Chapter 14

Normalization
Materials To Have Handy

- A paper copy of the StaffBranch relation (pg. 369 of the textbook and on or about slide 9)
- A paper copy of the Staff Branch function dependencies (pg. 375 of the textbook and on or about slide 29)
- The textbook
What is Normalization?

- A technique to decompose relations into groupings of logically related attributes based on functional dependencies between attributes.
- A bottom-up design technique
Purpose of Normalization

- Normalization is a technique for producing a set of suitable relations that attempt to
  1. minimize the likelihood of introducing inconsistent data into the database, and
  2. minimize the amount of redundancy in the database
How Normalization Supports Database Design

- Use top-down approach such as ER modeling
  - Data sources
  - Users
  - Users’ requirements specification
  - Forms/reports that are used or generated by the enterprise
  - Sources describing the enterprise such as data dictionary and corporate data model
  - Set of well-designed relations
  - ER model is mapped to a set of relations

Approach 1
- Use normalization as a bottom-up technique to create set of relations. (This approach is described in this chapter and the next)

Approach 2
- Use normalization as a validation technique to check structure of relations. (This approach is described in Chapter 16, Step 2.2)
Data Redundancy and Update Anomalies

- Problems associated with data redundancy are illustrated by looking at the StaffBranch relation.

<table>
<thead>
<tr>
<th>staffNo</th>
<th>sName</th>
<th>position</th>
<th>salary</th>
<th>branchNo</th>
<th>bAddress</th>
</tr>
</thead>
<tbody>
<tr>
<td>SL21</td>
<td>John White</td>
<td>Manager</td>
<td>30000</td>
<td>B005</td>
<td>22 Deer Rd, London</td>
</tr>
<tr>
<td>SG37</td>
<td>Ann Beech</td>
<td>Assistant</td>
<td>12000</td>
<td>B003</td>
<td>163 Main St, Glasgow</td>
</tr>
<tr>
<td>SG14</td>
<td>David Ford</td>
<td>Supervisor</td>
<td>18000</td>
<td>B003</td>
<td>163 Main St, Glasgow</td>
</tr>
<tr>
<td>SA9</td>
<td>Mary Howe</td>
<td>Assistant</td>
<td>9000</td>
<td>B007</td>
<td>16 Argyll St, Aberdeen</td>
</tr>
<tr>
<td>SG5</td>
<td>Susan Brand</td>
<td>Manager</td>
<td>24000</td>
<td>B003</td>
<td>163 Main St, Glasgow</td>
</tr>
<tr>
<td>SL41</td>
<td>Julie Lee</td>
<td>Assistant</td>
<td>9000</td>
<td>B005</td>
<td>22 Deer Rd, London</td>
</tr>
</tbody>
</table>
Data Redundancy and Insertion Anomalies

- StaffBranch relation has redundant data; the details of a branch are repeated for every member of staff.
- If I want to insert a branch without any staff, I must insert NULL values for the staff
  - Since NULL values are not allowed in staffNo, a primary key, the insert fails
Update and Deletion Anomalies

- **Update Anomaly**: If I change a person’s branch number, I must remember to also change their branch address.

- **Update Anomaly**: If branch’s address changes, it must be updated multiple times.

- **Deletion Anomaly**: If all staff associated with a branch are deleted, the branch details gets deleted as well.
Data Redundancy and Update Anomalies

- Relations that contain redundant information may potentially suffer from update anomalies.

