DBMS

Week 07

Extended E-R Features

Specialization

- Specialization is a process in the E-R model that involves dividing an entity set into distinct subgroupings based on certain characteristics or attributes that are not shared by all entities in the original set. It allows us to create subcategories or specialized entity sets within a broader entity set.
- For example, within the "person" entity set, we can specialize it into "employee" and "student" subsets, each with its own unique attributes like "salary" for employees and "total credits" for students.

Generalization

- Generalization is the opposite of specialization and involves combining multiple lower-level entity sets with similar attributes and relationships into a higher-level entity set. It emphasizes the common features shared among these lower-level entities.
- For example, "instructor" and "secretary" entities can be generalized into the higher-level entity "employee" because they share attributes like "ID," "name," "street," and "city."

Attribute Inheritance

- Attribute inheritance is a key concept in specialization and generalization. It means that lower-level entity sets inherit attributes from higher-level entity sets. In other words, attributes defined at the higher level are also applicable to lower-level entities.
- For instance, when "student" is specialized from "person," it inherits attributes like "ID," "name," "street," and "city" from the "person" entity.

Aggregation

- Aggregation is a modeling technique used in the E-R model to represent relationships among relationships. It allows the creation of higher-level entity sets that treat relationships as entities themselves.
- For example, if there's a relationship called "proj guide" involving "instructor," "student," and "project," and each relationship has additional attributes, you can use aggregation to create a higher-level entity set called "proj guide" to represent these relationships.

Entity-Relationship Design Issues

Incorrect Use of Attribute

- One common mistake is using the primary key of an entity set as an attribute of another entity set. For example, having the department name as an attribute of the "student" entity is incorrect. Instead, the relationship "stud_dept" should be used to represent the connection between students and departments.

Erroneous Use of Relationship Attributes

- Another mistake is designating the primary key attributes of related entity sets as attributes of the relationship set. For example, using "ID" (primary key of student) and "ID" (primary key of instructor) as attributes of the "advisor" relationship is incorrect. These attributes are already implicit in the relationship set and should not be repeated.

Using Single-Valued Attribute for Multivalued Information

- It's important to use the appropriate type of attribute for the information you're modeling. Using a single-valued attribute when a multivalued attribute is needed can lead to data modeling issues. For example, representing student marks for different assignments with attributes in the "takes" relationship may limit the system's flexibility.

Binary vs. n-ary Relationship Sets

- Relationships in databases are often binary, but sometimes non-binary (n-ary) relationships are needed. It's possible to replace n-ary relationships with several binary relationships, but this may not always be the best choice. The decision depends on the specific requirements and constraints of the system being modeled.

Use of Entity Sets vs. Relationship Sets

- Deciding whether to use an entity set or a relationship set can sometimes be challenging. One guideline is to use a relationship set when modeling an action that occurs between entities. The choice depends on the semantics of the relationship and the structure of the real-world enterprise.

Consideration for Entity Sets with Attributes

- When deciding whether something should be an attribute or an entity set, it depends on the specific structure and semantics of the data. Some attributes are best expressed as entities when additional information or relationships are associated with them.

Breaking Up Large E-R Diagrams

- Large E-R diagrams should be divided into smaller pieces for clarity. When depicting an entity set in multiple places, attributes should only be shown once in the first occurrence to avoid redundancy and inconsistency.

Alternative Notations for Modeling Data

Importance of Diagrammatic Representation

- Diagrams are essential in database schema design as they provide an intuitive way to represent the data model of an application. They serve as a common language for communication between data modeling experts and domain experts who may not be familiar with data modeling.

Various Data Modeling Notations

- There are multiple notations available for modeling data, with Entity-Relationship (E-R) diagrams and UML class diagrams being the most widely used. These notations facilitate the visualization and communication of data structures.

No Universal E-R Notation Standard

- Unlike UML, there is no universal standard for E-R diagram notation. Different sources and tools may use variations of notations, which can create confusion.

Comparison with UML Class Diagrams

- The passage provides a comparison between E-R diagram constructs and their equivalent representations in UML class diagrams. It highlights the similarities and differences between the two notations.
  - In UML class diagrams, classes represent entities, and attributes and methods are associated with these classes.
  - Relationships between entities are represented by associations, with cardinality constraints and roles specifying the nature of the relationships.
  - Generalization and specialization hierarchies, as well as aggregation and composition, are supported in UML class diagrams.
  - UML uses a slightly different notation for cardinality constraints compared to E-R diagrams.

