NATIONAL UNIVERSITY
COLLEGE OF COMPUTING AND
INFORMATION TECHNOLOGIES

VER 1.0

Course MANUAL

INFORMATION MANAGEMENT

This course provides students with the theoretical knowledge and practical skills in the utilization of databases and database management systems in the ICT applications. The logical design, physical design and implementation of relational databases are all covered in this course.

INFORMATION TECHNOLOGY

JULIE ANNE ANGELES-CRYSTAL
ROGER C. PRIMO JR.
MARK JAMES G. CAYABYAB
JOHN IVAN C. MAURAT

INFORMATION MANAGEMENT

Table of Contents

MODULE

#05

NATIONAL UNIVERSITY

FLEX Course Material

Advance Data Modeling

WEEK 9 MODULE

MODULE GOALS

The Extended Entity Relationship model

Entity Clustering

Entity Integrity: Selecting Primary Keys

Design Cases: Learning Flexible Database Design

Education that works.

LESSON #

#5.1

The Extended Entity Relationship Model Entity Clustering

The Extended Entity-Relationship (EER) Model is an extension of the original Entity-Relationship (ER) model that includes additional concepts and features to represent more complex relationships and constraints in a database system.
One of the features of the EER model is "Entity Clustering."

Advance Data Modeling

KEY ASPECTS:

Advanced Data Modeling refers to a sophisticated and comprehensive approach to representing and structuring data in a database system.

It involves the use of advanced techniques and methodologies to design data models that accurately reflect the relationships and constraints within a given domain.

This goes beyond basic data modeling concepts and incorporates more complex elements to address specific requirements and challenges.

Normalization:

- While normalization is a fundamental concept, advanced data modeling involves a deeper understanding and application of normalization techniques. This includes reaching higher normal forms (4NF, 5NF) to minimize data redundancy and improve data integrity.

Dimensional Modeling:

- Commonly used in data warehousing, dimensional modeling focuses on designing data structures for efficient querying and reporting. It involves the creation of star schemas and snowflake schemas, which are optimized for analytical processing.

Agile Data Modeling:

- Agile methodologies emphasize flexibility and responsiveness to changing requirements. Agile Data Modeling involves iterative and incremental approaches to modeling, allowing for quick adaptations to evolving business needs.

KEY ASPECTS:

Temporal Data Modeling:

- Temporal data modeling deals with representing and managing data changes over time. It includes concepts such as valid time and transaction time, enabling the tracking of historical data and temporal querying.

Graph Data Modeling:

- Graph databases and modeling are used to represent complex relationships in interconnected data. Graph data modeling is especially valuable for scenarios where relationships are as important as entities, such as social networks, fraud detection, and recommendation systems.

Object-Relational Mapping (ORM):

- ORM is a technique used to bridge the gap between object-oriented programming languages and relational databases. Advanced ORM tools and techniques facilitate the seamless integration of object-oriented models with relational databases.

NoSQL Data Modeling:

- In the era of NoSQL databases, data modeling extends beyond the relational paradigm. NoSQL data modeling involves designing schemas for document-oriented databases, key-value stores, column-family stores, and graph databases.

Advance Data Modeling

KEY ASPECTS:

Business Process Modeling:

- Advanced data modeling incorporates business process modeling to understand how data flows through an organization. It involves the creation of process models, activity diagrams, and workflow diagrams to capture the end-to-end business processes.

Data Vault Modeling:

- Data Vault Modeling is a method for modeling the data warehouse in a way that is scalable, flexible, and resilient to changes. It includes concepts like Hubs (business keys), Links (associations), and Satellites (historical data).

Metadata Management:

- Advanced data modeling involves robust metadata management practices. This includes documenting and managing metadata (data about data) to ensure that there is a clear understanding of the meaning and context of the data.

Machine Learning Data Modeling:

- As machine learning becomes more prevalent, data modeling extends to include features and attributes relevant to machine learning models. This involves identifying and engineering features that contribute to predictive modeling.

Advance Data Modeling

KEY ASPECTS:

Security and Privacy Modeling:

- Advanced data modeling considers security and privacy concerns. This involves modeling access controls, encryption, and ensuring compliance with data protection regulations.

