UNIT – I DATABASE MODELLING, MANAGEMENT AND DEVELOPMENT

1. DATABASE DESIGN AND MODELLING

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

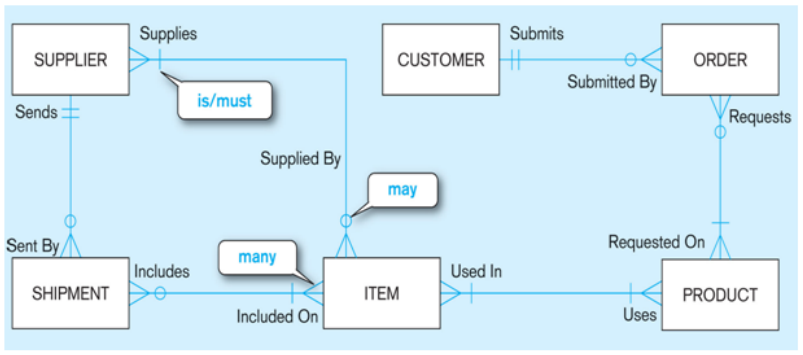
- An entity-relationship model (E-R model) is a detailed, logical representation of the data for an organization or for a business area.
- The E-R model is expressed in terms of
  - entities in the business environment
  - the relationships (or associations) among those entities
  - the attributes (or 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.

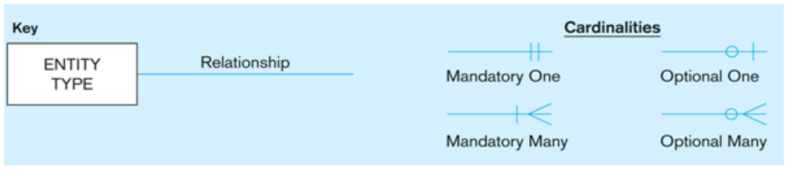
SAMPLE ER MODEL

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

1.1 MODELING ENTITIES AND ATTRIBUTES

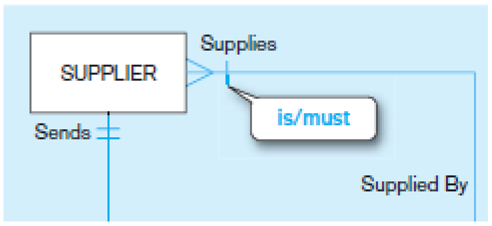
- The basic constructs of the E-R model are entities, relationships, and attributes.
- The richness of the E-R model allows designers to model real-world situations accurately and expressively, which helps account for the popularity of the model.

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.

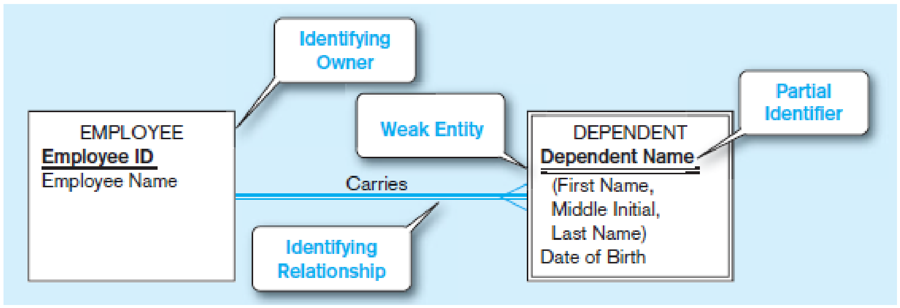
Entity type EMPLOYEE with two instances

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

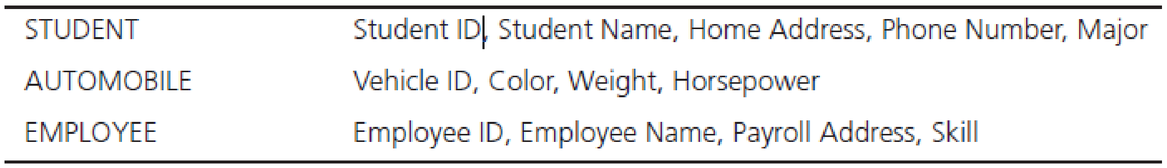
There are a few special guidelines for *naming* entity types, which follow:

- 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:

