1 Normalization

Normalization is an algorithmic approach to **generating** a set of **relation schemas** from attributes and functional dependencies, that are not redundant (no unnecessary repetitions), and is able to represent all necessary information.

2 Functional Dependency

A **functional dependency** is a a generalization of a key. It simply states that

> in relation schema R, if A and B are **attribute sets**, $A \rightarrow B$ (A determines B) is a functional dependency, meaning that whenever 2 tuples in a relation over R are equal on attributes A, the tuples are equal on attributes B.

An example is 'CPR $\rightarrow$ firstname' - if we know 'CPR', we know 'firstname', but not the other way around. Functional dependencies are decided by DB designers and are the **semantics** of the DB.

A **key constraint** is a functional dependency: 'K $\rightarrow$ R', where the superkey determines all the attributes in a relation. Whenever the K attributes of the 2 tuples are equal, all the attributes are equal.

3 Boyce-Codd Normal Form (BCNF)

Boyce-Codd Normal Form (BCNF) is a DB decomposition requirement that **avoids most redundancy**, has **lossless join**, meaning that joining subrelations must yield the original relation. However, BCNF is **not always dependency preserving**, BCNF decompositions may violate functional dependencies.

We say that a relation schema R is in BCNF if all FDs X $\rightarrow$ Y associated with R are **trivial** (Y $\subseteq$ X), e.g. B $\rightarrow$ B, or X is a superkey for R, e.g. B $\rightarrow$ R. In plain terms, this means that the only allowed FDs for a relation schema in BCNF are superkeys.

3.1 Attribute Set Closure

The **attribute set closure** X^+, which is **all the attributes that X determine**, can be computed by repeatedly applying **Armstrongs axioms**

reflexivity	$Y \subseteq X \implies X \rightarrow Y$ (find trivial FDs)
augmentation	$X \rightarrow Y \implies AX \rightarrow AY$ (implicit $\cup$)
transitivity	$X \rightarrow Y$ and $Y \rightarrow Z \implies X \rightarrow Z$

and derivations thereof. The attribute set closure is **sound**, meaning that it only generates correct FDs and it is **complete**, meaning that it generates *all* FDs.

3.2 Decomposition

A schema in BCNF can be created algorithmically by iteratively decomposing the original schema.

If there is a FD $X \rightarrow Y$ associated with a relation schema R that violates BCNF, the schema can be decomposed into subrelations XY and R-Y, e.g.

$$R = name, CPR, pet-license-ID, pet-name$$

FDs:

$CPR \rightarrow name, pet-license-ID$ and $pet-license-ID \rightarrow pet-name$. The latter violates BCNF, because '$pet-license-ID^{+} \neq R$

Decomposition:

P	=	{ pet-license-ID, pet-name } (pet)
O	=	{ name, CPR, pet-license-ID } (owner)

3.3 Canonical Cover

A **canonical cover** for a set of FDs is an equivalent set of FDs **without redundance**, meaning that there may not be extraneous attributes, and the left sides of FDs are unique. The canonical cover can be computed algorithmically with a **fixpoint algorithm**, that apply rules iteratively, until the result set does not change.

4 Third Normal Form (3NF)

All schemas in BCNF are in **Third Normal Form** (BCNF is stricter), but in contrast to BCNF it is always possible to find a 3NF decomposition that is **dependency preserving** (FDs hold), has **lossless join** (joining yields original). However, 3NF decompositions may be **redundant**.

We say that a relation schema R is in 3NF if:

> all FDs and their derivations (F^+) X $\rightarrow$ Y associated with R are either **trivial** or X is a superkey for R, or **each attribute in Y-X is contained in a candidate key for R.**

The **decomposition algorithm** for 3NF normalization works as follows:

```
for each FD X->Y in F+
    if no schema contains XY
        create new schema X,Y
if no schema contains candidate key for R
    create new schema with candidate key for R
```