- Types of update anomalies include
  - Insertion
  - Deletion
  - Modification
Lossless-join and Dependency Preservation Properties

◆ Two important properties of decomposition.
  – *Lossless-join property* enables us to find any instance of the original relation from corresponding instances in the smaller relations.
  – *Dependency preservation property* enables us to enforce a constraint on the original relation by enforcing some constraint on each of the smaller relations.
Functional Dependencies

- Functional dependency describes relationship between attributes.
- For example, if A and B are attributes of relation R, B is functionally dependent on A (denoted A → B), if each value of A in R is associated with exactly one value of B in R.
An Example Functional Dependency

staffNo functionally determines position

(a)

Staff number SL21  Manager

(b)

position does not functionally determine staffNo

Staff number SL21  Staff number SG5
Determinant

- The *determinant* of a functional dependency refers to the attribute or group of attributes on the left-hand side of the arrow.
Functional Dependencies exist only if they hold for all time

- Consider the values shown in staffNo and sName attributes of the StaffBranch relation

- Based on sample data, the following functional dependencies appear to hold.

$$ staffNo \rightarrow sName $$
$$ sName \rightarrow staffNo $$
Example Functional Dependency that holds for all Time

- However, the only functional dependency that remains true for all possible values for the staffNo and sName attributes of the Staff relation is:

  \[ \text{staffNo} \rightarrow \text{sName} \]
Characteristics of Functional Dependencies

- Determinants should have the minimal number of attributes necessary to maintain the functional dependency with the attribute(s) on the right hand-side.

- This requirement is called *full functional dependency*.
Example of a Partial Dependency

- The following functional dependency exists in the StaffBranch relation

  \[ \text{staffNo, sName} \rightarrow \text{branchNo} \]

- However, branchNo is also functionally dependent on a subset of (staffNo, sName), namely staffNo. Example above is a partial dependency.
Characteristics of Functional Dependencies

◆ Main characteristics of functional dependencies used in normalization:
  – There is a \textit{one-to-one} relationship between the left-hand side and right-hand side attributes
  – Holds for \textit{all} time.
  – The determinant has the \textit{minimal} number of necessary attributes
Transitive Dependencies

- Important to recognize a transitive dependency because its existence in a relation can potentially cause update anomalies.

- Transitive dependency describes a condition where A, B, and C are attributes of a relation such that if A $\rightarrow$ B and B $\rightarrow$ C, then C is transitively dependent on A via B.
Example Transitive Dependency

- Consider functional dependencies in the StaffBranch relation

  \[ \text{staffNo} \rightarrow \text{sName, position, salary, branchNo} \]
  \[ \text{branchNo} \rightarrow \text{bAddress} \]

- bAddress has a transitive dependency, on staffNo via branchNo.
Example - Identifying a set of functional dependencies for the StaffBranch relation

- Examine semantics of attributes in StaffBranch relation. Assume that position held and branch determine a member of staff’s salary.
Example - Identifying a set of functional dependencies for the StaffBranch relation

- With sufficient information available, identify the functional dependencies for the StaffBranch relation as:

  - `staffNo → sName, position, salary, branchNo`
  - `branchNo → bAddress`
  - `bAddress → branchNo`
  - `branchNo, position → salary`
  - `bAddress, position → salary`
Main purpose of identifying a set of functional dependencies for a relation is to specify the set of integrity constraints that must hold on a relation.

An important integrity constraint to consider first is the identification of candidate keys, one of which is selected to be the primary key for the relation.
Example - Identify Primary Key for StaffBranch Relation

- To identify all candidate key(s), identify the attribute (or group of attributes) that uniquely identifies each tuple in this relation.
Example - Identifying Primary Key for StaffBranch Relation

- All attributes that are not part of a candidate key should be functionally dependent on the key.

- The only candidate key and therefore primary key for StaffBranch relation, is staffNo, as all other attributes of the relation are functionally dependent on staffNo.
The Process of Normalization

- Formal technique for analyzing a relation based on its primary key and the functional dependencies between the attributes of that relation.

- Often executed as a series of steps. Each step corresponds to a specific normal form, which has known properties.
The Process of Normalization

- As normalization proceeds, the relations become progressively more restricted (stronger) in format and also less vulnerable to update anomalies.
The Process of Normalization
Unnormalized Form (UNF)

- A table that contains one or more repeating groups (see pg. 382 of the book)

- To create an unnormalized table
  - Transform the data from the information source (e.g. form) into table format with columns and rows.
First Normal Form (1NF)

◆ A relation in which the intersection of each row and column contains one and only one value.
UNF to 1NF

- Nominate an attribute or group of attributes to act as the key for the unnormalized table.