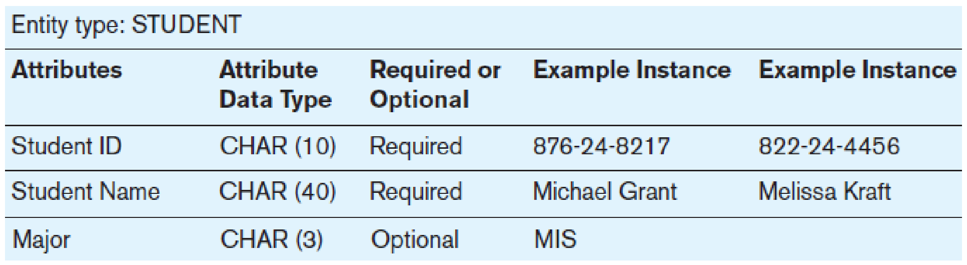


STUDENT	Student ID, Student Name, Home Address, Phone Number, Major
AUTOMOBILE	Vehicle ID, Color, Weight, Horsepower
EMPLOYEE	Employee ID, Employee Name, Payroll Address, Skill

- 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.

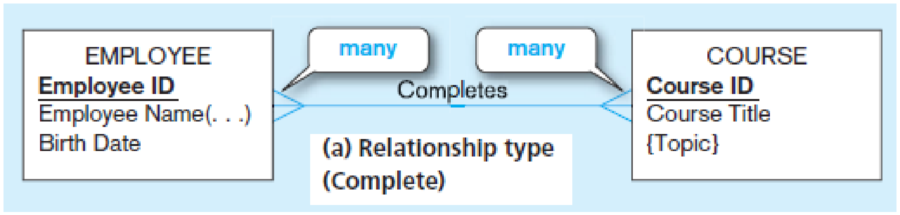
NAMING AND DEFINING ATTRIBUTES

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

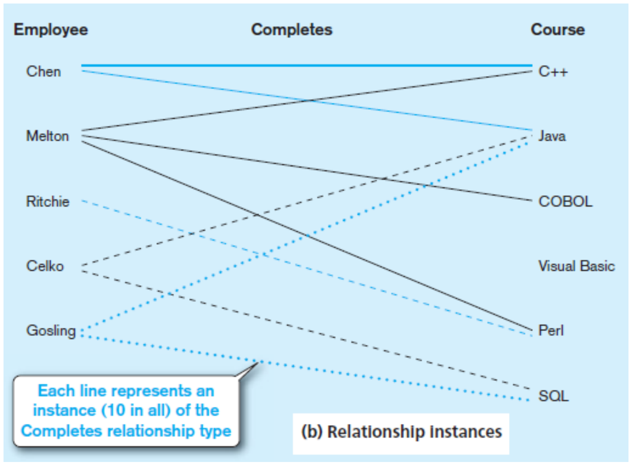
- An attribute name is a *singular noun or noun phrase* (such as Customer ID, Age, or Major).
- An attribute name should be *unique*. No two attributes of the same entity type may have the same name
- *Each attribute name should follow a standard format*

1.2 MODELING RELATIONSHIP AND THEIR BASIC CONCEPTS

- 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).
- 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.



(a) Relationship type (Complete)


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

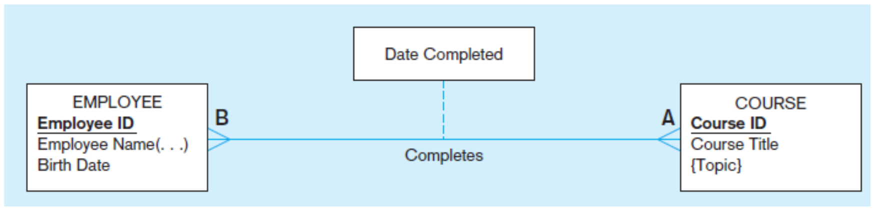
BASIC CONCEPTS AND DEFINITIONS IN RELATIONSHIPS

- 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

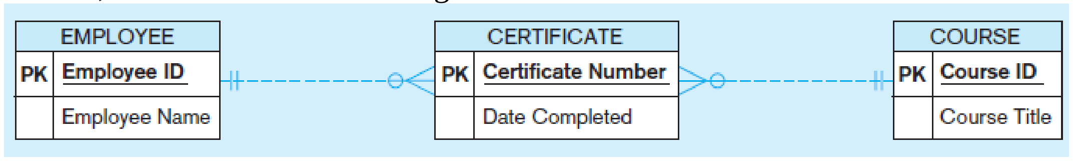
- Example, suppose the organization wishes to record the date (month andyear) when an employee completes each course.
- This attribute is named Date Completed.
  In the above diagram Date Completed has not been associated with either the EMPLOYEE or COURSE entity.
- That is because Date Completed is 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

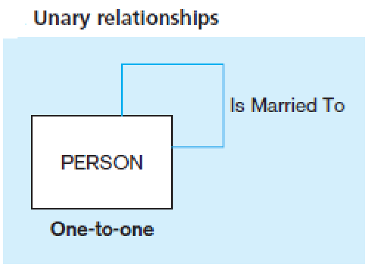


DEGREE OF A RELATIONSHIP

- 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).

