Deep Dive into Advanced Relational Model Concepts

Estimated time: 10 minutes

Objective

By the end of this reading, you will:

1. Understand functional dependencies
2. Learn about multi-valued dependencies
3. Focus on candidate keys
4. Practice the concept on a practical scenario

In the world of managing databases, it's super important to understand some fancy concepts to make databases work well. Three big ones are functional dependencies, multi-valued dependencies, and candidate keys. These are like the building blocks for organizing and speeding up databases.

1. Functional Dependencies (FDs)

FDs are fundamental building blocks for ensuring data integrity and consistency in relational databases. They represent a specific type of relationship between attributes in a relation, where the value of one attribute (determinant) uniquely determines the value of another attribute (dependent). In simpler terms, if you know the value of the determinant, you can always identify the exact value of the dependent attribute.

- **For example**, if you know an employee's ID number, you can look it up in the table and find out their first and last name. In this case, the employee ID "determines" the first and last name. This is a functional dependency.
  Think of it as a special kind of connection between pieces of information. The determining attribute acts like a key that unlocks the related information.

Properties of FDs:

- **Notation**: FDs are typically written as X -> Y, where X is the determinant and Y is the dependent attribute.
- **Properties**:
  - Reflexivity: X -> X (every attribute determines itself).
  - Transitivity: If X -> Y and Y -> Z, then X -> Z (dependencies can chain together).
  - Closure: The minimal set of FDs that implies all other FDs in a relation.
- **Key points**
  - FDs are essential for maintaining data accuracy and minimizing redundancy.
  - They play a crucial role in database normalization, ensuring the efficient organization of data.
  - FDs help in eliminating unnecessary repetition and ensuring the correctness of data.

2. Multi-Valued Dependencies (MVDs)

MVDs are more complex than FDs. With an MVD one attribute, the determinant, determines a set of possible values for another attribute, the dependent. In other words, knowing the value of the determinant narrows down the potential values the dependent attribute can hold.

Here's how it works:

- If you know an employee's ID (let's say ID 123), you can't just look it up and find their one assigned project.
- With MVD, knowing ID 123 tells you there will be multiple rows in the table, each listing a different project that employee 123 works on.
- So, the employee ID "multi-determines" the project because it doesn't give you a single answer, but lets you know there are potentially several project assignments for that employee in separate rows. This is different from a regular dependency, where one piece of information leads to one or two specific others.

Properties of MVDs:

- **Notation**: MVDs are written as X -> $\{Y_1, Y_2, \ldots, Y_n\}$, where X is the determinant and $Y_1, Y_2, \ldots, Y_n$ are the possible values for the dependent attribute.
- **Properties**:
  - MVDs have similar properties to FDs, like reflexivity and transitivity. However, they lack the closure property.
  - Identifying MVDs helps understand the complex relationships between attributes and ensures data consistency within those relationships. Violations of MVDs can lead to invalid data entries.
- **Key points:**
  - MVDs are essential for organizing complex relationships between sets of attributes.
  - They help in avoiding mix-ups and ensuring proper organization of data.
  - Understanding MVDs is crucial for maintaining data integrity and optimizing database performance.

3. Candidate Keys

Candidate keys uniquely identify each row in a relation. They represent a minimal set of attributes that collectively guarantee no duplicates exist in the table. In other words, if you know the values of the candidate key, you can pinpoint a specific row and only that row. The keys are unique within that table.

Imagine going back to the basic table with just employee ID, first name, and last name.

- An employee ID is a strong candidate key on its own. Why? Because every employee has a unique ID, knowing the ID instantly tells you which record belongs to that particular employee.
- Here's the twist: This table might have another candidate key! Think about it: what if two employees have the same first and last name (less common, but possible)? In that case, the combination of first name and last name wouldn't be enough to distinguish between them.

- But the employee ID, by itself, will always be unique and pinpoint the exact employee. That's why a single attribute (employee ID) can be a candidate key in this scenario.

Key points:

- Uniqueness: Each combination of values in the candidate key must uniquely identify a distinct row.
- Minimality: No proper subset of the candidate key should be able to uniquely identify a row. This ensures that the candidate key is the smallest possible set of attributes needed for unique identification.
- Performance: Having well-defined candidate keys significantly improves query performance. Queries searching for specific rows can utilize indexing on the candidate key for faster data retrieval. It also helps maintain data integrity by preventing duplicate entries.
- Multiple keys: A relation can have multiple candidate keys, meaning different minimal sets of attributes can uniquely identify each row.

