



Data

Analysis

Idea

Lecture 4
**Database Management
System**

Er. Shiva Kunwar
Lecturer, GU

Lesson 1: Introduction to DBMS (5hrs)

1. Overview of Database and DBMS
2. Characteristics and Applications
3. Data Abstraction and Independence
4. Database Users and Administrator
5. Application Architecture
6. **Basics of Database Language (DDL, DML, DCL) + Lab**

Database Language

- A DBMS has appropriate languages and interfaces to express database queries and updates.
- Database languages can be used to read, store and update the data in the database.
- Types of database languages are :
 - **DDL – Data Definition Language**
 - **DML – Data Manipulation Language**
 - **DCL – Data Control Language**
 - **TCL – Transaction Control Language**

DDL – Data Definition Language

Definition: DDL is used to define and manage the structure of the database.

Content:

- Creating tables, views, and indexes.
- Modifying the structure of existing tables.
- Defining constraints (e.g., primary keys, foreign keys).
- Data definition language is used to store the information of metadata like the number of tables and schemas, their names, indexes, columns in each table, constraints, etc.

DDL – Data Definition Language

Here are some tasks that come under DDL:

- **Create:** It is used to create database and its objects in the database.
- **Alter:** It is used to alter the structure of the existing database.
- **Drop:** It is used to delete objects from the database.
- **Truncate:** It is used to remove all records from a table.
- **Rename:** It is used to rename an object.
- **Comment:** It is used to comment on the data dictionary.

These commands are used to update the database schema that's why they come under Data definition language.

DML – Data Manipulation Language

Definition: DML is responsible for manipulating data stored in the database.

- It is used for accessing and manipulating data in a database.
- It handles user requests.
- It includes operations like SELECT, INSERT, UPDATE, and DELETE.
- DML establishes communication between user and database.

DML – Data Manipulation Language

There are two types of DML

(a) **Procedural DML**: user required to specify what data are needed and how they get those data.

(b) **Nonprocedural (Declarative) DML**: user only required to what data needed without specifying how to get those data.

Declarative DMLs are usually easier to learn and use than procedural DMLs.

However, since a user does not have to specify how to get data, the database system must figure out an efficient means of accessing data. The DML component of SQL is nonprocedural.

DML – Data Manipulation Language

Here are some tasks that come under DML

- **Select:** It is used to retrieve data from a database.
- **Insert:** It is used to insert data into a table.
- **Update:** It is used to update existing data within a table.
- **Delete:** It is used to delete all records from a table.
- **Merge:** It performs UPSERT operation, i.e., insert or update operations.
- **Call:** It is used to call a structured query language or a Java subprogram.
- **Explain Plan:** It has the parameter of explaining data or data access path.
- **Lock Table:** It controls concurrency.

DCL – Data Control Language

Definition: DCL is focused on managing access control and permissions within the database.

- It is used to retrieve the stored or saved data.
- It gives different levels of access to the objects in the database.
- The DCL execution is transactional. It also has rollback parameters.
- (But in Oracle database, the execution of data control language does not have the feature of rolling back.)

DCL – Data Control Language

Here are some tasks that come under DCL:

- **Grant:** It is used to give user access privileges to a database.
- **Revoke:** It is used to take back permissions from the user.

There are the following operations which have the authorization of Revoke:

- CONNECT, INSERT, USAGE, EXECUTE, DELETE, UPDATE and SELECT.

TCL – Transaction Control Language

- TCL is used to run the changes made by the DML statement.
- TCL can be grouped into a logical transaction.

Here are some tasks that come under TCL:

- **Commit:** It is used to save the transaction on the database.
- **Rollback:** It is used to restore the database to original since the last Commit.
- **Save Point:** It is used to save the data on the temporary basis in the database

END OF LECTURE 4

- SHIVA.KUNWAR@HOTMAIL.COM
- +977-9819123654

Google classroom code : drbzdcf

PREVIEW FOR LECTURE 5

LAB SESSION
DATA MODEL



Analysis

Idea

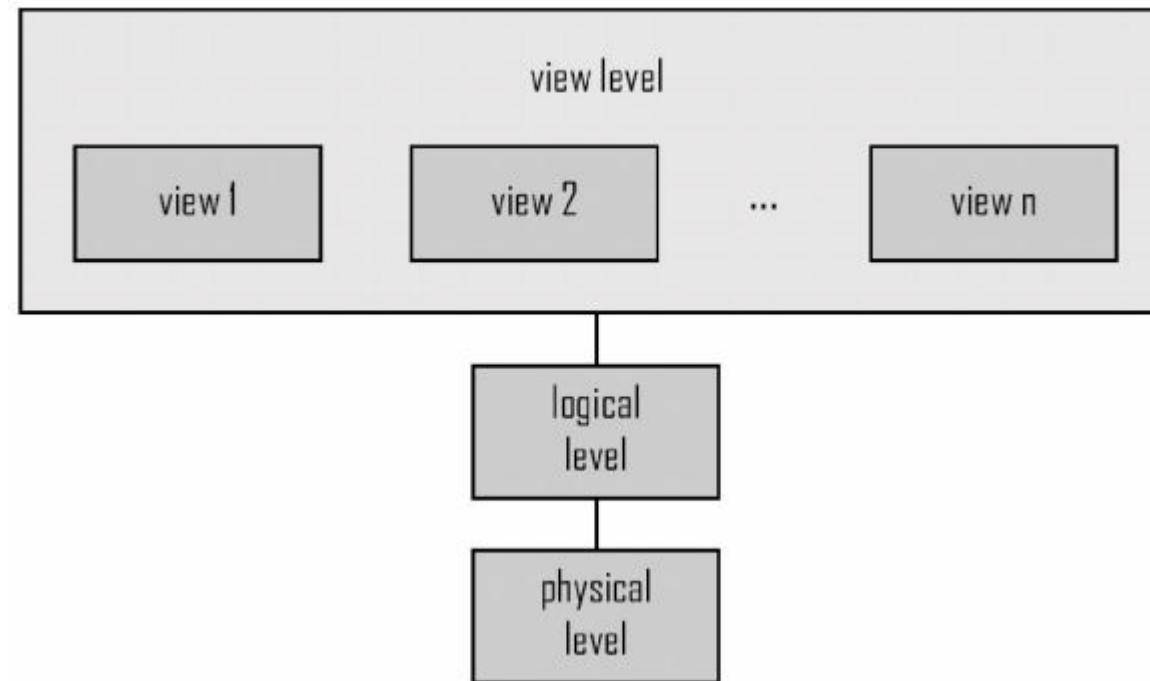
Lecture 5
**Database Management
System**

Er. Shiva Kunwar
Lecturer, GU

Lesson 2: Data Model (6hrs)

- 1. Different Data Model Concepts**
- 2. E-R Model**
- 3. Network/Hierarchical Model**
- 4. Relational Model**
- 5. Entities, Relationships and Attributes**
- 6. E-R Diagrams, Keys**
- 7. Generalization, Specialization and Aggregation**

Data Model Concept

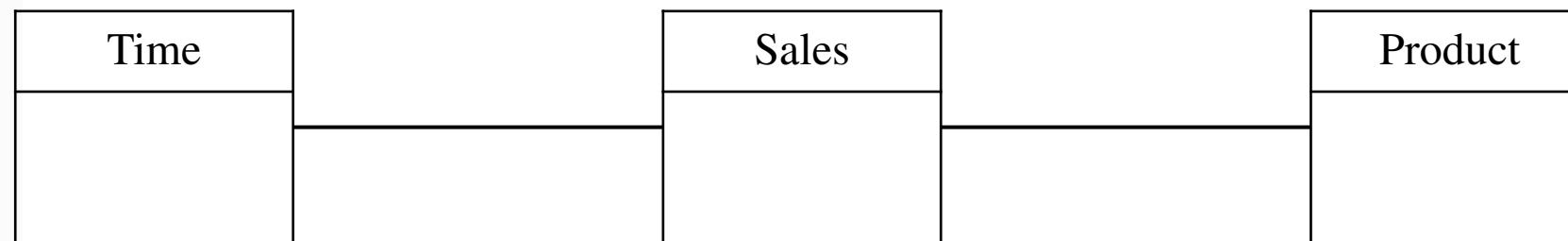


Data Model Concept

- It refers to the collection of concepts that can be used to describe the structure of a database, describing data, data relations, semantics, etc.
- a) Conceptual Data Model**
 - b) Logical Data Model**
 - c) Physical Data Model**

Data Model Concept: Conceptual Data Model

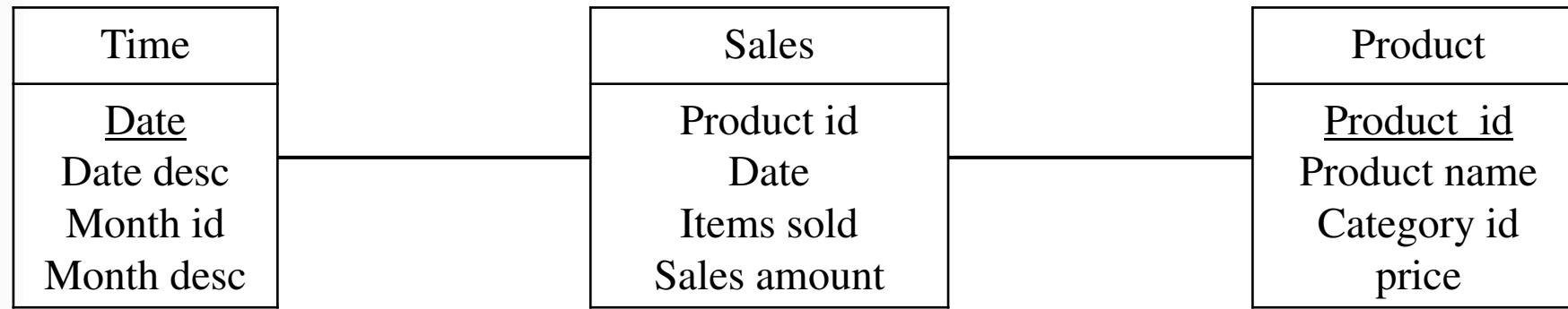
- Only sees entity (table).
- Eagle eye view concept.
- Identify the highest-level relationships between different entities.
- Includes important entities and relationships among them.
- No attributes and primary key are defined.



Data Model Concept: Logical Data Model

- Aka Implementation or Representation data model.
- Describes the data in as much detail as possible without regard to how they will be physically implemented in database.
- It includes all entities and relationships among them.
- All attributes are specified.
- Primary and foreign keys are specified.
- Normalization occurs at this level.

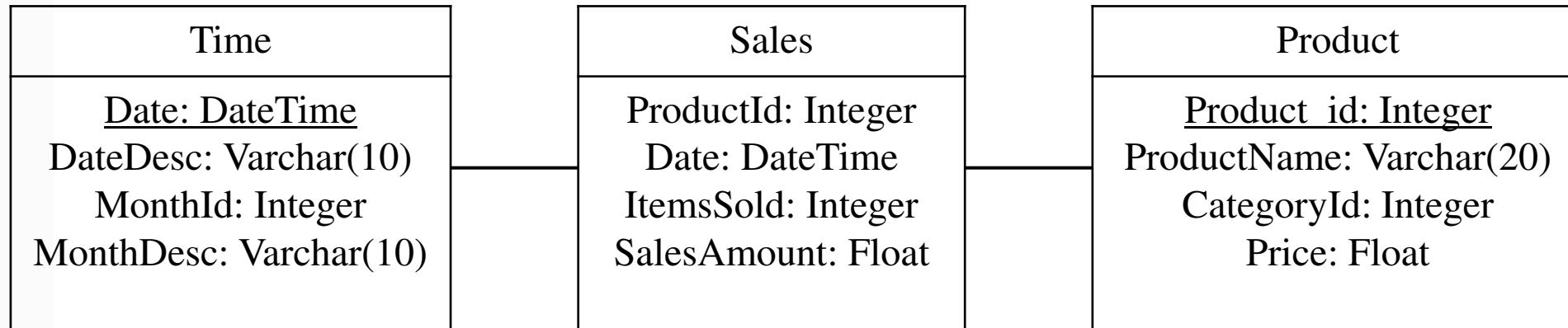
Data Model Concept: Logical Data Model



Data Model Concept: Physical Data Model

- Describes the details of how data are stored in the system.
- The concepts are generally for computer specialists.
- It shows all table structure including column name, column data type, column constraints, primary key, foreign key, relationships, etc.
- Denormalization may occur based on user requirements.

Data Model Concept: Physical Data Model



Data Model Concept

- Complexity increases from conceptual data model to logical data model to physical data model.

Data Model

- Data models describe the underlying structure of database.
- It is a conceptual tool for describing data, relationship among data, data semantics and consistency constraints.
- There are several data models which can be group into three categories.
 - a) Object-based Logical Models.**
 - b) Record-based Logical Models.**
 - c) Physical Data Models.**

Object-based Logical Models

- Object based logical model describe data at the logical and view levels.
- It has flexible structuring capabilities.
- It allows to specify data constraints explicitly.
- Under object-based logical model there are sever data models
 - **Entity-relationship model**
 - **Object-oriented model**
- **Object-based models focus on representing real-world entities and the relationships between them.**

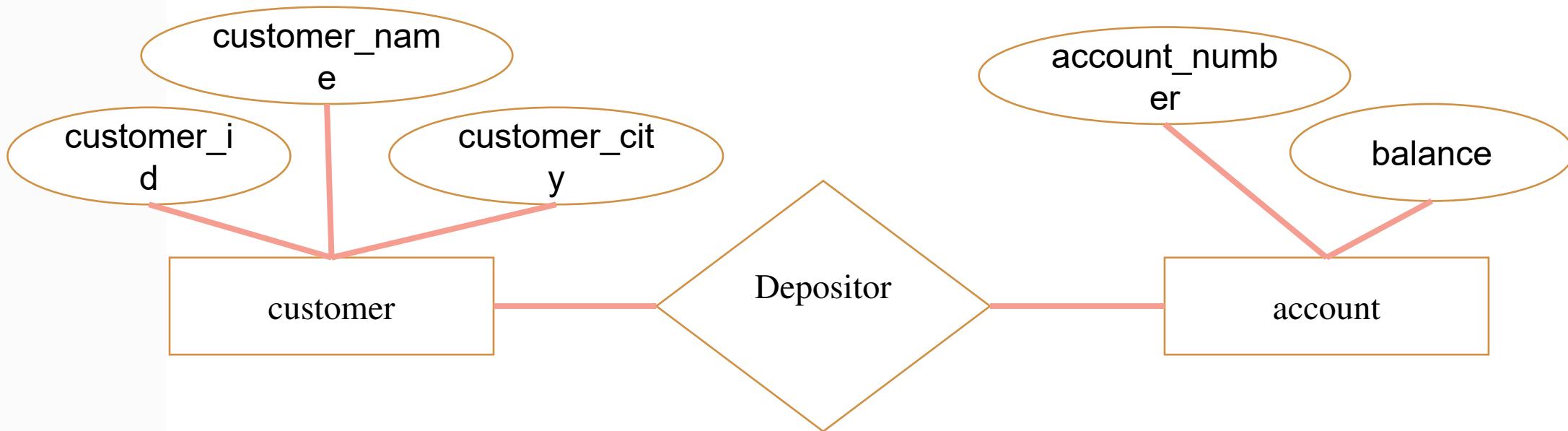
Entity-relationship model

- E-R model describes the design of database in terms of entities and relationship among them.
- An entity is a “thing” or “object” in real world that are distinguishable from other objects.
- An entity is described by a set of attributes. For example
 - Attributes account_number and balance may describe entity “account”.
 - Attributes customer_id, customer_name, customer_city may describe entity “customer”.
- E-R model graphically express overall logical structure of a database by an E-R diagram.

Entity-relationship model

- A relationship is an association among several entities. For example, a depositor relationship associates a customer with each account he or she has.
- The set of all entities of same type called entity set and similarly set of all relationship of the same type called relationship set.
- Components of E-R diagram are as follows:
 - rectangles: represent entity sets
 - ellipses: represent attributes
 - diamonds: represent relationships among entity sets
 - lines: link attributes to entity sets and entity sets to relationships

Entity-relationship model



Object-oriented model

- Object oriented data model is extension to E-R model with the notion of encapsulation, methods (functions) and object identity.
- It is based on collection of objects, like the E-R model.
- An object contains values stored in instance variables within the object.
- These values are themselves objects. That is, objects can contain objects to an arbitrarily deep level of nesting.
- The only way in which one object can access the data of another object is by invoking the method of that other object. This is called sending a message to the object.

Record-based Logical Models

- Record-based logical model also describes data at logical and view level.
- It describes logical structure of database in more detail for implementation point of view.
- It describes database structure in terms of fixed-format records of different types.
- Each table contains records of a particular type.
- And each record type defines fixed number of fields or attributes.
- Each field is usually of a fixed length.
- **Record-based models emphasize the organization of data in records and tables.**

Record-based Logical Models

- The three most widely-accepted models under record-based logical models are:
 - **Relational model**
 - **Network model**
 - **Hierarchical**

Relational model

- Relational model describes database design by a collection of tables (relations).
- It represents both data and their relationships among those data.
- Each table consist number of columns (attributes) with unique names.
- It is a most widely used data model.
- Relational model is lower-level abstraction than E-R model.
- Database model are often carried out in E-R model and then translated into relational mode.

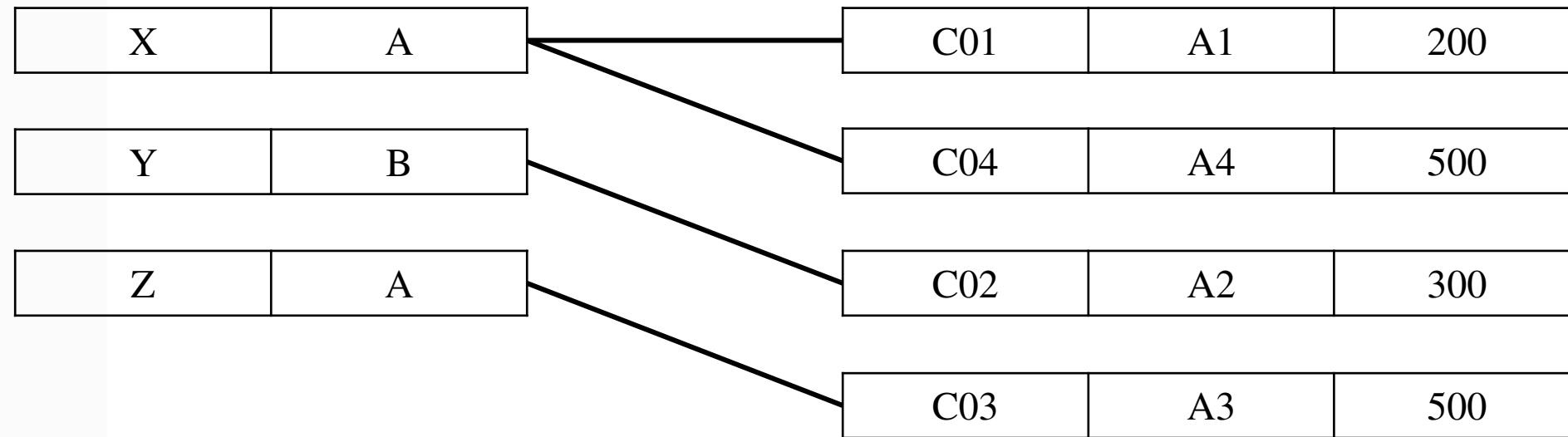
Relational model

- Previous describe E-R model can be expressed in relational model as:

Customer	Customer_na	Customer_city		Account_numbe	balance
_id	me			r	
C01	X	A		A1	200
C02	Y	B		A2	300
C03	Z	A		A3	500
C04	X	A		A4	500
Customer relation			Customer_id	Account numb	account relation
				er	
			C01	A1	
			C02	A2	
			C03	A3	
depositor relation			C04	A4	

Network model

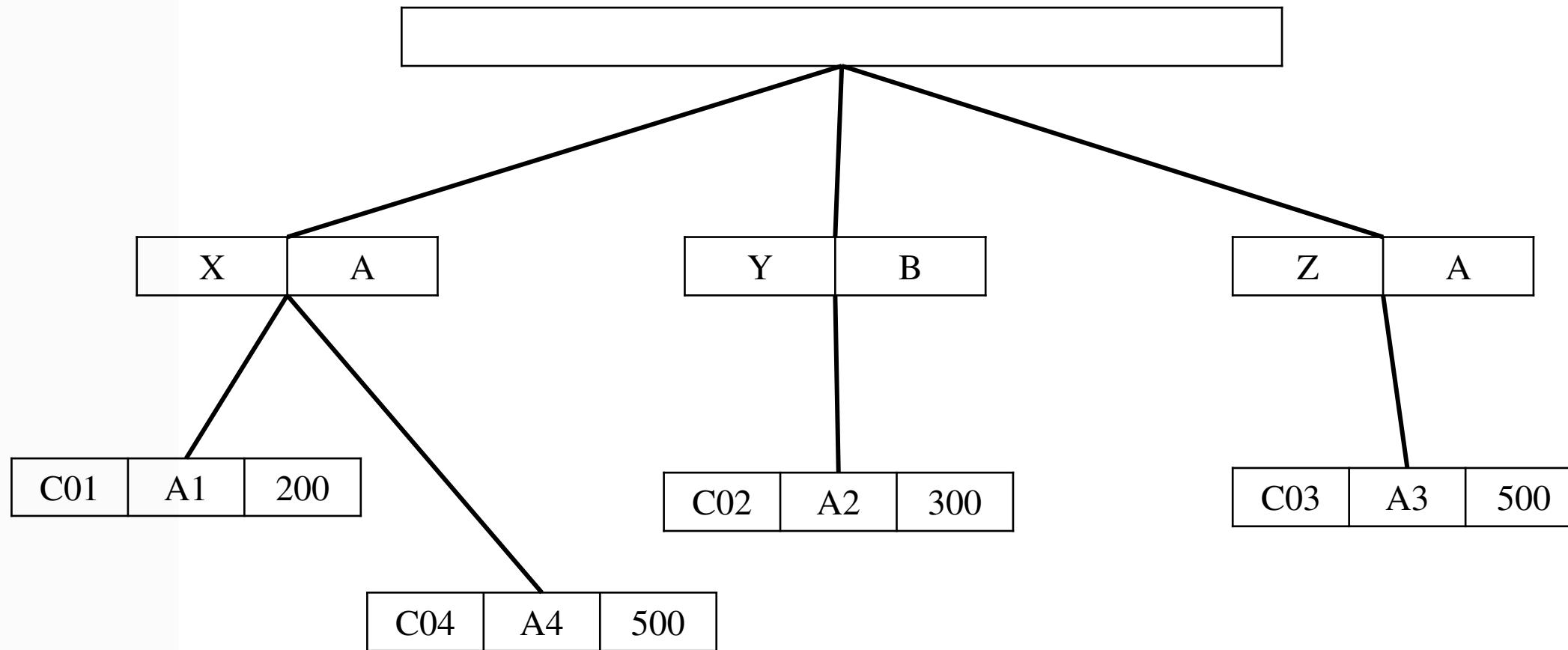
- In network model, data are represented by the set of records and
 - relationships among data are represented by links.



Hierarchical Models

- Hierarchical model also represents data by a set of records, but records are organized in hierarchical, or order structure and database is a collection of such disjoint trees.
- The nodes of the tree represent record types.
- Hierarchical tree consists one root record type along with zero or more occurrences of its dependent subtree and each dependent subtree is again hierarchical.
- In hierarchical model, no dependent record can occur without its parent record.
- Furthermore, no dependent record may be connected to more than one parent record.

Hierarchical Models



Physical Data Models

- Physical data models are used to describe data at the lowest level.
- Example: Indexed Sequential Access Method (ISAM) is a physical data model that defines how data is physically organized to optimize storage and retrieval.
- **Physical data models describe how data is stored and accessed at the physical level.**

END OF LECTURE 5

- SHIVA.KUNWAR@HOTMAIL.COM
- +977-9819123654

Google classroom code : drbzdcf

PREVIEW FOR LECTURE 6

ENTITIES, RELATIONSHIPS AND ATTRIBUTES

E-R DIAGRAMS, KEYS

GENERALIZATION, SPECIALIZATION AND AGGREGATION



Data Analysis Idea

Lecture 6
**Database Management
System**

Er. Shiva Kunwar
Lecturer, GU

Lesson 2: Data Model (6hrs)

1. Different Data Model Concepts
2. E-R Model
3. Network/Hierarchical Model
4. Relational Model
5. **Entities, Relationships and Attributes**
6. **E-R Diagrams, Keys**
7. Generalization, Specialization and Aggregation

Topics

- Entities
- Entities Types and Entities Set
- Strong Entity Sets and Weak Entity Sets
- Relationship and Relationship Set
- Attributes and Keys; Primary Key, Candidate Key, Super Key
- E-R diagram
- Reducing E-R diagram to tables,

Entities

- An entity is a real-world object or concept that has a distinct existence and is distinguishable from other objects.
- In the context of a database, entities represent the main objects about which data is stored.
- Entities have attributes that describe their properties or characteristics.
- Each entity instance is uniquely identifiable, often by a primary key.
- Represented by rectangles in ER diagrams.
- *In a university database, "Student", "Professor" and "Course" can be entities.*

Entities Types

- An entity type is a collection of entities that share common characteristics or properties.
- An entity type represents a category or class of objects in the real world.
- It defines a set of properties (attributes) applicable to all entities within that type.
- Represented by a rectangle in an ER diagram.
- *In a university database, "Student" is an entity type that includes individual students. Each student has attributes such as StudentID, Name, and DateOfBirth.*

Entities Set

- An entity set is a collection of all entities of a particular entity type at a given point in time.
- An entity set represents the current snapshot of all instances belonging to an entity type.
- The set evolves as entities are added or removed over time.
- Represented by the entire collection of rectangles of a specific entity type.
- *The set of all "Student" entities currently enrolled in a university forms the "Student" entity set.*

Strong Entity

- A strong entity is an entity that can be uniquely identified by its attributes.
- They can exist independently without relying on other entities for identification.
- Strong entities have a primary key that serves as a unique identifier.
- Represented by a rectangle in an ER diagram, with the entity name inside the rectangle.
- *In a university database, the "Student" entity is a strong entity. Each student can be uniquely identified by a StudentID, making it a strong entity.*

Weak Entity

- Weak entities rely on a strong entity, known as the "owner," for identification.
- They have a partial key, which is a set of attributes that, in combination with the owner entity's primary key, uniquely identifies instances.
- Represented by a double rectangle in an ER diagram, indicating its dependency on another entity.
- *In a database for a company's employees, a "Dependent" entity might be weak. It would depend on the "Employee" entity for identification, and its partial key could include attributes like "DependentName" and "Relationship" (relationship to the employee).*

Relationships

- A relationship is an association between two or more entities that signifies a real-world connection.
- It represents how instances of entities interact or are related to each other.
- Relationships have a name that describes the nature of the association (e.g., "Works for," "Enrolls in," "Manages").
- Represented by a diamond shape connecting related entities in an ER diagram.
- *In a university database, a "Teaches" relationship might exist between the "Professor" and "Course" entities. The relationship signifies which professors teach which courses.*

Degree of Relationship Set

1. Unary Relationship (Recursive Relationship)

- When there is only ONE entity set participating in a relation, the relationship is called a unary relationship.
- "Manages" relationship where an employee manages another employee.*

2. Binary Relationship

- When there are TWO entities set participating in a relationship, the relationship is called a binary relationship.
- "Works_In" relationship between "Employee" and "Department."*
- "Enrolls_In" relationship between "Student" and "Course."*

Degree of Relationship Set

3. Ternary Relationship

- When there are THREE entities set participating in a relationship, the relationship is called a ternary relationship.
- "*Teaches*" relationship involving "*Professor*," "*Course*," and "*Department*."

