Notes on Normalization of Databases

Normalization is due to E. F. Codd -- creator of the relational database management system model. Normal forms are based on anomalies discovered by Codd as he researched the relational DBMS.

Anomalies: unexpected results from an operation

Codd identified three kinds of anomalies. These are sometimes referred to as modification or update anomalies from the three update operations - delete, insert and update. They are:

- **delete**: when deleting a value for an attribute, you inadvertently lose the value for some other attribute
- **insert**: you need to store a value for a particular attribute but can't because you need some other value to include that occurrence (don't have key value)
- **update**: like insert but to change a value, you need to know all instances which may be hard to find.

Codd originally specified three normal forms and showed that any or all of the anomalies could occur in the various normal forms. Higher normal forms have since been described by Codd and others. The process of normalization is that of creating relations which are in a high enough normal form so as to avoid anomalies. Generally, the higher the normal form the better.

The process of taking a relation and splitting it up into multiple relations is called decomposition. The purpose of decomposing relations is to avoid anomalies.

Background terms needed to understand the process are:

- **key**: a unique attribute (or field) which can be used to identify the entire tuple (or record) as unique
- **candidate key**: the set of all attributes (or combinations) which might serve as a key
- **primary key**: key selected by the database administrator as the key we will use for that relation
- **composed (or composite) key**: a key of two or more fields
- **key attribute**: an attribute that is part of the primary key
- **functional dependency**: state in which an attribute (or combination) determines what another attribute’s value is

For example, Social Security Number determines last name so last name is
functionally dependent on SSN but the reverse is not true.

Sometimes you might ask yourself the question "If I know what X is do I automatically know what Y will be?" If so, then $X > Y$ (or simply said: "X determines Y.")

A *functional dependency diagram* (FDD) is often helpful. Put each attribute in a block. Dependencies are shown with arrows. (Don't confuse this with E-R diagrams.) Key fields are usually shown on the left.

![Functional Dependency Diagram Example]

If you have multiple fields in a key (a composed key), each of the key attributes would be enclosed in an outer box with dependency arrows from the appropriate block (hopefully the outside one.)

![Composed Key Example]

**determinant**: an attribute which functionally determines another

**transitive dependency**: a dependency not involving any part of the primary key (here, given that A is the key, the arrow from B to C there would indicate a transitive dependency)

![Transitive Dependency Example]

**full functional dependency**: a situation in which there is no dependency on a subset of the determinant, the determinant is composed of the minimum number of attributes to form the dependency (given that A and B are both required for the key in the next diagram, the dependency arrow to D is acceptable but the arrow from A to C will prove to be problematic as it is a *partial dependency*)

![Full Functional Dependency Example]
Use of Codd's **Standard Relational Notation (SRN)** is also desirable -- especially if we don't want to use sample tables. It consists of the relation name followed by all the attributes for that relation shown in parentheses. In addition the primary key is underlined. For example:

```
STUDENT ( SID, LNAME, FNAME, ADDR, PHONE )
```

or

```
COURSE-REG ( SID, CNUM, INSTR, TERM )
```

Dependency arrows can be used with this representation too but a designer has to be diligent to avoid missing something.

Normalization of relations is solely to avoid anomalies. It is an intuitive process -- an art rather than a science. The process involves putting all attributes in one large relation and examining dependencies based on either sample data or what we know about the enterprise and its business rules (or both.) We **decompose** a single relation into two or more smaller relations.

The **normal forms** are hierarchical in nature -- each next higher form requires that all lower forms exist. Decomposition should cause the resulting relations to achieve a higher normal form. Here we will concern ourselves with four normal forms but there are still higher forms which a relation may require to avoid anomalies.

A relation is in **First Normal Form (1NF)** if and only if all underlying values are **atomic**. That means only one piece of data can be stored within the field (attribute) of a particular record (tuple).

A relation is in **Second Normal Form (2NF)** if and only if it (a) is in 1NF and (b) every non-key attribute is fully dependent on the primary key. Dependency on other attributes may also occur but dependency on a part of the primary key is not sufficient.

A relation is in **Third Normal Form (3NF)** if and only if (a) it is in 2NF and (b) each non-key attribute is not transitively dependent. An attribute not in the key can not be determined by another attribute that is not in the key.

A relation is in **Boyce-Codd Normal Form (BCNF)** if and only if every determinant is a candidate key.
A decomposition is a **good decomposition** only if the resulting relations can be **joined** (an operation similar to matrix multiplication) so that **all** (and only) the original data is retained. In essence, the join should reproduce the very relation we decomposed. A **bad decomposition** indicates improper application of the normalization technique and will result in less (or more!) tuples than in the original relation.

It may well be that a given relation could be decomposed properly in more than one way. Invariably, one way will suit the current needs of the enterprise better than any others. Choose the decomposition that is most useful.

This relation with a transitive dependency from B to C

![Diagram of relation with transitive dependency from B to C](image)

...can be decomposed into these two relations (a **good decomposition** – join on A),

![Diagram of good decomposition 1](image)

...or these two relations (a **good decomposition** – join on B),

![Diagram of good decomposition 2](image)

...or these two relations (a **bad decomposition** – join on C).

![Diagram of bad decomposition](image)