# Information Systems in Healthcare – Michel Kana, PhD - Homework

Solution by Katarzyna Dunikowska

## 1. Database normalization

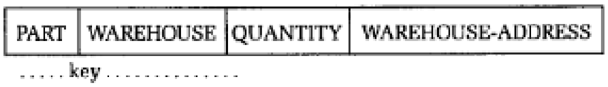
**Describe all normal forms in a relational database provide an example for each form.**

Database normalization is the process of organizing the fields and tables of a relational database to minimize redundancy and dependency. Normalization usually involves dividing large tables into smaller tables and defining relationships between them.

Forms in a relational database:

- First normal form
  -all occurrences of a record type must contain the same number of fields
  -first normal form excludes variable repeating fields and groups

- Second normal form
  Second normal form is violated when a non-key field is a fact about a subset of a key. It is only relevant when the key is composite, i.e., consists of several fields.



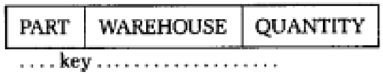
| PART | WAREHOUSE | QUANTITY | WAREHOUSE-ADDRESS |
|---|---|---|---|

. . . . . key . . . . . . . . . . . . . .

The key here consists of the PART and WAREHOUSE fields together, but WAREHOUSE-ADDRESS is a fact about the WAREHOUSE alone. The basic problems with this design are:

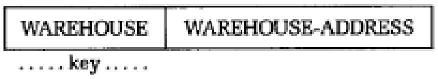
- The warehouse address is repeated in every record that refers to a part stored in that warehouse.
- If the address of the warehouse changes, every record referring to a part stored in that warehouse must be updated.
- Because of the redundancy, the data might become inconsistent, with different records showing different addresses for the same warehouse.
- If at some point in time there are no parts stored in the warehouse, there may be no record in which to keep the warehouse's address.

The record should be decomposed into (replaced by) the two records:



| PART | WAREHOUSE | QUANTITY |
|---|---|---|

. . . . key . . . . . . . . . . . . . . . . . . .

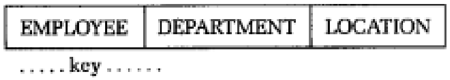


| WAREHOUSE | WAREHOUSE-ADDRESS |
|---|---|

. . . . . key . . . . .

- Third normal form

  Third normal form is violated when a non-key field is a fact about another non-key field, as in:



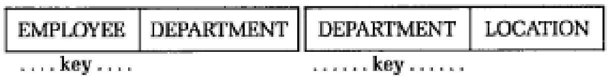
| EMPLOYEE | DEPARTMENT | LOCATION |
|---|---|---|

..... key ......

The EMPLOYEE field is the key. If each department is located in one place, then the LOCATION field is a fact about the DEPARTMENT -- in addition to being a fact about the EMPLOYEE. The problems with this design are the same as those caused by violations of second normal form:

- The department's location is repeated in the record of every employee assigned to that department.
- If the location of the department changes, every such record must be updated.
- Because of the redundancy, the data might become inconsistent, with different records showing different locations for the same department.
- If a department has no employees, there may be no record in which to keep the department's location.

To satisfy third normal form, the record shown above should be decomposed into the two records:



| EMPLOYEE | DEPARTMENT |
|---|---|

.... key ....

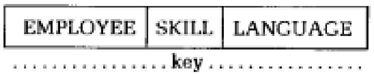
| DEPARTMENT | LOCATION |
|---|---|

...... key ......

- Fourth normal form

  Fourth normal form deal with multi-valued facts. The multi-valued fact may correspond to a many-to-many relationship, or to a many-to-one relationship.

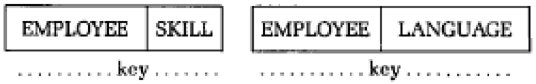
Under fourth normal form, a record type should not contain two or more independent multi-valued facts about an entity.



| EMPLOYEE | SKILL | LANGUAGE |
|---|---|---|

................ key ................

Instead, they should be represented in the two records:



| EMPLOYEE | SKILL |
|---|---|

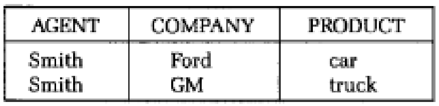
.......... key .......

| EMPLOYEE | LANGUAGE |
|---|---|

........... key ...........

- Fifth normal form
  Fifth normal form deals with cases where information can be reconstructed from smaller pieces of information that can be maintained with less redundancy.

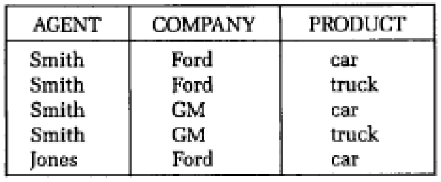
This information could be kept in one record type with three fields:



| AGENT | COMPANY | PRODUCT |
|---|---|---|
| Smith | Ford | car |
| Smith | GM | truck |

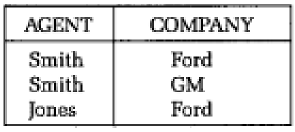
This form is necessary in the general case. For example, although agent Smith sells cars made by Ford and trucks made by GM, he does not sell Ford trucks or GM cars. Thus we need the combination of three fields to know which combinations are valid and which are not.

But suppose that a certain rule was in effect: if an agent sells a certain product, and he represents a company making that product, then he sells that product for that company.

