**Chapter 5**

**Normalization**

1. **Introduction**
2. **First normal form 1NF**
3. **Second normal form 2NF**
4. **Third normal form 3NF**

**5.1 What is Normalization?**

**Normalization** is a *process* in which we systematically examine relations for *anomalies* and, when detected, remove those anomalies by splitting up the relation into two new, related, relations.

Normalization is an important part of the database development process: Often during normalization, the database designers get their first real look into how the data are going to interact in the database.

Finding problems with the database structure at this stage is strongly preferred to finding problems further along in the development process because at this point it is fairly easy to cycle back to the conceptual model (Entity Relationship model) and make changes.

Normalization can also be thought of as a trade-off between data redundancy and performance. Normalizing a relation reduces data redundancy but introduces the need for joins when all of the data is required by an application such as a report query.

Recall, the Relational Model consists of the elements: relations, which are made up of attributes.

A **relation** is a set of attributes with values for each attribute such that:

* 1. Each attribute (column) value must be a single value only.
  2. All values for a given attribute (column ) must be of the same data type.
  3. Each attribute (column) name must be unique.
  4. The order of attributes (columns) is insignificant
  5. No two tuples (rows) in a relation can be identical.
  6. The order of the tuples (rows) is insignificant.

**Normalization Benefits:**

1. Facilitates data integration.
2. Reduces data redundancy.
3. Provides a robust architecture for retrieving and maintaining data.
4. Compliments data modeling.
5. Reduces the chances of data anomalies occurring.

**5.2 Problem Without Normalization**

Without Normalization, it becomes difficult to handle and update the database, without facing data loss. **Insertion, Updating and Deletion Anomalies** are very frequent if Database is not Normalized. To understand these anomalies let us take an example of **Student** table.

|  |  |  |  |
| --- | --- | --- | --- |
| **S\_id** | **S\_Name** | **S\_Address** | **Subject\_opted** |
| 401 | Adam | Noida | Bio |
| 402 | Alex | Panipat | Maths |
| 403 | Stuart | Jammu | Maths |
| 404 | Adam | Noida | Physics |

* **Updating Anomaly :** To update address of a student who occurs twice or more than twice in a table, we will have to update **S\_Address** column in all the rows, else data will become inconsistent.
* **Insertion Anomaly :** Suppose for a new admission, we have a Student id(S\_id), name and address of a student but if student has not opted for any subjects yet then we have to insert **NULL** there, leading to Insertion Anamoly.
* **Deletion Anomaly :** If (S\_id) 401 has only one subject and temporarily he drops it, when we delete that row, entire student record will be deleted along with it.

**5.3 Functional Dependencies**

The single most important concept in relational schema design theory is that of a functional dependency.

A functional dependency is a constraint between two sets of attributes from the database. Suppose that our relational database schema has *n* attributes *A*1, *A*2, ..., *An*.

If we think of the whole database as being described by a single **universal** relation schema *R* = {*A*1, *A*2, ... , *An*}.

A **functional dependency**, denoted by *X →* *Y*, between two sets of attributes *X* and *Y* that are subsets of *R, such that* any two tuples *t*1 and *t*2 in *r* that have *t*1[*X*] = *t*2[*X*], they must also have *t*1[*Y*] = *t*2[*Y*].

This means that the values of the *Y* component of a tuple in *r* depend on, or are *determined by,* the values of the *X* component;

We say that the values of the *X* component of a tuple uniquely (or **functionally**) *determine* the values of the *Y* component.

We say that there is a functional dependency from *X* to *Y*, or that *Y* is **functionally dependent** on *X*.

Functional dependency is represented as **FD** or **f.d.** The set of attributes *X* is called the **left-hand side** of the FD, and *Y* is called the **right-hand side**.

*X* functionally determines *Y* in a relation schema *R* if, and only if, whenever two tuples of *r*(*R*) agree on their *X*-value, they must necessarily agree on their *Y-*value.