4. n-ary Relationship

- When there are n entities set participating in a relation, the relationship is called an n-ary relationship.

Constraints

- E-R model has a capability to enforce constraints.
- Two most important type of constraints in ER model are-
- **Mapping Cardinalities (Cardinality ratio)**
- **Participation Constraints**

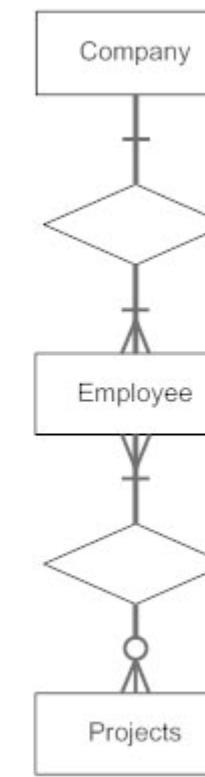
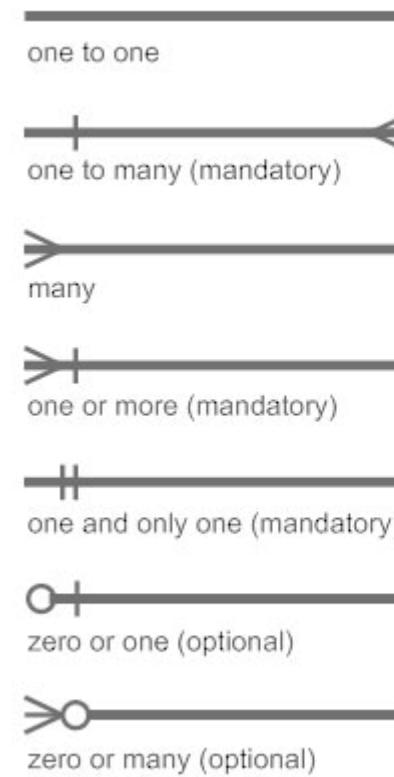
Mapping Cardinalities in Relationship

- Mapping Cardinalities describes no. of entities to which another entity can be associated via relationship set.
- Mapping cardinalities are most useful in describing binary relationship sets but it can also describe relationship sets that involve more than two entity sets.

Mapping Cardinalities in Relationship

- For binary relationship set between entity set A and B mapping cardinality must one of the following.
- **One to one:** An entity in A is associated with at most one entity in B and entity in B is associated with at most one entity in A.
- **One to many:** An entity in A is associated with zero or more entities in B but entity in B can be associated with at most one entity in A.
- **Many to one:** An entity in A is associated with at most one entity in B but an entity in B can be associated with zero or more entities in A.
- **Many to many:** An entity in A is associated with zero or more entities in B, and an entity in b is associated with zero or more entities in A.

Mapping Cardinalities in Relationship



Participation Constraints

- The participation of an entity set E in a relationship set R is said to be **total** if every entities in E participates in at least one relationship in R.
- If only some entities in E participate in relationship in R, then participation of entity set E in relationship set R is said to be **partial**.
- *The participation of loan in the relationship set borrower is total but customer entity set in borrower relationship set is partial since not all customers necessarily take loan from bank, customer may also those who are only account holder.*

Attributes

- An attribute is a property or characteristic of an entity that provides more information about the entity.
- Attributes describe the features, qualities, or traits associated with entities.
- Entities can have multiple attributes.
- Attributes have data types, such as string, number, or date, defining the kind of information they can store.
- Represented by ovals connected to the respective entity in ER diagrams.
- *Attributes for a "Student" entity might include "StudentID," "Name," and "DateOfBirth."*

Attributes Types

- **Key Attribute:** which uniquely identifies each entity *e.g.. Student_id*
- **Simple Attribute:** which cannot be divided further *e.g.. Phone_no*
- **Composite Attribute:** made of more than one simple attribute
e.g.. name => first name, last name
- **Derived Attribute:** do not exist in physical database, and values derived from another attribute *e.g.. Age derived from DOB from database, here DOB is Stored Attribute*
- **Single Valued Attribute:** having single value *e.g.. Student_id*
- **Multi Valued Attribute:** having multiple values *e.g.. Phone_no, email*
- **Descriptive Attributes:** attributes of the relationship

Keys

- A key is a set of one or more attributes that uniquely identifies an entity or a record within a database.
- Keys are essential for establishing relationships between tables and ensuring the accuracy and integrity of the data.
- In ER diagrams, primary keys are typically underlined.
- Foreign keys are represented as arrows pointing to the referenced table's primary key.
- *In a "Student" table, the "StudentID" could be the primary key, uniquely identifying each student. If there's a "Course" table, and we want to establish a relationship, the "StudentID" in the "Course" table becomes a foreign key, referencing the "Student" table's primary key.*

Keys Characteristics

Uniqueness

- Keys must ensure the uniqueness of values within a table.
- Each record should be uniquely identified by its key.

Uniqueness Across Tables

- In the case of a foreign key, the values should be unique in the referenced table's primary key.

Non-Null

- Keys should not contain null values.
- They should have valid and meaningful values to ensure reliable identification.

Keys Types

Primary Key

- A primary key is a unique identifier for each record in a table.
- No two records in the table can have the same primary key value.
- It ensures data integrity and serves as a reference for relationships with other tables.

Foreign Key

- A foreign key is a field in a table that refers to the primary key in another table.
- It establishes a link between the two tables, creating a relationship.
- Foreign keys maintain referential integrity.

Keys Types

Candidate Key

- A candidate key is a set of attributes that could serve as the primary key.
- It satisfies the uniqueness and irreducibility requirements for a key.
- *In the "Employee" table of a company database, both "EmployeeID" and "Email" could be candidate keys.*

Composite Key

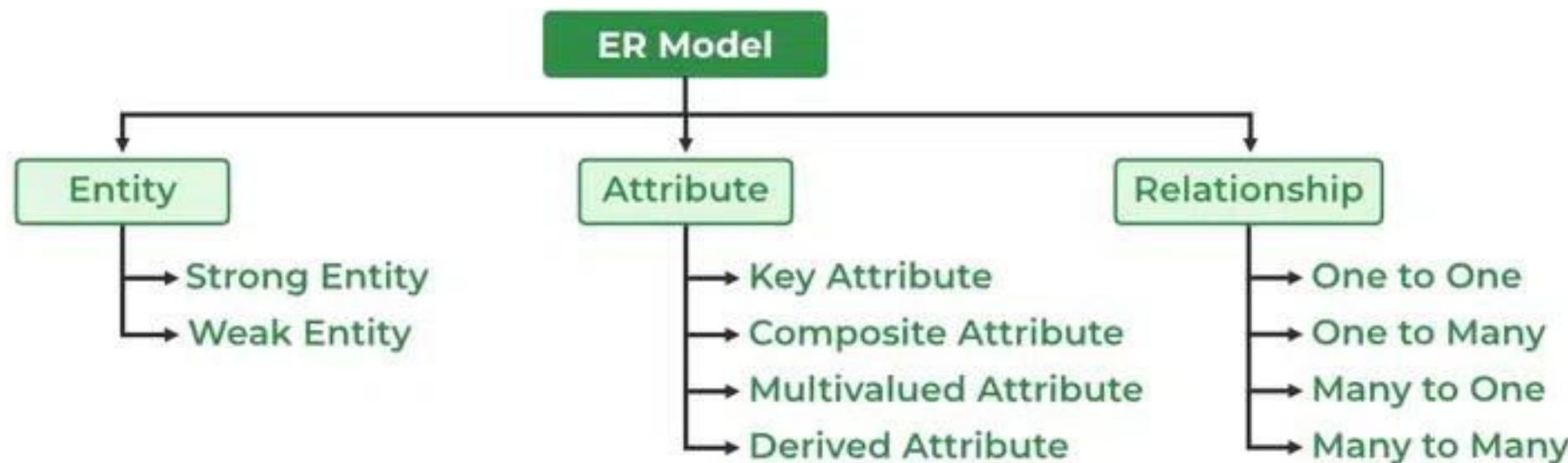
- A composite key is a key that consists of multiple attributes.
- It is used when a single attribute is not sufficient to uniquely identify a record.
- *In a "Purchase" table of an online store database, a composite key could be a combination of "CustomerID" and "ProductID" to uniquely identify each purchase.*

Keys Types

Super Key

- A super key is a set of one or more attributes (columns) that, taken together, can uniquely identify a tuple (row) in a relation (table) within a database.
- It is a broader concept than a primary key and can include more attributes than needed for uniqueness.
- A super key is essentially any combination of attributes that uniquely identifies a record.
- All primary keys are super keys, not all super keys are primary keys.
- *In the "Employee" table of a company database, both "EmployeeID" and "Email" could be candidate keys.*

Components of ER Model



ER Diagram

- Is a data modeling technique that creates a graphical representation of entities and their relationship within an information system.
- It illustrates the logical structure of the database.

Components of ER Diagram

- **Rectangles:** representing entity sets.
- **Ellipses:** representing attributes.
- **Diamonds:** representing relationship sets.
- **Lines:** linking attributes to entity sets and entity sets to relationship sets.
- **Double ellipses:** represents multivalued attributes.
- **Dashed ellipses:** represents derived attributes
- **Double lines:** represents total participation of entity in relationship sets
- **Double rectangles:** represents weak entity sets
- **Underline:** indicates primary key attributes

Important Tips Before Preparing an ER Diagram

- **Understand the Domain**
- **Identify Entities**
- **Define Relationships**
- **Attribute Identification**
- **Primary Keys**
- **Avoid Redundancy**
- **Use Consistent Naming Conventions**
- **Think About Cardinality**

Important Tips Before Preparing an ER Diagram

- **Understand the Domain**
- Before creating an ER diagram, it's crucial to thoroughly understand the domain or the application for which you are designing the database.
- This involves gaining insights into the business processes, user requirements, and the nature of the data involved.
- *Suppose you are designing a database for a library management system.*
- *Understanding the domain involves learning about library processes, user interactions, and the type of information stored.*
- *This could include knowledge about books, authors, borrowers, and library transactions.*

Important Tips Before Preparing an ER Diagram

- **Identify Entities**
- Start by identifying the main entities in the system.
- Entities are the fundamental building blocks that represent real-world objects or concepts.
- They could be objects like "Customer," "Product," or "Order" in a retail management system.
- *In the library management system, entities could include "Book," "Author," "Borrower," and "Transaction."*
- *Each of these represents a distinct object or concept in the library domain.*

Important Tips Before Preparing an ER Diagram

- **Define Relationships**
- Determine how entities are related to each other.
- Relationships define the associations between entities.
- It differs with types like one-to-one, one-to-many, or many-to-many.
- *In the library management system, there could be a "Written by" relationship between "Author" and "Book," indicating that an author writes one or more books.*
- *This relationship is crucial for understanding how authors and books are connected.*

Important Tips Before Preparing an ER Diagram

- **Attribute Identification**
- List the attributes for each entity.
- Attributes describe the properties or characteristics of entities.
- Be comprehensive in identifying attributes to capture all relevant information.
- *In the library management system, for the "Book" entity, attributes could include "ISBN," "Title," "Genre," and "Publication Year."*
- *For the "Author" entity, attributes might include "AuthorID," "Name," and "Nationality."*

Important Tips Before Preparing an ER Diagram

- **Primary Keys:**
- Clearly define primary keys for each entity.
- The primary key uniquely identifies each instance of an entity.
- Ensure that primary keys are unique and essential for data integrity.
- *In the library management system, in the "Book" entity, the "ISBN" could be the primary key.*
- *In the "Author" entity, the "AuthorID" could serve as the primary key.*
- *This ensures unique identification of each book and author.*

Important Tips Before Preparing an ER Diagram

- **Avoid Redundancy:**
- Minimize redundancy in the ER diagram by normalizing the model
- Normalization involves organizing data to reduce duplication and dependency.
- This ensures that data is stored efficiently and consistently, contributing to overall database integrity.
- *In the library management system, for example, there is information about the author's nationality in both the "Author" and "Book" entities, it might be redundant.*
- *Normalization would involve creating a separate "AuthorDetails" entity to store non-repetitive information about authors.*

Important Tips Before Preparing an ER Diagram

- **Use Consistent Naming Conventions:**
- Adopt consistent and meaningful names for entities, attributes, and relationships.
- Consistent naming conventions improve the clarity and maintainability of the ER diagram.
- Ensure that names are easily understandable and follow a standardized format.
- *In the library management system, use "BookTitle" in one entity and "AuthorName" in another, rather than mixing terms like "Title" and "Name."*
- *Consistency in naming enhances readability.*

Important Tips Before Preparing an ER Diagram

- **Think About Cardinality**
- Determine the cardinality of relationships.
- Cardinality specifies the number of instances of one entity associated with another.
- *In the library management system, in the "Borrower-Transaction" relationship, a borrower may have multiple transactions (one-to-many), but each transaction is associated with one borrower (one-to-one).*
- *Understanding cardinality is crucial for designing the database schema accurately.*

Example

- In a university, a Student enrolls in Courses. A student must be assigned to at least one or more Courses. Each course is taught by a single Professor. To maintain instruction quality, a Professor can deliver only one course.

Example

- Step 1) Entity Identification
- We have three entities
 - Student
 - Course
 - Professor

Example

- Step 2) Relationship Identification
 - We have the following two relationships
-
- The student is assigned a course
 - Professor delivers a course

Example

- Step 4) Identify Attributes
- Once, you have a list of Attributes, you need to map them to the identified entities. Ensure an attribute is to be paired with exactly one entity. If you think an attribute should belong to more than one entity, use a modifier to make it unique.
- Once the mapping is done, identify the primary Keys. If a unique key is not readily available, create one.

Example

- Step 4) Identify Attributes
- Entity Primary Key Attribute
- Student Student_ID StudentName
- Professor Employee_ID ProfessorName
- Course Course_ID CourseName

Example

- Step 4) Cardinality Identification
- For them problem statement we know that,
- A student can be assigned multiple courses
- A Professor can deliver only one course

Practical

- Construct an ER diagram for a car insurance company with a registration id and an address and have name and address of its customers. Each customer own one or more cars which has its model, color and engine number. Each car is associated with zero or any number of recorded accidents.

Practical

- UPS prides itself on having up-to-date information on the processing and current location of each shipped item. To do this, UPS relies on a company-wide information system. Shipped items are the heart of the UPS product tracking information system.
- Shipped items can be characterized by item number (unique), weight, dimensions, insurance amount, destination, and final delivery date. Shipped items are received into the UPS system at a single retail center.
- Retail centers are characterized by their type, uniqueID, and address. Shipped items make their way to their destination via one or more standard UPS transportation events (i.e., flights, truck deliveries). These transportation events are characterized by a unique scheduleNumber, a type (e.g, flight, truck), and a deliveryRoute.
- Prepare an ER Diagram that captures this information.

END OF LECTURE 6

- SHIVA.KUNWAR@HOTMAIL.COM
- +977-9819123654

Google classroom code : drbzdcf

PREVIEW FOR LECTURE 7

ER DIAGRAM Contd.



Data Analysis Idea

Lecture 7
**Database Management
System**

Er. Shiva Kunwar
Lecturer, GU

Lesson 2: Data Model (6hrs)

1. Different Data Model Concepts
2. E-R Model
3. Network/Hierarchical Model
4. Relational Model
5. Entities, Relationships and Attributes
6. **E-R Diagrams, Keys**
7. **Generalization, Specialization and Aggregation**

Topics

- E-R diagram
- Reducing E-R diagram to tables
- Generalization, Specialization and Aggregation

ER Diagram Example

- In a university, a Student enrolls in Courses. A student must be assigned to at least one or more Courses. Each course is taught by a single Professor. To maintain instruction quality, a Professor can deliver only one course.

Example

- Step 1) Entity Identification
- We have three entities
 - Student
 - Course
 - Professor

Example

- Step 2) Relationship Identification
 - We have the following two relationships
-
- The student is assigned a course
 - Professor delivers a course

Example

- Step 4) Identify Attributes
- Once, you have a list of Attributes, you need to map them to the identified entities. Ensure an attribute is to be paired with exactly one entity. If you think an attribute should belong to more than one entity, use a modifier to make it unique.
- Once the mapping is done, identify the primary Keys. If a unique key is not readily available, create one.

Example

- Step 4) Identify Attributes
- Entity Primary Key Attribute
- Student Student_ID StudentName
- Professor Employee_ID ProfessorName
- Course Course_ID CourseName

Example

- Step 4) Cardinality Identification
- For them problem statement we know that,
- A student can be assigned multiple courses
- A Professor can deliver only one course

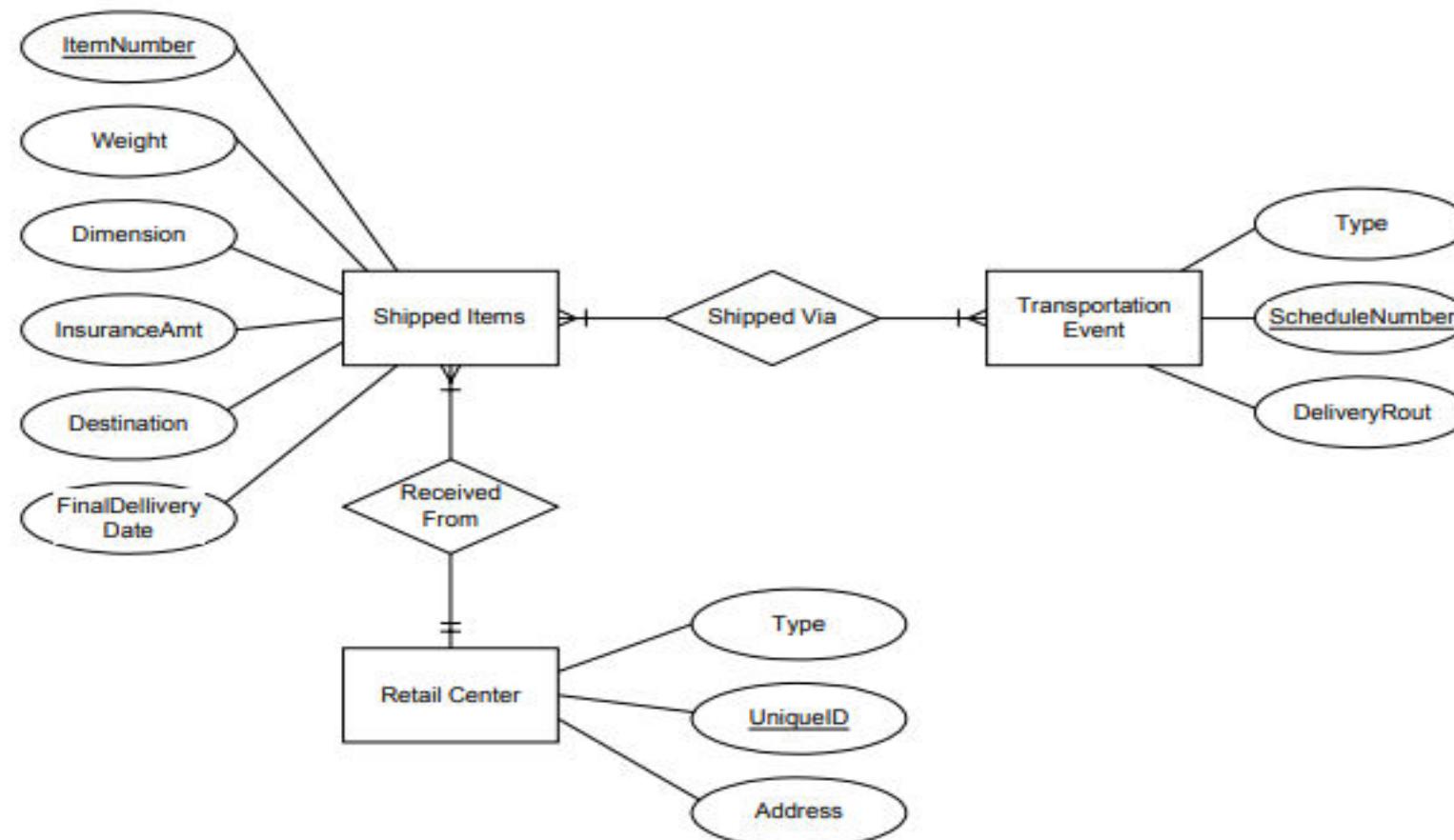
Practical

- Construct an ER diagram for a car insurance company with a registration id and an address and have name and address of its customers. Each customer own one or more cars which has its model, color and engine number. Each car is associated with zero or any number of recorded accidents.

Practical

- UPS prides itself on having up-to-date information on the processing and current location of each shipped item. To do this, UPS relies on a company-wide information system. Shipped items are the heart of the UPS product tracking information system.
- Shipped items can be characterized by item number (unique), weight, dimensions, insurance amount, destination, and final delivery date. Shipped items are received into the UPS system at a single retail center.
- Retail centers are characterized by their type, uniqueID, and address. Shipped items make their way to their destination via one or more standard UPS transportation events (i.e., flights, truck deliveries). These transportation events are characterized by a unique scheduleNumber, a type (e.g, flight, truck), and a deliveryRoute.
- Prepare an ER Diagram that captures this information.

Practical



Reducing ER Diagrams to Table

- We can represent the E-R database schema by a set of tables.
- Each entity sets and each relationship sets in E-R schema can be represented by their corresponding tables.
- Each attributes of entity sets, and relationship sets are map as columns of their corresponding tables.
- Similarly, constraints specified in E-R diagram such as primary key, cardinality constraints etc. are mapped to tables generated from E-R diagram.
- In fact, representing E-R schema into tables is converting E-R model of database into relational model

Reducing ER Diagrams to Table

- Consider a simple ER diagram with two entities: "Student" and "Course," and a many-to-many relationship "Enrolls In."
- Student: StudentID, Name, GPA*
- Course: CourseID, Title, Department*
- Enrolls In: EnrollmentID (PK), StudentID (FK), CourseID (FK)*
- Create Tables Student table, Course table, and enrolls table that represent many-to-many relation

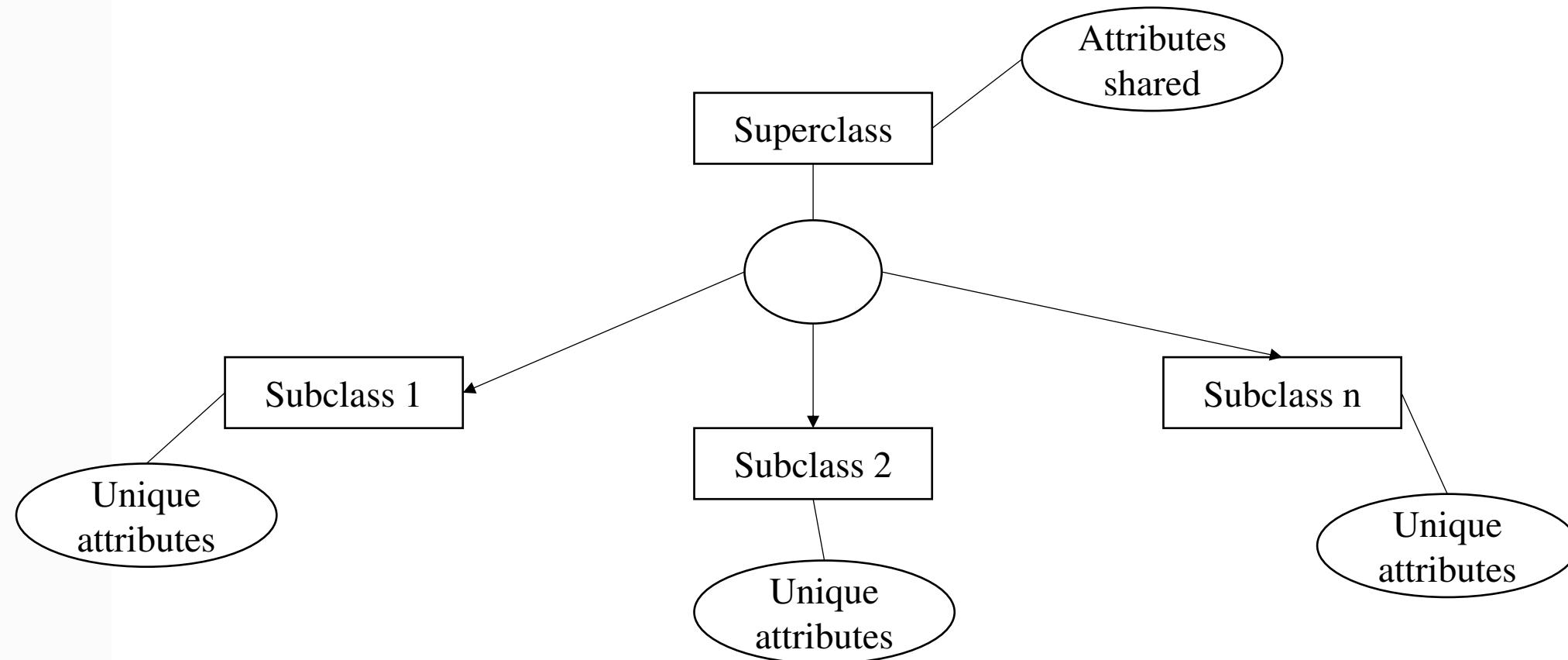
Extended ER Features

- In the pursuit of enhancing the ER model, several terms and concepts have been incorporated.
- Generalization refers to the process of abstracting common features from multiple entities into a more generalized entity, reducing redundancy and promoting a higher level of abstraction.
- Specialization, conversely, involves refining a generalized entity into more specialized entities, tailoring attributes and relationships to specific subsets of data.
- Aggregation, facilitates the modeling of relationships involving higher-level entities composed of or related to lower-level entities.

Superclass & Subclass

- **Superclass**
- A superclass is a generalized entity that contains common attributes and relationships shared by multiple specialized entities or subclasses.
- It represents the more abstract, generalized concept.
- **Subclass**
- A subclass is a specialized entity that inherits attributes and relationships from a superclass.
- It represents a more specific, specialized concept.
- Attributes are unique from other subclasses.