- Identify the repeating group(s) in the unnormalized table which repeats for the key attribute(s).
UNF to 1NF

- **Remove the repeating group by**
  - Creating for each row a number of columns equal to the maximum occurrences of a value in a multi-valued attribute.
    - If there are at most 3 phone #’s, create 3 columns
    - Enter appropriate data into the columns of rows containing the repeating data (‘flattening’ the table).

  Or by

  - Placing the repeating data along with a copy of the original key attribute(s) into a separate relation.
Second Normal Form (2NF)

- **Conceptual Definition:** A relation that is in 1NF and every non-primary-key attribute is fully functionally dependent on the primary key.

- **Operational Transformation from 1NF:** Remove all partial dependencies on the primary key.
1NF to 2NF

- Identify the primary key for the 1NF relation.

- Identify the functional dependencies in the relation.

- If partial dependencies exist on the primary key remove them by placing them in a new relation along with a copy of their determinant.
Example: A sample clientRental relation with functional dependencies

<table>
<thead>
<tr>
<th>clientNo</th>
<th>prpertyNo</th>
<th>cName</th>
<th>pAddress</th>
<th>rentStart</th>
<th>rentFinish</th>
<th>rent</th>
<th>ownerNo</th>
<th>oName</th>
</tr>
</thead>
</table>

- **primary key**
- **candidate key**
- **candidate key**
Example: 1\textsuperscript{st} \rightarrow 2\textsuperscript{nd} Normal Form

- The partial dependency of \textit{cName} on \textit{clientNo} means that we should place these two attributes in a new relation and remove \textit{cName} from \textit{clientRental}

- The partial dependency \textit{propertyNo} $\rightarrow$ \textit{pAddress}, rent, ownerNo, oName means these attributes should be placed in a new relation and remove \textit{pAddress}, rent, ownerNo, and oName from \textit{clientRental}
New Relations

<table>
<thead>
<tr>
<th>Client</th>
</tr>
</thead>
<tbody>
<tr>
<td>clientNo</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PropertyOwner</th>
</tr>
</thead>
<tbody>
<tr>
<td>propertyNo</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Rental</th>
</tr>
</thead>
<tbody>
<tr>
<td>clientNo</td>
</tr>
</tbody>
</table>

Note that original clientRental relation can be reconstituted via joins
Third Normal Form (3NF)

- A relation that is in 1NF and 2NF and in which no non-primary-key attribute is transitively dependent on the primary key.

<table>
<thead>
<tr>
<th>propertyNo</th>
<th>pAddress</th>
<th>rent</th>
<th>ownerNo</th>
<th>oName</th>
</tr>
</thead>
</table>

The above relation is not in 3NF because of the transitive dependency ownerNo → oName
2NF to 3NF

◆ Identify the primary key in the 2NF relation.

◆ Identify functional dependencies in the relation.

◆ If transitive dependencies exist on the primary key remove them by placing them in a new relation along with a copy of their determinant.
Example

- We convert PropertyOwner to 3NF by creating a new relation with the attributes (ownerNo, oName) and deleting oName from PropertyOwner

<table>
<thead>
<tr>
<th>PropertyForRent</th>
</tr>
</thead>
<tbody>
<tr>
<td>propertyNo</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Owner</th>
</tr>
</thead>
<tbody>
<tr>
<td>ownerNo</td>
</tr>
</tbody>
</table>
General Definitions of 2NF and 3NF

◆ Previous definitions in this chapter applied only to primary keys. More general definitions follow:

◆ Second normal form (2NF)
  – A relation that is in first normal form and every non-primary-key attribute is fully functionally dependent on any candidate key.

◆ Third normal form (3NF)
  – A relation that is in first and second normal form and in which no non-primary-key attribute is transitively dependent on any candidate key.
Trade-Off between PK Normal Form and General Normal Form

- PK Normalization is simpler
- General Normalization provides more opportunities for eliminating redundancies and data inconsistencies