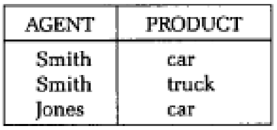


| AGENT | COMPANY | PRODUCT |
|---|---|---|
| Smith | Ford | car |
| Smith | Ford | truck |
| Smith | GM | car |
| Smith | GM | truck |
| Jones | Ford | car |

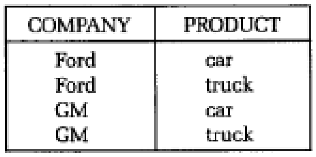
In this case, it turns out that we can reconstruct all the true facts from a normalized form consisting of three separate record types, each containing two fields:



| AGENT | COMPANY |
|---|---|
| Smith | Ford |
| Smith | GM |
| Jones | Ford |



| AGENT | PRODUCT |
|---|---|
| Smith | car |
| Smith | truck |
| Jones | car |



| COMPANY | PRODUCT |
|---|---|
| Ford | car |
| Ford | truck |
| GM | car |
| GM | truck |

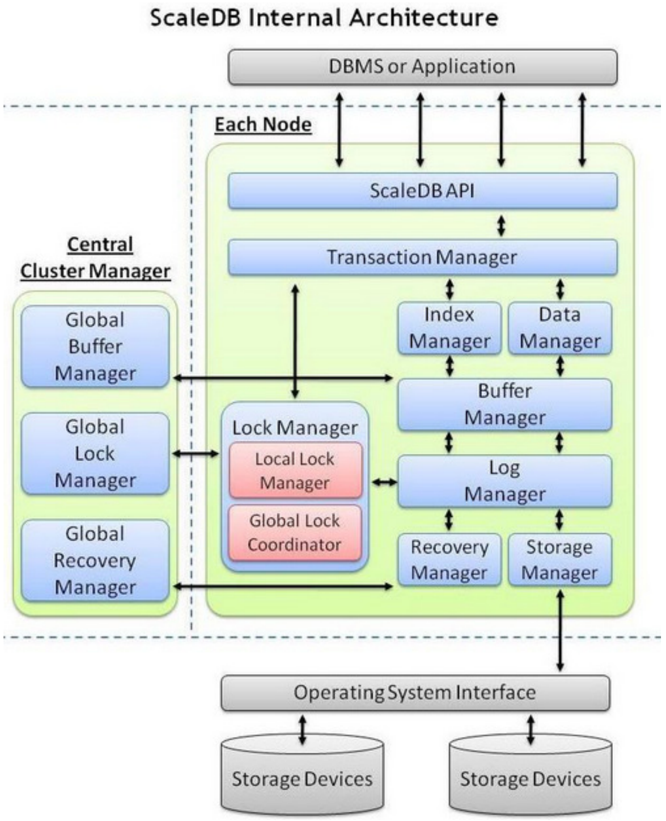
These three record types are in fifth normal form, whereas the corresponding three-field record shown previously is not.

Record type is in fifth normal form when its information content cannot be reconstructed from several smaller record types, i.e., from record types each having fewer fields than the original record.

2. **Describe the major parts of a database management system. Illustrate your answer with MySQL.**

The major parts of a database management system:

- ScaleDB API
- Transaction manager
- Index manager
- Data manager
- Buffer manager
- Log manager
- Recovery manager
- Storage manager



Major parts of MySql [2]

- ScaleDB API – Exposes the ScaleDB functions to DBMS and applications that leverage ScaleDB. ScaleDB publishes an API that any application can use to exploit ScaleDB. In addition, ScaleDB provides connectors for MySQL in open source that you can download and modify. There is also a generic connector that enables your application to directly access ScaleDB without a DBMS.

- Transaction Manager – A transaction manager is the part of an application that is responsible for coordinating transactions across one or more resources. Ensures that the transactions are safe, guaranteeing completion or else rolling-back the uncommitted transaction in case of failure.

- Index Manager – This leverages our Multi-Table Indexing engine to facilitate rapid access to data.

- Data Manager – Coordinates reading and writing of data to files. It provides several classes to wrap MySQL database access connections, execute SQL queries and retrieve query result rows and column fields.

- Buffer Manager – Manages the machines' local cache on each node to improve efficiency and performance. It coordinates with the Global Buffer Manager on the Cluster Manager to ensure that each node is aware of changes made by other nodes.

- Log Manager – Maintains the local log, which is used for rolling back uncommitted transactions and also for failure recovery.

- Lock Manager – Manages local lock management and coordinates with global locking. This insures, among other things, that no two nodes are changing the same information at the same time.

- Recovery Manager – Recovery Manager to ensure that upon failure of the node, it can be recovered gracefully. The Recovery Manager's job is to keep copies of data for retrieval later, in case of loss of data. It also logs commands that modify the data and other significant events inside the database.

- Storage Manager – This coordinates the flushing of the data to disk. The storage manager interfaces with the operating system to write data to the disk efficiently. Because the storage functions reside in a separate subsystem, the MYSQL engine operates at a level of abstraction away from the operating system. The storage manager writes to disk all of the data in the user tables, indexes, and logs as well as the internal system data.

[1] A SIMPLE GUIDE TO FIVE NORMAL FORMS IN RELATIONAL DATABASE THEORY

http://www.iai.uni-bonn.de/III//lehre/vorlesungen/TDWA/WS07/1.pdf

[2] http://www.scaledb.com/images/ScaleDB_Architecture.jpg