NAME: Nicholas Jacob
STUDENT ID: # 113578513
GRADED HOMEWORK NUMBER: 4
COURSE: CS/DSA 4513 DATABASE MANAGEMENT
SECTION: ONLINE
SEMESTER: FALL 2023
INSTRUCTOR: DR. LE GRUENWALD
SCORE:

1. (a) Find the name of each employee who works for "Big Bank".

$$\Pi_{person\_name}(\sigma_{company\_name=\text{``}BigBank\text{''}}(works))$$

(b) Find the name and city of each employee who works for "Big Bank".

$$\Pi_{person\_name,city}(\sigma_{company\_name=\text{``}BigBank\text{''}}(employee \bowtie_{person\_name} works))$$

(c) Find the name, street address, and city of each employee who works for "Big Bank" and earns more than \$10 000.

$$\Pi_{person\_name,street,city}(\sigma_{(salary>10000)\wedge(company\_name=\text{``}BigBank\text{''})}(employee \bowtie_{person\_name} works)))$$

(d) Find the name of each employee who lives in the same city as the company for which they work.

$$\Pi_{person\_name}(\sigma_{employee.city=company.city}((employee \bowtie_{person\_name} works) \bowtie_{company\_name} company))$$

2. Given the relational schema Employee(id, name, classid,gender, manager,salary) and the set of Functional Dependencies

FD =

{(classid,id,gender)→ (salary, manager),

name → (age,id),

id→ name,

manager → (gender, age, classid,id)}

(a) To find all candidate keys, we will first examine the closure FD^+. Using decomposition, we see that

manager → gender,

manager → age,

manager → classid,

manager → id,

Then we grab the penultimate and the last here and use transitivity so,

manager → name.

To show manager is a candidate key, I still need salary. We see from decomposition that

(classid,id,gender)$\rightarrow$ salary, and

manager $\rightarrow$ (gender, classid,id) so transitivity gives us

manager $\rightarrow$ salary. Thus manager is a superkey. You cannot simplify manager so it is also a candidate key.

The other superkey is (classid, id, gender). We see right away that its closure contains salary and manager (decomposition rule). With this addition, we can also add age (decomposition again). We also get id trivially (reflexivity) so name will be included. Thus we have shown that all elements belong to the superkey (classid, id, gender). We note that this set is minimal. Closure on any of the individual or combination of elements will not give you manager. Thus (classid, id, gender) is a candidate key.

We do not see any other keys. id and name had the best shot but both do not imply manager so you are stuck.

(b) Normal Forms

i. This relational schema is in the first normal form since all entries are atomic.

ii. To check for second normal form, we would need for any non-prime attribute to be fully dependent on the candidate key. We see that name is partially dependent on the candidate key (classid, id, gender) since id $\rightarrow$ name.

iii. We are not third normal form either. name $\rightarrow$ age is in the closure (by decomposition). But age is not in any of the candidate keys so not third normal form as age is not a prime attribute.

iv. Since we were not third normal form, we will not be BCNF. We can see this by definition too. name $\rightarrow$ age is again a issue. name is not a superkey.

(c) I think the decomposition should be based on the canonical functional dependencies with all subsets removed.

R_1(CIGS)

R_2(CIGM)

R_3(NA)

R_4(NI)

R_5(MA)

then superkeys are (cig), (cig)m, n, (n)(i), m. Dependencies preserved are cig$\rightarrow$ s,cig$\leftrightarrow$ m, n$\rightarrow$a, n$\leftrightarrow$i, m$\rightarrow$ a. The only issue we have to check to make sure still in 3NF is the R_4. The issue there is it has both functional dependencies n$\leftrightarrow$i. We note that since i is a superkey, we are okay for 3NF.

(d) To show that this is a lossless join is a bit tricky. In part that neither the book nor videos discussed what to do in a non-binary case. I see in the literature (and on the stackexchange) this is called 'the chase'. I'll try to apply the chase here. Joining R_1 and R_2 is natural. The intersection of those two sets is CIG which was a superkey for both of those sets. I join R_{12} with R_5. Again this is lossless as the intersection is M and that was a superkey for R_5. Next I take R_{125} and intersect wtih R_4. Intersection there is I and I is a superkey for R_4. Lastly join this mega set with R_3. Intersection is NA. Since it contains N and N is a super key for R_3 we have chased all sets back together and have lost none of the information.

(e) To show that this reduction to 3NF is dependency preserving (it must be by construction...), we must show that the union of the functional dependencies closure is the same as the the closure of the initial functional dependencies. There is actually nothing to show here since all of the functional dependencies appeared after the decomposition. Some of them were simplified or combined but all appeared in the union before you attempt a closure. Thus it is dependency preserving decomposition.

