Chapter 5: Logical Database Design and the Relational Model
Part 2: Normalization

Modern Database Management 6th Edition
Jeffrey A. Hoffer, Mary B. Prescott, Fred R. McFadden
Robert C. Nickerson
ISYS 464 – Spring 2003
Topic 22

Introduction to Normalization

- **Normalization**: The process of transforming relations into forms that make them easier to manage in certain situations.
- Objective of normalization: To eliminate modification anomalies from a relation, which are problems that can occur when a relation is modified (that is, when data is added, deleted, or changed).

Normal Forms

- **Normal forms**: Forms that a relation can take during normalization.
- First normal form – 1NF: Lower
- Second normal form – 2NF
- Third normal form – 3NF
- Boyce-Codd normal form - BCNF
- Fourth normal form – 4NF
- Fifth normal form – 5NF
- Domain/key normal form – DKNF: Higher

Practical View of Normalization

- Only up through 4NF is practical.
- We don't even know fully the consequences of a relation not being in 5NF.
- It is very difficult to determine if a relation is in DKNF.
- Not all relations can be put in DKNF.

Normal Forms

- Any relation in a higher normal form is also in all lower normal forms.
- Each higher normal form results in fewer modification anomalies.
- A relation in DKNF has no modification anomalies.

Practical View of Normalization

- Higher normal forms make it easier to update a database. In lower normal forms, modification anomalies have to be handled through complex programming that takes longer to execute.
- But, higher normal forms can make it harder to query a database. In higher normal forms, some queries require complex programming and take longer to execute.
De-normalization

- **De-normalization**: The process of changing a relation into a lower normal form to make the database easier to query even though there will be more modification anomalies in it.

Place of Normalization in Database Development

- Some people consider normalization to be a critical step in logical design. They recommend that all relations be put in as high a normal form as possible (**fully normalize all relations**).
- My view:
  - Normalization is related to performance tradeoff: What trade off are we willing to make between ease of modifying the database and ease of querying the database?

Place of Normalization in Database Development

- Good conceptual design usually results in relations in a relatively high normal form.
- Normal forms should be checked after completing the logical design (which is based on the conceptual design) to determine if the relations are in an acceptable normal form given the modification/query tradeoff.

Data Normalization

- Primarily a tool to validate and improve a logical design so that it satisfies certain constraints that **avoid unnecessary duplication of data**
- The process of decomposing relations with anomalies to produce smaller, well-structured relations

Well-Structured Relations

- A relation that contains minimal data redundancy and allows users to insert, delete, and update rows without causing data inconsistencies
- Goal is to avoid modification anomalies
  - **Insertion Anomaly** – adding new rows forces user to create duplicate data; adding data about more than one entity
  - **Deletion Anomaly** – deleting rows may cause a loss of data that would be needed for other future rows; deleting data about more than one entity
  - **Modification (update) Anomaly** – changing data in a row forces changes to other rows because of duplication; changing data about entity in more than one tuple
- General rule of thumb: a table should not pertain to more than one entity type

Example – Figure 5.2b

<table>
<thead>
<tr>
<th>EMPLOYEE</th>
<th>Name</th>
<th>Dept. Name</th>
<th>Salary</th>
<th>Course_Title</th>
<th>Date_Completed</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>Margaret Singer</td>
<td>Marketing</td>
<td>60,000</td>
<td>C++</td>
<td>6/12/2000</td>
</tr>
<tr>
<td>120</td>
<td>John Doe</td>
<td>Marketing</td>
<td>50,000</td>
<td>Java</td>
<td>6/12/2000</td>
</tr>
<tr>
<td>130</td>
<td>Jane Smith</td>
<td>Sales</td>
<td>65,000</td>
<td>Accounting</td>
<td>6/12/2000</td>
</tr>
<tr>
<td>140</td>
<td>Alice Johnson</td>
<td>Accounting</td>
<td>70,000</td>
<td>Finance</td>
<td>6/12/2000</td>
</tr>
<tr>
<td>150</td>
<td>bob</td>
<td>Finance</td>
<td>80,000</td>
<td>C++</td>
<td>6/12/2000</td>
</tr>
<tr>
<td>160</td>
<td>Tom Brown</td>
<td>Sales</td>
<td>62,000</td>
<td>Java</td>
<td>6/12/2000</td>
</tr>
<tr>
<td>170</td>
<td>tim</td>
<td>Sales</td>
<td>60,000</td>
<td>C++</td>
<td>6/12/2000</td>
</tr>
</tbody>
</table>

