This chapter presents the principles of relational database design. Undergraduates frequently find this chapter difficult. It is acceptable to cover only Sections 7.1 and 7.3 for classes that find the material particularly difficult. However, a careful study of data dependencies and normalization is a good way to introduce students to the formal aspects of relational database theory.

There are many ways of stating the definitions of the normal forms. We have chosen a style which we think is the easiest to present and which most clearly conveys the intuition of the normal forms.

Exercises

7.17 Explain what is meant by repetition of information and inability to represent information. Explain why each of these properties may indicate a bad relational database design.

Answer:

- Repetition of information is a condition in a relational database where the values of one attribute are determined by the values of another attribute in the same relation, and both values are repeated throughout the relation. This is a bad relational database design because it increases the storage required for the relation and it makes updating the relation more difficult.

- Inability to represent information is a condition where a relationship exists among only a proper subset of the attributes in a relation. This is bad relational database design because all the unrelated attributes must be filled with null values otherwise a tuple without the unrelated information cannot be inserted into the relation.

- Loss of information is a condition of a relational database which results from the decomposition of one relation into two relations and which cannot be
combined to recreate the original relation. It is a bad relational database design because certain queries cannot be answered using the reconstructed relation that could have been answered using the original relation.

7.18 Why are certain functional dependencies called trivial functional dependencies?
Answer: Certain functional dependencies are called trivial functional dependencies because they are satisfied by all relations.

7.19 Use the definition of functional dependency to argue that each of Armstrong’s axioms (reflexivity, augmentation, and transitivity) is sound.
Answer: The definition of functional dependency is:
\( \alpha \rightarrow \beta \) holds on \( R \) if in any legal relation \( r(R) \), for all pairs of tuples \( t_1 \) and \( t_2 \) in \( r \) such that \( t_1[\alpha] = t_2[\alpha] \), it is also the case that \( t_1[\beta] = t_2[\beta] \).

Reflexivity rule: if \( \alpha \) is a set of attributes, and \( \beta \subseteq \alpha \), then \( \alpha \rightarrow \beta \).

Assume \( \exists t_1, t_2 \) such that \( t_1[\alpha] = t_2[\alpha] \)
\( t_1[\beta] = t_2[\beta] \) since \( \beta \subseteq \alpha \)
\( \alpha \rightarrow \beta \) definition of FD

Augmentation rule: if \( \alpha \rightarrow \beta \) and \( \gamma \) is a set of attributes, then \( \gamma \alpha \rightarrow \gamma \beta \).

Assume \( \exists t_1, t_2 \) such that \( t_1[\gamma \alpha] = t_2[\gamma \alpha] \)
\( t_1[\gamma] = t_2[\gamma] \) since \( \gamma \subseteq \gamma \alpha \)
\( t_1[\alpha] = t_2[\alpha] \) since \( \alpha \subseteq \gamma \alpha \)
\( t_1[\beta] = t_2[\beta] \) definition of \( \alpha \rightarrow \beta \)
\( t_1[\gamma \beta] = t_2[\gamma \beta] \) \( \gamma \beta = \gamma \cup \beta \)
\( \gamma \alpha \rightarrow \gamma \beta \) definition of FD

Transitivity rule: if \( \alpha \rightarrow \beta \) and \( \beta \rightarrow \gamma \), then \( \alpha \rightarrow \gamma \).

Assume \( \exists t_1, t_2 \) such that \( t_1[\alpha] = t_2[\alpha] \)
\( t_1[\beta] = t_2[\beta] \) definition of \( \alpha \rightarrow \beta \)
\( t_1[\gamma] = t_2[\gamma] \) definition of \( \beta \rightarrow \gamma \)
\( \alpha \rightarrow \gamma \) definition of FD

7.20 Consider the following proposed rule for functional dependencies: If \( \alpha \rightarrow \beta \) and \( \gamma \rightarrow \beta \), then \( \alpha \rightarrow \gamma \). Prove that this rule is not sound by showing a relation \( r \) that satisfies \( \alpha \rightarrow \beta \) and \( \gamma \rightarrow \beta \), but does not satisfy \( \alpha \rightarrow \gamma \).
Answer: Consider the following rule: if \( A \rightarrow B \) and \( C \rightarrow B \), then \( A \rightarrow C \). That is, \( \alpha = A, \beta = B, \gamma = C \). The following relation \( r \) is a counterexample to the rule.