Here's the important thing: A table can have multiple candidate keys. As long as a set of attributes guarantees finding one specific employee record and no others, it qualifies as a candidate key. The key thing is that it uniquely identifies a row in the table.

Practical scenario

Let's consider a hypothetical scenario where we have a database for tracking employee projects. Each employee can work on multiple projects, and each project can involve multiple employees. We'll create a table to represent this scenario:

EmployeeID	ProjectID	EmployeeName	ProjectName	Department
1	101	Alice	Project X	HR
1	102	Alice	Project Y	Finance
2	101	Bob	Project X	HR
3	101	Charlie	Project X	IT
3	102	Charlie	Project Y	Finance

In this table:

- **EmployeeID**: Unique identifier for each employee.
- **ProjectID**: Unique identifier for each project.
- **EmployeeName**: Name of the employee.
- **ProjectName**: Name of the project.
- **Department**: Department to which the project belongs.

Functional Dependencies (FDs)

EmployeeID -> EmployeeName

- Knowing the EmployeeID uniquely determines the EmployeeName.
- For instance, if you look up EmployeeID 1 in the table, it uniquely identifies the EmployeeName as "Alice".

  EmployeeID -> EmployeeName
  1 -> Alice

ProjectID -> ProjectName

- Knowing the ProjectID uniquely determines the ProjectName.
- In the table, ProjectID 101 corresponds to "Project X" and ProjectID 102 corresponds to "Project Y", demonstrating a one-to-one relationship.

  ProjectID ↔ ProjectName
  101 ↔ Project X
  102 ↔ Project Y

Multi-Valued Dependencies (MVDs)

{EmployeeID} ->> {ProjectID}

- Knowing the EmployeeID tells us all the projects that employees are working on, independent of each other.

Let's revisit the example of EmployeeID 1 (Alice) from the table:

- EmployeeID 1 is associated with ProjectID 101 (Project X) in the HR department.
- EmployeeID 1 is also associated with ProjectID 102 (Project Y) in the Finance department.

  EmployeeID – {associated with} –> ProjectID (department)
  1 — — —> 101 (HR)
  1 — — —> 102 (Finance)

Knowing only EmployeeID 1 doesn't tell you which specific project Alice works on. It indicates there are potentially several project assignments for her, reflected in separate rows in the table. This highlights the many-to-many relationship between EmployeeID and ProjectID.

Candidate Keys

There are two candidate keys (CKs):

- **EmployeeID**: As discussed earlier, each employee has a unique identifier (EmployeeID) that pinpoints their specific record. You can identify any employee solely based on their EmployeeID, making it a candidate key.

EmployeeID -> EmployeeName
1 -> Alice

- **Combination of EmployeeID, ProjectID**: This might seem surprising, but consider a scenario where every employee has a distinct name within a project (no duplicates within projects). In that case, the combination of EmployeeName and ProjectID would uniquely identify each employee record.

  EmployeeID and ProjectID determine the Department
  EmployeeID -> ProjectID ==> Department
  1 -> 101 => HR

For instance, if Alice was assigned to Project X and Charlie to Project Y (assuming unique names within projects), then the combination of "John" and "Project X" or "Mary" and "Project Y" would uniquely identify their respective rows.

Summary

Aspect	Functional Dependencies	Multi-Valued Dependencies (MVDs)	Candidate Keys
Definition	Describe relationships between attributes	Extend the concept to groups of attributes	Sets of attributes uniquely identifying rows
Essence	Values of certain attributes determined by others	Dependencies between sets of attributes	Uniquely identify each row in a table
Example	Knowing one attribute allows finding another	Describe how sets of attributes determine each other	Combination of attributes uniquely identifying each record
Purpose	Ensure data accuracy and minimize redundancy	Organize data effectively, avoiding mix-ups	Enforce entity integrity constraints
Usage	Essential for database normalization	Crucial for maintaining data integrity	Establish relationships between tables

Author(s)

Akansha Yadav