- Question – Is this a relation?  
  - Answer – Yes: unique rows and no multivalued attributes
- Question – What’s the primary key?  
  - Answer – Composite: Emp_ID, Course_Title
Anomalies in this Table
- **Insertion** – can’t enter a new employee without having the employee take a class
- **Deletion** – if we remove employee 140, we lose information about the existence of a Tax Acc class
- **Modification (update)** – giving a salary increase to employee 100 forces us to update multiple records

Why do these anomalies exist?
Because we’ve combined two themes (entity types) into one relation. This results in duplication, and an unnecessary dependency between the entities

Functional Dependencies
- 1NF though BCNF deal with functional dependencies
- **Functional dependency (FD):** Given a relation \( R(X,Y,\ldots) \), \( Y \) is functionally dependent on \( X \) (or \( X \) functionally determines \( Y \)) if and only if every value of \( X \) has associated with it only one value of \( Y \) at any one time. (Given a value of \( X \), there is only one value of \( Y \) associated with it at any one time.)
- Notation: \( X \rightarrow Y \) (functional dependency formula)

<table>
<thead>
<tr>
<th>Functionally Dependent</th>
<th>Not functionally Dependent</th>
</tr>
</thead>
<tbody>
<tr>
<td>X ( \rightarrow ) Y</td>
<td>X ( \not\rightarrow ) Y</td>
</tr>
<tr>
<td>1A</td>
<td>1A</td>
</tr>
<tr>
<td>1A</td>
<td>1B</td>
</tr>
<tr>
<td>2B</td>
<td>2B</td>
</tr>
</tbody>
</table>

Functional Dependencies and Keys
- **Functional Dependency:** The value of one attribute (the determinant) determines the value of another attribute
- **Candidate Key:**
  - A unique identifier. One of the candidate keys will become the primary key
    - E.g. perhaps there is both credit card number and SS# in a table...in this case both are candidate keys
  - Each non-key field is functionally dependent on every candidate key

Diagram: 5.22 -Steps in normalization
**First Normal Form**

- No multivalued attributes
- Every attribute value is atomic
- Fig. 5-2a on page 168 is not in 1st Normal Form (multivalued attributes) → it is not a relation
- EmpID, Name, Dept_Name, Salary, (Course_Title, Date_Completed)
- Fig. 5-2b is in 1st Normal form
- EmpID, Course_Title, Name, Dept_Name, Salary, Date_Completed
- *All relations are in 1st Normal Form*

**Second Normal Form**

- 1NF *plus* every non-key attribute is fully functionally dependent on the ENTIRE primary key
- Every non-key attribute must be defined by the entire key, not by only part of the key
- No partial functional dependencies
- Fig. 5-2b is NOT in 2nd Normal Form (see fig 5-23b)

**Modification Anomalies**

- What are the modification anomalies in this example?
- Insertion anomaly: Adding an employee requires adding course data
- Deletion anomaly: Deleting an employee involves deleting course data
- Update anomaly: Updating an employees salary may involve updating it in several tuples

**Transitive Dependency**

- Transitive dependency: Given a relation R(A,B,C,…), C is *transitively dependent* on A if A → B and B → C. (That is, one attribute functionally determines a second, which functionally determines a third.)
Modification Anomalies

- What are the modification anomalies in this example?
  - Insertion anomaly: Inserting a customer requires inserting data (region) about a salesperson
  - Deletion anomaly: Deleting a customer deletes data (region) about a salesperson
  - Update anomaly: Changing a salesperson’s region requires changing data in more than one tuple

Third Normal Form

- 2NF PLUS no transitive dependencies involving non-key attributes
Boyce-Codd Normal Form (Appendix B)

- All determinants are candidate keys
- 3NF relations can sometimes have anomalies related to functional dependencies
- BCNF relations eliminate these anomalies

Other Normal Forms (from Appendix B)

- 4th NF
  - No multivalued dependencies
- 5th NF
  - No “lossless joins”
- Domain-key NF
  - The “ultimate” NF…perfect elimination of all possible anomalies

Is a higher normal form always better than a lower one?

- Emp_address (ID, Name, Street, City, State, Zip)
- ID  →  Name, Street, City, State, Zip
- Street, City, State  →  Zip
- In 2NF, but not in 3NF because of transitive dependency
- To put in 3NF, put Zip into separate relation:
  - Emp_address (ID, Name, Street, City, State)
  - Zip (Street, City, State, Zip)
- But then join required each time address is needed
- Better to keep in 2NF because Zips rarely change

Integrity Constraints

- Domain Constraints
  - Allowable values for an attribute.
- Entity Integrity
  - No primary key attribute may be null. All primary key fields \textbf{MUST} have data
- Referential Integrity
  - Any foreign key value (on the relation of the many side) \textbf{MUST} match a primary key value in the relation of the one side. (Or the foreign key can be null)

Figure 5-5:
Referential integrity constraints (Pine Valley Furniture)