Superclass & Subclass



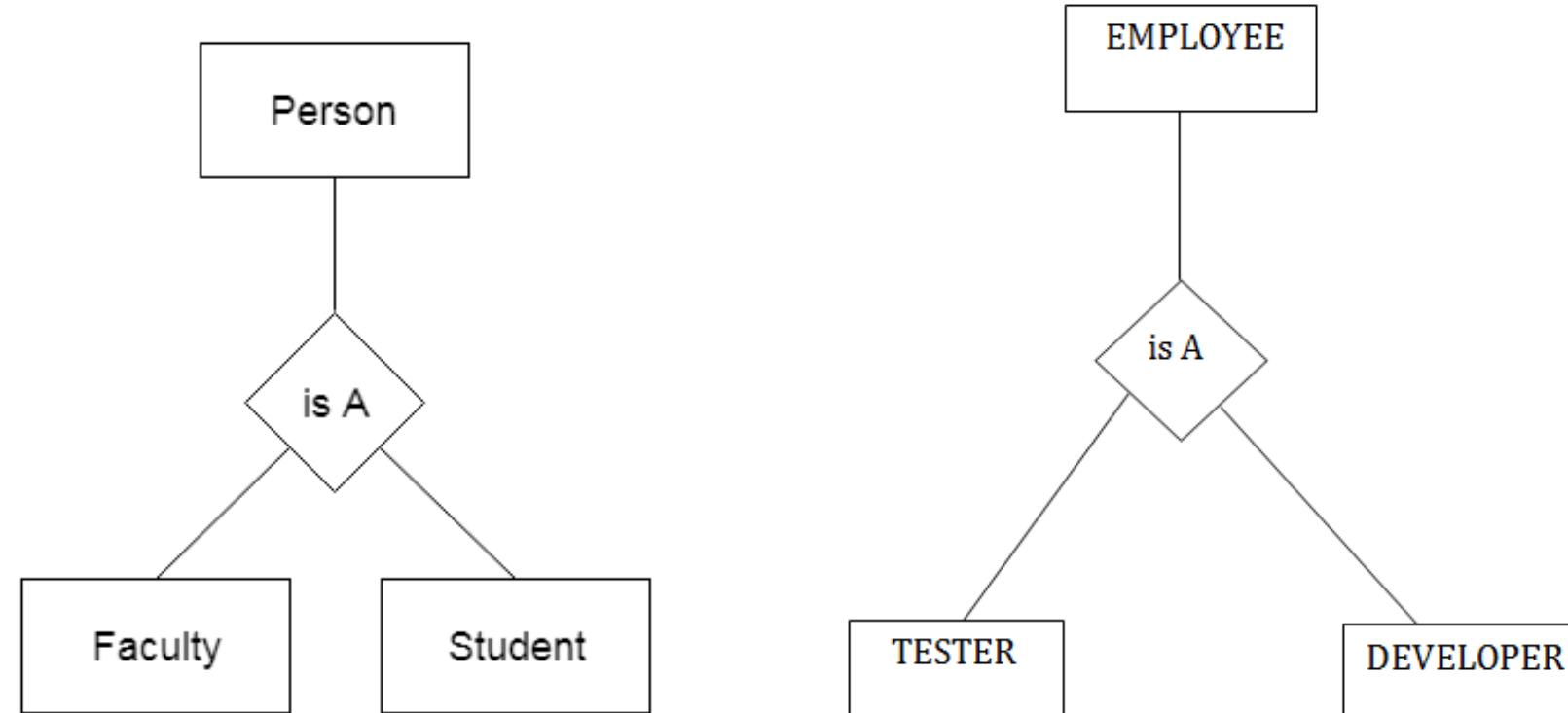
Generalization

- It involves abstracting common features from multiple entities to create a more generalized entity.
- It is a bottom-up approach.
- **Characteristics:**
 - Reduces redundancy by identifying and representing commonalities.
 - Enhances simplicity by creating a higher-level, more abstract entity.
- *Consider entities like 'Car' and 'Truck.' These entities share common attributes such as 'Make,' 'Model,' and 'Year.' Generalization allows the creation of a more abstract entity called 'Vehicle,' which captures the shared characteristics of both 'Car' and 'Truck.'*

Specialization

- It involves refining a generalized entity into more specialized entities.
- It is a top-down approach.
- **Characteristics:**
- Allows for the representation of diverse entities with unique characteristics.
- Enables the modeling of specific subtypes within a broader category.
- *Building upon the 'Vehicle' entity, specialization creates more specific entities like 'Car' and 'Truck,' each inheriting the common attributes from the 'Vehicle' entity while adding attributes specific to its type.*
- This also explains the concept of Inheritance.

Generalization-Specialization



Generalization-Specialization Constraints

- **Attribute Constraints**
- Attributes can be associated with a superclass or a subclass.
- An attribute associated with a superclass is shared by all its subclasses.
- An attribute associated with a subclass is specific to that subclass.
- *If 'Person' has a 'Name' attribute, it is shared by both 'Employee' and 'Customer.'*
If 'Employee' has a 'Salary' attribute, it is specific to the 'Employee' subclass.

Generalization-Specialization Constraints

- **Disjoint Constraint**
- Specifies that a superclass entity can belong to only one subclass.
- Denoted by a letter "d" next to the line connecting the superclass to the specialization symbol.
- *If 'Animal' is disjointly specialized into 'Mammal' and 'Bird,' an 'Animal' can be either a 'Mammal' or a 'Bird,' but not both.*

Generalization-Specialization Constraints

- **Overlap Constraint**
- Allows a superclass entity to belong to more than one subclass.
- Denoted by a letter "o" next to the line connecting the superclass to the specialization symbol.
- *If 'Fruit' is overlapped specialized into 'Citrus' and 'Berry,' a 'Fruit' can be both a 'Citrus' and a 'Berry.'*

Generalization-Specialization Constraints

- **Total Completeness Constraint**
- Specifies that every superclass entity must belong to at least one subclass.
- Denoted by a double line connecting the superclass to the specialization symbol.
- *If 'Vehicle' is totally specialized into 'Car' and 'Truck,' every 'Vehicle' must be either a 'Car' or a 'Truck.'*

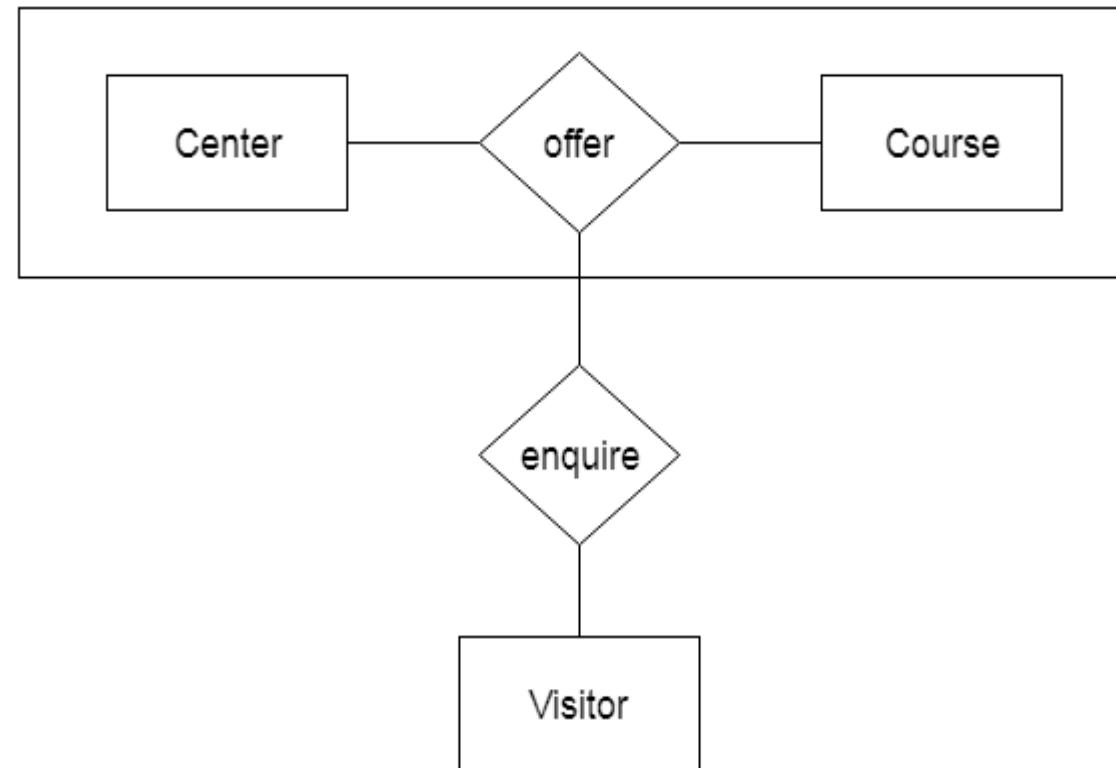
Generalization-Specialization Constraints

- **Partial Completeness Constraint**
- Allows superclass entities to exist without being members of any subclass.
- Denoted by a single line connecting the superclass to the specialization symbol.
- *If 'Person' is partially specialized into 'Employee' and 'Customer,' a 'Person' can exist without being an 'Employee' or 'Customer.'*

Aggregation

- Allows modeling of relationships involving higher-level entities composed of or related to lower-level entities.
- It helps in representing complex relationships more effectively.
- **Characteristics:**
- Useful for expressing complex relationships and hierarchies.
- Allows for the creation of composite entities representing a whole-part relationship.
- *Consider an 'Order' entity that aggregates 'Product' entities. The 'Order' entity represents a higher-level concept that consists of individual 'Product' entities.*

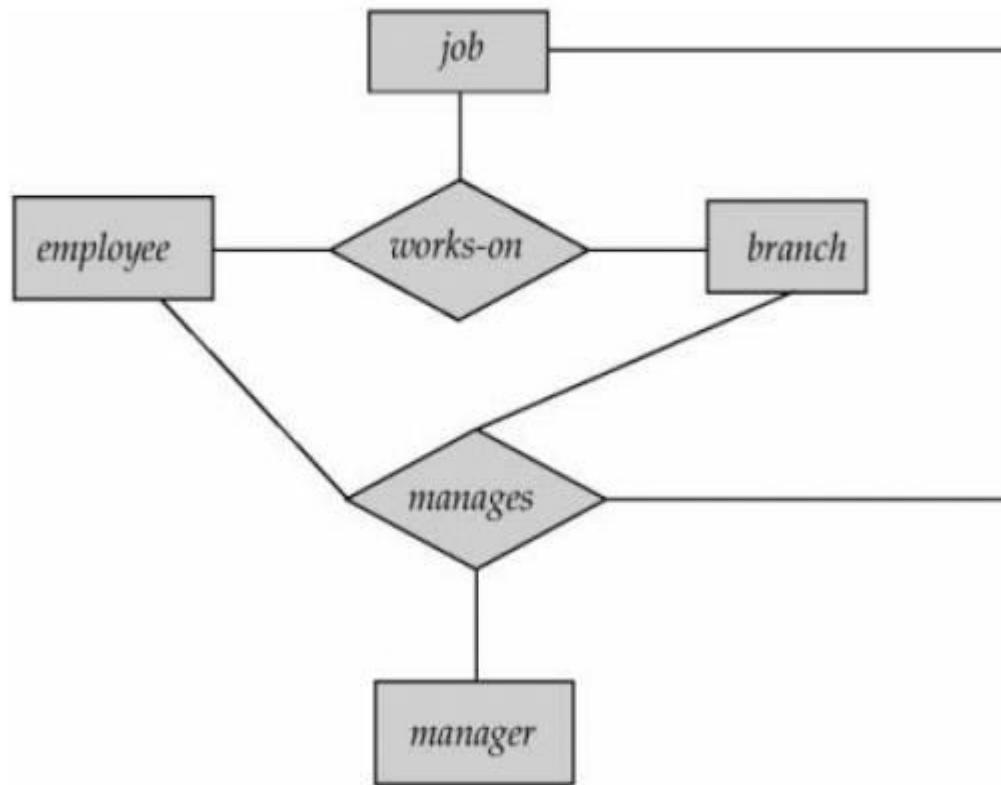
Aggregation



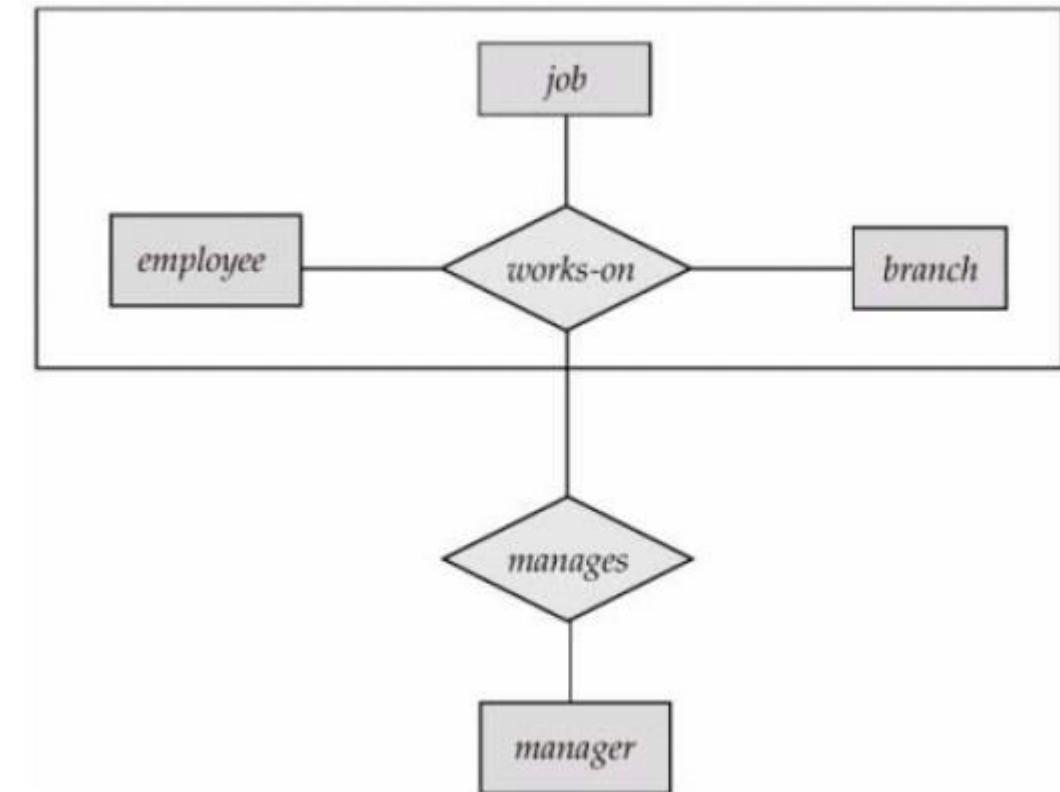
Aggregation Scenario

- Managers who manages particular job/task performed by particular employee at particular branch.

Aggregation



E-R diagram with redundant relationships.



E-R Diagram with aggregation

Practical

- Draw the ER diagram for a small database for a bookstore. The database will store information about books for sale. Each book has an ISBN, title, price and short description. Each book is published by a publisher in a certain publishing year. For each publisher, the database maintains the name, address and phone number.
- Each book is written by one or more authors. For each author, the database maintains his/her ID, name and a short introduction. Each book is stored in exactly one warehouse with a particular quantity. For each warehouse, the database maintains the warehouse name, the location and the phone number. Each book has one or more sellers, which may be either companies (corporate vendors) or individuals (individual vendors).

Practical

- For each company, the database maintains a name of the company, its address, its phone numbers (there could be more than one phone number, each with a number and a description) and its contact person. For each individual vendor, the database keeps a name, a phone number and an email address. A contact person whose company sells a book cannot be selling the same book as an individual vendor at the same time (he/she may sell other books as an individual seller).

Practical

- Identify all the entities
- -AUTHOR
- -PUBLISHER
- -BOOK
- -CUSTOMER
- -SHOPPING_BASKET
- -WAREHOUSE

Practical

- find the relations
- 1. Each book is written by a author
- 2. Each book has a publisher
- 3. Some shopping baskets may contain more than one copy of same book
- 4. The warehouse stocks several books
- 5. A customer owns several shopping basket

Practical

- Identify the key attribute
- *AUTHOR- name
- *PUBLISHER- name
- *BOOK- ISBN
- *CUSTOMER- email
- *SHOPPING_BASKET- basket_ID
- *WAREHOUSE- code

Practical

- Identify other relevant attributes
- *AUTHOR- name,address,URL
- *PUBLISHER- name,address,URL,phone
- *BOOK- ISBN,year,title,price
- *CUSTOMER- email, name,address,phone
- *SHOPPING_BASKET- basket_ID
- *WAREHOUSE- code, address,phone

Practical

relations

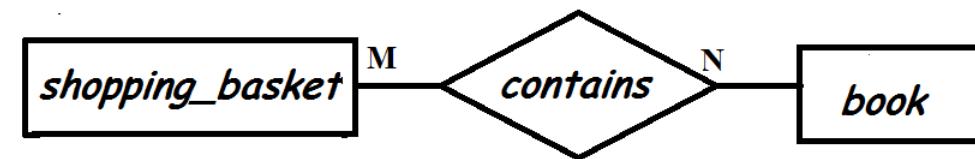
1. Each book is written by a author



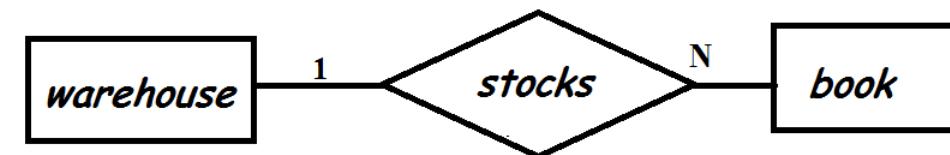
2. Each book has a publisher



3. Some shopping baskets may contain more than one copy of same book

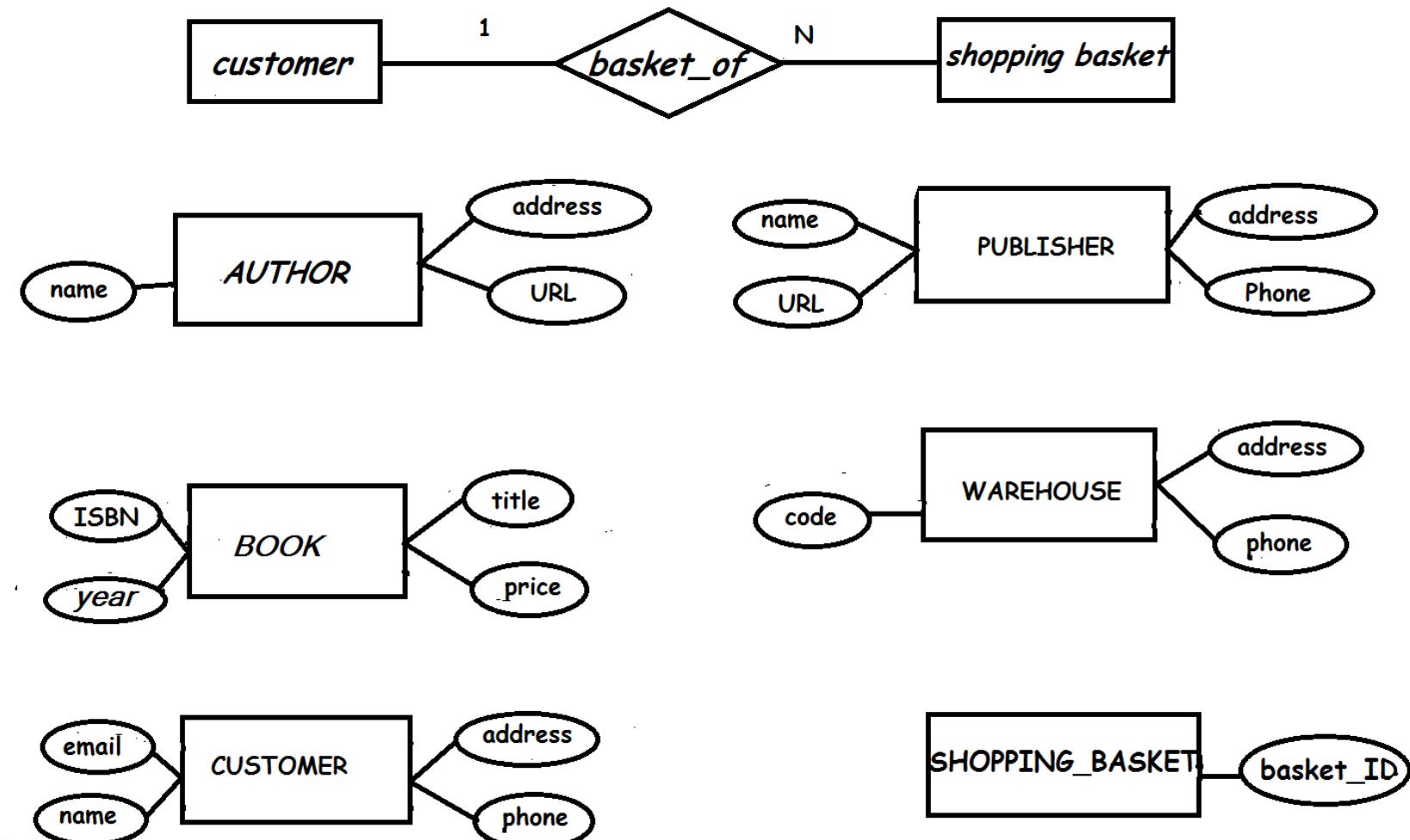


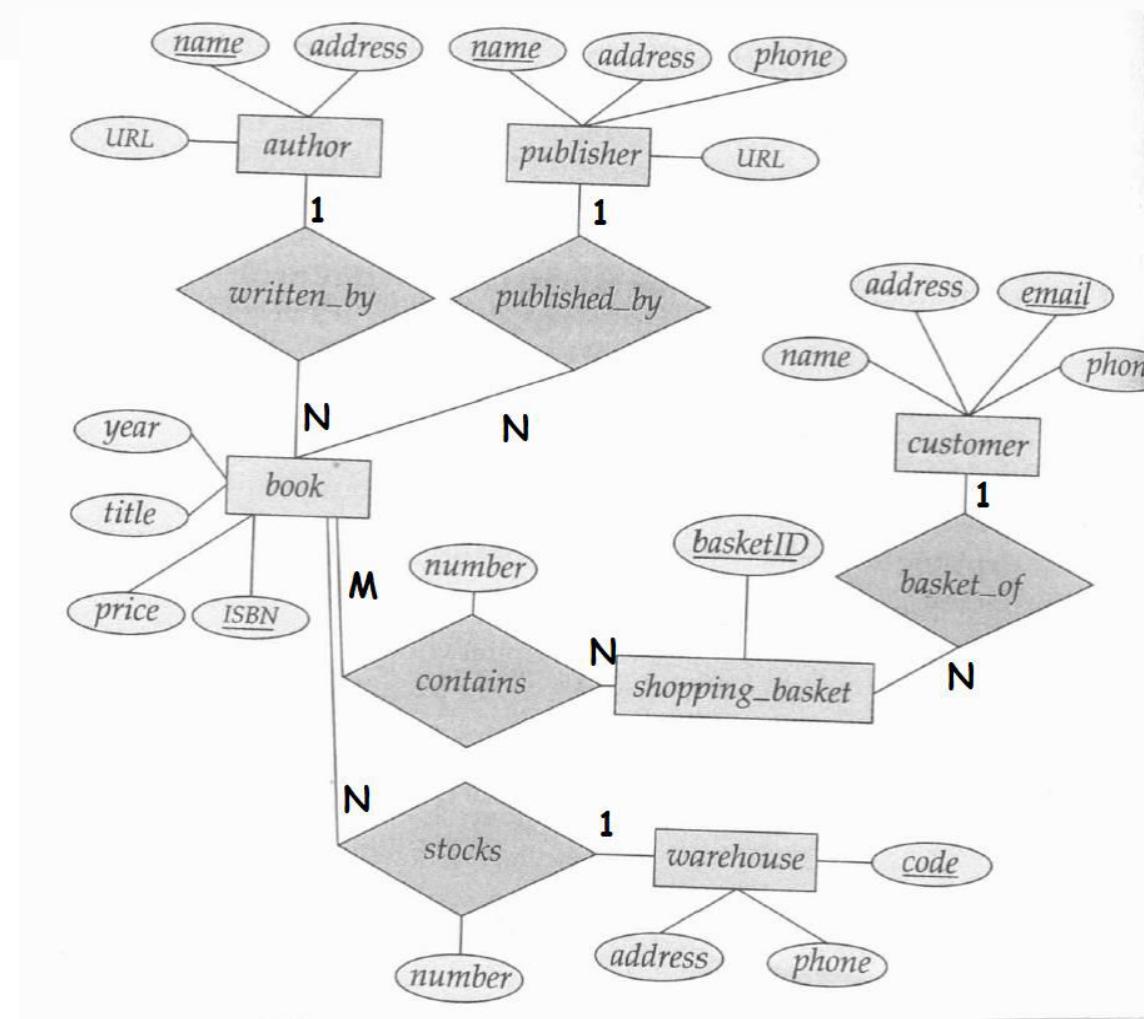
4. The warehouse stocks several books



Practical

5. A customer owns several shopping basket





E-R DIAGRAM OF ONLINE BOOKSTORE

END OF LECTURE 7

- SHIVA.KUNWAR@HOTMAIL.COM
- +977-9819123654

Google classroom code : drbzdcf

PREVIEW FOR LECTURE 8

RELATIONAL MODEL

INTRODUCTION TO RELATIONAL DATABASE

DATABASE SCHEMA AND VIEWS

RELATIONAL MODEL CONSTRAINTS



Analysis

Idea

Lecture 8
**Database Management
System**

Er. Shiva Kunwar
Lecturer, GU

Lesson 1: Introduction to DBMS (5hrs)

1. Overview of Database and DBMS
2. Characteristics and Applications
3. Data Abstraction and Independence
4. Database Users and Administrator
5. Application Architecture
6. **Basics of Database Language (DDL, DML, DCL) + Lab**

DDL – Data Definition Language

Definition: DDL is used to define and manage the structure of the database.

Content:

- Creating tables, views, and indexes.
- Modifying the structure of existing tables.
- Defining constraints (e.g., primary keys, foreign keys).
- Data definition language is used to store the information of metadata like the number of tables and schemas, their names, indexes, columns in each table, constraints, etc.

DDL – Data Definition Language

Here are some tasks that come under DDL:

- **Create:** It is used to create database and its objects in the database.
- **Alter:** It is used to alter the structure of the existing database.
- **Drop:** It is used to delete objects from the database.
- **Truncate:** It is used to remove all records from a table.
- **Rename:** It is used to rename an object.
- **Comment:** It is used to comment on the data dictionary.

These commands are used to update the database schema that's why they come under Data definition language.

Practical DDL

- The general syntax for logging in to MySQL from the command line is:
- *mysql -u <username> -p*
- With root as username,
- *mysql -u root -p*
- If password is blank, -p can be omitted.