If a constraint on *R* states that there cannot be more than one tuple with a given *X*-value in any relation instance *r*(*R*)—that is, *X* is a **candidate key** of *R—* this implies that *X →Y* for any subset of attributes *Y* of *R.*

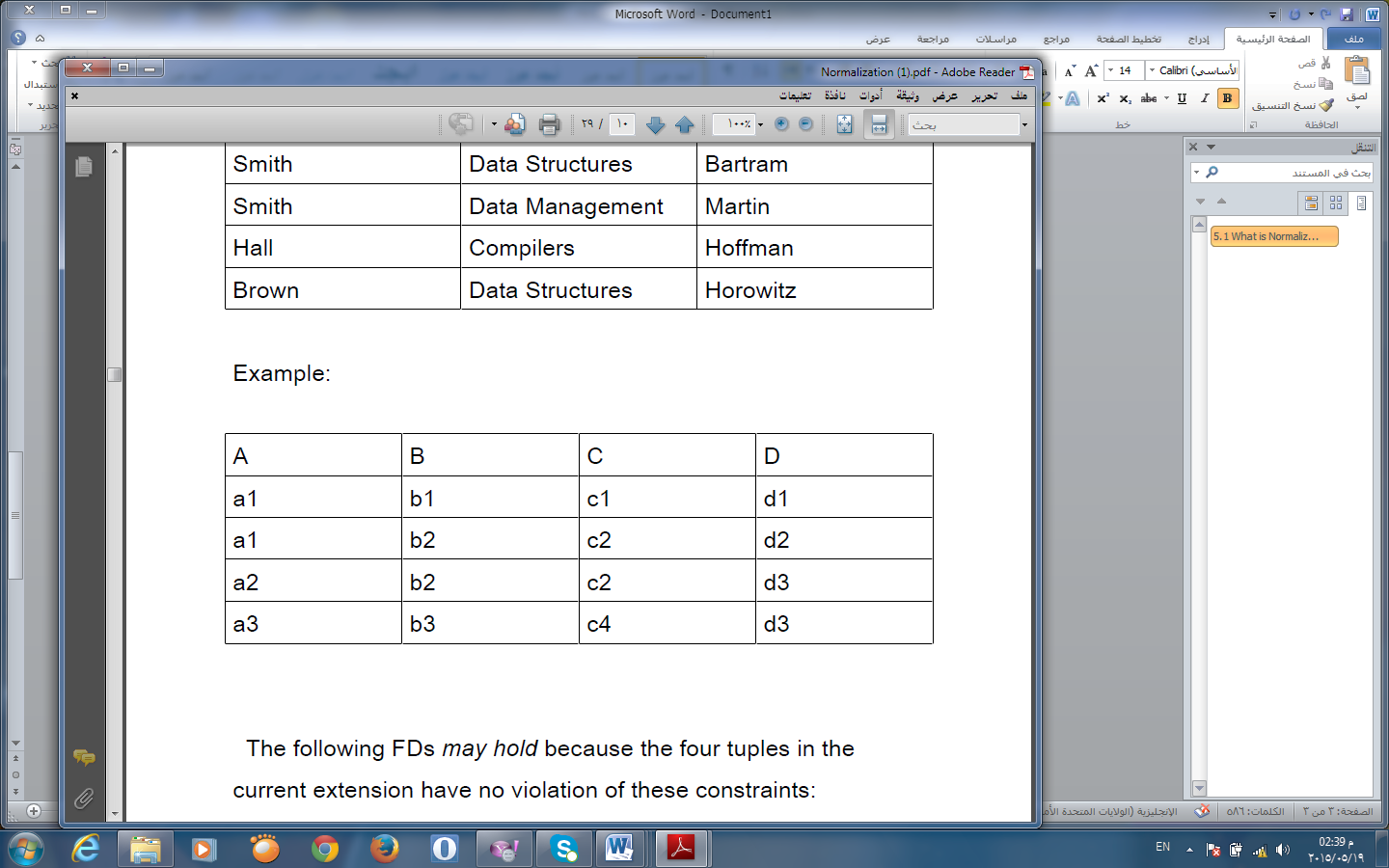
If *X* is a candidate key of *R*, then *X →R.*

If *X→Y* in *R*, this does not imply that *Y→X* in *R*.

A functional dependency is a property of the **semantics** or **meaning of the attributes**.

Whenever the semantics of two sets of attributes in *R* indicate that a functional dependency should hold, we specify the **dependency as a constraint.**

**Example :**



The following FDs *may hold* because the four tuples in the

current extension have no violation of these constraints:

*B* *C; C* *B;* {*A, B*} *C;* {*A, B*} *D;* and {*C, D*} *B*

However, the following *do not* hold because we already have

violations of them in the given extension*:*

*A* *B* (tuples 1 and 2 violate this constraint);

*B* *A* (tuples 2 and 3 violate this constraint);

*D**C* (tuples 3 and 4 violate it).