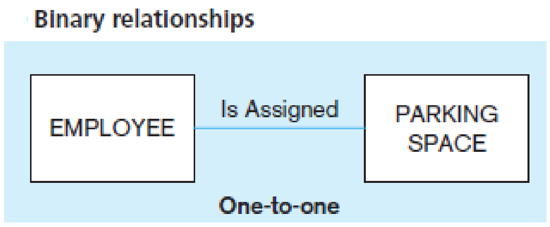
UNARY RELATIONSHIP

- 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.



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.

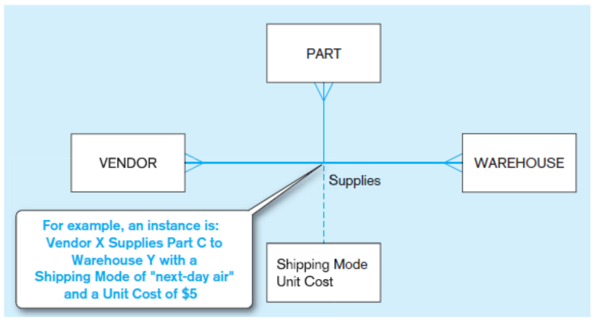


TERNARY RELATIONSHIP

- 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.

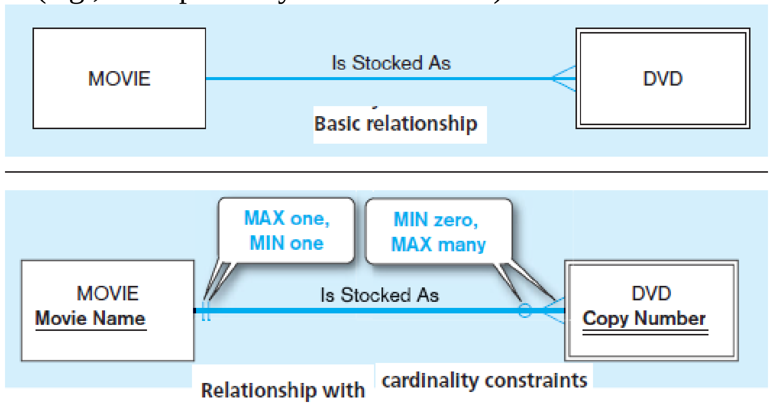
Ternary relationship


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.

1.3 <u>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.

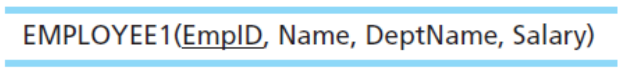
- The first of these, at IBM's San Jose Research Laboratory, led to the development of System R.
- The second, at the University of California at Berkeley, led to the development of an academically oriented RDBMS. Commercial
- RDBMS products from numerous vendors started to appear about 1980.
- Today 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(<u>EmpID</u>, 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:

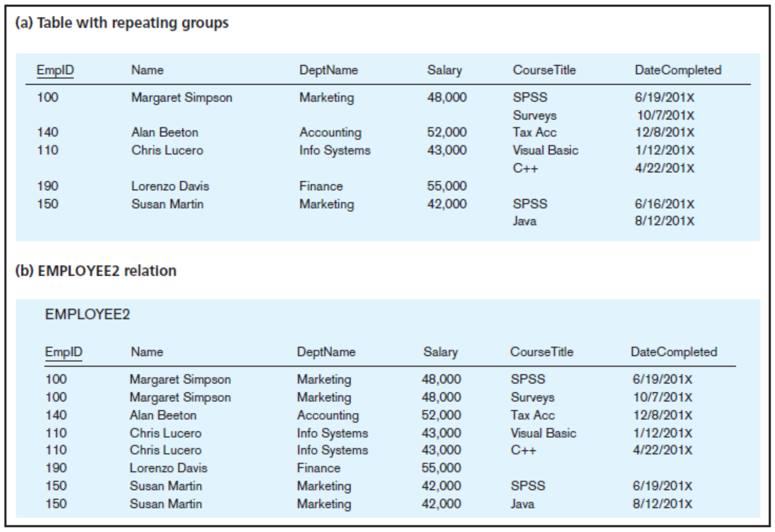
- Each relation (or table) in a database has a unique name.
- An entry at the intersection of each row and column is atomic (or single valued).
  - There can be only one value associated with each attribute on a specific row of a table; no multivalued attributes are allowed in a relation.
- Each row 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

