Relational Design and Functional Dependency

The most important logical criteria in database design are reduction/elimination of redundancy and maintenance of database consistency. Therefore, our principle goal in relational design is to design relations that store each fact (tuple) only once in the database and that remain consistent following database operations (updates, insertions and deletions). Such relations are said to be in normal form.

There are a number of different normal forms. Each normal form satisfies specific criteria for that form, eliminating a different kind of redundancy. The goal is usually to get relations into the highest (or at least a high) normal form, without loss of information.

A relation is converted into a normal form by the process of decomposition; dividing the relation into smaller relations (i.e. relations that each have fewer attributes). This must be done in such a manner that the original relation can always be recovered (using a natural join). Such decomposition is called lossless. The general term for converting relations of a database into relations in a particular normal form is called normalisation.

Example of a Relation Requiring Normalisation

To gain an insight into the anomalies which may occur if a relation is not in normal form, examine the following relation, and complete the tasks, noting the inconsistencies and loss of data which arise.

1. Add tuple (S85,35,P1,9). Results in inconsistency in project budget for P1.
2. Update tuple (S75,32,P1,7) to be (S75,35,P1,7). The result is inconsistent budget information.

**RELATION: ASSIGN**

<table>
<thead>
<tr>
<th>Person-ID</th>
<th>Project-Budget</th>
<th>Project</th>
<th>Time-spent-by-person-on-project</th>
</tr>
</thead>
<tbody>
<tr>
<td>S75</td>
<td>32</td>
<td>P1</td>
<td>7</td>
</tr>
<tr>
<td>S75</td>
<td>40</td>
<td>P2</td>
<td>3</td>
</tr>
<tr>
<td>S79</td>
<td>32</td>
<td>P1</td>
<td>4</td>
</tr>
<tr>
<td>S79</td>
<td>27</td>
<td>P3</td>
<td>1</td>
</tr>
<tr>
<td>S80</td>
<td>40</td>
<td>P2</td>
<td>5</td>
</tr>
</tbody>
</table>
Another way of stating the goal of relational design is to represent user information using relations that do not create anomalies following insert, delete or update operations.

Normalisation of tables (putting them into normal form) consists of decomposing a relation (by projection) into more than one relation, each of which will not lead to anomalies in the data after updates, insertions or deletions of tuples. However, we must be careful not to lose information during decomposition. Generally, the loss of information manifests by way of ‘spurious’ tuples being generated when the tables are joined.

Consider the below decomposition of the ASSIGN relation. Note that there is no redundant data and that inconsistencies cannot arise from insertions, deletions and updates.

<table>
<thead>
<tr>
<th>RELATION: PROJECT</th>
<th>RELATION: ASSIGN_1</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Project</strong></td>
<td><strong>Person-ID</strong></td>
</tr>
<tr>
<td>P1</td>
<td>32</td>
</tr>
<tr>
<td>P2</td>
<td>40</td>
</tr>
<tr>
<td>P3</td>
<td>27</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Note also that we can recover the original table using a natural join, as below. This indicates that the decomposition is lossless.

\[
R1 = \text{JOIN PROJECT, ASSIGN}_1 \text{ OVER Project}
\]

Note that it will not always necessarily be true that a decomposed relation can be joined to recover exactly the original table. This loss of information in decomposition must be avoided. We will return to a treatment of this topic later.

Before addressing normal forms and decomposition formally, we must first examine the concept of **functional dependencies (FDs)** and some related concepts.
Functional Dependency

**Functional Dependency (FD)** is a term derived from the mathematical theory that underpins relational database theory. It concerns the dependence of the values of one set of attributes on those of another set of attributes.

Consider the relation:

**STUDENTS** = \{Student_ID, First_Name, Last_Name\}

We may state that the set of attributes \{First_Name, Last_Name\} is *functionally dependent* on the attribute set \{Student_ID\}. This means that, given a value for Student_ID, we can always uniquely determine the value of First_Name and the value of Last_Name. Note that, for this relation, the opposite would not be true. For example, if there are three students with the same first and last name we will get a list of three student IDs. That is we cannot uniquely determine a value for Student_ID given values of the attributes \{First_Name, Last_Name\}.

Student_ID determines \{First_Name, Last_Name\} but \{First_Name, Last_Name\} does not determine Student_ID.