Practical DDL

- Use Create command to create a database.
- *create database gu;*
- Use Show command to show the list of all available database in the server.
- *show databases;*
- Use use command to activate the newly created database.
- *use gu;*
- Use show tables command to show list of all tables in the active database.
- *show tables;*

Practical DDL - Create

- CREATE TABLE: This is used to create a new relation (table).
- Syntax: CREATE TABLE <relation_name/table_name >
(field_1 data_type(size),field_2 data_type(size), . . .)
- *CREATE TABLE Students (StudentID INT PRIMARY KEY,
FirstName VARCHAR(50),
LastName VARCHAR(50),
Age INT ;*

Practical DDL - Alter

- ALTER TABLE ...ADD...: This is used to add some extra fields into existing relation.
- Syntax: ALTER TABLE relation_name ADD (new field_1 data_type(size), new field_2 data_type(size),...);
- *ALTER TABLE Students*
- *ADD COLUMN Email VARCHAR(100);*

Practical DDL - Alter

- ALTER TABLE...MODIFY...: This is used to change the width as well as data type of fields of existing relations.
- Syntax: ALTER TABLE relation_name MODIFY (field_1 newdata_type(Size), field_2 newdata_type(Size), ... field_n newdata_type(Size))
- *ALTER TABLE Students*
- *ALTER COLUMN GPA DECIMAL(3, 2);*

Practical DDL - Alter

- ALTER TABLE..DROP.... This is used to remove any field of existing relations.
- Syntax: ALTER TABLE relation_name DROP COLUMN (field_name);
- *ALTER TABLE Students*
- *DROP COLUMN Age;*

Practical DDL - Alter

- ALTER TABLE..RENAME...: This is used to change the name of fields in existing relations.
- Syntax: ALTER TABLE relation_name RENAME COLUMN (OLD field_name) to (NEW field_name)
- *ALTER TABLE Students*
- *RENAME COLUMN FirstName TO First_Name;*

Practical DDL - Drop

- **DROP TABLE:** This is used to delete the structure of a relation. It permanently deletes the records in the table.
- Syntax: `DROP TABLE relation_name;`
- *DROP TABLE Teachers;*

Practical DDL - Rename

- RENAME: It is used to modify the name of the existing database object.
- Syntax: RENAME TABLE old_relation_name TO new_relation_name;
- *RENAME TABLE Students TO AllStudents;*

Practical DDL - Describe

- DESCRIBE: It is useful for exposing details about a table (columns, data types)..
- Syntax: DESCRIBE TABLE relation_name;
- *DESCRIBE TABLE Students;*

Practical DDL - Truncate

- TRUNCATE: It is useful for removing all records from a table while keeping the table structure.
- Syntax: TRUNCATE TABLE relation_name;
- *TRUNCATE TABLE Teachers;*

Practical DDL - Comment

- COMMENT: It is useful for adding comments or descriptions to database objects.
- Syntax: COMMENT ON TABLE relation_name IS ‘comments to add’;
- *COMMENT ON TABLE Students*
- *IS 'Contains information about students';*

Practical DDL - Assignment

- *Create a table EMPLOYEE with following schema:*
- *(Emp_no, E_name, E_address, E_ph_no, Dept_no, Dept_name, Job_id , Salary)*
- *Add a new column; HIREDATE to the existing relation.*
- *Change the datatype of JOB_ID from char to varchar2.*
- *Change the name of column/field Emp_no to E_no.*
- *Modify the column width of the job field of emp table.*

DML – Data Manipulation Language

Definition: DML is responsible for manipulating data stored in the database.

- It is used for accessing and manipulating data in a database.
- It handles user requests.
- It includes operations like SELECT, INSERT, UPDATE, and DELETE.
- DML establishes communication between user and database.

DML – Data Manipulation Language

Here are some tasks that come under DML

- **Select:** It is used to retrieve data from a database.
- **Insert:** It is used to insert data into a table.
- **Update:** It is used to update existing data within a table.
- **Delete:** It is used to delete all records from a table.
- **Merge:** It performs UPSERT operation, i.e., insert or update operations.
- **Call:** It is used to call a structured query language or a Java subprogram.
- **Explain Plan:** It has the parameter of explaining data or data access path.
- **Lock Table:** It controls concurrency.

Practical DML - Select

- SELECT: Used to retrieve data from one or more tables in a database.
- Syntax: `SELECT column1, column2, ...`
- `FROM table_name`
- `WHERE condition;`
- *SELECT FirstName, LastName*
- *FROM Students*
- *WHERE GPA > 3.0;*
- *Note: * can be used to select all columns.*

Practical DML - Insert

- INSERT: Used to insert new records into a table.
- Syntax: `INSERT INTO table_name (column1, column2, ...)`
- `VALUES (value1, value2, ...);`
- *INSERT INTO Students (StudentID, FirstName, LastName, GPA)*
- *VALUES (1, 'John', 'Doe', 3.5);*

Practical DML - Update

- UPDATE: Used to modify existing data in a table.
- Syntax: UPDATE table_name
- SET column1 = value1, column2 = value2, ...
- WHERE condition;

- *UPDATE Students*
- *SET GPA = 3.8*
- *WHERE StudentID = 1;*

Practical DML - Delete

- DELETE: Used to delete records from a table based on specified conditions.
- Syntax: `DELETE FROM table_name`
- WHERE condition;
- *DELETE FROM Students*
- *WHERE GPA < 2.0;*

Practical DML - Merge

- MERGE: Performs an operation that either inserts new rows or updates existing rows, based on a specified condition.
- Generally used for UPSERT operations (INSERT or UPDATE).
- *MERGE INTO target_table USING source_table*
- *ON (condition)*
- *WHEN MATCHED THEN*
- *UPDATE SET column1 = value1, ...*
- *WHEN NOT MATCHED THEN*
- *INSERT (column1, column2, ...) VALUES (value1, value2, ...);*

Practical DDL - Assignment

- *Create a table EMPLOYEE with following schema:
(Emp_no, E_name, E_address, E_ph_no, Dept_no, Dept_name, Job_id , Salary)*
- *Insert at least 5 rows in the table.*
- *Display all the information of EMP table.*
- *Display the record of each employee who works in department ELECTRONICS.*
- *Update the city of Emp_no-12 with current city as Pokhara.*
- *Display the details of Employee who works in department ACCOUNTS.*
- *Delete the email_id of employee James.*
- *Display the complete record of employees working in SALES Department.*

END OF LECTURE 8

- SHIVA.KUNWAR@HOTMAIL.COM
- +977-9819123654

Google classroom code : drbzdcf

PREVIEW FOR LECTURE 9

RELATIONAL MODEL



Data Analysis Idea

Lecture 9

Database Management System

Er. Shiva Kunwar
Lecturer, GU

Lesson 3: Relational Model (4hrs)

- 1. Introduction to Relational Database**
- 2. Database Schema and Views**
- 3. Relational Model Constraints**
- 4. Relational Operations and Algebra**

Relational Model

- Primary data model widely used for data storage and processing.
- Was proposed by Dr. E. F. Codd of IBM in 1970.
- The relational model provides a flexible and efficient way to organize and manage data, offering a standardized approach to database design and querying.
- It has been widely adopted in the development of relational database management systems (RDBMS).
- SQL (Structured Query Language) is the language commonly used to interact with relational databases based on this model.
- Here are the main elements of the relational model:

Tables (Relations)

- In the relational model, data is organized into tables, which are also referred to as relations.
- Each table consists of rows and columns.
- Tables represent entities, and each row in a table represents a specific instance or record of that entity.

Rows (Tuples)

- Rows, also known as tuples, represent individual records or data instances within a table.
- Each row contains a set of values, one for each column in the table.
- A row is uniquely identified by a primary key, which is a column or a combination of columns with unique values.

Columns (Attributes)

- Columns, also known as attributes, represent the properties or characteristics of the entities.
- Each column has a name and a specific data type (e.g., integer, varchar, date).
- Columns define the structure of the data and provide a way to categorize and organize information.

Relationships

- Relationships establish connections between tables by linking columns in one table to columns in another.
- Relationships are defined using keys, such as primary keys and foreign keys.
- Foreign keys establish referential integrity between tables, ensuring that relationships are valid and consistent.

Primary Keys

- A primary key is a unique identifier for a row in a table.
- It ensures that each row is uniquely identified within the table.
- Primary keys are used to establish relationships between tables.

Foreign Keys

- A foreign key is a column or a set of columns in one table that refers to the primary key in another table.
- It establishes a link between tables, enforcing referential integrity.
- Foreign keys are used to create relationships between tables.

Relational Model Summary

- A Relational Database Model consists of relations to connect them by key fields.
- The relation is represented in rows and columns.
- Each column of the relation is called an attribute.
- Each row in the relation is called a tuple.
- Each relation can have one unique column i.e. primary key.
- Each relation can have n-columns and n-tuple.
- The relational model represents data in the form of relations or tables.

Relational Model

- Relational Schema R is denoted by $R(A_1, A_2, \dots, A_n)$ where R is name of relation and A_1, A_2, \dots, A_n are list of attributes.
- Each relation is preceded by the name of that relation.
- The fields of the relations are separated by commas and placed within the parentheses of the relation.
- *Example: Relational Model can be represented as shown below*
- *STUDENT (StudNo, Sname)*
- *ENROLLMENT (StudNo, Subcode, marks)*

Integrity Constraints

- Integrity refers to accuracy, consistency and reliability of data in the database.
- Integrity constraints prevent the occurrence of errors and inconsistencies although changes are made to database by authorized users.
- Integrity constraints are defined by some key constraints like primary key foreign key.
- Two general integrity rules in relational database system are:
 - Entity integrity
 - Referential integrity

Entity integrity

- Entity integrity ensures that each row (or record) in a table is uniquely identifiable by a primary key and that the primary key column(s) do not contain null values.
- It guarantees the uniqueness of each record in a table and provides a means for uniquely identifying and accessing individual rows.

```
CREATE TABLE Employees (  
  EmployeeID INT PRIMARY KEY,  
  FirstName VARCHAR(50) NOT NULL,  
  LastName VARCHAR(50) NOT NULL  
);
```

Referential integrity

- Referential integrity ensures the consistency and accuracy of relationships between tables.
- It is maintained using foreign keys, where the values in a foreign key column must match the values in the corresponding primary key column of another table or be null.

```
CREATE TABLE Orders (  
        OrderID INT PRIMARY KEY,  
        CustomerID INT,  
    FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)  
);
```

Relational Database

- A relational database is a type of database that uses the relational model for organizing and structuring data.
- It consists of a collection of tables, each of which is related to one another by common fields, often known as keys.
- Normalization is the process of organizing data in a way that minimizes redundancy and dependency which involves breaking down large tables into smaller, related tables.
- Relational databases adhere to the ACID properties (Atomicity, Consistency, Isolation, Durability) to ensure the reliability of transactions.

Relational Database

- The relational database management system (RDBMS) is the software that enables users to interact with and manage the relational database.
- Some terminologies for RDBMS are:
- Record or Relation Oriented: (Relation, Domain, Tuple, Attribute, Degree, Keys)
- Table Oriented: (Table, Set of Permitted Values, Row, Column, No of rows, No of columns, Unique + Not Null)

Relational Database

- Record or Relation Oriented:
 - Relation: Set of values of field in rows and columns
 - Domain: valid set of values that an attribute can take.
 - Tuple: value at each row of relation
 - Attribute: value at each column of relation
 - Degree: number of attributes of its relational schema
 - Keys

Database Schema

- A database schema is a logical and structural design that defines the organization and relationships of data within a database.
- It provides a blueprint for how data is stored, accessed, and managed.
- A schema includes information about tables, columns, data types, constraints, and other database objects.

Database Schema Components

- Tables: Entities and their relationships are represented as tables.
- Columns: Attributes or fields within each table.
- Relationships: Links between tables, typically expressed through primary and foreign key constraints.
- Constraints: Rules that define the integrity of the data, such as primary key, foreign key, unique, and check constraints.

Database Schema Example

```
CREATE TABLE Students (
    StudentID INT PRIMARY KEY,
    FirstName VARCHAR(50),
    LastName VARCHAR(50),
    CourseID INT FOREIGN KEY REFERENCES Courses(CourseID)
);
```

```
CREATE TABLE Courses (
    CourseID INT PRIMARY KEY,
    CourseName VARCHAR(100),
    Instructor VARCHAR(50)
);
```

Database View

- A database view is a virtual table or result set that is based on existing tables.
- A view does not store the data itself.
- It provides a way to represent data stored in one or more underlying tables.
- Views are useful for simplifying complex queries and providing a customized perspective on the data.
- Views allows users to
 - Structure data in a way that users want
 - Restrict access to data such that user can see and sometime modify exactly what they need
 - Summarize data from various tables, which can be used to generate reports

Database View

- Creating a View:

CREATE VIEW view_name AS

 SELECT column1, column2 FROM table_name WHERE condition;

- Querying a View:

 SELECT * FROM view_name;

- Updating a View:

 UPDATE view_name SET column=value WHERE condition;

- Dropping a View:

 DROP VIEW view_name;

Database View

```
CREATE VIEW SalesSummary AS
SELECT
    Orders.OrderID,
    Customers.CustomerName,
    Products.ProductName,
    OrderDetails.Quantity,
    OrderDetails.UnitPrice,
    OrderDetails.Quantity * OrderDetails.UnitPrice AS TotalPrice
FROM
    Orders
JOIN Customers ON Orders.CustomerID = Customers.CustomerID
JOIN OrderDetails ON Orders.OrderID = OrderDetails.OrderID
JOIN Products ON OrderDetails.ProductID = Products.ProductID;
```

Database View

- In this example, the SalesSummary view provides a consolidated view of sales-related information by joining multiple tables (Orders, Customers, OrderDetails, and Products).
- Users can query this view to obtain a summary of sales data without dealing directly with the complexity of multiple joins.

Relational Model Constraints

- In the relational model, constraints are rules or conditions applied to the data in a database table to ensure its integrity and consistency.
- These constraints define the relationships between tables, the uniqueness of data, and other properties.
- They prevent the insertion of invalid or inconsistent data and contribute to the overall quality of the database.
- Here are some common constraints in the relational model:

Primary Key Constraint

- Ensures that a column or set of columns uniquely identifies each row in a table.
- No two rows can have the same values for the primary key column(s).

```
CREATE TABLE Students (  
    StudentID INT PRIMARY KEY,  
    FirstName VARCHAR(50),  
    LastName VARCHAR(50)  
);
```

Unique Constraint

- Ensures that all values in a specific column or set of columns are unique.
- Unlike the primary key, a unique constraint allows for null values.

```
CREATE TABLE Employees (
    EmployeeID INT UNIQUE,
    FirstName VARCHAR(50),
    LastName VARCHAR(50)
);
```

Foreign Key Constraint

- Establishes a link between two tables by referencing the primary key of one table in another.
- Enforces referential integrity between related tables.

```
CREATE TABLE Orders (  
        OrderID INT PRIMARY KEY,  
        CustomerID INT,  
        OrderDate DATE,  
        FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)  
);
```

Check Constraint

- Specifies a condition that must be true for each row in a table.
- Used to limit the range of allowed values in a column.

```
CREATE TABLE Employees (
    EmployeeID INT PRIMARY KEY,
    Salary DECIMAL(10, 2) CHECK (Salary >= 0)
);
```

Not Null Constraint

- Ensures that a column cannot have a null (undefined) value.

```
CREATE TABLE Customers (  
    CustomerID INT PRIMARY KEY,  
    FirstName VARCHAR(50) NOT NULL,  
    LastName VARCHAR(50) NOT NULL  
);
```

Default Constraint

- Specifies a default value for a column when no explicit value is provided.

```
CREATE TABLE Products (  
    ProductID INT PRIMARY KEY,  
    ProductName VARCHAR(100),  
    Price DECIMAL(10, 2) DEFAULT 0.00  
);
```

END OF LECTURE 9

- SHIVA.KUNWAR@HOTMAIL.COM
- +977-9819123654

Google classroom code : drbzdcf

PREVIEW FOR LECTURE 10

RELATIONAL OPERATIONS
RELATIONAL ALGEBRA



Data Analysis Idea

Lecture 10

Database Management System

Er. Shiva Kunwar
Lecturer, GU

Lesson 3: Relational Model (4hrs)

1. Introduction to Relational Database
2. Database Schema and Views
3. Relational Model Constraints
4. **Relational Operations and Algebra**

Relational Algebra

- The relational algebra is a procedural query language.
- It consists of a set of operations that takes one or more relations as inputs and produces a new relation as output.
- It helps to understand the query execution and optimization in RDBMS.

Relational Operations

- The fundamental operations in relational algebra are
- **Selection**
- **Projection**
- **Union**
- **Set Difference**
- **Cartesian Product**
- **Rename.**

SELECTION

- Selects a subset of tuples from relation that satisfies certain criteria
- denoted by σ (Sigma)
- Syntax: $\sigma \text{ condition } (\text{relation})$

SELECTION

- Employee (Eid, Ename, Eaddress, Salary)
- Find employes whose Eid equals to 4.
- $\sigma_{Eid=4} (employee)$
- Find all employees whose salary is greater than 10000
- $\sigma_{salary > 10000} (employee)$
- Find all employees whose salary is greater than 5000 and Eid equals to 4.
- $\sigma_{salary > 5000 \text{ AND } Eid = 4} (employee)$

SELECTION

- We may use logical operators like \wedge , \vee , $!$ and relational operators like $=$, \neq , $>$, $<$, \leq , \geq with the selection condition.
- Selection operator only selects the required tuples according to the selection condition. It does not display the selected tuples. To display the selected tuples, projection operator is used.
- Selection operator always selects the entire tuple. It can not select a section or part of a tuple.
- The number of rows returned by a selection operation is obviously less than or equal to the number of rows in the original table
- Minimum Cardinality = 0, Maximum Cardinality = $|R|$

PROJECTION

- Select some of the columns from table and discards other columns
- used only if we are interested in some attributes
- result is vertical partition of relation into two relations
- denoted by π (pi)
- Syntax: $\pi \text{ attribute_list} (\text{relation})$

PROJECTION

- Employee (Eid, Ename, Eaddress, Salary)
- Select "Ename" and "Salary" from all records in the "employee" relation.
- $\pi Ename, Salary(employee)$
- Select "Ename" and "Salary" only for records where the salary is greater than 10000.
- $\pi Ename, Salary (\sigma salary > 10000(employee))$
- Select "Ename" and "Salary" from all records in the "employee" relation and then filters out records where the salary is not greater than 10000.
- $\sigma salary > 10000 (\pi Ename, Salary(employee))$

UNION

- It builds relation from tuple appearing in either or both specified relation
- It eliminates duplicate records
- two relations are union compatible if
 - they have same number of attributes
 - attribute domain must be compatible
 - names of attribute are same in both the relations
- denoted by \cup .
- Syntax: $\pi \text{ attribute_list } (\text{relation}) \cup \pi \text{ attribute_list } (\text{relation})$

UNION

- $\pi_{customerName}(depositor) \cup \pi_{customerName}(borrower)$
- All customers of bank who have an account or a loan
- $A \cup B = B \cup A$

SET DIFFERENCE

- Allows to find tuples that are in one relation but not in another
- Union compatible should exist.
- denoted by – (Minus).
- Syntax: $\pi \text{ attribute_list } (\text{relation}) - \pi \text{ attribute_list } (\text{relation})$

SET DIFFERENCE

- $\pi \text{customerName}(\text{depositor}) - \pi \text{customerName}(\text{borrower})$
- All customers of bank who have an account but not loan
- $A - B \neq B - A$

CARTESIAN PRODUCT

- It allows to combine information from any 2 relations
- It builds a relation with concatenation of every row in first relation with every row in the second relation
- Also known as Cross Product or Cross Join
- denoted by \times (Cross)
- Syntax: $\pi \text{ attribute_list}(A \times B)$
- $\pi EId, Ename, DeptName (Employee.DeptId = Department.DeptId) (Employee \times Department)$

CARTESIAN PRODUCT

Borrower(Cname, Loan_no)

Cname	Loan_no
X	L01
Y	L02

Loan(Loan_no, Branch_name, Amount)

Loan_no	Branch_name	Amount
L01	B1	5000
L02	B2	6000

R = borrower X loan

Cname	Borrower.Loan_no	Loan.Loan_no	Branch	Amount
X	L01	L01	B1	5000
X	L01	L02	B2	6000
Y	L02	L01	B1	5000
Y	L02	L02	B2	6000

Query: Find all customer who taken loan from branch “B1”.

CARTESIAN PRODUCT

Query: Find all customer who taken loan from branch “B1”.

$$\pi_{c_name}(\sigma_{borrower.loan_number = loan.loan_number} (\sigma_{branch_name = 'B1'} (borrower \times loan)))$$

Step 2: $\sigma_{branch_name = 'B1'} (borrower \times loan)$

Cname	Borrower.LOAN_no	Loan.LOAN_no	Branch	Amount
X	L01	L01	B1	5000
Y	L02	L01	B1	5000

CARTESIAN PRODUCT

Query: Find all customer who taken loan from branch “B1”.

$$\pi_{c_name}(\sigma_{borrower.loan_number = loan.loan_number} (\sigma_{branch_name = 'B1'} (borrower \times loan)))$$

Step 3: $\sigma_{borrower.loan_number = loan.loan_number} (\sigma_{branch_name = "B1"} (borrower \times loan))$

Cname	Borrower.LOAN_no	Loan.LOAN_no	Branch	Amount
X	L01	L01	B1	5000

Step 3: $\pi_{c_name} (\sigma \dots)$

Cname
X

RENAME

- It is used to rename the resultant relation.
- denoted by ρ (Rho)
- Syntax: $\rho \ name_of_relation(relation)$

RENAME

- $\rho_{\text{cse_students}} (\pi_{\text{Roll_no}, \text{Name}} (\sigma_{\text{Branch}=\text{'cse'}} (\text{Students})))$
- The selection of "Roll_no" and "Name" attributes from the "Students" relation where the "Branch" is 'cse,' and the result is renamed as "cse_students."

Additional Operations

- The fundamental operations of the relational algebra are sufficient to express any relational algebra query.
- But for complex query it is difficult.
- Additional operations (**set intersection, natural join, division, assignment**) simplify the common queries.

SET INTERSECTION

- Results tuples that appear in both relation
- The result must be union compatible
- denoted by intersection
- Syntax: $\pi \text{ attribute_list} (\text{relation}) \cap \pi \text{ attribute_list} (\text{relation})$

SET INTERSECTION

- $\pi \text{customerName}(\text{depositor}) \cap \pi \text{customerName}(\text{borrower})$
- All customers of bank who have an account and a loan
- $A \cap B = B \cap A$

JOIN OPERATION

- Allows users to combine 2 relation in a specified way
- The join operation can be stated in terms of a cartesian product followed by a selection operation
- It is very important as used frequently with specifying database queries
- three types of join exist
 - Theta join or condition join
 - Equi join or inner join
 - Natural join

THETA JOIN (θ) or CONDITION JOIN

- Combines two relation together based on a condition using a comparison operator (θ)
- θ is a predicate and consist of one of the comparison operators ($=$, $<$, $>$, \leq , \geq , \neq) and specifies join condition
- For any two relations A&B, theta join is
- Syntax: $A \bowtie_{\theta} B$ or $\sigma_{\theta}(A X B)$

Theta join (θ)

- $Employee \bowtie Employee.Eid = Department.Eid$ *Department*
- $\sigma Employee.Eid = Department.Eid$ (*Employee X Department*)
- This expression represents an equi-join between the "Employee" and "Department" relations based on the equality of the "Eid" attribute in both relations.

Theta join (θ)

- $S1 \bowtie_{(S1.Sid < R1.Sid)} R1$
- $S1 X R1$
- S1.Sid R1.Sid
- 22 22
- 22 58 *
- 31 22
- 31 58 *
- 58 22
- 58 58

Sid	R1	Day
22	101	10/10/2015
58	103	9/9/2016

Sid	S1	Rating	Age
22	Ram	7	45
31	Shyam	8	50
58	Hari	10	35

EQUI JOIN or INNER JOIN

- If θ is $=$ then theta join becomes equi join
- The $=$ condition is performed by primary and foreign keys
- The result must include two attributes with property that values of these two attributes are equal in every tuple in relation
- For any two relations A&B, Equi join is
- Syntax: $A \bowtie (A.Pk = B.Fk) B$

EQUI JOIN

- $S1 \bowtie_{(S1.Sid = R1.Sid)} R1$
- S1.Sid R1.Sid
- 22 22 *
- 22 58
- 31 22
- 31 58
- 58 22
- 58 58 *

Sid	R1	Day
22	101	10/10/2015
58	103	9/9/2016

Sid	S1	Rating	Age
22	Ram	7	45
31	Shyam	8	50
58	Hari	10	35

NATURAL JOIN

- It is represented by symbol \bowtie .
- Only relates those rows which have common attributes value.
- For relation A&B, natural join is represented as
- Syntax: $A \bowtie B = \text{column}(A) \cup \text{column}(B)$

NATURAL JOIN

Eid	Ename	DeptName
101	Ram	Finance
102	Shyam	Marketing
103	Hari	Finance

DeptName	Manager
Finance	Gopal
Marketing	Gita

Employee ⚠ Department

Eid	Ename	DeptName	DeptName	Manager
101	Ram	Finance	Finance	Gopal
102	Shyam	Marketing	Marketing	Gita
103	Hari	Finance	Finance	Gopal

OUTER JOIN

- The outer-join operation is extension to natural join.
- It has a capability to deal with missing information.
- There are three form of outer-join operation
 - a) Left outer-join
 - b) Right outer-join
 - c) Full outer-join

LEFT OUTER JOIN

- Takes all tuples in the left relation. If there are any tuples in right relation that does not match with tuple in left relation, simply pad these right relation tuples with null.
- Add them to the result of the left outer-join.

RIGHT OUTER JOIN

- Takes all tuples in the right relation. If there are any tuples in the left relation that does not match with tuple in right relation, simply pad left relation tuples with null.
- Add them to the result of the left outer-join.

FULL OUTER JOIN

- Pad tuples from the left relation that did not match any from the right relation
- Pad tuples from the right relation that did not match any from the left relation
- Add them to the result of full outer-join.

Employee (pname, street, city)

Company (cname, city)

Works (pname, cname, salary)

Manages (pname, managername)

1. Find the names of all employees who work for First Bank Corporation.

RA: $\pi \text{ pname} (\sigma \text{ cname} = \text{'First Bank Corporation'} (\text{Works}))$

SQL: SELECT DISTINCT pname FROM WORKS WHERE cname='First Bank Corporation';

Employee (pname, street, city)
Company (cname, city)

Works (pname, cname, salary)
Manages (pname, managername)

2. Find the names and cities of residence of all employees who work for First Bank Corporation.

RA: $\pi \text{ pname city} (\text{Employee} \bowtie (\sigma \text{ cname} = \text{'First Bank Corporation'} (\text{Works})))$

SQL: SELECT DISTINCT pname, city FROM employee NATURAL JOIN works
 WHERE cname='First Bank Corporation';

RA: $\pi \text{ pname, city} (\sigma \text{ cname} = \text{'First Bank Corporation'} (\text{Employee} \bowtie \text{Works}))$

RA: $\pi \text{ employee.pname, city} (\text{Employee} \bowtie \text{Employee.pname} = \text{Works.pname} (\sigma \text{ cname} = \text{'First Bank Corporation'} (\text{Works})))$

Employee (pname, street, city)
Company (cname, city)

Works (pname, cname, salary)
Manages (pname, managername)

3. Find the names, street address, and cities of residence of all employees who work for First Bank Corporation and earn more than \$10,000 per annum.