3. My full conversation with ChatGPT can be found here

Me

How are functional dependencies preserved in database applications

ChatGPT

Functional dependencies are important in database design to ensure data integrity and minimize redundancy. Functional dependencies describe the relationships between attributes in a relational database. When you perform various operations in a database application, it's crucial to preserve these dependencies to maintain the consistency and accuracy of the data. Here are some ways in which functional dependencies are preserved in database applications:

Normalization: The primary method for preserving functional dependencies is through the normalization process. Normalization involves organizing the database schema to reduce redundancy and dependency issues. The normalization process, which includes decomposing tables into smaller, related tables, helps in preserving functional dependencies and ensures that data is stored in a way that minimizes update anomalies.

Foreign Key Constraints: Foreign keys are used to establish relationships between tables in a relational database. By defining foreign key constraints, you enforce referential integrity between tables. Foreign keys help preserve functional dependencies by ensuring that values in a column of one table match the values in the referenced column of another table. This maintains the relationships between tables and prevents inconsistent or orphaned data.

Transaction Management: Transactions play a crucial role in maintaining consistency and preserving functional dependencies. A transaction is a sequence of one or more SQL statements that are executed as a single unit of work. Transactions help ensure that changes to the database, such as inserts, updates, or deletes, are atomic and either fully completed or fully rolled back. This prevents partial updates that could violate functional dependencies.

Database Constraints: In addition to foreign key constraints, other database constraints such as unique constraints and check constraints help preserve functional dependencies. Unique constraints ensure that no two rows in a table have the same values in specified columns, preserving functional dependencies related to uniqueness. Check constraints allow you to define conditions that must be met for the data to be valid, helping to enforce functional dependencies.

Application Logic: The logic implemented in the database application itself can play a role in preserving functional dependencies. Application developers need to be aware of the underlying database schema and design their application logic to adhere to the defined functional dependencies.

Data Validation: Proper data validation at the application level ensures that only valid and consistent data is sent to the database. This helps in preserving functional dependencies by preventing the insertion of data that could violate those dependencies.

By combining these methods, database designers and application developers can work together to create a system that preserves functional dependencies and maintains data integrity throughout the application's lifecycle. User How can functional dependencies be used with entity relationship models ChatGPT

Functional dependencies play a crucial role in database design, including the development of entity-relationship models. In entity-relationship modeling, functional dependencies help define the relationships between entities and attributes. Here's how functional dependencies can be used in conjunction with entity-relationship models:

Identifying Functional Dependencies: Before creating an entity-relationship model, it's essential to identify the functional dependencies that exist in the real-world scenario being modeled. This involves understanding the relationships between attributes within entities.

Entity-Relationship Diagrams (ERD): In entity-relationship modeling, an ERD is a graphical representation of the entities, attributes, and relationships in a database. Functional dependencies influence the structure and connections within the ERD. Attributes within entities represent properties or characteristics, and understanding their functional dependencies helps in organizing them appropriately.

Normalization: Functional dependencies are closely related to normalization, a process used to organize data in a relational database to eliminate redundancy and dependency issues. During normalization, entities are decomposed into smaller tables to minimize the potential for update anomalies. This decomposition is guided by the functional dependencies identified in the data.

Primary and Foreign Keys: Functional dependencies guide the determination of primary keys for entities. The primary key is a unique identifier for each record in an entity and is often chosen based on the functional dependencies in the data. Foreign keys, which establish relationships between entities, are also influenced by functional dependencies. These keys help maintain referential integrity and ensure that relationships are consistent.

Relationship Cardinality: Functional dependencies contribute to determining the cardinality of relationships between entities in an ERD. Cardinality defines the number of instances of one entity that can be related to another entity. By understanding the functional dependencies, you can identify whether the relationship between entities is one-to-one, one-to-many, or many-to-many.

Attributes and Normal Forms: Functional dependencies guide the placement of attributes within entities and help ensure that each entity is in an appropriate normal form. Normal forms are stages of normalization, and each form has specific criteria regarding functional dependencies.

Attribute Constraints: Functional dependencies are used to define constraints on attributes. For example, if an attribute is functionally dependent on another, it may have constraints such as uniqueness or a specific range of values.

ERD Refinement: As you progress in the database design process, you may need to refine your ERD based on feedback and additional functional dependencies that become apparent. Refinement involves adjusting the structure of entities and relationships to better represent the real-world scenario.

In summary, functional dependencies are foundational to the design of entity-relationship models. They guide decisions related to the identification of entities, attributes, primary and foreign keys, and the relationships between entities. By incorporating functional dependencies into the modeling process, you create a database design that reflects the inherent structure and dependencies of the data you are working with.