Big Data Modeling:

- Big Data introduces challenges in terms of volume, velocity, and variety. Advanced data modeling for Big Data involves designing models that can handle massive volumes of data, streaming data, and diverse data types.

Data Modeling Tools:

- Advanced data modeling often leverages sophisticated modeling tools that support collaborative modeling, version control, and integration with other development tools.

Advanced Data Modeling is a multidimensional and evolving field that adapts to the changing landscape of data management.

It involves selecting the right modeling techniques and tools based on the specific requirements and characteristics of the data and the business processes it supports.

LESSON #

#5.2

Entity Clustering

Entity clustering in the EER model involves grouping related entities together into a higher-level construct known as a "cluster" or "superclass." This grouping is based on shared characteristics or relationships among the entities.

Essentially, entity clustering allows for the abstraction of common features or relationships among entities, making the model more organized and representing a higher level of abstraction.

ENTITY CLUSTERING:

Key Points:

1. **Abstraction:** Entity clustering provides a way to abstract commonalities among entities, simplifying the overall model.
2. **Relationships:** Entities within a cluster share relationships with each other, and the cluster itself can participate in relationships with other entities or clusters.
3. **Attributes:** Common attributes among entities within a cluster can be grouped at the cluster level, reducing redundancy and enhancing maintainability.
4. **Hierarchical Structure:** Entity clusters can be organized in a hierarchical structure, allowing for a more organized and modular representation of the data model.
5. **Complex Relationships:** Entity clustering is particularly useful when dealing with complex relationships that involve multiple entities sharing common characteristics.

ENTITY CLUSTERING

OBJECTIVES OF ENTITY CLUSTERING

- **Duplicate Detection:**
  - Identify and group together duplicate records or representations of the same entity in a dataset.
- **Data Integration:**
  - Integrate data from multiple sources by linking records that refer to the same real-world entities.
- **Data Quality Improvement:**
  - Enhance the quality of data by resolving discrepancies and inconsistencies in entity representations.
- **Entity Matching:**
  - Match entities across different datasets, databases, or systems, even when identifiers or attributes vary.
- **Analytics and Reporting:**
  - Improve the accuracy of analytics and reporting by ensuring that data is consolidated and represents real-world entities.

ENTITY CLUSTERING

TECHNIQUES OF ENTITY CLUSTERING

- **Deterministic Matching:**
  - Exact matching based on predefined rules or algorithms. It requires a high degree of confidence in the matching criteria.
- **Probabilistic Matching:**
  - Matching based on probabilities and statistical methods. It accommodates variations and uncertainties in the data.
- **Tokenization and String Matching:**
  - Breaking down strings (names, addresses) into tokens and comparing them to identify similarities. Techniques include Jaccard similarity and Levenshtein distance.
- **Blocking:**
  - Grouping records into blocks or clusters based on certain attributes, narrowing down the comparison space for efficiency.
- **Machine Learning-Based Matching:**
  - Leveraging machine learning algorithms, such as clustering algorithms or classification models, to learn patterns and make matching decisions.
- **Graph-Based Matching:**
  - Representing entities and their relationships as a graph and using graph algorithms to identify clusters.
- **Fuzzy Matching:**
  - Matching that considers similarity rather than exact matches. It allows for variations and partial matches.

ENTITY CLUSTERING

CHALLENGES OF ENTITY CLUSTERING

- **Data Quality:**
  - Poor data quality, including missing or incorrect values, can hinder the effectiveness of clustering algorithms.
- **Scalability:**
  - Clustering large datasets efficiently can be challenging, especially when dealing with high-dimensional or diverse data.
- **Algorithm Selection:**
  - Choosing the right clustering algorithm depends on the characteristics of the data and the specific matching requirements.
- **Handling Variability:**
  - Dealing with variations in entity representations, such as typos, abbreviations, and different formats.
- **Computational Complexity:**
  - Some clustering algorithms have high computational complexity, making them resource-intensive.

