This lecture describes Boyce-Codd normal form and 3rd normal form.

Normal forms

There is a substantial theory of various types of normal forms; essentially these are guarantees that if a relation schema does not have various types of functional dependency, then certain types of problem with redundancy cannot occur.

Associated with each type of normal form is a theory of decomposition which explains under what circumstances a schema can be decomposed in such a way that the resulting relations are in the desired normal form.

Normal form hierarchy

The normal forms based on FDs alone form a hierarchy with increasingly strict limitations on the allowed functional dependencies as follows:

- First normal form (1NF)
- Second normal form (2NF)
- Third normal form (3NF)
- Boyce-Codd normal form (BCNF)

Thus any schema that is in BCNF is automatically in 3NF, 2NF and 1NF.
**BCNF**

A relational schema is in *Boyce-Codd normal form* if for every functional dependency $X \rightarrow A$ (where $X$ is a subset of the attributes and $A$ is a single attribute) either

- $A \in X$ (that is, $X \rightarrow A$ is a trivial FD), or
- $X$ is a superkey.

In other words, the only functional dependencies are either the trivial ones (which always hold) or ones based on the keys of the relation. If a relational schema is in BCNF, then there is *no redundancy* within the relations.

**No redundancy in BCNF**

Loosely speaking, a relational schema in BCNF is already in its “leanest” possible form — each attribute is determined by the key(s) alone so nothing that is stored can be deduced from a smaller amount of information.

The student number / name / unit code / mark relation $SNUM$ from last lecture is *not* in BCNF because there is a functional dependency $S \rightarrow N$ but $S$ is *not* a superkey.

**BCNF decomposition**

Suppose a relation $R$ is *not* in BCNF. Then there must be some functional dependency $X \rightarrow A$ where $X$ is not a superkey.

In this case, the relation can be *decomposed* into the two relations

$$R_1 = R - A \quad R_2 = XA$$

where $R - A$ simply means dropping the attribute $A$ from the schema, and $XA$ is the relation whose attributes are those in $X$ together with $A$.

**BCNF decomposition cont.**

If either $R_1$ or $R_2$ is not in BCNF then the process can be continued, by decomposing them in the same fashion.

By continually decomposing any relation not in BCNF into smaller relations, we must eventually end up with a collection of relations that are in BCNF.

Therefore any initial schema can be decomposed into BCNF.
Example from text

Consider a relation schema \(CSJDPQV\) that represents contracts for suppliers to supply certain parts to certain projects.
- \(C\) is the Contract ID
- \(S\) is the Supplier ID
- \(J\) is the Project ID
- \(D\) is the Department ID
- \(P\) is the Part ID
- \(Q\) is the Quantity
- \(V\) is the Value

Example cont.

Now we add some constraints that can be expressed as FDs.
- The contract ID is a key
  \(C \rightarrow CSJDPQV\)
- A project purchases each type of part in a single contract
  \(JP \rightarrow C\)
- A department purchases at most one part from each supplier
  \(SD \rightarrow P\)

Is this schema in BCNF? The first FD \(JP \rightarrow C\) does not violate BCNF because \(JP\) is a key. However \(SD\) is not a key, and so \(SD \rightarrow P\) violates BCNF.

Decompose the relation

Guided by the FD \(SD \rightarrow P\) we get the decomposition

\[R_1 = CSJDPQV \quad R_2 = SDP\]

where both schemas are in BCNF.

(Note that this is not obvious — it requires using Armstrong’s axioms to calculate the closure of all the given FDs and checking that none of them violate the BCNF conditions.)

At first sight it would seem that we have resolved the problems with redundancy — simply decompose into BCNF at all times! However this is unfortunately not true.

A subtle problem

There is a subtle problem with the decomposition of \(R\) into

\[R_1 = CSJDPQV \quad R_2 = SDP\]

which is related to maintaining the key constraint \(JP \rightarrow C\).