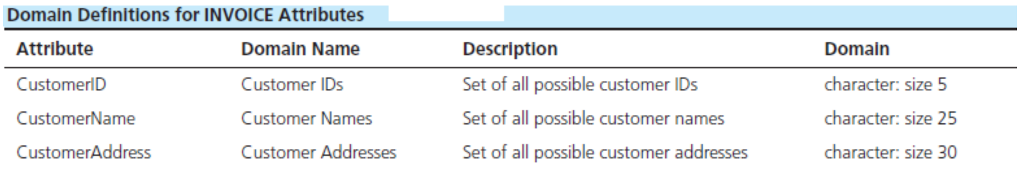
1.4 RELATIONAL DATA MODEL CONSTRAINTS

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.

---

2. BUSINESS RULES AND RELATIONSHIP

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.

---

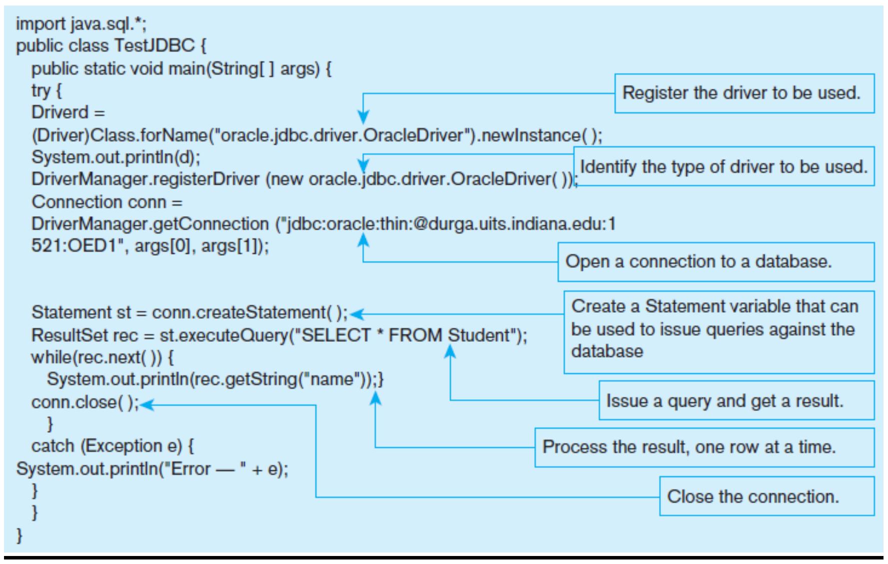
3. JAVA DATABASE CONNECTIVITY (JDBC)

- 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.