Use of Composition in UML

- UML uses a "composition" notation, denoted by a line with a small shaded diamond at one end, to represent a strong containment relationship between entities. This corresponds roughly to the concept of a weak entity set in E-R diagrams, indicating existence dependency.

Support for Object-Oriented Features

- UML class diagrams also support object-oriented features like interfaces, allowing for a more comprehensive representation of software systems.

Choice of Notation

- The choice between E-R diagrams and UML class diagrams depends on the specific needs of the data modeling task and the familiarity of the designers with these notations.

Other Aspects of Database Design

Functional Requirements

- Database design is not just about the structure of the database schema. It also involves planning the functionality that the database should support. This includes defining the transactions that update data and queries to retrieve data. Additionally, designers must plan the interfaces that users will interact with to perform these functions.

Authorization Mechanisms

- Not all users have the same level of access to data or permissions to perform all actions within the database. Implementing an effective authorization mechanism is crucial. This mechanism can control who can view specific data, perform transactions, or access certain functionalities. It can operate at both the database level and higher-level functionalities or interfaces.

Data Flow and Workflow

- Database applications are often part of larger enterprise applications that involve complex data flows and workflows. Data may move through various stages, users, and processes. Workflow systems help manage these processes, and they interact with the database system. Understanding these workflows and their interactions with the database is essential for comprehensive database design.

Schema Evolution

- Database design is an ongoing process because the needs of an organization evolve over time. The data requirements of an organization are subject to change, and the database schema must adapt to these changes. Schema evolution can occur at various levels: conceptual, logical, or physical. Good database design should anticipate future needs and minimize the impact of schema changes as the organization evolves.

Distinguishing Permanent and Changeable Constraints

- It's important to distinguish between fundamental constraints that are unlikely to change and constraints that may change due to evolving organizational policies or requirements. A flexible database design can accommodate anticipated changes without requiring major modifications.

Human-Oriented Activity

- Database design is a human-centered activity involving interaction with various stakeholders. End-users are people who interact with the system, and the database designer must collaborate with domain experts to understand the specific data requirements and needs of the application domain. Effective communication and collaboration with all stakeholders are essential for successful database design and deployment.

Class activity

- Design Process
  - Conceptual-design
  - Logical-design
  - Physical-design
- Entity-relationship (E-R) data model
- Entity and entity set
  - Simple and composite attributes
  - Single-valued and multivalued attributes
  - Derived attribute
- Key
  - Superkey
  - Candidate key
  - Primary key
- Relationship and relationship set
  - Binary relationship set
  - Degree of relationship set
  - Descriptive attributes
  - Superkey, candidate key, and primary key
  - Role
  - Recursive relationship set
- E-R diagram
- Mapping cardinality:
  - One-to-one relationship
  - One-to-many relationship
  - Many-to-one relationship
  - Many-to-many relationship
- Total and partial participation
- Weak entity sets and strong entity sets
  - Discriminator attributes
  - Identifying relationship
- Specialization and generalization
- Aggregation
- Design choices
- United Modeling Language (UML)

Tools

- Dia:
  - Dia is a free and open-source diagram editor that is available on multiple platforms, including Linux and Windows.
  - It supports both E-R diagrams and UML class diagrams.
  - For representing entities with attributes, you can use UML classes or tables from the Database library provided by Dia.
  - Dia provides flexibility in diagram creation and is a great choice for users who prefer open-source solutions.
- LucidChart:
  - LucidChart is an online diagram editor that offers support for creating E-R diagrams.
  - It allows entities to be represented similarly to standard E-R notation.
  - Relationships can be created using diamonds from the Flowchart shape collection.
  - LucidChart is a web-based tool, making it accessible from various platforms and devices.
- Draw.io:
  - Draw.io is another online diagram editor that supports E-R diagrams.
  - Users can create relationships using its features.
  - Draw.io is user-friendly and offers collaborative capabilities for diagram creation.
- Commercial Tools:
  - Commercial options like IBM Rational Rose Modeler, Microsoft Visio, ERwin Data Modeler, Poseidon for UML, and SmartDraw provide comprehensive solutions for database design and modeling.
  - These tools offer a wide range of features, including support for E-R diagrams, UML class diagrams, and more.
  - They often include advanced features for database design, forward and reverse engineering, and database management.

Chapter 7 relational database design

Relational database design is the process of creating a structured and efficient database schema based on the principles of the relational model. This involves organizing data into tables, defining relationships between tables, and ensuring data integrity through normalization and proper constraints.