RA: π pname, street, city (σ cname='First Bank Corporation' AND salary > 10000
(Employee \bowtie Works))

SQL: SELECT DISTINCT pname, street, city FROM employee NATURAL JOIN works
WHERE cname='First Bank Corporation' and salary > 10000;

Employee (pname, street, city)
Company (cname, city)

Works (pname, cname, salary)
Manages (pname, managername)

4. Find the names of all employees in this database who live in the same city as the company for which they work.

RA: $\pi \text{Employee.pname} (\sigma \text{Employee.city} = \text{Company.city} \text{ AND } \text{Employee.pname} = \text{Works.pname} \text{ AND } \text{Works.cname} = \text{Company cname})$ ($\text{Employee} \times \text{Works} \times \text{Company}$)

SQL: `SELECT DISTINCT employee.pname FROM employee, works, employee WHERE employee.city = company.city AND Employee.pname = Works.pname AND Works.cname = Company.cname;`

RA: $\pi \text{pname} (\text{Employee} \bowtie \text{Works} \bowtie \text{Company})$

SQL: `SELECT DISTINCT pname FROM employee NATURAL JOIN works NATURAL JOIN company;`

Employee (pname, street, city)
Company (cname, city)

Works (pname, cname, salary)
Manages (pname, managername)

5. Find the names of all employees who live in the same city and on the same street as do their managers.

RA: $\pi_{e.pname} (\rho_e(\text{Employee}) \times \text{Manages} \bowtie (e.pname = \text{manages}.pname \text{ AND } \text{managername} = m.pname \text{ AND } e.street = m.street \text{ AND } e.city = m.city) \rho_m(\text{Employee}))$

SQL: SELECT DISTINCT e.name FROM employee e, manages, employee m WHERE e.pname = manages.pname AND managername = m.pname AND e.street = m.street AND e.city = m.city;

Employee (pname, street, city)

Company (cname, city)

Works (pname, cname, salary)

Manages (pname, managername)

6. Find the names of all employees in this database who do not work for First Bank Corporation.

RA: $\pi \text{ pname} (\sigma \text{ cname} \neq \text{'First Bank Corporation'} (\text{Works}))$

SQL: `SELECT DISTINCT pname FROM works WHERE cname <> 'First Bank Corporation';`

RA: $\pi \text{ pname} (\text{Employee}) - \pi \text{ pname} (\sigma \text{ cname} = \text{'First Bank Corporation'} (\text{Works}))$

SQL: `SELECT DISTINCT pname FROM employee EXCEPT SELECT DISTINCT pname FROM works WHERE cname = 'First Bank Corporation';`

Employee (pname, street, city)
Company (cname, city)

Works (pname, cname, salary)
Manages (pname, managername)

7. Find the names of all employees who earn more than every employee of Small Bank Corporation.

RA: $\pi \text{ pname} (\text{Works}) - \pi \text{ pname} (\rho_{w1}(\text{Works})) \bowtie w1.\text{salary} \leq w2.\text{salary} \text{ AND } w2.\text{cname} = \text{'First Bank Corporation'} (\rho_{w2}(\text{Works}))$

SQL: SELECT DISTINCT pname FROM works WHERE salary > ALL (SELECT salary FROM Works WHERE cname = 'Small Bank Corporation');

END OF LECTURE 10

- SHIVA.KUNWAR@HOTMAIL.COM
- +977-9819123654

Google classroom code : drbzdcf

PREVIEW FOR LECTURE 11

INTRODUCTION TO SQL

SQL QUERIES AND SUBQUERIES

JOINED AND DERIVED RELATIONS



Data Analysis Idea

Lecture 11

Database Management System

Er. Shiva Kunwar
Lecturer, GU

Lesson 4: Relational Language and Database Constraints (11hrs)

- 1. Introduction to SQL**
- 2. SQL Queries and Subqueries**
- 3. Joined and Derived Relations**
- 4. DDL and DML Operations**
- 5. Views and Modification of Database**
- 6. Transaction Control Language**
- 7. Integrity Constraints and Triggers**
- 8. Referential Integrity in SQL**

Introduction to SQL

- SQL stands for Structured Query Language. It is used for storing and managing data in relational database management system (RDMS).
- It is a standard language for Relational Database System. It enables a user to create, read, update and delete relational databases and tables.
- All the RDBMS like MySQL, Informix, Oracle, MS Access and SQL Server use SQL as their standard database language.
- SQL allows users to query the database in several ways, using English-like statements.

Introduction to SQL

- SQL follows the following rules:
- Structure query language is not case sensitive. Generally, keywords of SQL are written in uppercase.
- Statements of SQL are dependent on text lines. We can use a single SQL statement on one or multiple text line.
- Using the SQL statements, you can perform most of the actions in a database.
- SQL depends on tuple relational calculus and relational algebra.

SQL Commands

A. DDL - Data Definition Language

1. CREATE Statement in SQL
2. TRUNCATE Statement in SQL
3. DROP Statement in SQL
4. ALTER Statement in SQL
5. COMMENT in SQL
6. RENAME in SQL

SQL Commands

B. DQL - Data Query Language

1. SELECT Query in SQL

C. DML - Data Manipulation Language

1. INSERT INTO in SQL
2. UPDATE Statement in SQL
3. DELETE Statement in SQL
4. LOCK Table Statement in SQL

SQL Commands

D. DCL - Data Control Language

1. GRANT Privilege in SQL - This command gives the users the privileges of access to a database. This command can be used to grant UPDATE, INSERT, DELETE, and SELECT privileges to its users on several tables or on a single table.

- Here is the Syntax of the GRANT Command:
- *GRANT SELECT, UPDATE ON USER_TABLE TO OTHER_USER;*

Eg: *GRANT INSERT, SELECT ON users TO Ram*

- Thus, using the grant command, Ram has been granted permission on the “users” database objects, and he can insert or query the “users” database.

SQL Commands

D. DCL - Data Control Language

2. REVOKE Privilege in SQL - The REVOKE command is used to take permissions back from the user. This command is mainly used to revoke a privilege.

- Here is the syntax of the REVOKE Command:
- *REVOKE name_of_privilege ON name_of_object FROM {name_of_user | PUBLIC | name_of_role}*

Eg: *REVOKE INSERT, SELECT ON users FROM Ram*

- Thus, when we use this command, the permissions of Ram (like insert or query) on the “users” database objects have been removed.

SQL Commands

D. TCL - Transaction Control Language

1. TRANSACTIONS in SQL

- A TRANSACTION is a TCL command, and it groups a set of various tasks into one single unit of execution.
- Every transaction begins with some specific task, and it ends when all of the tasks in a group are completed successfully.
- If any of these tasks happen to fail, the transaction will ultimately fail.
- Thus, a transaction will only have two results: failure or success.
- Thus, by definition, a database transaction must be atomic, consistent, isolated, and durable. So these are usually known as the ACID Properties.

SQL Commands

1. TRANSACTIONS in SQL

a) **COMMIT:** This command is used to save all the transactions in the DB.

- Syntax: *COMMIT;*

Eg: *UPDATE Student SET DOB='2005-03-27' WHERE SName='Ram';*

COMMIT;

- Thus, this example would insert the DOB in the given table, which has the name = Ram and then COMMIT these changes in the DB.

SQL Commands

1. TRANSACTIONS in SQL

- b) **ROLLBACK** : This command could only used to reverse transactions that occurred since the last ROLLBACK or COMMIT command. All the modifications must be cancelled if any SQL grouped statements produce a certain error.
 - Syntax: *ROLLBACK*;

Eg: *UPDATE Student SET DOB= '2005-03-27' WHERE SName= 'Ram';*

ROLLBACK;

- Thus, this example would insert the DOB in the given table, which has the name = Ram and then ROLLBACK these changes in the DB. Thus, this type of operation would not create an impact on the table.

SQL Queries

- SQL (Structured Query Language) queries are commands used to interact with a relational database.
- They are used for various operations such as retrieving, updating, inserting, and deleting data in the database.
- Here are some key aspects of SQL queries:
 1. **Basic SELECT Query:** Retrieve all columns from a table.
 - *SELECT * FROM TableName;*
 2. **Specifying Columns:** Retrieve specific columns from a table.
 - *SELECT Column1, Column2 FROM TableName;*

SQL Queries

- 3. Filtering Rows with WHERE:** Retrieve rows based on a condition.
 - *SELECT * FROM TableName WHERE Condition;*
- 4. Sorting Results with ORDER BY:** Sort the result set based on one or more columns.
 - *SELECT * FROM TableName ORDER BY Column1 ASC, Column2 DESC;*
- 5. Grouping Data:** Group rows based on a column's values.
 - *SELECT Column1, COUNT(*) FROM TableName GROUP BY Column1;*
- 6. Joining Tables:** Combine rows from two or more tables based on a related column.
 - *SELECT * FROM Table1 INNER JOIN Table2 ON Table1.Column = Table2.Column;*

SQL Queries

7. **Aggregate Functions:** Perform calculations on data (e.g., SUM, AVG, COUNT, MAX, MIN). It returns a single value.
 - *SELECT AVG(Column1), COUNT(*), MAX(Column2) FROM TableName;*
8. **Wildcards:** Wildcards are special characters used in conjunction with the LIKE operator to perform pattern-matching searches in string data. They allow to match patterns rather than exact values. The two main wildcards in SQL are the percent sign % and the underscore _.
 - a. **Percent Sign %**
 - b. **Underscore _**

SQL Queries

a. Percent Sign %

- The percent sign represents zero or more characters.
- It can be used at the beginning, middle, or end of a pattern.
- Example 1: Match any string ending with "ing".
*SELECT * FROM Words WHERE Word LIKE '%ing';*
- Example 2: Match any string containing "at".
*SELECT * FROM Words WHERE Word LIKE '%at%';*
- Example 3: Match any string starting with "b".
*SELECT * FROM Words WHERE Word LIKE 'b%';*

SQL Queries

b. Underscore _

- The underscore represents a single character.
- It can be used to match a specific character at a specific position.
- Example 1: Match any four-letter word ending with "at".
- *SELECT * FROM Words WHERE Word LIKE '_ _ at';*
- Example 2: Match any word starting with "b" and having a total of four characters.
- *SELECT * FROM Words WHERE Word LIKE 'b_ _ _';*

SQL Subqueries

- SQL subqueries are queries embedded within other queries.
 - They can be used in various parts of a main query, such as the SELECT clause, FROM clause, WHERE clause, or HAVING clause.
 - Subqueries are useful for performing operations on the result set of another query.
 - Here are some key aspects of SQL subqueries:
 1. **Basic Subquery:** Use a subquery within a WHERE clause.
- SELECT * FROM TableName WHERE Column1 = (SELECT Column1 FROM AnotherTable WHERE Condition);*

SQL Subqueries

- 2. Subquery with IN Operator:** Check if a value is within a set of values returned by a subquery.
 - *SELECT * FROM TableName WHERE Column1 IN (SELECT Column1 FROM AnotherTable WHERE Condition);*
- 3. Subquery with EXISTS Operator:** Check if a subquery returns any rows.
 - *SELECT * FROM TableName WHERE EXISTS (SELECT * FROM AnotherTable WHERE Condition);*
- 4. Scalar Subquery:** Use a subquery that returns a single value.
 - *SELECT Column1, (SELECT MAX(Column2) FROM AnotherTable) AS MaxValue FROM TableName;*

SQL Subqueries

- 5. Correlated Subquery:** Reference columns from the outer query within the subquery.
 - `SELECT * FROM TableName t1 WHERE Column1 > (SELECT AVG(Column2) FROM TableName t2 WHERE t1.Column3 = t2.Column3);`
- 6. Subquery in SELECT Clause:** Use a subquery to calculate a value in the SELECT clause.
 - `SELECT Column1, (SELECT COUNT(*) FROM AnotherTable WHERE Condition) AS CountFromSubque FROM TableName;`

Joined Relations

- Joined relations involve combining rows from two or more tables based on related columns.
- The JOIN operation is used to create a joined relation.

*SELECT * FROM Orders*

JOIN Customers ON Orders.CustomerID = Customers.CustomerID;

- This query joins the "Orders" table and the "Customers" table based on the common column "CustomerID."
- The result is a joined relation that includes columns from both tables for rows where the "CustomerID" matches.

Derived Relations

- Derived relations involve creating a new table or result set based on existing tables through operations like projection, selection, or combination.
- The result is often a virtual table, not necessarily stored in the database.

```
SELECT CustomerID, COUNT(OrderID) AS OrderCount FROM Orders  
GROUP BY CustomerID;
```

- This query creates a derived relation by counting the number of orders (OrderCount) for each unique customer (CustomerID) in the "Orders" table.
- The result is a derived relation that shows the customer ID and the count of orders for each customer.

SQL JOIN

- In SQL, JOIN clause is used to combine the records from two or more tables in a database.
- Types of SQL JOIN
 - **INNER JOIN**
 - **LEFT JOIN**
 - **RIGHT JOIN**
 - **FULL JOIN**

INNER JOIN (EQUI JOIN)

- In SQL, INNER JOIN selects records that have matching values in both tables as long as the condition is satisfied.
- It returns the combination of all rows from both the tables where the condition satisfies.

SELECT table1.column1, table1.column2, table2.column1,....

FROM table1

INNER JOIN table2

ON table1.matching_column = table2.matching_column;

LEFT JOIN

- The SQL left join returns all the values from left table and the matching values from the right table.
- If there is no matching join value, it will return NULL.

SELECT table1.column1, table1.column2, table2.column1,....

FROM table1

LEFT JOIN table2

ON table1.matching_column = table2.matching_column;

RIGHT JOIN

- In SQL, RIGHT JOIN returns all the values from the right table and the matched values from the left table.
- If there is no matching in both tables, it will return NULL.

SELECT table1.column1, table1.column2, table2.column1,....

FROM table1

RIGHT JOIN table2

ON table1.matching_column = table2.matching_column;

FULL JOIN

- In SQL, FULL JOIN is the result of a combination of both left and right outer join. So, Join tables have all the records from both tables.
- It puts NULL on the place of matches not found.

SELECT table1.column1, table1.column2, table2.column1,....

FROM table1

FULL JOIN table2

ON table1.matching_column = table2.matching_column;

SELF JOIN

- We use the self join in SQL to join any table to itself in a way that we assume that the table is two tables.
- This way, we temporarily rename at least one of the tables in an SQL statement.

SELECT x.name_of_column, y.name_of_column...

FROM table_1 x, table_1 y

WHERE x.common_field = y.common_field;

Loan		
Loan_no	Branch_name	Amount
L-170	Kathmandu	3000
L-230	Bhaktapur	4000
L-260	Lalitpur	1700

LEFT JOIN			
Loan_no	Branch_name	Amount	Customer_name
L-170	Kathmandu	3000	Ram
L-230	Bhaktapur	4000	Shyam
L-260	Lalitpur	1700	null

Borrower	
Customer_name	Loan_no
Ram	L-170
Shyam	L-230
Hari	L-155

RIGHT JOIN				
Loan_no	Branch_name	Amount	Loan_no	Customer_name
L-170	Kathmandu	3000	L-170	Ram
L-230	Bhaktapur	4000	L-230	Shyam
null	null	null	L-155	Hari

FULL OUTER JOIN			
Loan_no	Branch_name	Amount	Customer_name
L-170	Kathmandu	3000	Ram
L-230	Bhaktapur	4000	Shyam
L-260	Lalitpur	1700	null
null	null	null	Hari

Commands

- *SELECT Loan.Loan_no, Loan.Branch_name, Loan.Amount,
Borrower.Customer_name FROM Loan
LEFT OUTER JOIN Borrower ON Loan.Loan_no = Borrower.Loan_no*
- *SELECT Loan.Loan_no, Loan.Branch_name, Loan.amount, Borrower.Loan_no
Borrower.Customer_name FROM Loan
RIGHT OUTER JOIN Borrower ON Loan.Loan_no = Borrower.Loan_no;*
- *SELECT Loan.Loan_no, Loan.Branch_name, Loan.Amount,
Borrower.Customer_name FROM Loan
FULL OUTER JOIN Borrower ON Loan.Loan_no = Borrower.Loan_no;*

Assignment

- Differentiate between JOIN and UNION in SQL with syntax.

END OF LECTURE 11

- SHIVA.KUNWAR@HOTMAIL.COM
- +977-9819123654

Google classroom code : drbzdcf

PREVIEW FOR LECTURE 12

VIEWS AND MODIFICATION OF DATABASE CONSTRAINTS



Data Analysis Idea

Lecture 12

Database Management System

Er. Shiva Kunwar
Lecturer, GU

Lesson 4: Relational Language and Database Constraints (11hrs)

1. Introduction to SQL
2. SQL Queries and Subqueries
3. Joined and Derived Relations
4. DDL and DML Operations
5. **Views and Modification of Database**
6. Transaction Control Language
7. **Integrity Constraints & Referential Integrity in SQL**
8. **Assertions and Triggers**

Views and Modification of Database

- A database view is a virtual table or result set that is based on existing tables.
- A view does not store the data itself.
- It provides a way to represent data stored in one or more underlying tables.
- Views are useful for simplifying complex queries and providing a customized perspective on the data.
- Views allows users to
 - Structure data in a way that users want
 - Restrict access to data such that user can see and sometime modify exactly what they need
 - Summarize data from various tables, which can be used to generate reports.

Views and Modification of Database

- Creating a View:

```
CREATE VIEW view_name AS
```

```
    SELECT column1, column2 FROM table_name WHERE condition;
```

- Querying a View:

```
SELECT * FROM view_name;
```

- Updating a View:

```
UPDATE view_name SET column=value WHERE condition;
```

- Dropping a View:

```
DROP VIEW view_name;
```

Views and Modification of Database

```
CREATE VIEW SalesSummary AS
SELECT
    Orders.OrderID,
    Customers.CustomerName,
    Products.ProductName,
    OrderDetails.Quantity,
    OrderDetails.UnitPrice,
    OrderDetails.Quantity * OrderDetails.UnitPrice AS TotalPrice
FROM
    Orders
JOIN Customers ON Orders.CustomerID = Customers.CustomerID
JOIN OrderDetails ON Orders.OrderID = OrderDetails.OrderID
JOIN Products ON OrderDetails.ProductID = Products.ProductID;
```

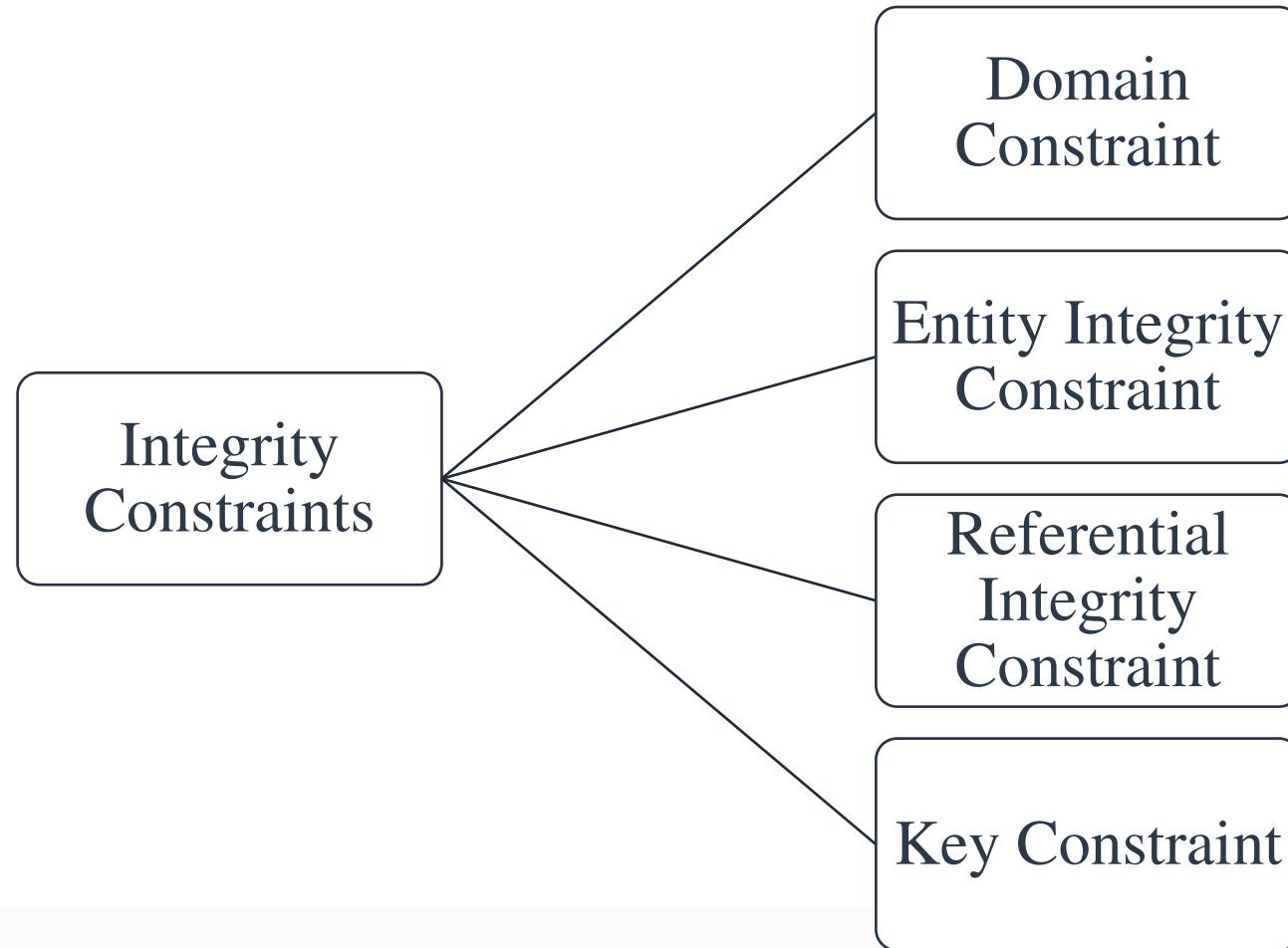
Views and Modification of Database

- In this example, the SalesSummary view provides a consolidated view of sales-related information by joining multiple tables (Orders, Customers, OrderDetails, and Products).
- Users can query this view to obtain a summary of sales data without dealing directly with the complexity of multiple joins.

Integrity Constraints

- Integrity refers to accuracy, consistency and reliability of data in the database.
- Integrity constraints prevent the occurrence of errors and inconsistencies although changes are made to database by authorized users.
- Integrity constraints are a set of rules that are used to maintain the quality of information.
- Integrity constraints ensure that the data insertion, updating, and other processes have to be performed in such a way that data integrity is not affected.
- Thus, integrity constraint is used to guard against accidental damage to the database.

Integrity Constraints



Domain Constraints

- Domain constraints can be defined as the definition of a valid set of values for an attribute.
- Domain constraints are the most elementary form of integrity constraint.
- It is tested by database system whenever a new data item is entered into database.
- System test values inserted in the database and test queries to ensure that the comparisons make sense.

Domain Constraints

```
CREATE TABLE Students (
    ID INT PRIMARY KEY,
    NAME VARCHAR(50) NOT NULL,
    SEMESTER VARCHAR(50) NOT NULL,
    AGE INT NOT NULL
);
```

ID	NAME	SEMESTER	AGE
1000	Ram	1 st	17
1001	Sita	2 nd	24
1002	Leo	5 th	21
1003	Kate	3 rd	A

Not allowed. Because AGE is an integer attribute.

Domain types in SQL

- **Char (n)**: A fixed length character string with user specified length n.
- **Varchar(n)**: A variable length string with user specified maximum length n.
- **Int**: An integer (Machine dependant).
- **Smallint**: A small integer.
- **Numeric (p,d)**: A fixed point number with user specified precision.
- **Real, double precision**: Floating point and double precision floating point numbers.
- **Float (n)**: A floating point number with precision of at least n digits.
- **Date**: A calendar date containing a four digit year, month and day of the month.
- **Time**: The time of a day, in hours, minutes and seconds.
- **Timestamp**: A combination of date and time

Entity Integrity Constraints

- The entity integrity constraint states that primary key value can't be null.
- This is because the primary key value is used to identify individual rows in relation and if the primary key has a null value, then we can't identify those rows.
- A table can contain a null value other than the primary key field.

Entity Integrity Constraints

ID	NAME	SEMESTER	AGE
1000	Ram	1 st	17
1001	Sita	2 nd	24
1002	Leo	5 th	21
	Kate	3 rd	A



Not allowed as PRIMARY KEY cannot contain a NULL value.

Key Constraints

- Keys are the entity set that is used to identify an entity within its entity set uniquely.
- An entity set can have multiple keys, but out of which one key will be the primary key.
- A primary key can contain a unique and null value in the relational table.

Key Constraints

ID	NAME	SEMESTER	AGE
1000	Ram	1 st	17
1001	Sita	2 nd	24
1002	Leo	5 th	21
1002	Kate	3 rd	A



Not allowed as all id rows must be unique.

Referential Integrity

- Referential integrity ensures the consistency and accuracy of relationships between tables.
- It is maintained using foreign keys, where the values in a foreign key column must match the values in the corresponding primary key column of another table or be null.

```
CREATE TABLE Orders (  
        OrderID INT PRIMARY KEY,  
        CustomerID INT,  
    FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)  
);
```

Referential Integrity

```
CREATE TABLE Orders (
    OrderID INT PRIMARY KEY,
    CustomerID INT,
    FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)
);
```

<u>OrderID</u>	<u>CustomerID</u>
1000	11
1001	12
1002	16
1002	14

Not allowed as
16 primary key not found.

<u>CustomerID</u>	<u>Name</u>
11	Ram
12	Sita
13	Leo
14	Kate

Referential Integrity

- Consider two relation department and employee as follows
department(deptno#,dname) | employee(empno#,ename,deptno)
- **Deletion** of particular department from department table
 - need to delete records of employees they belongs to that particular department
 - or delete need not be allowed if there is any employee that is associated to that particular department that we are going to delete.
- Any **update** made in deptno in department table ensures deptno in employee to be updated automatically.
- This implies primary key acts as a referential integrity constraint in a relation.

Cascading Action-Referential Integrity

- *create table account(foreign key(branch-name) references branch on delete cascade on update cascade)*
- **on delete cascade:** if a delete of a tuple in branch results referential-integrity constraint violation, it also delete tuples in relation account that refers to the branch that was deleted.
- **on update cascade :** if a update of a tuple in branch results referential-integrity constraint violation, it updates tuples in relation account that refers to the branch that was updated.