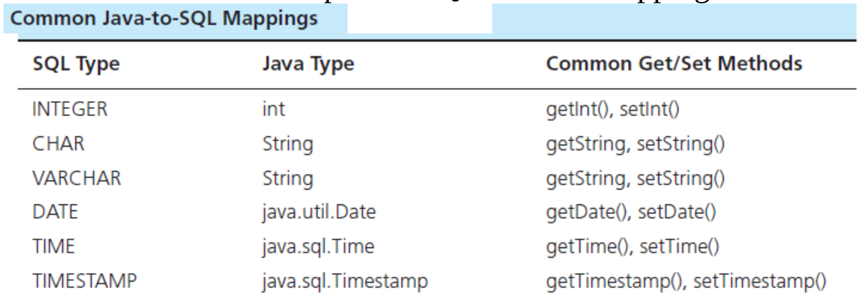
- The ResultSet object has a mechanism, called the cursor that points to its current row of data.
- When the ResultSet object is first initialized, the cursor is positioned before the first row.



```
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.

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



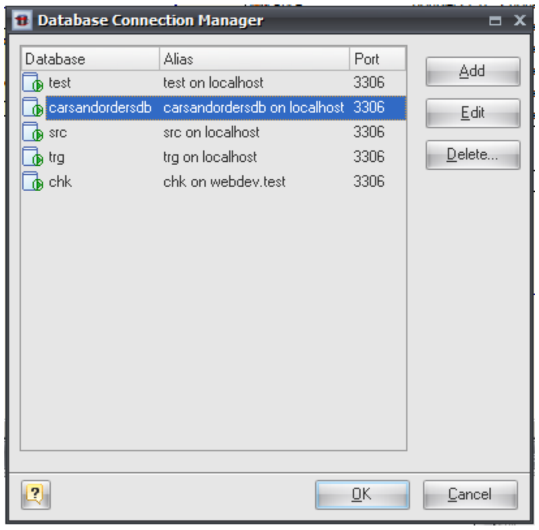
Common Java-to-SQL Mappings

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.

4. 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, 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	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.

5. STORED PROCEDURES

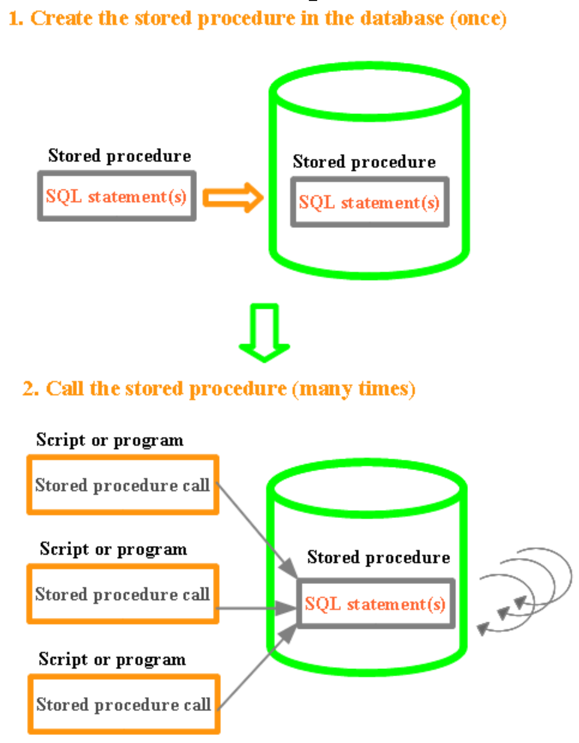
STORED PROCEDURES

- 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:**

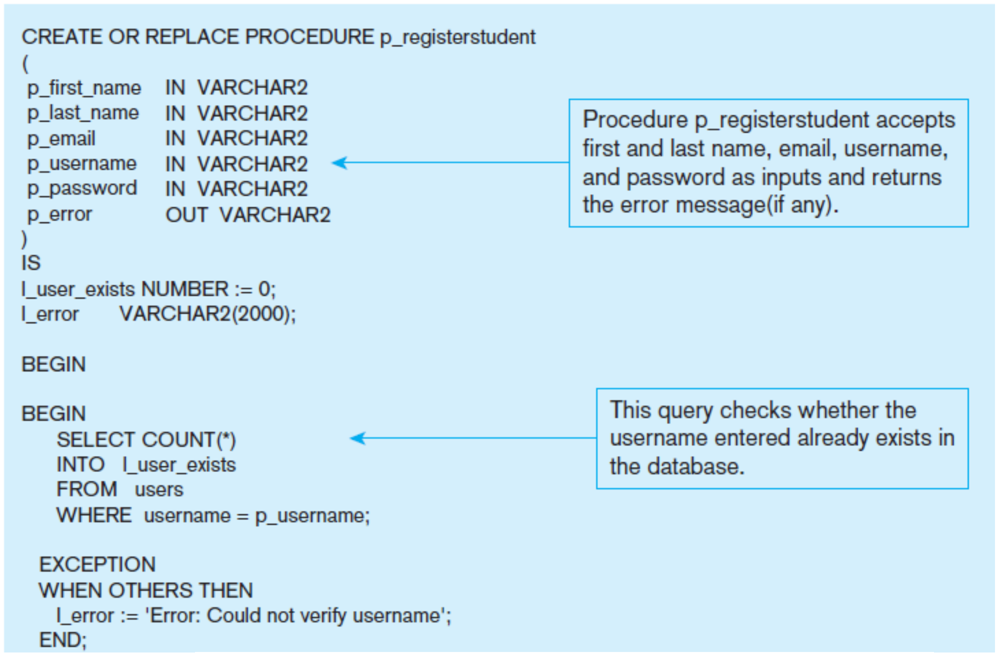


This technique commonly used in a professional environment:

- 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

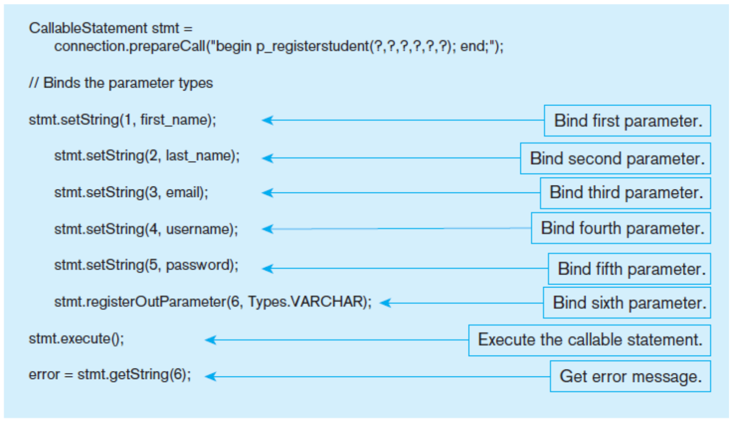