**5.3.1 Fully functional dependency (composite key)**

If attribute B is functionally dependent on a composite key A but not on any subset of that composite key, the attribute B is fully functionally dependent on A.

**5.3.2 Partial Dependency:**

When there is a functional dependence in which the determinant is only part of the primary key, then there is a partial dependency.

For example if (A, B)  (C, D) and B C and (A, B) is the primary key, then the functional dependence B C is a partial dependency.

**5.3.3 Transitive Dependency:**

When there are the following functional dependencies such that XY, Y Z and X is the primary key, then XZ is a transitive dependency because X determines the value of Z via Y.

Whenever a functional dependency is detected amongst nonprime,

there is a transitive dependency.

The advantage of removing transitive dependency is,

* Amount of data duplication is reduced.
* Data integrity achieved.

**5.4 Normalization of Relations**

The normalization process, as first proposed by Codd (1972a),takes a relation schema through a series of tests to *certify* whether it satisfies a certain **normal form**.

The process, which proceeds in a top-down fashion by evaluating each relation against the criteria for normal forms and decomposing relations as necessary, can thus be considered as *relational design by analysis.*

Initially, Codd proposed three normal forms, which he called first, second, and third normal form.

**5.4.1 How do you divide your tables?**

The basic rule is that **each table should describe one type of things, each row in the table should contain about one such thing, and the data we stored for each thing should exist in only one row**.

This can often be sufficient to know. If one follows this basic rule, ones databases will get a good design and one avoids problems with redundancy, things that will not be possible to store, and tables that is hard to understand.

However, sometimes it is difficult to actually know what kind of “things” it is that one would like to store and which data that is related to them. Then we can take use of the theory of normalization.

It helps us to see exactly how different columns within a table are related and shows us how to divide the table to avoid our problems. Therefore we will start looking at the different **normal forms** that the theory of normalization describes. Normal forms are conditions that tables should fulfill. The simplest form is the first normal form and by adding more conditions one can define the second normal form, third normal form and further on.

**5.4.2 The First Normal Form ( 1NF )**

A database is in first normal form if it satisfies the following conditions:

1. All the key attributes are defined.
2. There are no repeating groups in the table.
3. The value of record must be atomic.
4. All attributes are dependent on the primary key.

**Example1:** Consider the following table **stud** **:**

**stud**

|  |  |  |
| --- | --- | --- |
| **STU-ID** | **L-NAME** | **F-NAME** |
| 001 | Smith | John |
| 002 | Smith | Susan |
| 003 | Beal | Fred |
| 004 | Thomoson | Marie |
| 005 | Tom | Jake |
| 002 | Smith | Susan |
| 004 | Thomoson | Marie |
| 003 | Beal | Fred |

To bring this table to first normal form, we delete the duplication row, and now we have the following table stud1:

**Stud1**

|  |  |  |
| --- | --- | --- |
| **STU-ID** | **L-NAME** | **F-NAME** |
| 001 | Smith | John |
| 002 | Smith | Susan |
| 003 | Beal | Fred |
| 004 | Thomoson | Marie |
| 005 | Tom | Jake |

**Example2: Consider the following table student:**

**Student**

|  |  |  |
| --- | --- | --- |
| **STU-ID** | **CNAME** | **GRADE** |
| 001 | English , Italian | A |
| 002 | German , English | B |
| 003 | Italian | C |

To bring this table to first normal form, we convert data to **atomic value** , and now we have the following table student1:

**Student1**

|  |  |  |
| --- | --- | --- |
| **STU-ID** | **CNAME** | **GRADE** |
| 001 | English | A |
| 001 | Italian | A |
| 002 | German | B |
| 002 | English | B |
| 003 | Italian | C |

**5.4.3 Second Normal Form (2NF)**

A database is in second normal form if it satisfies the following conditions:

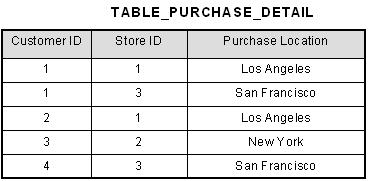
* It is in first normal form
* All non-key attributes are fully functional dependent on the primary key