Assertions

- An assertion is a predicate expressing a condition we wish the database to always satisfy.
- Domain constraints, functional dependency and referential integrity are special forms of assertion.
- If a constraint cannot be expressed in these forms, we use an assertion.
- E.g. Sum of loan amounts for each branch is less than the sum of all account balances at the branch.
- Every loan customer keeps a minimum of \$1000 in an account.
- General syntax for creating assertion in SQL is
create assertion <assertion-name> check <predicate>

Assertions

- Example 1: sum of loan amounts for each branch is less than the sum of all account balances at the branch.

create assertion sum-constraint check

*(not exists (select * from branch*

where (select sum(amount) from loan

where loan.branch-name = branch.branch-name)

>= (select sum(amount) from account

where loan.branch-name = branch.branch-name)))

Assertions

- Example 2: every customer must have minimum balance 1000 in an account who are loan holder.

create assertion balance-constraint check

```
(not exists (
    select * from loan
    where not exists (
        select * from borrower, depositor, account
        where loan.loan-number = borrower.loan-number
        and borrower.customer-name = depositor.customer-name
        and depositor.account-number = account.account-number
        and account.balance >= 1000)))
```

Triggers

- A trigger is a statement that is automatically executed by the system as a side effect of a modification to the database.
- While writing a trigger we must specify
 - conditions under which the trigger is executed
 - actions to be taken when trigger executes
- Triggers are useful mechanism to perform certain task automatically when certain condition/s met.
- Sometime trigger is also called rule or action rule.

Triggers

- Basic syntax for trigger

```
CREATE OR REPLACE TRIGGER <TRIGGER NAME>
{BEFORE,AFTER}
{INSERT|DELETE|UPDATE [OF column, . .]} ON <table name>
[REFERENCING {OLD AS <old>, NEW AS <new>}]
[FOR EACH ROW [WHEN <condition>]]
DECLARE
    Variable declaration;
BEGIN
    ...
END;
```

Triggers

```
CREATE TRIGGER overdraft-trigger
AFTER UPDATE ON account
REFERENCING new row AS nrow
FOR EACH ROW WHEN nrow.balance < 0
BEGIN
    insert into loan values
    (nrow.account_number, nrow.branch-name, nrow.balance);
    update account
    set balance = 0
    where account.account_number = nrow.account_number
END;
```

END OF LECTURE 12

- SHIVA.KUNWAR@HOTMAIL.COM
- +977-9819123654

Google classroom code : drbzdcf

PREVIEW FOR LECTURE 13

FUNCTIONAL DEPENDENCIES



Data Analysis Idea

Lecture 13

Database Management System

Er. Shiva Kunwar
Lecturer, GU

Lesson 5: Relational Database Design (7hrs)

- 1. Features of Good Database Design**
- 2. Functional Dependencies**
3. Decomposition using Functional Dependencies
4. Multi-valued and Joined Dependencies
5. Normalization and Different Normal Forms

Features of Good Database Design

- A good database design is crucial for the efficient and effective management of data.
- Enforce data integrity with **primary keys, foreign keys, and constraints**.
- Maintain consistency in **naming conventions** for tables, columns, and other database objects.
- Use standardized and **meaningful names** to improve readability and understanding.
- Implement **access controls** and **authentication** mechanisms to protect sensitive data.
- Enforce proper **user roles and permissions** to restrict access based on user responsibilities.

Features of Good Database Design

- Maintain **comprehensive documentation** for the database design, including entity-relationship diagrams, data dictionaries, and schema documentation.
- Use appropriate **normalization** techniques to eliminate data redundancy and maintain data consistency.
- Ensure that each piece of information is **stored in only one place** to avoid update anomalies.
- Normalize the database to at least the **Third Normal Form (3NF) or higher** as needed.
- Choose the appropriate type of **relationship** (one-to-one, one-to-many, many-to-many) based on the nature of the data.

Features of Good Database Design

- Implement **regular backup and recovery** procedures to protect against data loss.
- Have a **well-defined plan** for restoring the database to a consistent state in case of failures.
- Promote **data independence** by separating the logical data model from the physical storage details.
- Allow for **modifications to the database schema** without affecting the application layer.

Functional Dependencies

- Functional dependencies are constraints on the set of legal relations.
- Functional dependencies are interrelationship among attributes of a relation.
- It defines attributes of relation, how they are related to each other.
- It determines unique value for a certain set of attributes to the value for another set of attributes.
- It typically exists between the primary key and non-key attribute within a table.

Functional Dependencies

- It is abbreviated as FD or f.d. and is represented as $X \rightarrow Y$.
- The left side of FD is known as a determinant, the right side is known as a dependent.
- X will be the primary key attribute, while Y will be a dependent non-key attribute from a similar table as the primary key.
- It shows that the primary key attribute X is functionally dependent on the non-key attribute Y.

Functional Dependencies

- Assume we have an employee table with attributes: Emp_Id, Emp_Name, Emp_Address.
- Here Emp_Id attribute can uniquely identify the Emp_Name attribute of employee table because if we know the Emp_Id, we can tell that employee name associated with it.
- Functional dependency can be written as:
- $Emp_Id \rightarrow Emp_Name$
- We can say that Emp_Name is functionally dependent on Emp_Id.
- But, $Emp_Name \rightarrow Emp_Address$ does not hold functional dependency since Emp_Name cannot uniquely identify Emp_Address.

How to find Functional Dependencies for a Relation?

- Functional Dependencies in a relation are dependent on the domain of the relation. Consider a relation:
- *Student (Stud_No, Stud_Name, Stud_Phone, Stud_State, Stud_Country)*
- We know that *Stud_No* is unique for each student.
- So, $\text{Stud_No} \rightarrow \text{Stud_Name}$, $\text{Stud_No} \rightarrow \text{Stud_Phone}$, $\text{Stud_No} \rightarrow \text{Stud_State}$ and $\text{Stud_No} \rightarrow \text{Stud_Country}$ all will be true.
- Similarly, $\text{Stud_State} \rightarrow \text{Stud_Country}$ will be true as if two records have same *Stud_State*, they will have same *Stud_Country* as well.

How to find Functional Dependency Set for a Relation?

- Functional Dependency set or FD set of a relation is the set of all FDs present in the relation.
- For Example, FD set for relation Student is:
- $\{ Stud_No \rightarrow Stud_Name,$
 $Stud_No \rightarrow Stud_Phone,$
 $Stud_No \rightarrow Stud_State,$
 $Stud_No \rightarrow Stud_Country,$
 $Stud_State \rightarrow Stud_Country \}$

Attribute Closure

- Attribute closure of an attribute set can be defined as set of attributes which can be functionally determined from it.
- If F is a set of functional dependencies, then the closure of F is denoted as F^+ .
- How to find attribute closure of an attribute set?**
- To find attribute closure of an attribute set:
 - Add elements of attribute set to the result set.
 - Recursively add elements to the result set which can be functionally determined from the elements of the result set.

Attribute Closure

- Using FD set of previous example, attribute closure can be determined as:
- $(\text{Stud_No})^+ = \{ \text{Stud_No}, \text{Stud_Name}, \text{Stud_Phone}, \text{Stud_State}, \text{Stud_Country} \}$
- $(\text{Stud_State})^+ = \{ \text{Stud_State}, \text{Stud_Country} \}$

How to Find Candidate Keys and Super Keys Using Attribute Closure?

- If attribute closure of an attribute set contains all attributes of relation, the attribute set will be super key of the relation.
- If no subset of this attribute set can functionally determine all attributes of the relation, the set will be candidate key as well.
- For Example, using FD set of previous example,
- $(\text{Stud_No}, \text{Stud_Name})^+ = \{ \text{Stud_No}, \text{Stud_Name}, \text{Stud_Phone}, \text{Stud_State}, \text{Stud_Country} \}$
- $(\text{Stud_No})^+ = \{ \text{Stud_No}, \text{Stud_Name}, \text{Stud_Phone}, \text{Stud_State}, \text{Stud_Country} \}$

How to Find Candidate Keys and Super Keys Using Attribute Closure?

- $(\text{Stud_No}, \text{Stud_Name})$ will be super key but not candidate key because its subset $(\text{Stud_No})^+$ is equal to all attributes of the relation.
- So, Stud_No will be a candidate key.
- Attributes which are parts of any candidate key of relation are as prime attribute, others are non-prime attributes.
- For Example, Stud_No in Student relation is prime attribute, others are non-prime attribute.

Practice

- Q.1: Consider the relation scheme $R = (E, F, G, H, I, J, K, L, M, N)$ and the set of functional dependencies $\{\{E, F\} \rightarrow \{G\}, \{F\} \rightarrow \{I, J\}, \{E, H\} \rightarrow \{K, L\}, K \rightarrow \{M\}, L \rightarrow \{N\}\}$ on R . What is the key for R ?

- A. $\{E, F\}$
- B. $\{E, F, H\}$
- C. $\{E, F, H, K, L\}$
- D. $\{E\}$

Solution:

Finding attribute closure of all given options, we get:

$$\{E, F\}^+ = \{E, F, G, I, J\}$$

$$\{E, F, H\}^+ = \{E, F, H, G, I, J, K, L, M, N\}$$

$$\{E, F, H, K, L\}^+ = \{\{E, F, H, G, I, J, K, L, M, N\}\}$$

$$\{E\}^+ = \{E\}$$

$\{EFH\}^+$ and $\{EFHKL\}^+$ results in set of all attributes, but EFH is minimal. So, it will be candidate key.

So correct option is (B).

Practice

- Q.2: In a schema with attributes A, B, C, D and E following set of functional dependencies are given $\{ A \rightarrow B, A \rightarrow C, CD \rightarrow E, B \rightarrow D, E \rightarrow A \}$
Which of the following functional dependencies is NOT implied by the above set?

- A. $CD \rightarrow AC$
- B. $BD \rightarrow CD$
- C. $BC \rightarrow CD$
- D. $AC \rightarrow BC$

Solution:

Using FD set given in question,

$(CD)^+ = \{ C,D,E,A,B \}$ which means $CD \rightarrow AC$ holds true.

$(BD)^+ = \{ B,D \}$ which means $BD \rightarrow CD$ can't hold true.

So, this FD is not implied in FD set. So (B) is the required option.

$(BC)^+ = \{ B,C,D,E,A \}$ which means $BC \rightarrow CD$ holds true.

$(AC)^+ = \{ A,B,C,D,E \}$ which means $AC \rightarrow BC$ holds true.

Practice

- Q.3: Consider a relation scheme $R = (A, B, C, D, E, H)$ on which the following functional dependencies hold: $\{A \rightarrow B, BC \rightarrow D, E \rightarrow C, D \rightarrow A\}$.
What are the candidate keys of R ?

- A. AE, BE
- B. AE, BE, DE
- C. AEH, BEH, BCH
- D. AEH, BEH, DEH

Solution:

$(AE)^+ = \{ A, B, E, C, D \}$ which is not set of all attributes.

So AE is not a candidate key.

Hence option A and B are wrong.

$(AEH)^+ = \{ A, B, C, D, E, H \}$

$(BEH)^+ = \{ B, E, H, C, D, A \}$

$(BCH)^+ = \{ B, C, H, D, A \}$ which is not set of all attributes.

So, BCH is not a candidate key.

Hence option C is wrong.

So correct answer is D.

Armstrong Axioms of Functional Dependency

- The term Armstrong Axioms refers to the complete set of inference rules or axioms, introduced by William W. Armstrong, that is used to test the logical implication of functional dependencies.
- If F is a set of functional dependencies, then the closure of F , denoted as F^+ , is the set of all functional dependencies logically implied by F .
- Armstrong's Axioms are a set of rules, that when applied repeatedly, generates a closure of functional dependencies.
- We use Armstrong Axioms to determine the functional dependency in the database.
- Generally, it is used to derive other functional dependency in the database using the given functional dependency.

Armstrong Axioms

- **Axiom of Reflexivity:** If A is a set of attributes and B is a subset of A, then A holds B. If $B \subseteq A$ then $A \rightarrow B$. This property is trivial property.
- **Axiom of Augmentation:** If $A \rightarrow B$ holds and Y is the attribute set, then $AY \rightarrow BY$ also holds. That is adding attributes to dependencies, does not change the basic dependencies. If $A \rightarrow B$, then $AC \rightarrow BC$ for any C.
- **Axiom of Transitivity:** Same as the transitive rule in algebra, if $A \rightarrow B$ holds and $B \rightarrow C$ holds, then $A \rightarrow C$ also holds. $A \rightarrow B$ is called A functionally which determines B. If $X \rightarrow Y$ and $Y \rightarrow Z$, then $X \rightarrow Z$.

Armstrong Axioms – Secondary Rules

- **Union:** If $A \rightarrow B$ holds and $A \rightarrow C$ holds, then $A \rightarrow BC$ holds.
- **Composition:** If $A \rightarrow B$ and $X \rightarrow Y$ hold, then $AX \rightarrow BY$ holds.
- **Decomposition:** If $A \rightarrow BC$ holds then $A \rightarrow B$ and $A \rightarrow C$ hold.
- **Pseudo Transitivity:** If $A \rightarrow B$ holds and $BC \rightarrow D$ holds, then $AC \rightarrow D$ holds.
- **Self Determination:** It is similar to the Axiom of Reflexivity, i.e. $A \rightarrow A$ for any A.
- **Extensivity:** Extensivity is a case of augmentation. If $AC \rightarrow A$, and $A \rightarrow B$, then $AC \rightarrow B$. Similarly, $AC \rightarrow ABC$ and $ABC \rightarrow BC$. This leads to $AC \rightarrow BC$.

Functional Dependencies Types

- **Trivial functional dependency**
- If A is a set of attributes and B is a subset of A, then A holds B.
- If $B \subseteq A$ then $A \rightarrow B$.
- This has trivial functional dependency.
- The following dependencies are also trivial like: $A \rightarrow A$, $B \rightarrow B$

Functional Dependencies Types

- Consider a table with two columns Employee_Id and Employee_Name.
- $\{Employee_id, Employee_Name\} \rightarrow Employee_Id$ is a trivial functional dependency as Employee_Id is a subset of $\{Employee_Id, Employee_Name\}$.
- Also, $Employee_Id \rightarrow Employee_Id$ and $Employee_Name \rightarrow Employee_Name$ are trivial dependencies too.

Functional Dependencies Types

- **Non-Trivial Functional Dependency**
- $A \rightarrow B$ has a non-trivial functional dependency if B is not a subset of A.
- **Complete Non-Trivial Functional Dependency**
- When A intersection B is NULL, then $A \rightarrow B$ is called as complete non-trivial.

$ID \rightarrow Name$,

$Name \rightarrow DOB$

END OF LECTURE 13

- SHIVA.KUNWAR@HOTMAIL.COM
- +977-9819123654

Google classroom code : drbzdcf

PREVIEW FOR LECTURE 14

DECOMPOSITION USING FUNCTIONSL DEPENDENCIES
MULTI-VALUED AND JOINED DEPENDENCIES
NORMALIZATIONS



Data Analysis Idea

Lecture 14

Database Management System

Er. Shiva Kunwar
Lecturer, GU

Lesson 5: Relational Database Design (7hrs)

1. Features of Good Database Design
2. Functional Dependencies
- 3. Decomposition using Functional Dependencies**
- 4. Multi-valued and Joined Dependencies**
- 5. Normalization and Different Normal Forms**

Decomposition using Functional Dependencies

- The term decomposition refers to the process in which we break down a table in a database into various elements or parts.
- Thus, decomposition replaces a given relation with a collection of various smaller relations to collect a particular set of data.
- Breaking a relation X into $\{X_1, X_2, \dots, X_n\}$ is decomposition.
- Decomposition must always be lossless. This way, we can rest assured that the data/information that was there in the original relation can be reconstructed accurately based on the decomposed relations.
- In case the relation is not decomposed properly, then it may eventually lead to problems such as information loss.

Types of Decomposition

- Decomposition is of two major types in DBMS:
 - Lossless Decomposition
 - Lossy Decomposition

Students (#name, phone_no, major)
 Transcript (#name, course, grade)
 Teacher (course, prof)

Student_info (#name, course, phone_no, major, prof, grade)

name	course	phone_no	major	prof	grade
John	353	374537	Computer Science	Smith	A
Scott	329	427993	Mathematics	James	S
John	328	374537	Computer Science	Adams	A
Allen	432	729312	Physics	Blake	C
Turner	523	252731	Chemistry	Miller	B
John	320	374537	Computer Science	Martin	A
Scott	328	727993	Mathematics	Ford	B

Lossless Decomposition

- A decomposition is said to be lossless when it is feasible to reconstruct the original relation R using joins from the decomposed tables.
- It is the most preferred choice. This way, the information will not be lost from the relation after its decomposition.
- A lossless join would eventually result in the original relation that is very similar.
- For example,
- Let us take ‘A’ as the Relational Schema, having an instance of ‘a’. Consider that it is decomposed into: A1, A2, A3, . . . An; with instance: a1, a2, a3, . . . An.
- If $a_1 \bowtie a_2 \bowtie a_3 \dots \bowtie a_n$, then it is known as ‘Lossless Join Decomposition’.

Lossless Decomposition

- In the case of lossless decomposition, one selects the common attribute.
- Here, the criteria used for the selection of a common attribute as this attribute has to be a super key or a candidate key in either relation X1, relation X2, or either of them.
- In simpler words, the decomposition of the relation X into the relations X1 and X2 will be a lossless join decomposition in DBMS when a minimum of one of these functional dependencies is in F+ (Functional dependency closure).
$$X1 \cap X2 \rightarrow X1 \quad \text{OR} \quad X1 \cap X2 \rightarrow X2$$

Conditions for Lossless Decomposition

- Let us consider a relation X. In case we decompose this relation into relation X1 and relation X2 sub-parts. This decomposition will be referred to as a lossless decomposition in case it satisfies these statements:
 - If we **union** the sub relations X1 and X2, then it should **consist of all the attributes** available before the decomposition in the original relation X.
 - The **intersections** of X1 and X2 can **never be Null**. There must be a common attribute in the sub relation. This common attribute must consist of some unique data/information.
- Here, the common attribute needs to be the super key of the sub relations, either X1 or X2.

Conditions for Lossless Decomposition

- $X = (P, Q, R)$
- $X_1 = (P, Q), X_2 = (Q, R)$
- The relation X here consists of three attributes P , Q , and R .
- X decomposes into two separate relations X_1 and X_2 .
- Thus, each of these X_1 and X_2 both have two attributes. The common attribute among each of these is Q .
- Combining X_1 and X_2 would generate the original relation $X = (P, Q, R)$.
- Remember that the value present in column Q has to be unique. In case it consists of a duplicate value, then a lossless-join decomposition would not be possible here.

Lossy Decomposition

- Lossy decomposition is when a relation gets decomposed into multiple relational schemas, in such a way that retrieving the original relation leads to a loss of information.
- Thus, a lossy decomposition is bound to lose information.
- Careless decomposition is another name for lossy join decomposition.
- It is because there is an introduction of various extraneous tuples in the sub relations' natural join.
- These extraneous tuples make it very difficult to identify the original tuples.

Lossy Decomposition

- $X = (P, Q, R)$
- $X_1 = (P, Q), X_2 = (R)$
- Now, if we try to join both the tables mentioned above, we won't be able to do it- since the relation X_1 isn't part of the relation X_2 .
- Combining X_1 and X_2 would generate completely different relation than original relation.

Lossy Decomposition

- Addition of extraneous tuples. $X = (P, Q, R)$, $X_1 = (P, Q)$, $X_2 = (P, R)$

X		
P	Q	R
P1	Q1	R1
P2	Q1	R1
P1	Q2	R2
P1	Q3	R3

X1	
P	Q
P1	Q1
P2	Q1
P1	Q2
P1	Q3

X2	
P	R
P1	R1
P2	R1
P1	R2
P1	R3

X		
P	Q	R
P1	Q1	R1
P1	Q1	R2
P2	Q1	R1
P1	Q2	R2
P1	Q2	R1
P1	Q3	R3
P1	Q3	R1

Properties of Decomposition

- Decomposition must have the following properties:
 1. **Decomposition Must be Lossless**
 2. **Dependency Preservation**
 3. **Lack of Data Redundancy**

Properties of Decomposition

1. Decomposition Must be Lossless

- Decomposition must always be lossless, which means the information must never get lost from a decomposed relation.
- This way, we get a guarantee that when joining the relations, the join would eventually lead to the same relation in the result as it was actually decomposed.

Properties of Decomposition

2. Dependency Preservation

- Dependency is a crucial constraint on a database, and a minimum of one decomposed table must satisfy every dependency.
- If a relation R is decomposed into relation R1 and R2, then the dependencies of R either must be a part of R1 or R2 or must be derivable from the combination of functional dependencies of R1 and R2.
- Closure of functional dependencies after decomposition must be same as closure of FDs before decomposition for dependency preservation.**
- For example, suppose there is a relation R (A, B, C, D) with functional dependency set $(A \rightarrow BC)$. The relational R is decomposed into R1(ABC) and R2(AD) which is dependency preserving because FD $A \rightarrow BC$ is a part of relation R1(ABC).

Properties of Decomposition

3. Lack of Data Redundancy

- It is also commonly termed as a repetition of data/information.
- According to this property, decomposition must not suffer from data redundancy.
- When decomposition is careless, it may cause problems with the overall data in the database.
- When we perform normalization, we can easily achieve the property of lack of data redundancy.

Multi Valued Dependencies

- Multivalued dependency would occur whenever two separate attributes in a given table happen to be independent of each other.
- And yet, both of these depend on another third attribute.
- The multivalued dependency contains at least two of the attributes dependent on the third attribute.
- This is the reason why it always consists of at least three of the attributes.

Multi Valued Dependencies

- In this case, the columns Color and Manuf_Month are dependent on Car_Model, and they are independent of each other. Thus, we can call both of these columns multivalued. The dependencies are represented as:
- $\text{Car_Model} \rightarrow \rightarrow \text{Manuf_Month}$
- $\text{Car_Model} \rightarrow \rightarrow \text{Color}$
- We can read this as:
“Car_Model multidetermined Manuf_Month” and
“Car_Model multidetermined Color”.

Car_Model	Manuf_Month	Color
S2011	Jan	Yellow
S2011	Feb	Red
S3001	Mar	Yellow
S3001	Apr	Red
S4006	May	Yellow

Joined Dependencies

- Whenever we can recreate a table by simply joining various tables where each of these tables consists of a subset of the table's attribute, then this table is known as a Join Dependency.
- Thus, it is like a generalization of MVD.
- We can relate the JD to 5NF. Herein, a relation can be in 5NF only when it's already in the 4NF. Remember that it cannot be further decomposed.

Joined Dependencies

- We can decompose the table given above into these three tables given below. And thus, it is not in the Fifth Normal Form.
- Our Join Dependency would be:
- $\{(\text{Car_Model}, \text{Manuf_Month}), (\text{Car_Model}, \text{Color}), (\text{Manuf_Month}, \text{Color})\}$.
- The join relation of these three relations is equal to the very original relation <Car>

Car_Model	Manuf_Month	Color
S2011	Jan	Yellow
S2001	Feb	Red
S3001	Mar	Yellow
S4006	Apr	Red
S4006	May	Yellow

Normalizations

- Database normalization is the process of organizing the attributes of the database to reduce or eliminate data redundancy (having the same data but at different places).
- Data redundancy unnecessarily increases the size of the database as the same data is repeated in many places. Inconsistency problems also arise during insert, delete and update operations. This is solved by Normalization.
- Normalization is an important process in database design that helps in improving the efficiency, consistency, and accuracy of the database.
- It makes it easier to manage and maintain the data and ensures that the database is adaptable to changing business needs.

Features of DB Normalizations

- **Elimination of Data Redundancy:** Normalization helps in reducing or eliminating repetitions or redundancy, which can improve the efficiency and consistency of the database.
- **Ensuring Data Consistency:** By eliminating redundancy, normalization helps in preventing inconsistencies and contradictions that can arise due to different versions of the same data.
- **Simplification of Data Management:** By breaking down a complex data structure into simpler tables, normalization makes it easier to manage the data, update it, and retrieve it.

Features of DB Normalizations

- **Improved Database Design:** By organizing the data in a structured and systematic way, normalization makes it easier to design and maintain the database. It also makes the database more flexible and adaptable to changing business needs.
- **Avoiding Update Anomalies:** Normalization helps in avoiding update anomalies, which can occur when updating a single record in a table affects multiple records in other tables.
- **Standardization:** Normalization helps in standardizing the data in the database. By organizing the data into tables and defining relationships between them.

Normal Forms

- Normal forms are used to eliminate or reduce redundancy in database tables.
- In database management systems (DBMS), normal forms are a series of guidelines that help to ensure that the design of a database is efficient, organized, and free from data anomalies.
- **First Normal Form (1NF):** This is the most basic level of normalization. In 1NF, each table cell should contain only a single value, and each column should have a unique name. The first normal form helps to eliminate duplicate data and simplify queries.
- **Second Normal Form (2NF):** 2NF eliminates redundant data by requiring that each non-key attribute be dependent on the primary key. This means that each column should be directly related to the primary key, and not to other columns.

Normal Forms

- **Third Normal Form (3NF):** 3NF builds on 2NF by requiring that all non-key attributes are independent of each other. This means that each column should be directly related to the primary key, and not to any other columns in the same table.
- **Boyce-Codd Normal Form (BCNF):** BCNF is a stricter form of 3NF that ensures that each determinant in a table is a candidate key. In other words, BCNF ensures that each non-key attribute is dependent only on the candidate key.
- **Fourth Normal Form (4NF):** 4NF is a further refinement of BCNF that ensures that a table does not contain any multi-valued dependencies.
- **Fifth Normal Form (5NF):** 5NF is the highest level of normalization and involves decomposing a table into smaller tables to remove data redundancy and improve data integrity.

Advantages of Normal Form

- **Reduced data redundancy:** Normalization helps to eliminate duplicate data in tables, reducing the amount of storage space needed and improving database efficiency.
- **Improved data consistency:** Normalization ensures that data is stored in a consistent and organized manner, reducing the risk of data inconsistencies and errors.
- **Simplified database design:** Normalization provides guidelines for organizing tables and data relationships, making it easier to design and maintain a database.
- **Improved query performance:** Normalized tables are typically easier to search and retrieve data from, resulting in faster query performance.
- **Easier database maintenance:** Normalization reduces the complexity of a database by breaking it down into smaller, more manageable tables, making it easier to add, modify, and delete data.

END OF LECTURE 14

- SHIVA.KUNWAR@HOTMAIL.COM
- +977-9819123654

Google classroom code : drbzdcf

PREVIEW FOR LECTURE 15

DIFFERENT NORMAL FORMS



Data Analysis Idea

Lecture 15

Database Management System

Er. Shiva Kunwar
Lecturer, GU

Lesson 5: Relational Database Design (7hrs)

1. Features of Good Database Design
2. Functional Dependencies
3. Decomposition using Functional Dependencies
4. Multi-valued and Joined Dependencies
5. Normalization and **Different Normal Forms**

Normal Forms

- Normal forms are used to eliminate or reduce redundancy in database tables.
- In database management systems (DBMS), normal forms are a series of guidelines that help to ensure that the design of a database is efficient, organized, and free from data anomalies.
- **First Normal Form (1NF):** This is the most basic level of normalization. In 1NF, each table cell should contain only a single value, and each column should have a unique name. The first normal form helps to eliminate duplicate data and simplify queries.
- **Second Normal Form (2NF):** 2NF eliminates redundant data by requiring that each non-key attribute be dependent on the primary key. This means that each column should be directly related to the primary key, and not to other columns.