Features of Good Relational Designs

Entity-Relationship (E-R) Design

- The passage begins by highlighting the importance of starting with a well-designed Entity-Relationship model (E-R model) as the basis for creating a relational database design.

Repetition of Information

- It discusses the problem of redundancy, where the same information is repeated in multiple tuples. Redundancy can lead to data inconsistencies if not managed properly.

Example of Redundancy

- An example is given using a "university database" schema (dep) where department information is repeated for each instructor in the department. This redundancy is illustrated with the "in dep" relation.

Issues with Redundancy

- Redundancy can lead to data inconsistencies if a user updates one occurrence of redundant data but not others. Additionally, it can create problems when trying to represent new departments without instructors.

Decomposition

- To address redundancy, the passage suggests decomposing the schema into smaller schemas. This decomposition should be "lossless," meaning it does not result in a loss of information.

Lossless Decomposition

- The passage defines a "lossless decomposition" as one in which there is no loss of information when replacing the original schema with the decomposed schemas. It is determined using functional dependencies.

Normalization Theory

- The passage introduces the concept of normalization as a methodology for designing a relational database. The goal of normalization is to eliminate redundancy and ensure data integrity while allowing for efficient data retrieval.

Normal Forms

- The passage mentions the concept of normal forms, which are specific criteria for evaluating whether a relation schema is in "good form." Different normal forms, such as First Normal Form (1NF) and Second Normal Form (2NF), are used to guide the decomposition process.

Functional Dependencies

- To determine whether a schema is in good form, the passage suggests using functional dependencies. Functional dependencies are relationships between attributes that define how data should be structured to ensure integrity.

Decomposition Using Functional Dependencies

Constraints in Real-World Data

- The passage emphasizes that real-world data often has various constraints or rules that must be followed. In the context of a university database, it provides examples of constraints related to unique identification of students and instructors, single names, and department associations.

Legal Instances

- A legal instance of a relation is one that satisfies all the real-world constraints. In other words, it adheres to the rules defined for the data.

Notational Conventions

- The passage introduces notational conventions for discussing relational database design, including the use of Greek letters for sets of attributes, uppercase Roman letters for relation schemas, and lowercase letters for relations. It also explains how to refer to schemas and instances of relations.

Keys and Functional Dependencies

- The passage discusses the concepts of keys (Superkeys, candidate keys, primary keys) and functional dependencies. It defines a superkey as a set of attributes that can uniquely identify a tuple and functional dependencies as relationships between attributes that impose constraints on the data.

Functional Dependency Example

- An example is provided where the functional dependency A → C is satisfied in a given instance of a relation, but C → A is not satisfied, demonstrating how functional dependencies work.

Trivial Functional Dependencies

- Some functional dependencies are considered trivial because they are satisfied by all relations. Trivial dependencies are those where the right-hand side is a subset of the left-hand side.

Inference of Functional Dependencies

- The passage mentions that given a set of functional dependencies, it is possible to infer additional functional dependencies. The closure of a set of functional dependencies, denoted as F+, contains all the inferred dependencies.

Lossless Decomposition and Functional Dependencies

- Functional dependencies are used to determine when a decomposition of a schema into smaller schemas is lossless. The passage provides a rule for binary decomposition that ensures lossless decomposition if R1 ∩ R2 forms a superkey for either R1 or R2.

Constraints on Decomposed Schemas

- If a schema is decomposed into two schemas, certain constraints must be imposed to maintain consistency. These constraints include making the intersection of the decomposed schemas a primary key for one schema and a foreign key for the other.

Normal Forms

BCNF (Boyce-Codd Normal Form)

- BCNF is a desirable normal form for relational database schemas.
- BCNF eliminates all redundancy based on functional dependencies.
- A relation schema R is in BCNF with respect to a set F of functional dependencies if, for all functional dependencies $\alpha \rightarrow \beta$ in F+, where α and β are subsets of R:
- Either $\alpha \rightarrow \beta$ is a trivial functional dependency ($\beta \subseteq \alpha$).
- Or α is a superkey for schema R.
- A database design is in BCNF if each relation schema in the design is in BCNF.
- BCNF is a more restrictive normal form than 3NF.

3NF (Third Normal Form)

- 3NF relaxes the constraints of BCNF slightly.
- A relation schema R is in 3NF with respect to a set F of functional dependencies if, for all functional dependencies $\alpha \rightarrow \beta$ in F+, where α and β are subsets of R:
  - Either $\alpha \rightarrow \beta$ is a trivial functional dependency.
  - Or α is a superkey for schema R.
  - Or each attribute A in the set $(\beta - \alpha)$ is contained in a candidate key for schema R.
- 3NF allows certain nontrivial dependencies that BCNF does not allow.

