contracts • Common law
Cases brought by	• Government	• Government • Individuals and companies
Wronged party	• Society	• Individuals and companies
Remedy	• Jail, fine	• Damages, typically monetary

Compliance with Privacy Laws

- The first laws that addressed the protection of personal data were adopted in the United States in the late 1960's.
- Since that time, the Federal government and the States have passed hundreds of laws and regulations that pertain to the collection, processing, sharing or use of personal data.
  - Electronic Communications Privacy Act (ECPA);
  - Fair Credit Reporting Act (FCRA);
  - Fair and Accurate Credit Transaction Act (FACTA);
  - Health Insurance Portability and Accountability Act (HIPAA) and the related privacy rules;
  - Children Online Privacy Protection Act (COPPA);
  - Gramm Leach Bliley Act and the related privacy rules;
  - CAN SPAM Act;
  - Telephone Customer Protection Act;
  - Telephone Sales Rule;
  - Federal and State Unfair and Deceptive Practices Acts;
  - State Online Privacy Protection Act;
  - State medical information laws;
  - State driver's license laws;