Normal Forms

- **Third Normal Form (3NF):** 3NF builds on 2NF by requiring that all non-key attributes are independent of each other. This means that each column should be directly related to the primary key, and not to any other columns in the same table.
- **Boyce-Codd Normal Form (BCNF):** BCNF is a stricter form of 3NF that ensures that each determinant in a table is a candidate key. In other words, BCNF ensures that each non-key attribute is dependent only on the candidate key.
- **Fourth Normal Form (4NF):** 4NF is a further refinement of BCNF that ensures that a table does not contain any multi-valued dependencies.
- **Fifth Normal Form (5NF):** 5NF is the highest level of normalization and involves decomposing a table into smaller tables to remove data redundancy and improve data integrity.

Normal Forms

1NF	2NF	3NF	4NF	5NF	
Conditions	Eliminate Repeating Groups	Eliminate Partial Functional Dependency	Eliminate Transitive Dependency	Eliminate Multi-values Dependency	
Decomposition of Relation	R	R_{11} R_{12}	R_{21} R_{22} R_{23}	R_{31} R_{32} R_{33} R_{34}	R_{41} R_{42} R_{43} R_{44} R_{45}

First Normal Form

- A relational schema R is in 1NF if the domains of all attributes of R are atomic.
- Ensure each table has primary key and minimal sets of attributes which uniquely identify a record.
- Eliminate repeating group (attribute having more than one value for a primary key)
- First normal form disallows the multi-valued and composite attribute, or their combination
- **Each attribute must be atomic.**
- **The Domain of attributes must not change.**
- **Every Column/ Attribute must have a Unique Name.**
- **The order of Data does not matter.**

First Normal Form

TABLE_ITEMS		
Item No	Hues	Cost
1	Pink, Black	100
2	Red, Grey	200
3	Brown	300

Converts to 1NF

TABLE_HUES	
Item No	Hues
1	Pink
1	Black
2	Red
2	Grey
3	Brown

TABLE_COST	
Item No	Cost
1	100
2	200
3	300

First Normal Form

- First Normal Form (1NF) does not eliminate redundancy, but rather, it's that it eliminates repeating groups.

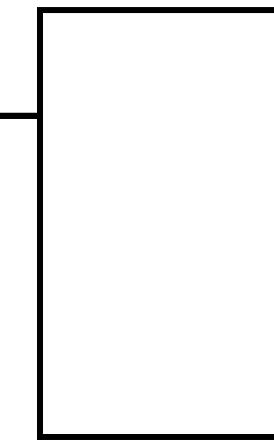
Second Normal Form

- A relation R is in 2NF if
 - it is in 1NF,
 - every non-key attribute is fully dependent on the primary key and
 - should not consist of partial dependency.
- It removes partial functional dependencies into a new relationship.

Second Normal Form

TABLE_Student		
<u>Stud_No</u>	<u>Course_No</u>	Course_Fee
1	C1	1000
2	C2	1500
1	C4	2000
4	C3	1000
4	C1	1000

Converts to 2NF



TABLE_COURSE	
<u>Stud_No</u>	Course_No
1	C1
2	C2
1	C4
4	C3
4	C1

TABLE_FEE	
<u>Course_No</u>	Course_Fee
C1	1000
C2	1500
C3	1000
C4	2000

COURSE_FEE would be a non-prime attribute

Second Normal Form

- 2NF tries to reduce the redundant data getting stored in memory.

Third Normal Form

- A relation R is in 3NF if
 - it is in 2NF and
 - every non-key attribute is non-transitively dependent on the primary key.
- All the non-key attributes must depend on the primary key but not any other non-key attribute.
- A relation is in third normal form if it holds at least one of the following conditions for every non-trivial function dependency $X \rightarrow Y$.
 - X is a super key.
 - Y is a prime attribute, i.e., each element of Y is part of some candidate key.

Third Normal Form

TABLE_Employee

Emp_id	Emp_Name	Emp_Zip	Emp_State	Emp_City
111	Ram	33700	Gandaki	Pokhara
222	Sita	44600	Bagmati	Lalitpur
333	Hari	32900	Lumbini	Butwal

Here, EMP_STATE & EMP_CITY dependent on EMP_ZIP and EMP_ZIP dependent on EMP_ID. The non-prime attributes (EMP_STATE, EMP_CITY) are transitively dependent on the super key(EMP_ID). It violates the rule of the third normal form.

TABLE_Employee_Zip

Emp_id	Emp_Name	Emp_Zip
111	Ram	33700
222	Sita	44600
333	Hari	32900

TABLE_Employee_Zip_City

Emp_Zip	Emp_State	Emp_City
33700	Gandaki	Pokhara
44600	Bagmati	Lalitpur
32900	Lumbini	Butwal

Third Normal Form

- 3NF is sufficient to design a normal relational database since a majority of third normal form tables stay free from anomalies of deletion, updating, and insertion.
- Added to this, 3NF is bound to ensure losslessness and preserve functional dependencies.

Boyce Codd Normal Form

- It involves decompositions involving relations with more than one candidate key, where candidate keys are composite and overlapping,
- A relation R is in BCNF if every determinant is a candidate key,
- BCNF is the advanced version of 3NF. It is stricter than 3NF.

Boyce Codd Normal Form

TABLE1

Emp_id	Emp_Country	Emp_Dept	Emp_Type	Emp_DeptNo
111	Nepal	Innovation	D394	283
111	Nepal	Marketing	D394	300
333	China	Developing	D283	232

TABLE2.1

Emp_id	Emp_Country
111	Nepal
333	China

TABLE2.2

Emp_Dept	Emp_Type	Emp_DeptNo
Innovation	D394	283
Marketing	D394	300
Developing	D283	232

In the table1, Functional dependencies are as follows:

$\text{EMP_ID} \rightarrow \text{EMP_COUNTRY}$

$\text{EMP_DEPT} \rightarrow \{\text{DEPT_TYPE}, \text{EMP_DEPT_NO}\}$

Candidate key: {EMP-ID, EMP-DEPT}

The table is not in BCNF because neither EMP_DEPT nor EMP_ID alone are keys.

TABLE3

Emp_id	Emp_DeptNo
111	283
111	300
333	232

Boyce Codd Vs 3NF

- Relation in BCNF has no information repetition but relation in 3NF has information repetition.
- Decomposition in 3NF is lossless and dependency-preserving but decomposition in BCNF is not dependency-preserving.

Fourth Normal Form

- A relation R is in 4NF if
 - it is in BCNF and
 - has no multivalued dependencies.
- For a dependency $A \rightarrow B$, if for a single value of A, multiple values of B exist, then the relation will be a multi-valued dependency.

Fourth Normal Form

- The given STUDENT table is in 3NF, but the COURSE and HOBBY are two independent entity. Hence, there is no relationship between COURSE and HOBBY.
- In the STUDENT relation, a student with STU_ID, 21 contains two courses, Computer and Math and two hobbies, Dancing and Singing. So there is a Multi-valued dependency on STU_ID, which leads to unnecessary repetition of data.
- So to make the above table into 4NF, we can decompose it into two tables:

STU_ID	COURSE	HOBBY
21	Computer	Dancing
21	Math	Singing
34	Chemistry	Dancing
74	Biology	Cricket
59	Physics	Hockey

STU_ID	COURSE

STU_ID	HOBBY

Fifth Normal Form

- A relation R is in 5NF if
 - it is in 4NF,
 - not contain any join dependency,
 - joining should be lossless and
 - every join dependency is implied by a candidate key.

Fifth Normal Form

- Multivalued dependencies are removed by 4NF, and join dependencies are removed by 5NF.
- The greatest degrees of database normalization, 4NF and 5NF, might not be required for every application.
- Normalizing to 4NF and 5NF might result in more complicated database structures and slower query speed, but it can also increase data accuracy, dependability, and simplicity.

END OF LECTURE 15

- SHIVA.KUNWAR@HOTMAIL.COM
- +977-9819123654

Google classroom code : drbzdcf

PREVIEW FOR LECTURE 16

TRANSACTION MANAGEMENT SYSTEM CONCURRENCY CONTROL



Data Analysis Idea

Lecture 16

Database Management System

Er. Shiva Kunwar
Lecturer, GU

Lesson 6: Transaction Management and Concurrency Control (5hrs)

1. Concept of Transaction and ACID Properties
2. Concurrency Control and Recovery
3. Serializability Concept
4. Lock-based and Timestamp-based Protocols

Transaction

- Logical unit of work formed from group of operations.
- It is the atomic unit of work that is either completed in its entirety or not at all.
- It may also be defined as the unit or piece of program execution that accesses and possibly updates the various data items.
- A transaction is the DBMS abstract view of a user program that consists of sequence of reads and writes.

Transaction

- Transaction access data using two operations:
 - Read(A): Read operations Read(A) or R(A) reads the value of A from the database and stores it in a buffer in the main memory.
 - Write (A): Write operation Write(A) or W(A) writes the value back to the database from the buffer.

Transaction

- Transaction processing is information processing that is derived into individual indivisible operations called transactions.
- Each transaction must succeed or fail as a complete unit but cannot remain in intermediate state.
- Transaction processing maintains a system in a consistent state.
- Eg: T1: read (A)
 A = A-100
 write(A)
 read(B)
 B=B+1000
 write(B)

Transaction

- Transaction Processing has two important operations:
- **Commit:** After all instructions of a transaction are successfully executed, the changes made by a transaction are made permanent in the database.
`DELETE FROM Employee WHERE AGE = 20;
COMMIT;`
- **Rollback:** If a transaction is not able to execute all operations successfully, all the changes made by a transaction are undone.
`DELETE FROM Employee WHERE AGE = 20;
ROLLBACK;`

Transaction ACID Properties

- **Atomicity:** either all operations of transactions are reflected properly in database or known at all.
- **Consistency:** execution of a transaction in isolation preserves the consistency of database.
- **Isolation:** even though multiple transactions may execute concurrently the system guarantees that for every pair of transaction TI and TJ, for TI either TJ finishes execution before TI started or TJ starts execution after finishing TI. Each transaction is unaware of other transaction.
- **Durability:** after successful completion of transaction the changes in database persists even if there are system failures.

ACID Properties - Atomicity

- Either the entire transaction takes place at once or doesn't happen at all.
- There is no midway i.e. transactions do not occur partially.
- Each transaction is considered as one unit and either runs to completion or is not executed at all.
- It involves the following two operations.
 - **Abort:** If a transaction aborts, changes made to the database are not visible.
 - **Commit:** If a transaction commits, changes made are visible.
- Atomicity is also known as the 'All or nothing rule'.

ACID Properties - Atomicity

- Consider the following transaction T consisting of T1 and T2: Transfer of 100 from account X to account Y.
- If the transaction fails after completion of T1 but before completion of T2. (say, after write(X) but before write(Y)), then the amount has been deducted from X but not added to Y.
- This results in an inconsistent database state.
- Therefore, the transaction must be executed in its entirety in order to ensure the correctness of the database state.

ACID Properties - Consistency

- This means that integrity constraints must be maintained so that the database is consistent before and after the transaction.
- It refers to the correctness of a database.
- Referring to the example above,
- The total amount before and after the transaction must be maintained.
- Total before T occurs = $500 + 200 = 700$.
- Total after T occurs = $400 + 300 = 700$.
- Therefore, the database is consistent. Inconsistency occurs in case T1 completes but T2 fails. As a result, T is incomplete.

ACID Properties - Isolation

- This property ensures that multiple transactions can occur concurrently without leading to the inconsistency of the database state.
- Transactions occur independently without interference.
- Changes occurring in a particular transaction will not be visible to any other transaction until that particular change in that transaction is written to memory or has been committed.
- This property ensures that the execution of transactions concurrently will result in a state that is equivalent to a state achieved these were executed serially in some order.

ACID Properties - Isolation

- Let $X = 500, Y = 500$.
- Consider two transactions T and T'' .
- Suppose T has been executed till Read (Y) and then T'' starts. As a result, interleaving of operations takes place due to which T'' reads the correct value of X but the incorrect value of Y and sum computed by
 $T'': (X+Y = 50, 000 + 500 = 50, 500)$
- is thus not consistent with the sum at end of the transaction:
- $T: (X+Y = 50, 000 + 450 = 50, 450)$.
- This results in database inconsistency, due to a loss of 50 units. Hence, transactions must take place in isolation and changes should be visible only after they have been made to the main memory.

ACID Properties - Durability

- This property ensures that once the transaction has completed execution, the updates and modifications to the database are stored in and written to disk and they persist even if a system failure occurs.
- These updates now become permanent and are stored in non-volatile memory.
- The effects of the transaction, thus, are never lost.

ACID Properties Benefits

- ACID properties are the four key characteristics that define the reliability and consistency of a transaction in a Database Management System (DBMS).
- ACID properties provide a framework for ensuring data consistency, integrity, and reliability in DBMS.
- They ensure that transactions are executed in a reliable and consistent manner, even in the presence of system failures, network issues, or other problems.
- These properties make DBMS a reliable and efficient tool for managing data in modern organizations.

Transaction States

- Active state
 - it is the initial state
 - transaction stays in this state while it is executing
- Partially Committed
 - transaction is in partially committed after the final statement has been executed
- Aborted
 - transaction is in aborted state after it has been rolled back and the database has been restored to its state prior to the start of transaction
- Committed
 - transaction is in committed state after successful compilation

Transaction States

- Transaction has committed only if it enters committed state.
- a transaction has aborted only if it enters aborted state.
- A transaction has terminated if either committed or aborted.
- After transaction being in aborted state, system has two options either it can restart transaction or rollback to initial state..
- Transaction processing system usually allow multiple transactions to run concurrently, this causes several complications with consistency of data.

Transaction

- Though transactions can be run serially to ensure consistency, concurrent execution of transaction has following advantages:
 - Improved throughput and resources utilization.
 - Reduced waiting time.
- Database system control the interactions among the concurrent transactions to prevent from different consistency of database through a variety of mechanisms called concurrency control schemes.
- When transactions are executing concurrently in an interleaved fashion then the order of execution of operations from various transactions is known as schedule.

Schedule

- A schedule S specifies the chronological order in which the operations of concurrent transactions are executed.
- Let transaction T1 transfers Rs 50 from account A to B.
- Transaction T2 transfers 10% of balance from A to B.
- Let $A=1000$, $B=2000$
- T1: read (A)
 $A=A-50$
write(A)
read(B)
 $B=B+50$
write(B)
- T2: read (A)
 $temp=A*0.1$
 $A=A-temp$
write(A)
read(B)
 $B=B+temp$
write(B)

Serial Schedule

$$(A+B)\text{initial} = (A+B)\text{final}$$

T1: read (A)	1000			
A=A-50	950			
write(A)	950			
read(B)	2000	T2: read (A)	950	
B=B+50	2050	temp=A*0.1	95	
write(B)	2050	A=A-temp	855	
		write(A)	855	
		read(B)	2050	
		B=B+temp	2145	
		write(B)	2145	

T2: read (A)	1000			
temp=A*0.1	100			
A=A-temp	900			
write(A)	900			
read(B)	2000	T1: read (A)	900	
B=B+temp	2100	A=A-100	850	
write(B)	2100	write(A)	850	
		read(B)	2100	
		B=B+50	2150	
		write(B)	2150	

Serial Schedule

- In the schedule shown, consistency by transferring Rs 50 from account A to account B is preserved. Since $(A+B)_{\text{initial}} = (A+B)_{\text{final}}$
- Similarly, if the order of execution of transaction is reversed i.e., execute T2 first and then execute T1, then also $(A+B)_{\text{initial}} = (A+B)_{\text{final}}$
- Hence consistency of data is preserved.

Non-Serial Schedule

T1: read (A)	1000		
A=A-50	950		
write(A)	950	T2: read (A)	950
		temp=A*0.1	95
		A=A-temp	855
read(B)	2000	write(A)	855
B=B+50	2050		
write(B)	2050	read(B)	2050
		B=B+temp	2145
		write(B)	2145

T1: read (A)	1000		
A=A-50	950	T2: read (A)	1000
		temp=A*0.1	100
		A=A-temp	900
		write(A)	900
write(A)	950	read(B)	2000
read(B)	2000		
B=B+50	2050		
write(B)	2050	B=B+temp	2150
		write(B)	2150

Non-Serial Schedule

- Here both 1st and 2nd box are non serial schedule.
- In 1st box, $(A+B)_{\text{initial}} = (A+B)_{\text{final}}$.
- Hence consistency is preserved, that means it is in correct state.
- In 2nd box,
- $(A+B)_{\text{initial}} = 1000+2000 = 3000$
- $(A+B)_{\text{final}} = 950 + 150 = 3100$.
- Hence consistency of data box is not maintained and is not in correct state.

Serializability Concept

- Basic Assumption – Each transaction preserves database consistency.
- Thus, serial execution of a set of transactions preserves database consistency.
- A (possibly concurrent) schedule is serializable if it is equivalent to a serial schedule.
- Main objective of serializability is to search non serial schedules that allow transaction to execute concurrently without interfering consistency.
- We can conclude that non serial schedule is correct if it produces same result as serial execution.

Serializability Rules

- Ordering of read/write is important.
- If two transactions only read data item, they do not conflict, and order is not important.
- If two transactions either read or write separate data items they do not conflict and order is not important.
- If one transaction writes to data item and another reads or writes same data item order of execution is important.

Conflict Serializability

- If your schedule S can be transferred into a schedule S' by a series of swaps of non conflicting instructions, we say that S and S' are conflict equivalent.
- We say that the schedule S is conflict serializable if it is conflict equivalent to a serial schedule.

Instruction I _i	Instruction I _j	Result
Read(Q)	Read(Q)	No conflict
Read(Q)	Write(Q)	Conflict
Write(Q)	Read(Q)	Conflict
Write(Q)	Write(Q)	Conflict

Conflict Serializability

- Here, schedule A can be transformed into serial schedule B by series of swaps of non conflicting instructions.
- Hence schedule A is conflict serializable.

Schedule A		Schedule B	
T1	T2	T1	T2
read(A)		read(A)	
write(A)		write(A)	
	read(A)	read(B)	
	write(A)	write(B)	
read(B)			read(A)
write(B)			write(A)
	read(B)		read(B)
	write(B)		write(B)

Conflict Serializability

- Here, we cannot transform schedule X into schedule Y by swapping non conflicting instruction
- Hence, schedule X is non conflict serializable.

Schedule X		Schedule Y	
T1	T2	T1	T2
read(A)		read(A)	
	write(A)	write(A)	
write(A)			write(A)
	read(B) write(B)		

View serializability

- We say that the schedule S is view serializable if it is view equivalent to a serial schedule.
- Let S and S' be two schedules with the same set of transactions. S and S' are view equivalent if the following three conditions are met:
 1. For each data item Q , if transaction T_i reads the initial value of Q in schedule S , then transaction T_i must, in schedule S' , also read the initial value of Q .
 2. For each data item Q if transaction T_i executes $\text{read}(Q)$ in schedule S , and that value was produced by transaction T_j (if any), then transaction T_i must in schedule S' also read the value of Q that was produced by transaction T_j .
 3. For each data item Q , the transaction (if any) that performs the final $\text{write}(Q)$ operation in schedule S must perform the final $\text{write}(Q)$ operation in schedule S'

Concurrency Control and Recovery

- Executing a single transaction at a time will **increase the waiting time** of the other transactions which may result in delay in the overall execution.
- Hence for increasing the overall throughput and efficiency of the system, several transactions are executed.
- Concurrently control is a very important concept of DBMS which ensures the simultaneous execution or manipulation of data by several processes or user without resulting in data inconsistency.
- Concurrency control provides a procedure that can control concurrent execution of the operations in the database.

Concurrency Control Problems

- **Dirty Read Problem(Write-Read conflict)**
- Dirty read problem occurs when one transaction updates an item
- but due to some unconditional events that transaction fails
- and before the transaction performs rollback,
- some other transaction reads the updated value.

Concurrency Control Problems

- **Dirty Read Problem(Write-Read conflict)**
- The lost update problem can be illustrated with the below scenario between two transactions T1 and T2.
 - Transaction T1 modifies a database record without committing the changes.
 - T2 reads the uncommitted data changed by T1
 - T1 performs rollback
 - T2 has already read the uncommitted data of T1 which is no longer valid, thus creating inconsistency in the database.

Concurrency Control Problems

- **Lost Update Problem**
- Lost update problem occurs
 - when two or more transactions modify the same data,
 - resulting in the update being overwritten or lost by another transaction.

Concurrency Control Problems

- **Lost Update Problem**
- The lost update problem can be illustrated with the below scenario between two transactions T1 and T2.
- T1 reads the value of an item from the database.
- T2 starts and reads the same database item.
- T1 updates the value of that data and performs a commit.
- T2 updates the same data item based on its initial read and performs commit.
- This results in the modification of T1 gets lost by the T2's write which causes a lost update problem in the database.

Concurrency Control Protocols

- Concurrency control protocols are the set of rules which are maintained to solve the concurrency control problems in the database.
- It ensures that the concurrent transactions can execute properly while maintaining the database integrity and consistency.
- The concurrent execution of a transaction is provided with atomicity, consistency, isolation, durability, and serializability via the concurrency control protocols.
- **Lock based concurrency control protocol**
- **Timestamp based concurrency control protocol**

Lock-based Protocols

- In lock based protocol, each transaction needs to acquire locks before they start accessing or modifying the data items.
- There are two types of locks used in databases.
- **Shared Lock**
- **Exclusive Lock**

Lock-based Protocols

- **Shared Lock**
- Shared lock is also known as read lock.
- It allows multiple transactions to read the data simultaneously.
- The transaction which is holding a shared lock can only read the data item.
- But it can not modify the data item.

Lock-based Protocols

- **Exclusive Lock**
- Exclusive lock is also known as the write lock.
- Exclusive lock allows a transaction to update a data item.
- Only one transaction can hold the exclusive lock on a data item at a time.
- While a transaction is holding an exclusive lock on a data item, no other transaction is allowed to acquire a shared/exclusive lock on the same data item.

Lock-based Protocols

- There are two kind of lock-based protocol mostly used in database:
- **Two Phase Locking Protocol**
- **Strict Two-Phase Locking Protocol**

Lock-based Protocols

- **Two Phase Locking Protocol**
- Two phase locking is a widely used technique which ensures strict ordering of lock acquisition and release.
- It ensures serializability.
- Transaction issues lock and unlock phases in two phases.

Lock-based Protocols

- Two phase locking protocol works in two phases. Growing and Shrinking Phase.
- **Growing Phase:**
- In this phase, the transaction starts acquiring locks before performing any modification on the data items.
- Once a transaction acquires a lock, that lock can not be released until the transaction reaches the end of the execution.

Lock-based Protocols

- Two phase locking protocol works in two phases. Growing and Shrinking Phase.
- **Shrinking Phase:**
- In this phase, the transaction releases all the acquired locks once it performs all the modifications on the data item.
- Once the transaction starts releasing the locks, it can not acquire any locks further.

Lock-based Protocols

- **Strict Two Phase Locking Protocol**
- It is almost like the two phase locking protocol.
- The only difference is that in two phase locking the transaction can release its locks before it commits,
- but in case of strict two phase locking the transactions are only allowed to release the locks only when they performs commits.

Timestamp-based Protocols

- In this protocol each transaction has a timestamp attached to it.
- Timestamp is the time in which a transaction enters the system.
- The conflicting pairs of operations can be resolved by the timestamp ordering protocol through the utilization of the timestamp values of the transactions
- Therefore, guaranteeing that the transactions take place in the correct order.

Advantages of Concurrency

- **Waiting Time:** It means if a process is in a ready state but still the process does not get the system to get execute is called waiting time. So, concurrency leads to less waiting time.
- **Response Time:** The time wasted in getting the response from the CPU for the first time, is called response time. So, concurrency leads to less Response Time.
- **Resource Utilization:** The amount of Resource utilization in a particular system is called Resource Utilization. Multiple transactions can run parallel in a system. So, concurrency leads to more Resource Utilization.
- **Efficiency:** The amount of output produced in comparison to given input is called efficiency. So, Concurrency leads to more Efficiency.

Disadvantages of Concurrency

- **Overhead:** Implementing concurrency control requires additional overhead, such as acquiring and releasing locks on database objects. This overhead can lead to slower performance and increased resource consumption, particularly in systems with high levels of concurrency.
- **Deadlocks:** Deadlocks can occur when two or more transactions are waiting for each other to release resources, causing a circular dependency that can prevent any of the transactions from completing. Deadlocks can be difficult to detect and resolve and can result in reduced throughput and increased latency.
- **Reduced concurrency:** Concurrency control can limit the number of users or applications that can access the database simultaneously. This can lead to reduced concurrency and slower performance in systems with high levels of concurrency.

Disadvantages of Concurrency

- **Complexity:** Implementing concurrency control can be complex, particularly in distributed systems or in systems with complex transactional logic. This complexity can lead to increased development and maintenance costs.
- **Inconsistency:** In some cases, concurrency control can lead to inconsistencies in the database. For example, a transaction that is rolled back may leave the database in an inconsistent state, or a long-running transaction may cause other transactions to wait for extended periods, leading to data staleness and reduced accuracy.

Summary

- Concurrency control ensures transaction atomicity, isolation, consistency, and serializability.
- Concurrency control issues occur when many transactions execute randomly.
- A dirty read happens when a transaction reads data changed by an uncommitted transaction.
- When two transactions update data simultaneously, the Lost Update issue occurs.
- Lock-based protocol prevents incorrect read/write activities.
- Timestamp-based protocols organize transactions by timestamp.

END OF LECTURE 16

- SHIVA.KUNWAR@HOTMAIL.COM
- +977-9819123654

Google classroom code : drbzdcf

PREVIEW FOR LECTURE 17

RECOVERY SYSTEM



Data Analysis Idea

Lecture 17

Database Management System

Er. Shiva Kunwar
Lecturer, GU
02/20/2024

Lesson 7: Recovery System (3hrs)

Basic Concept of Crash Recovery

Classification of Failure

Recovery and Atomicity

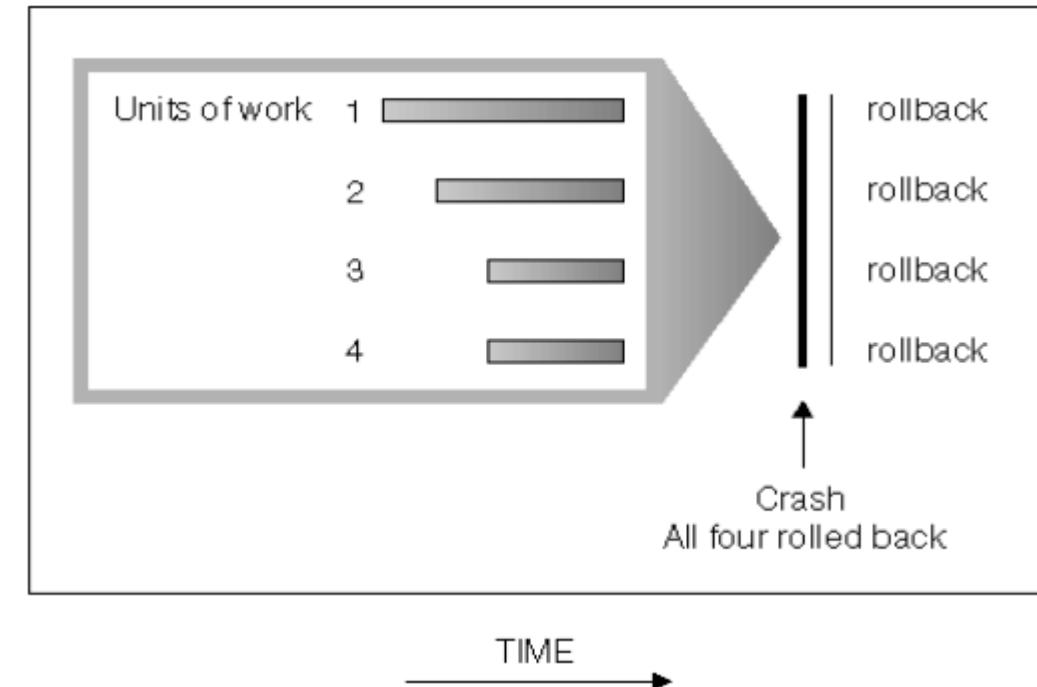
Log-based Recovery and Shadow Paging

Basic Concept of Crash Recovery

- During the transaction processing, the transactions on the database can be interrupted.
- If a failure occurs before all of the changes are made, the database is left in an inconsistent and unusable state.
- Crash recovery is the process by which the database is moved back to a consistent and usable state.
- After reaching the consistent state it has attained what is known as point of consistency.
- Crash Recovery can be done by **rolling back incomplete transactions** and **completing committed transactions** that were still in memory when the crash occurred.