Comparison of BCNF and 3NF

- BCNF and 3NF are two common normal forms used in relational database design.
- BCNF is more restrictive than 3NF; BCNF requires that all nontrivial dependencies be of the form $\alpha \rightarrow \beta$, where α is a superkey.
- 3NF relaxes this constraint by allowing certain nontrivial dependencies whose left side is not a superkey.
- When choosing between BCNF and 3NF, trade-offs must be considered, including the potential use of null values and the problem of information repetition.
- SQL does not provide a way to specify general functional dependencies easily, making it challenging to enforce functional dependencies other than primary key constraints efficiently.
- In some cases, the use of materialized views in database systems can help reduce the cost of enforcing functional dependencies.

Higher Normal Forms

- In some cases, functional dependencies alone may not be sufficient to avoid data redundancy and ensure data integrity.
- Higher normal forms beyond BCNF and 3NF have been defined to address such situations.

Functional-Dependency Theory

Closure of a Set of Functional Dependencies

- Given a set of functional dependencies (FDs) on a schema, you can prove that certain other FDs are logically implied by this set. The closure of a set of FDs, denoted as F+, is the set of all FDs logically implied by F. Closure is used when testing for normal forms like BCNF or 3NF.

Armstrong's Axioms

- These are a set of rules used to find logically implied FDs. The three main axioms are:
  - Reflexivity Rule: If α is a set of attributes, and β is a subset of α, then α → β holds.
  - Augmentation Rule: If α → β holds and γ is a set of attributes, then γα → γβ holds.
  - Transitivity Rule: If α → β holds and β → γ holds, then α → γ holds.

Additional Rules

- Besides Armstrong's axioms, there are additional rules for finding implied FDs, such as the Union Rule, Decomposition Rule, and Pseudotransitivity Rule.

Attribute Closure Algorithm

- This algorithm computes the closure of a set of attributes under a given set of FDs. It is used to determine if a set of attributes is a superkey and has other applications.

Canonical Cover

- A canonical cover is a simplified set of FDs equivalent to the original set. It is constructed by removing extraneous attributes and ensuring uniqueness of left sides of FDs.

Dependency Preservation

- When decomposing a relation into smaller ones, it's important to check if the original FDs are preserved. A decomposition is dependency preserving if the FDs that held in the original relation still hold in the decomposed relations.

Algorithms for Decomposition Using Functional Dependencies

BCNF (Boyce-Codd Normal Form)

- BCNF is a higher level of database normalization used in relational database design. A relation (table) is said to be in BCNF if, for every non-trivial functional dependency (a dependency in which the determination of one attribute uniquely determines another), the left-hand side (LHS) of the dependency is a superkey for the table. In other words, BCNF ensures that there are no partial dependencies where non-prime attributes depend on a proper subset of a candidate key.

3NF (Third Normal Form)

- 3NF is a level of database normalization that comes after 1NF (First Normal Form) and 2NF (Second Normal Form). A relation is in 3NF if, for every non-trivial functional dependency (a dependency in which the determination of one attribute uniquely determines another), the LHS of the dependency is a superkey or the attribute on the right-hand side (RHS) is a prime attribute (an attribute that is part of a candidate key). In other words, 3NF eliminates transitive dependencies, where non-prime attributes depend on other non-prime attributes through a candidate key.

Decomposition Using Multivalued Dependencies

BCNF (Boyce-Codd Normal Form)

- BCNF is a normal form that deals primarily with functional dependencies, which are relationships between attributes in a relation (table). A relation is considered to be in BCNF if, for every non-trivial functional dependency (a dependency where the determination of one attribute uniquely determines another), the left-hand side (LHS) of the dependency is a superkey for the relation. BCNF helps eliminate partial dependencies, ensuring that each attribute is fully functionally dependent on the candidate key(s).

4NF (Fourth Normal Form)

- 4NF is another normal form that goes beyond BCNF by considering both functional dependencies and multivalued dependencies. A relation is in 4NF if, for all non-trivial multivalued dependencies in the relation, either:
  - The multivalued dependency is trivial (meaning it involves the entire relation or the attributes on both sides are the same).
  - The attributes on the left-hand side (LHS) of the multivalued dependency form a superkey for the relation.
- In essence, 4NF is an extension of BCNF that ensures that there is no redundancy or unnecessary repetition of data due to multivalued dependencies.