APPLICATIONS OF ENTITY CLUSTERING

- **Customer Data Integration:**
    - Consolidating customer records from various sources to create a unified view.
- **Healthcare Data Integration:**
    - Linking patient records from different healthcare providers to create a comprehensive health history.
- **Fraud Detection:**
    - Identifying potentially fraudulent activities by clustering entities that exhibit suspicious behavior.
- **Academic and Research Data Integration:**
    - Integrating research publications, datasets, and author information to create a unified research repository.
- **Master Data Management:**
    - Managing master data by linking and consolidating records related to the same entities.
- **Social Network Analysis:**
    - Clustering entities in social networks to identify communities or groups of related individuals.

Entity clustering is a crucial step in data management, particularly when dealing with diverse and distributed datasets.

It contributes to improving data quality, enhancing decision-making, and enabling more accurate and meaningful analyses.

The choice of clustering techniques depends on the nature of the data and the specific requirements of the application.

LESSON #

#5.3

Entity Integrity: Selecting Primary Keys

Entity integrity is a fundamental concept in relational database management systems (RDBMS) that ensures the accuracy and consistency of data within a table.

One key aspect of entity integrity is the selection and enforcement of primary keys.

ENTITY INTEGRITY

CONSIDERATIONS:

Entity Integrity:

- Entity integrity ensures that each row (record) in a table is uniquely identified and that no key attributes (columns) contain null values.
- It is typically enforced through the use of primary keys.

Primary Keys:

- A primary key is a column or a set of columns in a table that uniquely identifies each record in that table.
- It provides a way to reference and link records across tables. The selection of an appropriate primary key is crucial for maintaining entity integrity.

ENTITY INTEGRITY

CONSIDERATIONS FOR PRIMARY KEYS

- **Uniqueness:**
  - A primary key must ensure the uniqueness of each record in the table. No two records should have the same primary key value.
- **Irreducibility:**
  - A primary key should be irreducible, meaning that it should consist of the minimum number of columns necessary to uniquely identify a record. Redundant columns should be avoided.
- **Immutability:**
  - Ideally, a primary key should be immutable, meaning that its values should not change over time. This ensures stability in references to the record.
- **Simplicity:**
  - Keep primary keys simple and easy to understand. Single-column primary keys are straightforward, but composite keys (multiple columns) may be necessary for certain scenarios.

CONSIDERATIONS FOR PRIMARY KEYS

- **Stability:**
  - Primary keys should be stable and not subject to frequent changes. Changing primary key values can lead to challenges in maintaining referential integrity.
- **Applicability:**
  - The primary key should be relevant to the business domain and reflect the natural way of identifying records. For example, in a "Students" table, a student ID might be a suitable primary key.
- **Availability:**
  - Ensure that the chosen primary key values are available and not reserved for other purposes. For example, using social security numbers might not be practical due to privacy concerns.
- **Consistency Across Tables:**
  - If a table needs to be related to another table, it's essential to ensure consistency in the primary key definition. For example, if one table uses an auto-incremented integer as a primary key, related tables should reference this integer.
- **Consider Auto-Incrementing Values:**
  - For surrogate keys (keys not derived from the natural attributes of the entity), consider using auto-incrementing integers. These provide a simple and efficient way to ensure uniqueness.

EXAMPLE:

- StudentID (Primary Key)
- FirstName
- LastName
- DateOfBirth

In this example, "StudentID" is chosen as the primary key.

It is unique for each student, irreducible, immutable, and stable over time.

Enforcing Entity Integrity:

- Entity integrity is typically enforced through the database management system's capabilities, such as the declaration of primary key constraints. The primary key constraint ensures that the primary key values are unique for each record and that they are not null.

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

- In this SQL example, the "StudentID" column is defined as the primary key for the "Students" table.
- By adhering to the considerations outlined above and enforcing primary key constraints, you contribute to the maintenance of entity integrity in your relational database.

LESSON #

#5.4

Design Cases: Learning Flexible Database Design

Designing a flexible and effective database is crucial for accommodating evolving business requirements, scalability, and ease of maintenance.

Let's explore two design cases that emphasize the principles of flexible database design.

CASE 1: E-COMMERCE PLATFORM

Requirements:

- **Products:**
  - Different types of products, each with unique attributes.
  - Ability to add new product types without modifying the database structure.
- **Orders:**
  - Customers can place orders for multiple products.
  - Support for various payment methods.
- **Users:**
  - Different types of users, such as customers, administrators, and vendors.