```
CREATE OR REPLACE PROCEDURE p_registerstudent
(
 p_first_name   IN  VARCHAR2
 p_last_name    IN  VARCHAR2
 p_email        IN  VARCHAR2
 p_username     IN  VARCHAR2
 p_password     IN  VARCHAR2
 p_error        OUT  VARCHAR2
)
IS
l_user_exists NUMBER := 0;
l_error      VARCHAR2(2000);

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);
```

6. TRENDS IN BIGDATA SYSTEMS

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

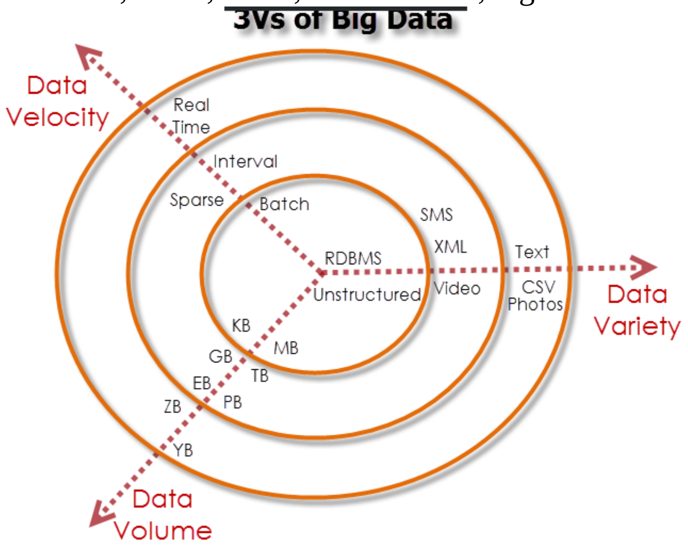
WHAT IS BIG DATA?

- 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

7. <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.

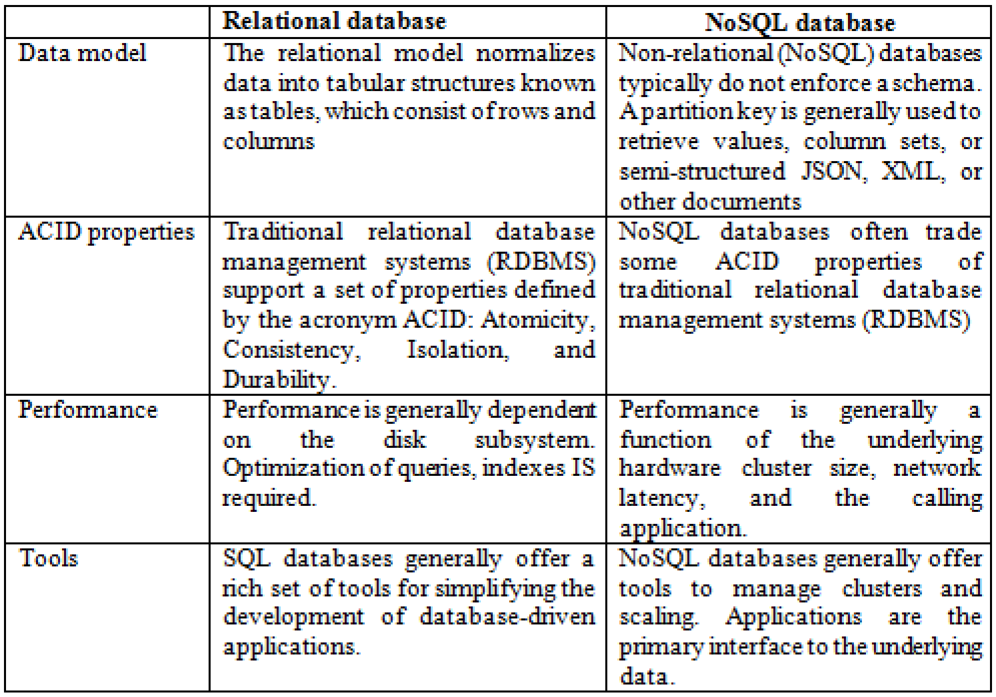
TYPES OF NOSQL DATABASES

- 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.

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.

- **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.

8. 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.

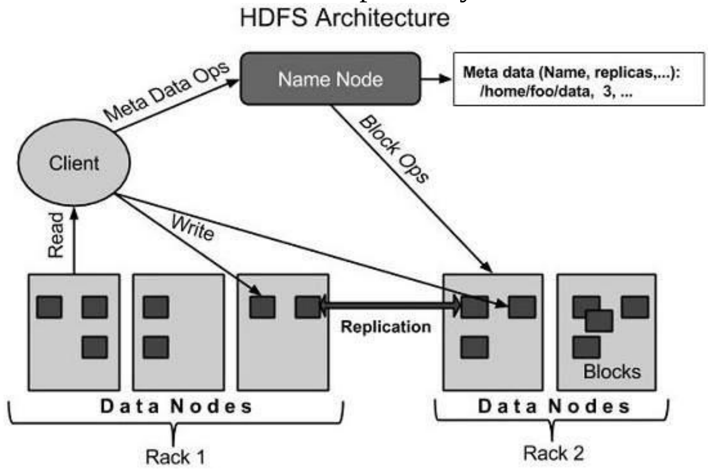
Features of HDFS

- 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
```