More Normal Forms

Project-Join Normal Form (PJNF), also known as Fifth Normal Form (5NF)

- PJNF is an extension of 4NF that deals with join dependencies. A join dependency specifies how a relation can be reconstructed by joining multiple other relations. In PJNF, a relation is in 5NF if it is in 4NF and satisfies all join dependencies implied by the functional and multivalued dependencies.
- Achieving PJNF requires dealing with complex join conditions and can be challenging in practice. It aims to minimize anomalies that arise when reconstructing relations from decomposed ones.

Domain-Key Normal Form (DKNF)

- DKNF is a higher level of normalization that addresses the constraints and dependencies in the database schema. A relation is in DKNF if, for every constraint that can be specified in the database language, the constraint is either a logical consequence of the definition of keys and domains or it can be enforced by the DBMS.
- DKNF aims to eliminate all types of redundancy and ensure that the database schema enforces data integrity constraints at the highest level.

Atomic Domains and First Normal Form

Atomic Attributes

- An attribute is considered atomic if its values are indivisible and do not have any substructure. In other words, atomic attributes have simple, single values.
- Common examples of atomic attributes include integers, real numbers, dates, and single pieces of data that do not need to be broken down further.

Non-Atomic Attributes

- Non-atomic attributes, on the other hand, have substructure or are composed of multiple components.
- Examples of non-atomic attributes include composite attributes (attributes composed of multiple simpler attributes) and set-valued attributes (attributes that can hold multiple values, such as sets or lists).

First Normal Form (1NF)

- A relation schema is said to be in first normal form (1NF) if all the attributes in the schema have atomic domains. In other words, there are no composite attributes or set-valued attributes in a 1NF schema.
- Achieving 1NF is the first step in the process of database normalization. It ensures that each attribute contains only atomic values, simplifying data storage and manipulation.

Use of Non-Atomic Values

- While 1NF promotes atomicity for attributes, there are cases where non-atomic values are useful and appropriate. For example, composite-valued attributes (e.g., an address with street, city, and state components) can represent complex real-world entities more naturally.
- Set-valued attributes (e.g., storing a set of values) are also valuable in certain scenarios, such as recording multiple phone numbers for a contact.

Database Integrity and Maintenance

- When non-atomic values are used in a database schema, it's essential to ensure data integrity and consistency through proper design and application logic. Updates to non-atomic values can be more complex and may require additional care to avoid inconsistencies.
- Redundant storage of data, as mentioned in your text, can occur when non-atomic attributes are not handled properly, leading to maintenance challenges.

Modern Database Systems

- Many modern database management systems (DBMS) support non-atomic values and offer ways to handle composite attributes and set-valued attributes efficiently.
- These systems provide mechanisms for querying and manipulating data with non-atomic attributes while maintaining data integrity.

Database-Design Process

Ways to Create Relation Schemas

- Relation schemas (denoted as r(R)) can be generated in various ways:
  - From an Entity-Relationship (E-R) diagram, ensuring a well-designed E-R diagram can result in schemas that require minimal further normalization.
  - As a single relation schema containing all relevant attributes, which can then be broken down through normalization.
  - As a result of ad hoc design, followed by formal verification for desired normal forms.

E-R Model and Normalization

- Properly designed E-R diagrams often lead to schemas in desired normal forms.
- Functional dependencies among attributes in generated schemas can be used to detect E-R design issues. Correcting these issues at the E-R modeling stage is preferable.

Naming of Attributes and Relationships

- Maintaining a unique-role assumption is essential to ensure that each attribute name has a unique meaning in the database.
- While distinct names should be used for attributes with different meanings, attributes with the same meaning in different schemas can use the same name.
- Order of attribute names typically does not matter, but listing primary-key attributes first is a common convention.

Denormalization for Performance

- Denormalization involves introducing redundancy into the schema to improve performance for specific applications.
- Redundant data requires extra effort to keep it consistent during updates.
- Materialized views can be used to achieve some of the benefits of denormalization while maintaining data integrity, with the database system handling view maintenance.

Other Design Issues

- Some design issues may not be addressed by normalization and can lead to suboptimal database design.
- The example of storing time-series data in separate relations for each year demonstrates a poor design practice.
- Using crosstab-like representations for storing data, with one column for each attribute value, can also lead to maintenance challenges and complex queries.
- While such representations are useful for display purposes, they are not recommended for database storage.