Conditions of transaction failure that needs crash recovery:

- A power failure on the machine, causing the database manager and the database partitions on it to go down.
- A hardware failure such as memory, disk, CPU, or network failure.
- A serious operating system error that causes to end the database to abnormally.



Classification of Failure

- To find that where the problem has occurred, we generalize a failure into the following categories:
 - Transaction failure
 - System crash
 - Disk failure



→ Transaction Failure

- The transaction failure occurs when it fails to execute or when it reaches a point from where it can't go any further.
- If a few transaction or processes is hurt, then this is called transaction failure.
- Reasons for a transaction failure could be -
 - Logical errors: If a transaction cannot be completed due to some code error or an internal error condition, then the logical error occurs. Such error can be challenging to detect and resolve.
 - Syntax error: It occurs when the DBMS itself terminates an active transaction because the database system is not able to execute it. For example, The system aborts an active transaction, in case of deadlock or resource unavailability.

System Crash

- System failure can occur due to power failure or other hardware or software failure.
Example: Operating system error.
- Fail-stop assumption: In the system crash, non-volatile storage is assumed not to be corrupted.

Disk Failure

- It occurs where hard-disk drives or storage drives used to fail frequently. It was a common problem in the early days of technology evolution.
- Disk failure occurs due to the formation of bad sectors, disk head crash, and unreachability to the disk or any other failure, which destroy all or part of disk storage.

Recovery and Atomicity

- When a system crashes, it may have several transactions being executed and various files opened for them to modify the data items.
- Transactions are made of various operations, which are atomic in nature.
- But according to ACID properties of DBMS, atomicity of transactions as a whole must be maintained, that is, either all the operations are executed or none.

Recovery and Atomicity

- When a DBMS recovers from a crash, it should maintain the following:
 - It should check the states of all the transactions, which were being executed.
 - A transaction may be in the middle of some operation; the DBMS must ensure the atomicity of the transaction in this case.
 - It should check whether the transaction can be completed now or it needs to be rolled back.
 - No transactions would be allowed to leave the DBMS in an inconsistent state.

Recovery and Atomicity

- There are two types of techniques, which can help a DBMS in recovering as well as maintaining the atomicity of a transaction –
- Maintaining the logs of each transaction and writing them onto some stable storage before actually modifying the database.
- Maintaining shadow paging, where the changes are done on a volatile memory, and later, the actual database is updated

Log-based Recovery

- Log is a sequence of records, which maintains the records of actions performed by a transaction.
- Log of each transaction is maintained in some stable storage so that if any failure occurs, then it can be recovered from there.
- If any operation is performed on the database, then it will be recorded in the log.
- But the process of storing the logs should be done before the actual transaction is applied in the database.

Log-based Recovery

- The log file is kept on a stable storage media.
- When a transaction enters the system and starts execution, it writes a log about it.
 - $\langle Tn, Start \rangle$
- When the transaction Tn modifies an item X , it write logs as follows
 - $\langle Tn, X, V1, V2 \rangle$
- Here, $V1$ =old data, $V2$ =new value
- It reads Tn has changed the value of X , from $V1$ to $V2$.
- When the transaction finishes, it logs :
 - $\langle Tn, commit \rangle$
- When the transaction Tn is aborted:
 - $\langle Tn, abort \rangle$

Log-based Recovery

- Let's assume there is a transaction to modify the City of a student. The following logs are written for this transaction.
- When the transaction is initiated, then it writes 'start' log.
 - $\langle T1, Start \rangle$
- When the transaction T1 modifies the City from 'Kathmandu' to 'Pokhara', then another log is written to the file.
 - $\langle T1, City, 'Kathmandu', 'Pokhara' \rangle$
- When the transaction is finished, then it writes another log to indicate the end of the transaction.
 - $\langle T1, Commit \rangle$
- When the transaction Tn is aborted:
 - $\langle T1, Abort \rangle$

Log-based Recovery

- Create a log for given transaction T1 and T2.

T1	T2	Log
Read A	Read A	$\langle T1, Start \rangle$
A=A-2000	A=A+5000	$\langle T1, A, 5000, 3000 \rangle$
Write A	Write A	$\langle T1, B, 8000, 10000 \rangle$
Read B	Read B	$\langle T1, Commit \rangle$
B=B+2000	B=B+7000	$\langle T2, Start \rangle$
Write B	Write B	$\langle T2, A, 3000, 8000 \rangle$ $\langle T2, B, 10000, 17000 \rangle$ $\langle T2, Commit \rangle$

Log-based Recovery

- There are two approaches to modifying the database:

1. Deferred database modification:

- The deferred modification technique occurs if the transaction does not modify the database until it has been committed.
- In this method, all the logs are created and stored in the stable storage, and the database is updated when a transaction is committed.

2. Immediate database modification:

- The Immediate modification technique occurs if database modification occurs while the transaction is still active.
- In this technique, the database is modified immediately after every operation. It follows an actual database modification.

Recovery using Log Records

- When a system with concurrent transactions crashes and recovers, it behaves in the following manner –
- The recovery system reads the logs backwards from the end to the last checkpoint.
- It maintains two lists, an undo-list and a redo-list.
- If the recovery system sees a log with $\langle Tn, Start \rangle$ and $\langle Tn, Commit \rangle$ or just $\langle Tn, Commit \rangle$, it puts the transaction in the redo-list.
- If the recovery system sees a log with $\langle Tn, Start \rangle$ but no commit or abort log found, it puts the transaction in undo-list.
- All the transactions in the undo-list are then undone and their logs are removed.
- All the transactions in the redo-list and their previous logs are removed and then redone before saving their logs.

Checkpoint

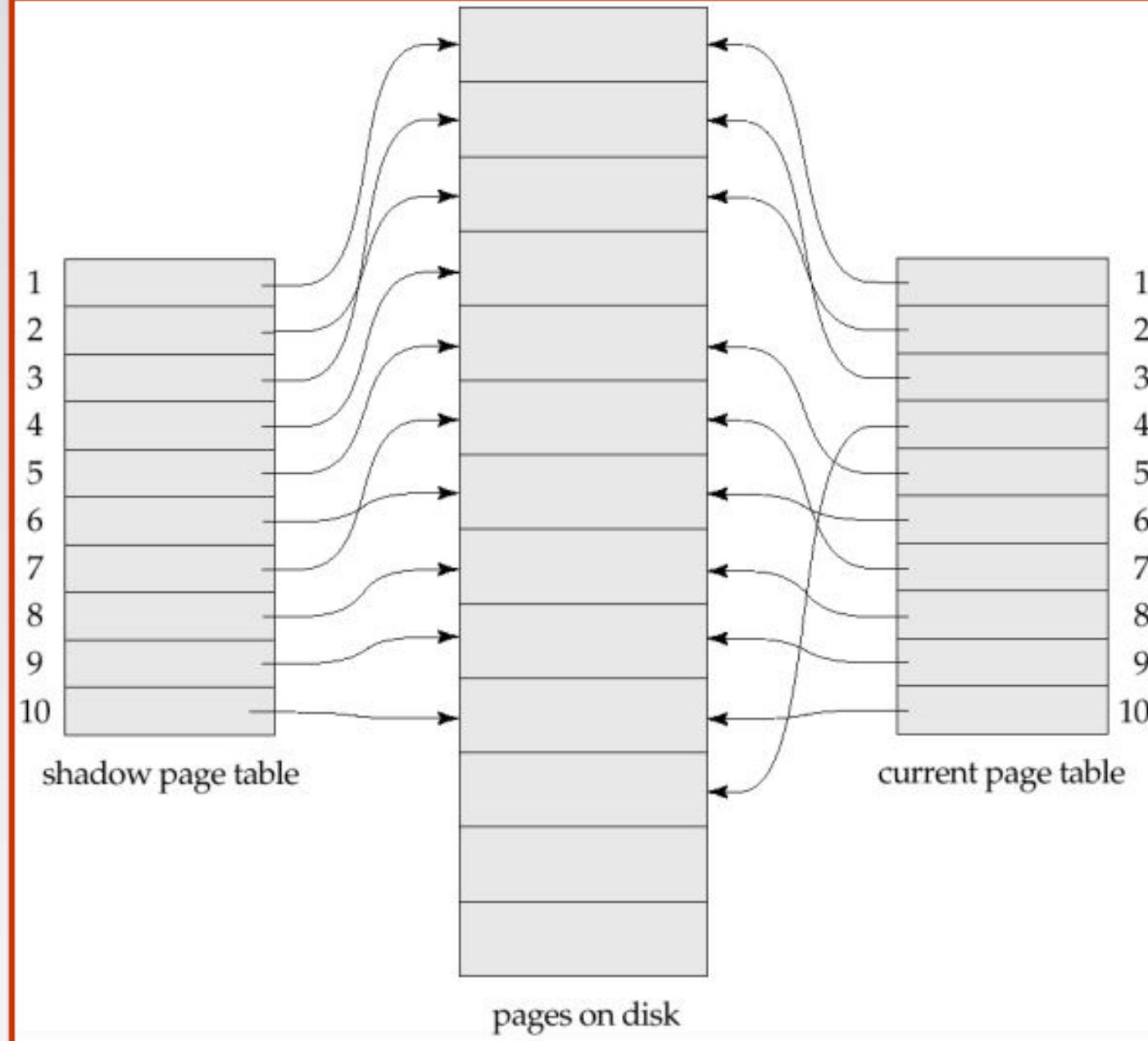
- The log file may grow too big to be handled at all as the time passes.
- Checkpoint is a mechanism where all the previous logs are removed from the system and stored permanently in a storage disk.
- Checkpoint declares a point before which the DBMS was in consistent state, and all the transactions were committed.

Shadow Paging

- Shadow paging is one of the techniques that is used to recover from failure and maintain database consistency in the time of failure.
- Concept of Shadow Paging:
- Step 1 – Page is a segment of memory. Page table is an index of pages. Each table entry points to a page on the disk.
- Step 2 – Two-page tables are used during the life of a transaction: the current page table and the shadow page table. Shadow page table is a copy of the current page table.
- Step 3 – When a transaction starts, both the tables look identical, the current table is updated for each write operation.

Shadow Paging

- Step 4 – The shadow page is never changed during the life of the transaction.
- Step 5 – When the current transaction is committed:
 - the shadow page entry becomes a copy of the current page table entry and
 - the disk block with the old data is released.
- Step 6 – The shadow page table is stored in non-volatile memory.
 - If the system crash occurs, then the shadow page table is copied to the current page table.
- The advantages of shadow paging are as follows –
 - No need for log records.
 - No undo/ Redo algorithm.
 - Recovery is faster



END OF LECTURE 17

- SHIVA.KUNWAR@HOTMAIL.COM
- +977-9819123654

Google classroom code : drbzdcf

PREVIEW FOR LECTURE 18

ADVANCED DATABASE MODEL



Data Analysis Idea

Lecture 18

Database Management System

Er. Shiva Kunwar
Lecturer, GU

Lesson 8: Advanced Database Model (4hrs)

- 1. Extension of Relational Model**
- 2. Object-Oriented Model**
- 3. Distributed Model**
- 4. NoSQL Systems and XML Database**

Extension of Relational Model

- Relational model enhance its power and make it more suitable for handling complex data structures.
- **Object-Relational Model**
- The Object-Relational Model (ORM) combines the features of the relational and object-oriented models.
- It allows for the storage of complex data types such as arrays, structures, and objects, as well as the ability to define user-defined types and functions.
- It provides an efficient way to handle complex data and enables the use of object-oriented programming concepts in database applications.

Extension of Relational Model

- **Extended Relational Model**
- The Extended Relational Model (ERM) extends the relational model by introducing additional constructs such as derived attributes, multi-valued dependencies, and recursive relationships.
- It allows for a more flexible representation of data and provides additional support for data modeling and design.
- **Semantic Model**
- The Semantic Model (SM) is a model that extends the relational model by adding semantic information to the database schema.
- It enables the representation of knowledge and relationships between entities and allows for more expressive queries that can take into account the meaning of data.

Extension of Relational Model

- **XML Model:**
- The XML Model is an extension of the relational model that supports the storage and manipulation of XML data.
- It provides an efficient way to store and retrieve complex hierarchical data structures such as documents and web pages.
- **NoSQL Model**
- The NoSQL Model is a non-relational model that is designed to handle large volumes of unstructured and semi-structured data.
- It provides a flexible and scalable way to store and retrieve data and is commonly used in big data and real-time applications.

Extension of Relational Model

- **Data Warehousing**
- A data warehouse is a large, centralized repository of data that is used for reporting and analysis.
- It is typically designed using a relational database management system (RDBMS) and provides a way to store and manage large volumes of data from multiple sources.
- The data warehouse uses a schema that is optimized for reporting and analysis and may include denormalized tables and aggregations to improve query performance.

Object-Oriented Model

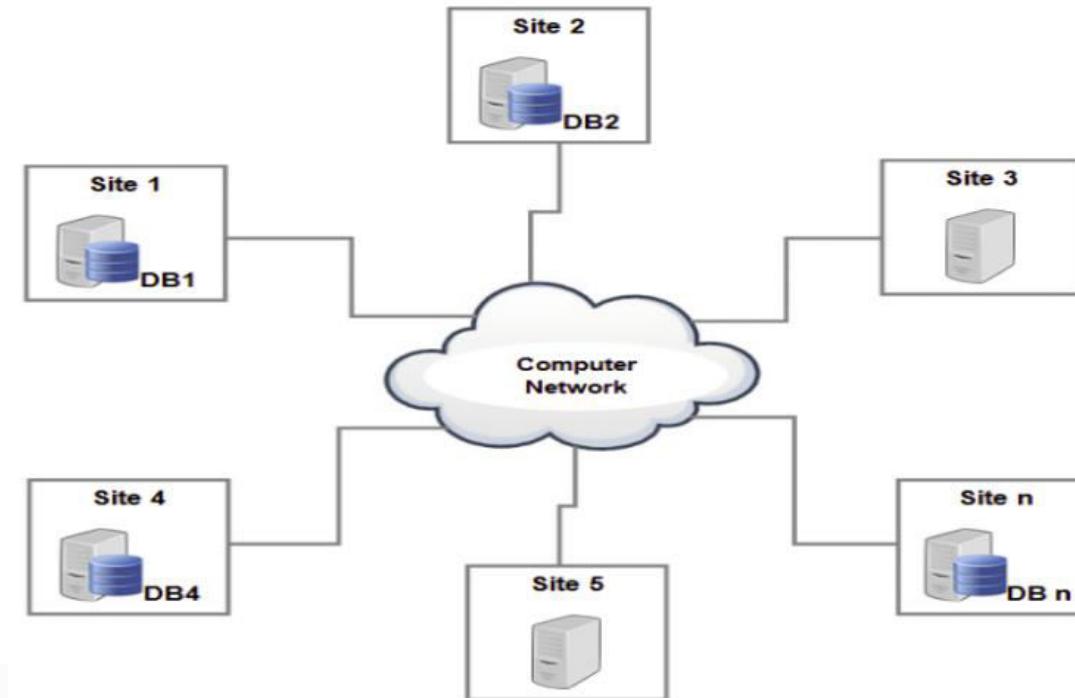
- It adds new features and capabilities that are not present in the traditional relational model.
- In the object-oriented model, data is represented as objects, which have properties and methods.
- Objects can be organized into classes, which define the properties and behavior of the objects.
- Classes can also inherit properties and methods from other classes, allowing for the creation of more complex and sophisticated data structures.

Object-Oriented Model

- Let's consider a simple database for a library.
- In a traditional relational database, you might have a table for books, with columns for the book title, author, publisher, and publication date.
- However, in an object-oriented database, you might represent a book as an object with properties such as title, author, and publication date, and methods such as borrow() and return().
- You might also have a class for books, which defines the properties and behavior of all book objects, and subclasses for different types of books.
- So, the object-oriented model provides a more flexible and expressive way to represent data than the traditional relational model.
- It also supports encapsulation, inheritance, and polymorphism.

Distributed Model

- Distributed database is a collection of multiple, logically related databases which are distributed across the nodes in the system.
- In a distributed model, data is distributed across multiple physical locations or nodes, which are connected by a network.



Distributed Model

- Example: Online Store → customers in different geographical regions.
- Consider an online store with customers in different geographical regions.
- You have a database that stores information about your customers, including their names, addresses, and order history.
- In a traditional, centralized database system, all the customer data would be stored in a single database located on a server in one location.
- This could lead to performance issues if there are a large number of customers or if customers are spread out over a large area.
- Additionally, if the server were to fail or go offline, the entire system would be unavailable.

Distributed Model

- To address these issues, a distributed database system is implemented which include setting up multiple database servers, each located in a different geographical region.
- Each server contains a complete copy of the customer database.
- Now, when a customer logs into your online store, their request is directed to the server closest to their location.
- The server processes the request using the local copy of the database, eliminating the need for data to be transferred across long distances.
- If one server were to fail, customers in that region would still be able to access the system using one of the other servers.

Distributed Model

- So, the example provides improved performance, availability, and fault tolerance.
- However, it does require additional resources to maintain multiple copies of the database and ensure that they remain in sync.
- Advantages
 - High availability
 - Scalability
 - Improved performance
- Disadvantages
 - Increased operational complexity
 - Increased cost

NoSQL Systems

- NoSQL is a type of database management system that is designed to handle large volumes of unstructured or semi-structured data, such as text, images, video, and social media data.
- Unlike traditional relational databases, NoSQL databases do not rely on a fixed schema or tabular structure.
- Example:

document = {"name": "Ram", "age": 20, "country": "Nepal"}

NoSQL Systems

- **Document database:** These databases store data in documents, typically in a JSON. Each document can contain a different structure, allowing for flexibility in the data model. Examples include MongoDB and Couchbase.
- **Key-value stores:** These databases store data as key-value pairs, with each key corresponding to a value. They are designed for fast retrieval of data and are often used for caching or session management. Example Amazon DynamoDB.
- **Column family stores:** These databases store data in columns, rather than rows, allowing for efficient querying and indexing of data. Examples include Apache Cassandra and HBase.
- **Graph databases:** These databases store data as nodes and edges, allowing for complex relationships and graph analysis. They are often used for social networking, recommendation engines, and fraud detection. Example OrientDB.

XML Database

- It abbreviates to Extensible Markup language.
- store huge amounts of information in the XML format.
- This data can be queried, transformed, exported, and returned to a calling system.
- The structure of XML data model is a tree model.
- XML model is used to store valuable internal data

Concept of Data Warehouse (Optional Topic)

- A data warehouse is a repository (or archive) of information gathered from multiple sources, stored under a unified schema, at a single site.
- Once gathered, the data are stored for a long time, permitting access to historical data.
- Thus, data warehouses provide the user with a single consolidated interface to data, making decision-support queries easier to write.
- Moreover, by accessing information for decision support from a data warehouse, the decision maker ensures that online transaction processing systems are not affected by the decision-support workload.

Concept of Data Warehouse

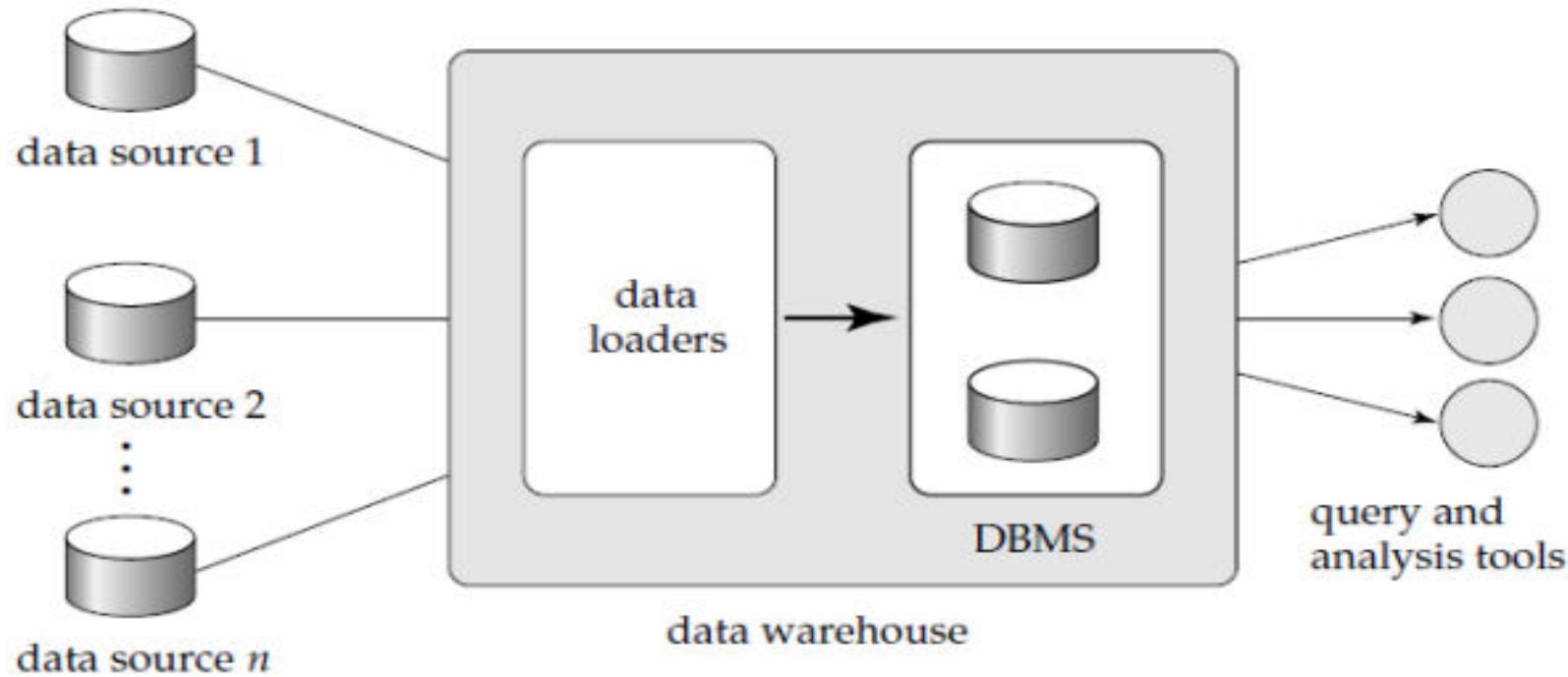


Figure 22.8 Data-warehouse architecture.

- Figure shows the architecture of a typical data warehouse, and illustrates the gathering of data, the storage of data, and the querying and data-analysis support.

Components of Data Warehouse

- Among the issues to be addressed in building a warehouse are the following:
- **When and how to gather data**
- In a source-driven architecture for gathering data, the data sources transmit new information, either continually (as transaction processing takes place), or periodically (nightly, for example).
- In a destination-driven architecture, the data warehouse periodically sends requests for new data to the sources.

Components of Data Warehouse

- **What schema to use?**
- Data sources that have been constructed independently are likely to have different schemas.
- They may even use different data models.
- Part of the task of a warehouse is to perform schema integration and convert data to the integrated schema before they are stored.
- As a result, the data stored in the warehouse are not just a copy of the data at the sources.
- Instead, they can be thought of as a materialized view of the data at the sources.

Components of Data Warehouse

- **Data cleansing**
- The task of correcting and preprocessing data is called data cleansing.
- Data sources often deliver data with numerous minor inconsistencies, that can be corrected. For example, names are often misspelled, and addresses may have street/area/city names misspelled, or zip codes entered incorrectly.
- These can be corrected to a reasonable extent by consulting a database of street names and zip codes in each city.
- Address lists collected from multiple sources may have duplicates that need to be eliminated in a merge–purge operation.
- Records for multiple individuals in a house may be grouped so only one mailing is sent to each house; this operation is called householding.

Components of Data Warehouse

- **How to propagate updates**
- Updates on relations at the data sources must be propagated to the data warehouse.
- If the relations at the data warehouse are the same as those at the data source, the propagation is straightforward.
- If they are not, the problem of propagating updates is the view-maintenance problem

Components of Data Warehouse

- **What data to summarize**
- The raw data generated by a transaction-processing system may be too large to store online.
- However, we can answer many queries by maintaining just summary data obtained by aggregation on a relation, rather than maintaining the entire relation.
- For example, instead of storing data about every sale of clothing, we can store total sales of clothing by item name and category.

Data Warehouse - Subject-Oriented

- Organized around major subjects, such as customer, product, sales
- Focusing on the modeling and analysis of data for decision-makers, not on daily operations or transaction processing
- Provide a simple and concise view around subject issues by excluding data that are not useful in the decision support process.

Data Warehouse - Integrated

- Constructed by integrating multiple, heterogeneous data sources
 - relational databases, flat files, and on-line transaction records.
- Data cleaning and data integration techniques are applied.
 - Ensure consistency in naming conventions, encoding structures, attribute measures, etc. among different data sources.
 - E.g., Hotel price: currency, tax, breakfast covered, etc.
 - When data is moved to the warehouse, it is converted.

Data Warehouse - Integrated

- The time horizon for the data warehouse is significantly longer than that of operational systems
 - Operational database: current value data
 - Data warehouse data: provide information from a historical perspective (e.g., past 5-10 years)
- Every key structure in the data warehouse
 - Contains an element of time, explicitly or implicitly
 - But the key to operational data may or may not contain “time element”

Data Warehouse - Nonvolatile

- A physically separate store of data transformed from the operational environment
- Operational update of data does not occur in the data warehouse environment
- Does not require transaction processing, recovery, and concurrency control mechanisms
- Requires only two operations in data accessing:
 - initial loading of data and access to data

END OF LECTURE 18

- SHIVA.KUNWAR@HOTMAIL.COM
- +977-9819123654

Google classroom code : drbzdcf

PREVIEW FOR LECTURE 19

DISCUSSION