Flexible Database Design:

- **Entity-Attribute-Value (EAV) for Products:**
  - Use an EAV model for products to accommodate different attributes for each product type. This allows the addition of new product types without altering the database schema.

```
CREATE TABLE Products (
   ProductID INT PRIMARY KEY,
   ProductType VARCHAR(50),
   -- Other common attributes
);

CREATE TABLE ProductAttributes (
   ProductID INT,
   AttributeName VARCHAR(50),
   AttributeValue VARCHAR(255),
   PRIMARY KEY (ProductID, AttributeName),
   FOREIGN KEY (ProductID) REFERENCES Products(ProductID)
);
```

DESIGN CASES:

CASE 1: E-COMMERCE PLATFORM (cont..)

- **Orders and Payment Methods:**
  - Design a flexible structure for orders to handle multiple products per order and different payment methods.

```
CREATE TABLE Orders (
   OrderID INT PRIMARY KEY,
   CustomerID INT,
   OrderDate DATETIME,
   -- Other common attributes
);

CREATE TABLE OrderItems (
   OrderID INT,
   ProductID INT,
   Quantity INT,
   PRIMARY KEY (OrderID, ProductID),
   FOREIGN KEY (OrderID) REFERENCES Orders(OrderID),
   FOREIGN KEY (ProductID) REFERENCES Products(ProductID)
);

CREATE TABLE Payments (
   PaymentID INT PRIMARY KEY,
   OrderID INT,
   Amount DECIMAL(10, 2),
   PaymentMethod VARCHAR(50),
   -- Other payment attributes
   FOREIGN KEY (OrderID) REFERENCES Orders(OrderID)
);
```

- **User Roles:**
  - Implement a flexible structure for user roles to accommodate different types of users.

```
CREATE TABLE Users (
   UserID INT PRIMARY KEY,
   Username VARCHAR(50),
   Password VARCHAR(255),
   -- Other common attributes
);

CREATE TABLE UserRoles (
   UserID INT,
   Role VARCHAR(50),
   PRIMARY KEY (UserID, Role),
   FOREIGN KEY (UserID) REFERENCES Users(UserID)
);
```

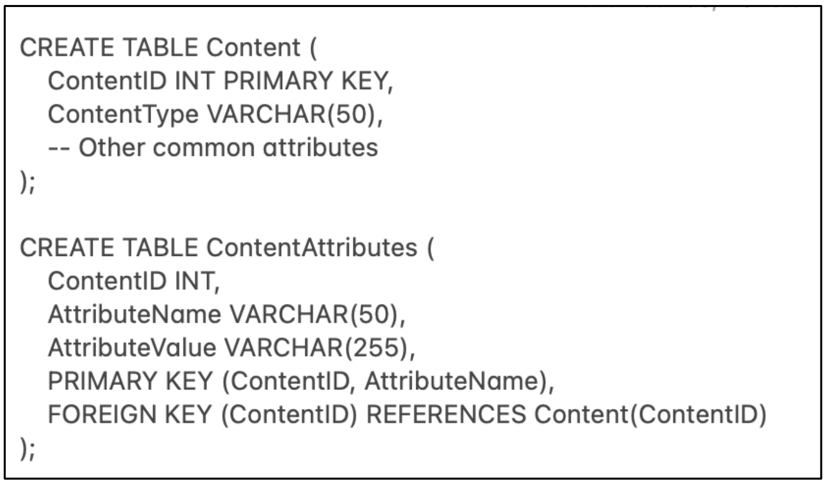
CASE 2: CONTENT MANAGEMENT SYSTEM

Requirements:

- **Content Types:**
  - Different types of content (articles, videos, images) with unique attributes.
  - Ability to introduce new content types without modifying the database schema.
- **Users and Permissions:**
  - Different user roles with varying permissions.
  - Support for user authentication and authorization.