We define functional dependency more formally:

A set of attributes \(Y\) is functionally dependent on a set of attributes \(X\) if a given set of values for each attribute in \(X\) determines unique values for the set of attributes in \(Y\). We use the notation:

\[ X \rightarrow Y \]

To denote that \(Y\) is functionally dependent on \(X\). The set of attributes \(X\) is known as the **determinant** of the FD \(X \rightarrow Y\).

Note that a set of FDs for a relation expresses ‘semantic’ information about the attributes of the relation, i.e. the meaning of attributes and the relationships between them. Exactly what FDs apply to a relation is determined by the designer’s interpretation of the data entities in the real-world application being modeled.
Looking again at the relation ASSIGN.

**RELATION: ASSIGN**

<table>
<thead>
<tr>
<th>Person-ID</th>
<th>Project-Budget</th>
<th>Project</th>
<th>Time-spent-by-person-on-project</th>
</tr>
</thead>
<tbody>
<tr>
<td>S75</td>
<td>32</td>
<td>P1</td>
<td>7</td>
</tr>
<tr>
<td>S75</td>
<td>40</td>
<td>P2</td>
<td>3</td>
</tr>
<tr>
<td>S79</td>
<td>32</td>
<td>P1</td>
<td>4</td>
</tr>
<tr>
<td>S79</td>
<td>27</td>
<td>P3</td>
<td>1</td>
</tr>
<tr>
<td>S80</td>
<td>40</td>
<td>P2</td>
<td>5</td>
</tr>
</tbody>
</table>

The attribute **Project-Budget** is functionally dependent on the attribute **Project**, because each project has one given project budget. Once a project name is known, a unique value of **Project-Budget** is also known. We can alternatively express this FD using the notation: **Project → Project-Budget**:

Another FD in ASSIGN is:

\[
\{\text{Person\_ID, Project}\} \rightarrow \text{Time-spent-by-person-on-project}
\]

We can express these FDs of the relation ASSIGN graphically using an **FD diagram** (as below) which represents attributes and sets of attributes as rectangles, and FDs as arrows directed from the determinant to the attribute(s) it determines.
Exercise 6

Write out each of the FDs depicted in the diagram below and describe what each one means. If all the attributes were part of the same relation, what would the relation key be?

Exercise 7

Again referring to the database schema from Exercise 3, write down the functional dependencies between the attributes in each relation.

STUDENT (Student_ID, First_Name, Last_Name)
REGISTER (Student_ID, Module_ID, Semester-Start-Date)
LECTURER (Lecturer_ID, First_Name, Last_Name)
MODULE (Module_ID, Module_Name, Lecturer_ID)
Full Functional Dependencies

Consider the following FD from our previous example relation:

\{\text{Person-ID, Project, Project-Budget}\} \rightarrow \{\text{Time-spent-by-person-on-project}\}

This FD is true, as it states that for each value of \text{Person-ID, Project} and \text{Project-Budget}, there is a unique value for the time spent. However, the attribute \text{Project-Budget} is not required in the determinant.

The term \textit{full functional dependency} is used to describe the minimum set of attributes in the determinant of an FD. The rules for full functional dependency are that if the set of attributes \(Y\) are to be fully dependent on the set of attributes \(X\), the following must hold:

- \(Y\) is functionally dependent on \(X\) and
- \(Y\) is not functionally dependent on any subset of \(X\)

According to these rules, the above FD is not fully dependent, whereas, by removing the attribute \text{Project-Budget}, we can acquire an FD which is fully functionally dependent. We say that the attribute \text{Project-Budget} is \textit{redundant}. It may be removed without losing information.

The converse of full FD: A functional dependency \(X \rightarrow Y\) is called a \textit{partial functional dependency} if some attribute \(A \in X\) can be removed from \(X\) and the dependency will still hold.

Question: How does the concept of a relation key relate to the concept of full functional dependency?
Trivial and Non-Trivial Functional Dependencies

It is often the case when examining the FDs associated with a relation that some FDs are of less interest because they are self-evident. For example consider the relation:

STUDENT (Student_ID, First_Name, Last_Name)

The FD \{First_Name, Last_Name\} → \{First_Name\} is true but of little informational interest.

We say that an FD is a **trivial FD** if and only if the right-hand side is a subset (not necessarily a proper subset) of the left-hand side.

**Non-trivial FDs** are, of course, those which are not by definition trivial.