Whenever a new tuple is added to the original relation \(R\) it is simple to enforce the key constraint \(JP \rightarrow C\), just by checking the tuples in that one table.

However, in the decomposed relation, entering the data about a new contract involves updating two relations with the \(J\) attribute going into \(R_1\) and the \(P\) attribute into \(R_2\).

Therefore to enforce the key constraint would require the DBMS to perform a join on \(R_1\) and \(R_2\) just to check that a particular \(JP\) pair does not appear twice.
Dependency-preserving decomposition

It would be desirable if any decomposition that we used was not only lossless-join, but also \textit{dependency preserving}.

Intuitively this means that any functional dependency in the original relation can be enforced by examining \textit{just one} of the relations in the decomposition.

If $F$ is a set of functional dependencies on a relation schema $R$, then a proposed decomposition of $R$ into relations with attributes $X$ and $Y$ is dependency preserving if and only if

$$(F_X \cup F_Y)^+ = F^+$$

where $F_X$ is the set of FDs involving only attributes from $X$, and similarly for $F_Y$.

Example cont.

The idea underlying a dependency-preserving decomposition is that the FDs on the original relation can be enforced just by enforcing $F_X$ on $R_1$ or $F_Y$ on $R_2$.

In the contracts example, the decomposition involves attribute sets

$X = CSJDQV \quad Y = SDP$

The problem occurs because the dependency $JP \rightarrow C$ is not in either $F_X$ nor in $F_Y$.

It is \textit{conceivable} that $JP \rightarrow C$ might be in $(F_X \cup F_Y)^+$ through some chain of inferences using Armstrong’s axioms, but it is not true in this case.

BCNF is too strong

In fact, there is \textit{no} dependency-preserving decomposition of the contracts relation into BCNF — essentially BCNF is “too restrictive” a condition.

This motivates the definition of \textit{3rd normal form} which is somewhat technical — essentially it relaxes the conditions for BCNF \textit{just enough} to guarantee that there will be a dependency-preserving decomposition of any relation schema into 3NF.

Thus we have a trade-off where allowing a controlled amount of redundancy gives us more freedom in decompositions.

3NF

A relational schema is in \textit{3NF} if for every functional dependency $X \rightarrow A$ (where $X$ is a subset of the attributes and $A$ is a single attribute) either

- $A \in X$ (that is, $X \rightarrow A$ is a trivial FD), or
- $X$ is a superkey, or
- $A$ is part of some key for $R$.

Thus the definition of 3NF permits a few additional FDs involving key attributes that are prohibited by BCNF.
The contracts example

The contracts example was not in BCNF because of the FD

\[ SD \rightarrow P \]

However as \( P \) is part of a key \((JP)\), this FD does not violate the conditions for 3NF. Therefore neither of the two given FDs violate the conditions for 3NF.

Checking all the other (implied) FDs shows that contracts is already in 3NF and thus should not be decomposed further.

Properties of 3NF

The most important property of 3NF is the result that there is always a lossless-join and dependency-preserving decomposition of any relational schema into 3NF.

While not conceptually difficult, the algorithm to decompose a relation schema into 3NF is quite fiddly and much more complicated than the simple algorithm for decomposition into BCNF.

Other normal forms

First normal form (1NF) is just the requirement that attribute values be atomic — that is, individual values rather than lists or sets — but places no restrictions on functional dependencies and hence redundancy.

Second normal form (2NF) is an intermediate normal form that completes the spectrum, but is not used in practice.

In addition there are 4th and 5th normal forms, which address redundancy problems that cannot be expressed using just FDs.

Practical Use of Normalization

In practice, the DBA should be guided by the theory of normalization, but not slavishly follow it.

When the ER diagram is converted into tables, any functional dependencies should be identified and a decision made as to whether the tables should be normalized or not.

Occasionally it will even be appropriate to de-normalize data if the most frequently occurring queries involve expensive joins of the normalized tables.