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>a1</td>
<td>b1</td>
<td>c1</td>
<td></td>
</tr>
<tr>
<td>a1</td>
<td>b1</td>
<td>c2</td>
<td></td>
</tr>
</tbody>
</table>

Note: \( A \rightarrow B \) and \( C \rightarrow B \) (since no 2 tuples have the same \( C \) value, \( C \rightarrow B \) is true trivially). However, it is not the case that \( A \rightarrow C \) since the same \( A \) value is in two tuples, but the \( C \) value in those tuples disagree.
7.21 Use Armstrong’s axioms to prove the soundness of the decomposition rule.

**Answer:** The decomposition rule, and its derivation from Armstrong’s axioms are given below:

if \( \alpha \rightarrow \beta \gamma \), then \( \alpha \rightarrow \beta \) and \( \alpha \rightarrow \gamma \).

\[ \begin{align*}
\alpha \rightarrow \beta \gamma & \text{ given} \\
\beta \gamma \rightarrow \beta & \text{ reflexivity rule} \\
\alpha \rightarrow \beta & \text{ transitivity rule} \\
\beta \gamma \rightarrow \gamma & \text{ reflexive rule} \\
\alpha \rightarrow \gamma & \text{ transitive rule}
\end{align*} \]

7.22 Using the functional dependencies of Practice Exercise 7.6, compute \( B^+ \).

**Answer:** Computing \( B^+ \) by the algorithm in Figure 7.9 we start with \( \text{result} = \{ B \} \). Considering FDs of the form \( \beta \rightarrow \gamma \) in \( F \), we find that the only dependencies satisfying \( \beta \subseteq \text{result} \) are \( B \rightarrow B \) and \( B \rightarrow D \). Therefore \( \text{result} = \{ B, D \} \). No more dependencies in \( F \) apply now. Therefore \( B^+ = \{ B, D \} \)

7.23 Show that the following decomposition of the schema \( R \) of Practice Exercise 7.1 is not a lossless-join decomposition:

\[
\begin{align*}
(A, B, C) \\
(C, D, E).
\end{align*}
\]

**Hint:** Give an example of a relation \( r \) on schema \( R \) such that

\[
\Pi_{A,B,C} (r) \Join \Pi_{C,D,E} (r) \neq r
\]

**Answer:** Following the hint, use the following example of \( r \):

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>( a_1 )</td>
<td>( b_1 )</td>
<td>( c_1 )</td>
<td>( d_1 )</td>
<td>( e_1 )</td>
</tr>
<tr>
<td>( a_2 )</td>
<td>( b_2 )</td>
<td>( c_1 )</td>
<td>( d_2 )</td>
<td>( e_2 )</td>
</tr>
</tbody>
</table>

With \( R_1 = (A, B, C), R_2 = (C, D, E) \):

a. \( \Pi_{R_1} (r) \) would be:

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>( a_1 )</td>
<td>( b_1 )</td>
<td>( c_1 )</td>
</tr>
<tr>
<td>( a_2 )</td>
<td>( b_2 )</td>
<td>( c_1 )</td>
</tr>
</tbody>
</table>

b. \( \Pi_{R_2} (r) \) would be:

<table>
<thead>
<tr>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>( c_1 )</td>
<td>( d_1 )</td>
<td>( e_1 )</td>
</tr>
<tr>
<td>( c_1 )</td>
<td>( d_2 )</td>
<td>( e_2 )</td>
</tr>
</tbody>
</table>

c. \( \Pi_{R_1} (r) \Join \Pi_{R_2} (r) \) would be:
Clearly, $\Pi_{R_1}(r) \not\cong \Pi_{R_2}(r) \neq r$. Therefore, this is a lossy join.

7.24 List the three design goals for relational databases, and explain why each is desirable.
Answer: The three design goals are lossless-join decompositions, dependency preserving decompositions, and minimization of repetition of information. They are desirable so we can maintain an accurate database, check correctness of updates quickly, and use the smallest amount of space possible.