In a table, if attribute B is functionally dependent on A, but is not functionally dependent on a proper subset of A, then B is considered fully functional dependent on A. Hence, in a 2NF table, all non-key

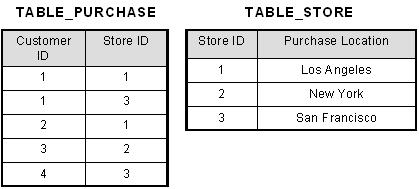
attributes cannot be dependent on a subset of the primary key. Note that if the primary key is not a composite key, all non-key attributes are always fully functional dependent on the primary key.

A table that is in 1st normal form and contains only a single key as the primary key is automatically in 2nd normal form.

**Example: Consider the following example:**



This table has a composite primary key [Customer ID, Store ID]. The non-key attribute is [Purchase Location]. In this case, [Purchase Location] only depends on [Store ID], which is only part of the primary key. Therefore, this table does not satisfy second normal form.

To bring this table to second normal form, we break the table into two tables, and now we have the following:

What we have done is to remove the partial functional dependency that we initially had. Now, in the table [TABLE\_STORE], the column [Purchase Location] is fully dependent on the primary key of that table, which is [Store ID].

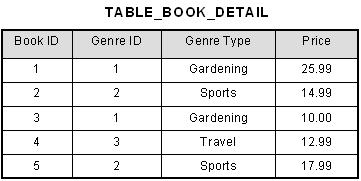
**5.4.4 Third Normal Form (3NF)**

A database is in third normal form if it satisfies the following conditions:

* It is in second normal form
* There is no transitive functional dependency

By transitive functional dependency, we mean we have the following relationships in the table: A is functionally dependent on B, and B is functionally dependent on C. In this case, C is transitively dependent on A via B.

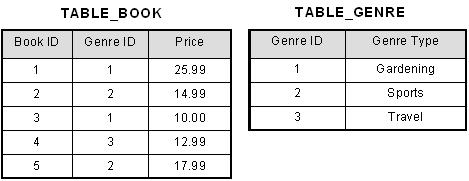
**Consider the following example:**



In the table able, [Book ID] determines [Genre ID], and [Genre ID] determines [Genre Type].

Therefore, [Book ID] determines [Genre Type] via [Genre ID] and we have transitive functional dependency, and this structure does not satisfy third normal form.

To bring this table to third normal form, we split the table into two as follows:



Now all non-key attributes are fully functional dependent only on the primary key. In [TABLE\_BOOK], both [Genre ID] and [Price] are only dependent on [Book ID]. In [TABLE\_GENRE], [Genre Type] is only dependent on [Genre ID].

**Exercises**:

|  |  |  |
| --- | --- | --- |
| **A** | **B** | **C** |
| a1 | b1 | c1 |
| a1 | b1 | c2 |
| a2 | b1 | c1 |
| a2 | b1 | c3 |