9.MAPREDUCE

- 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.

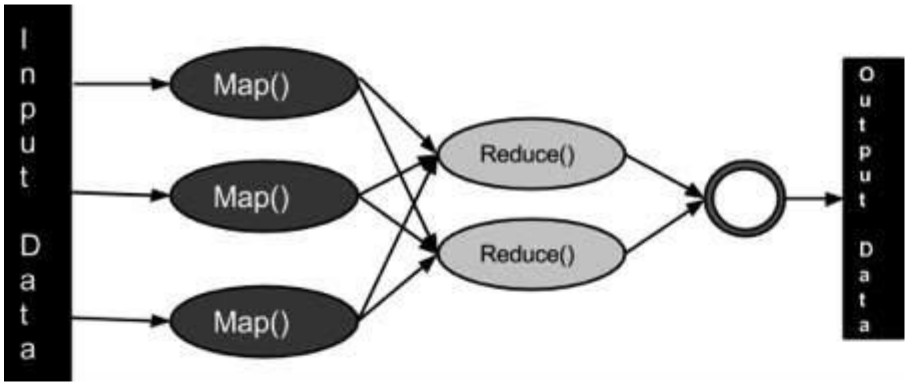
- Map takes a set of data and converts it into another set of data, where individual elements are broken down into tuples (key/value pairs).
- Secondly, reduce task, which takes the output from a map as an input and combines those data tuples into a smaller set of tuples.
- As the sequence of the name 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

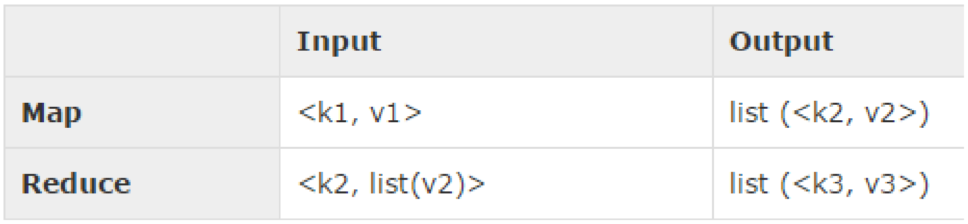
- Generally MapReduce paradigm is based on sending the computer to where the data resides!
- MapReduce program executes in three stages, namely map stage, shuffle stage, and reduce stage.
  - **Map stage** : The map or mapper's job is to process the input data. Generally the input data is in the form 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).



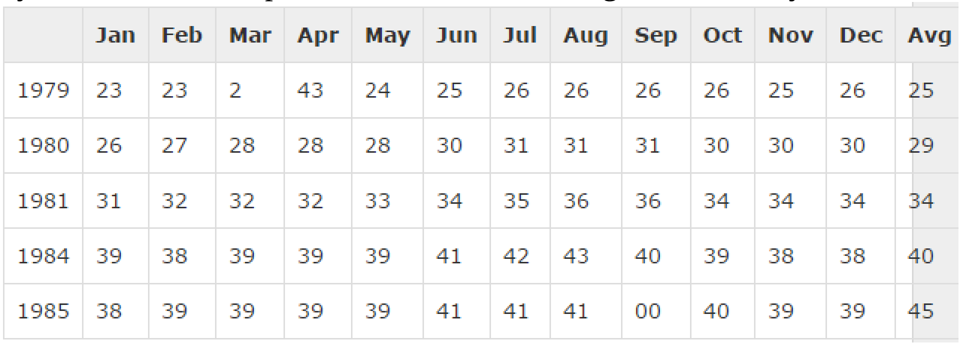
	Input	Output
Map	<k1, v1>	list (<k2, v2>)
Reduce	<k2, list(v2)>	list (<k3, v3>)