Flexible Database Design:

- **EAV for Content Types:**
  - Similar to Case 1, use EAV for content types to allow for the dynamic addition of attributes.



```
CREATE TABLE Content (
   ContentID INT PRIMARY KEY,
   ContentType VARCHAR(50),
   -- Other common attributes
);

CREATE TABLE ContentAttributes (
   ContentID INT,
   AttributeName VARCHAR(50),
   AttributeValue VARCHAR(255),
   PRIMARY KEY (ContentID, AttributeName),
   FOREIGN KEY (ContentID) REFERENCES Content(ContentID)
);
```

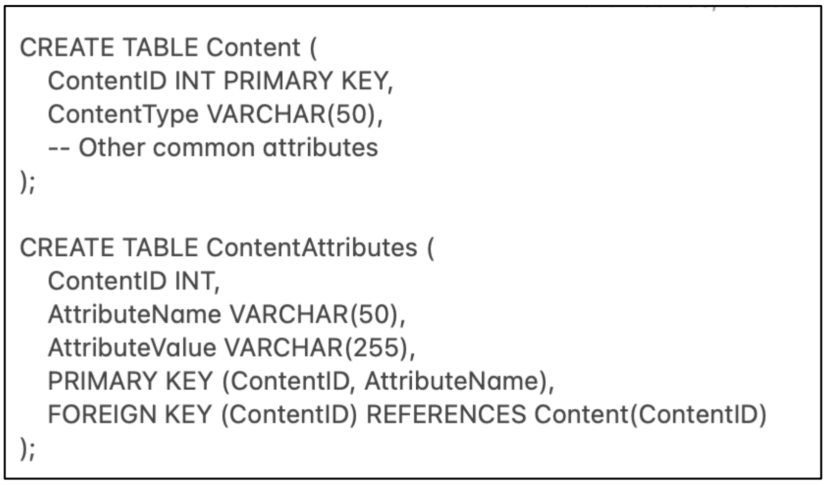
CASE 2: CONTENT MANAGEMENT SYSTEM

Requirements:

- **Content Types:**
  - Different types of content (articles, videos, images) with unique attributes.
  - Ability to introduce new content types without modifying the database schema.
- **Users and Permissions:**
  - Different user roles with varying permissions.
  - Support for user authentication and authorization.

Flexible Database Design:

- **EAV for Content Types:**
  - Similar to Case 1, use EAV for content types to allow for the dynamic addition of attributes.



```
CREATE TABLE Content (
   ContentID INT PRIMARY KEY,
   ContentType VARCHAR(50),
   -- Other common attributes
);

CREATE TABLE ContentAttributes (
   ContentID INT,
   AttributeName VARCHAR(50),
   AttributeValue VARCHAR(255),
   PRIMARY KEY (ContentID, AttributeName),
   FOREIGN KEY (ContentID) REFERENCES Content(ContentID)
);
```

CASE 2: CONTENT MANAGEMENT SYSTEM (CONT..)

User Authentication and Authorization:

Design a flexible structure for users, roles, and permissions.

```
CREATE TABLE Users (
    UserID INT PRIMARY KEY,
    Username VARCHAR(50),
    Password VARCHAR(255),
    -- Other common attributes
);

CREATE TABLE Roles (
    RoleID INT PRIMARY KEY,
    RoleName VARCHAR(50)
);

CREATE TABLE UserRoles (
    UserID INT,
    RoleID INT,
    PRIMARY KEY (UserID, RoleID),
    FOREIGN KEY (UserID) REFERENCES Users(UserID),
    FOREIGN KEY (RoleID) REFERENCES Roles(RoleID)
);

CREATE TABLE Permissions (
    PermissionID INT PRIMARY KEY,
    PermissionName VARCHAR(50)
);

CREATE TABLE RolePermissions (
    RoleID INT,
    PermissionID INT,
    PRIMARY KEY (RoleID, PermissionID),
    FOREIGN KEY (RoleID) REFERENCES Roles(RoleID),
    FOREIGN KEY (PermissionID) REFERENCES Permissions(PermissionID)
);
```

In both cases, the use of Entity-Attribute-Value (EAV) modeling for dynamic attributes and the design of flexible structures for related entities (products/orders and content/users) allow for a database that can adapt to changing business needs without significant schema modifications.

This flexibility supports scalability and facilitates easier maintenance in the long run. However, it's crucial to carefully consider the trade-offs and potential performance implications when opting for such flexible designs.