**5.1 List all functional dependencies satisfied by the relation of the following Figure:**

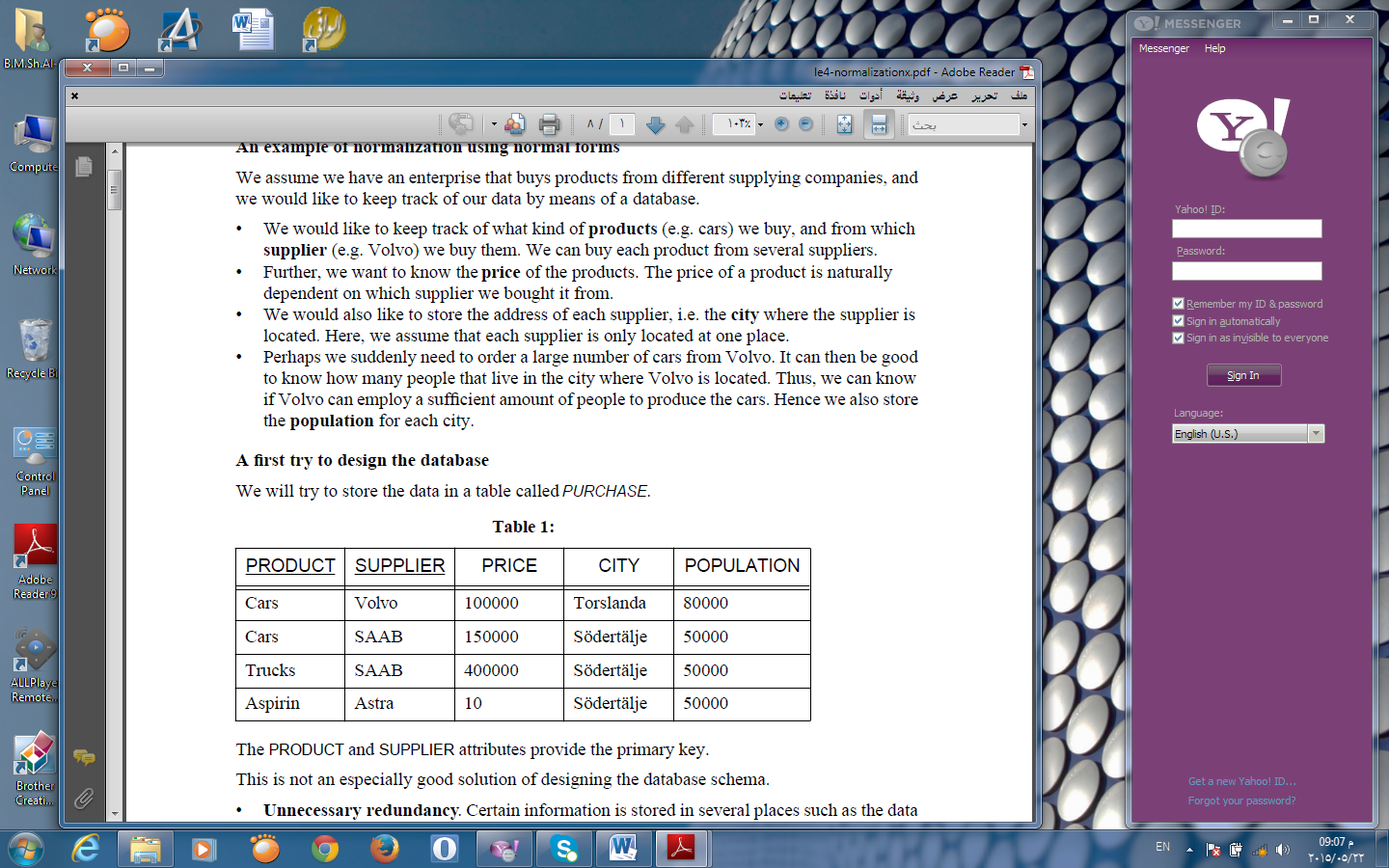
**5.2 List all functional dependencies satisfied by the relation of the following Figure:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Id** | **Name** | **Gender** | **Age** |
| 1 | Orlando | Male | 35 |
| 2 | John | Male | 35 |
| 3 | Jane | Female | 31 |
| 4 | Jane | Female | 30 |

**5.3 Using Normalization convert this table to 1NF,2NF,3NF.**

|  |  |  |
| --- | --- | --- |
| **Student** | **Age** | **Subject** |
| Adam | 15 | Biology, Maths |
| Alex | 14 | Maths |
| Stuart | 17 | Maths |

**5.4 Using Normalization convert this table to 1NF,2NF,3NF.**



**5.5 Using Normalization convert this table to 1NF,2NF,3NF.**

|  |  |  |  |
| --- | --- | --- | --- |
| **Street** | **Zipcode** | **City** | **Length** |
| Rydsvagen | 58248 | Linkoping | 19km |
| Mardtorpsgatan | 58248 | Linkoping | 0.7km |
| Storgatan | 58223 | Linkoping | 1.5km |
| Storgatan | 64631 | Gnesta | 0.014km |

**5.6 Using Normalization convert this table to 1NF,2NF,3NF.**

|  |  |  |  |
| --- | --- | --- | --- |
| **Student** | **Course-id** | **Grade** | **Address** |
| Erik | CIS331 | A | 80Ericsson Av. |
| Sven | CIS331 | B | 12Olafson ST. |
| Inge | CIS331 | C | 192Odin Blvd |
| Hildur | CIS362 | A | 212 Reyjavik ST. |