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
```

10. HIVE

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

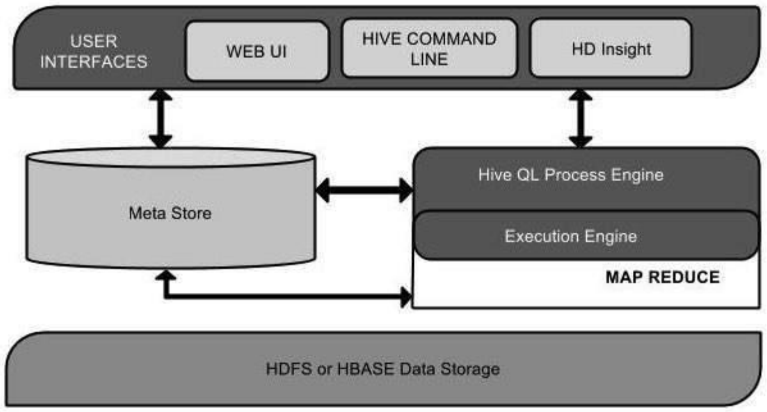
- A relational database
- A design for OnLine Transaction Processing (OLTP)
- A language for real-time queries and row-level updates

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.

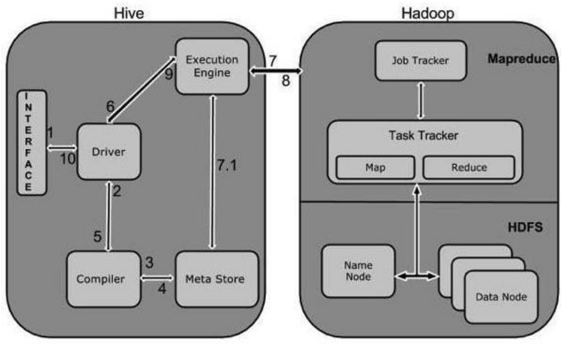
The following table describes each unit:

Unit Name	Operation
User Interface	Hive is a data warehouse infrastructure software

	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

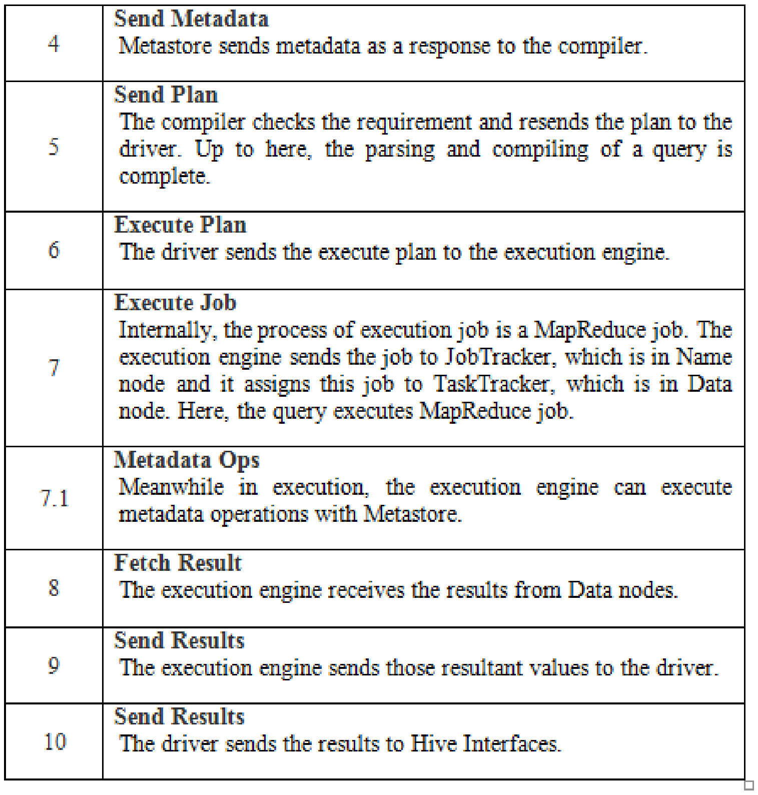
Working of Hive

- 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.