7.25 Give a lossless-join decomposition into BCNF of schema $R$ of Exercise 7.1.
Answer: From Exercise 7.6, we know that $B \rightarrow D$ is nontrivial and the left hand side is not a superkey. By the algorithm of Figure 7.12 we derive the relations $\{ (A, B, C, E), (B, D) \}$. This is in BCNF.

7.26 In designing a relational database, why might we choose a non-BCNF design?
Answer: BCNF is not always dependency preserving. Therefore, we may want to choose another normal form (specifically, 3NF) in order to make checking dependencies easier during updates. This would avoid joins to check dependencies and increase system performance.

7.27 Give a lossless-join, dependency-preserving decomposition into 3NF of schema $R$ of Practice Exercise 7.1.
Answer: First we note that the dependencies given in Practice Exercise 7.1 form a canonical cover. Generating the schema from the algorithm of Figure 7.13 we get

$$R' = \{(A, B, C), (C, D, E), (B, D), (E, A)\}.$$  

Schema $(A, B, C)$ contains a candidate key. Therefore $R'$ is a third normal form dependency-preserving lossless-join decomposition.

Note that the original schema $R = (A, B, C, D, E)$ is already in 3NF. Thus, it was not necessary to apply the algorithm as we have done above. The single original schema is trivially a lossless join, dependency-preserving decomposition.

7.28 Given the three goals of relational-database design, is there any reason to design a database schema that is in 2NF, but is in no higher-order normal form? (See Practice Exercise 7.15 for the definition of 2NF.)
Answer: The three design goals of relational databases are to avoid
- Repetition of information
- Inability to represent information
- Loss of information.
2NF does not prohibit as much repetition of information since the schema 
\((A, B, C)\) with dependencies \(A \rightarrow B\) and \(B \rightarrow C\) is allowed under 2NF, although the same \((B, C)\) pair could be associated with many \(A\) values, needlessly duplicating \(C\) values. To avoid this we must go to 3NF. Repetition of information is allowed in 3NF in some but not all of the cases where it is allowed in 2NF. Thus, in general, 3NF reduces repetition of information. Since we can always achieve a lossless join 3NF decomposition, there is no loss of information needed in going from 2NF to 3NF.

Note that the decomposition \\{\((A, B)\), \((B, C)\)\} is a dependency-preserving and lossless-join 3NF decomposition of the schema \((A, B, C)\). However, in case we choose this decomposition, retrieving information about the relationship between \(A, B\) and \(C\) requires a join of two relations, which is avoided in the corresponding 2NF decomposition.

Thus, the decision of which normal form to choose depends upon how the cost of dependency checking compares with the cost of the joins. Usually, the 3NF would be preferred. Dependency checks need to be made with every insert or update to the instances of a 2NF schema, whereas, only some queries will require the join of instances of a 3NF schema.

7.29 Given a relational schema \(r(A, B, C, D)\), does \(A \rightarrow BC\) logically imply \(A \rightarrow B\) and \(A \rightarrow C\)? If yes prove it, else give a counter example.

**Answer:** \(A \rightarrow BC\) holds on the following table:

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>1</td>
</tr>
</tbody>
</table>

If \(A \rightarrow B\), then we know that there exists \(t_1\) and \(t_3\) such that \(t_1[B] = t_3[B]\). Thus, we must choose one of the following for \(t_1\) and \(t_3\):

- \(t_1 = r_1\) and \(t_3 = r_3\), or \(t_1 = r_3\) and \(t_3 = r_1\):
  
  Choosing either \(t_2 = r_2\) or \(t_2 = r_4\), \(t_3[C] \neq t_2[C]\).

- \(t_1 = r_2\) and \(t_3 = r_4\), or \(t_1 = r_4\) and \(t_3 = r_2\):

  Choosing either \(t_2 = r_1\) or \(t_2 = r_3\), \(t_3[C] \neq t_2[C]\).

Therefore, the condition \(t_3[C] = t_2[C]\) can not be satisfied, so the conjecture is false.

7.30 Explain why 4NF is a normal form more desirable than BCNF.

**Answer:** 4NF is more desirable than BCNF because it reduces the repetition of information. If we consider a BCNF schema not in 4NF (see Practice Exercise 7.16), we observe that decomposition into 4NF does not lose information provided that a lossless join decomposition is used, yet redundancy is reduced.