Introduction to Databases, Fall 2003

IT University of Copenhagen

Lecture 4: Normalization

September 16, 2003

Lecturer: Rasmus Pagh

Today's lecture

- What you should remember from previously.
- Anomalies in relations.
- Decomposing relations.
- Functional dependencies.
- Boyce-Codd normal form (BCNF).
- 3rd normal form.
- Attribute value redundancy.

What you should remember from previously.

In this lecture I will assume that you remember:

- Key concepts of the relational data model:
  - Relation
  - Attributes
  - Relation schema
  - Relation instance
- Key concepts in SQL
  - Projection
  - Join
- Key concepts in E/R modeling:
  - Entity set
  - Relationship

Next: Anomalies in relations

Redundancy in a relation

Redundant (i.e., "unnecessary") information occurs in a relation if the same fact is repeated in several different tuples.

`[Figure 3.21 shown on slide]`

One obvious problem with redundant information is that we use more memory than is necessary. Redundancy is an example of an **anomaly** of the relation schema.

Other kinds of anomalies

The other principal kinds of unwanted anomalies are:

- **Update anomalies**. Occur when it is possible to change a fact in one tuple but leave the same fact unchanged in another. (E.g., the length of Star Wars in the `Movies` relation.)
- **Deletion anomalies**. Occur when deleting a tuple (recording some fact) may delete another fact from the database. (E.g., information on a movie in the `Movies` relation.)

`[Figure 3.21 shown on slide]`

Ideally, we would like relation schemas that do not allow anomalies. **Normalization** is a process that can often be used to arrive at such schemas.

Next: Decomposing relations

Decomposing relations

The anomalies in the example we saw can be eliminated by splitting (or **decomposing**) the relation schema

```
Movies(title, year, length, filmType, studioName, starName)
```

into two relation schemas

```
Movies1(title, year, length, filmType, studioName)
Movies2(title, year, starName)
```

[Figure 3.22 and 3.23 shown on slide]

Decomposition and projection

The relation instances for `Movies1` and `Movies2` were found by **projection** of `Movies` onto their attributes. In SQL, `Movies2` could be computed as follows:

```
SELECT title, year, starName
FROM Movies
```

This is a *general rule* when decomposing: The decomposed relation instances are found by projection of the original relation instance.

Recombining relations

We would like the decomposed relations to contain the same information as the original relation. In particular, we should be able to recombine them to recover the original relation.

Recombining can be done by **joining** the relations on attributes of the same name (this is called a **natural join**). `[Figure 3.28 shown on slide]`

Example: In SQL we can compute `Movies` as follows:

```
SELECT *
FROM Stars1, Stars2
WHERE Stars1.title = Stars2.title AND
Stars1.year = Stars2.year
```

Problem session (5 minutes)

Consider these two attempts at decomposing `Movies` into relations

`MoviesA(length, filmType, studioName)`

`MoviesB(title, year, starName)`

`MoviesX(title, year, length, filmType)`

`MoviesY(title, year, starName, studioName)`

What are the problems with these attempts?

Keys of a relation

A **key** for a relation is a set of its attributes that satisfy:

- **Uniqueness.** The values of the attributes uniquely identify a tuple.
- **Minimality.** No proper subset of the attributes has the uniqueness property.

If uniqueness is satisfied (but not necessarily minimality) the attributes are said to form a **superkey**.

Examples: `[Figure 3.21 shown on slide]`

- {`Title, year, starName`} is a key for the `Movies` relation.
- {`Title, year, starName, length`} is a superkey, but not a key, for the `Movies` relation.
- {`Title, year`} is not a superkey (or key) for the `Movies` relation.

Key terminology

Confusingly, what we call a superkey in the context of relations corresponds to what we called a key in the context of E/R models.

According to the book, the key/superkey terminology is *not* well-established.

Keys consisting of more than one attribute are sometimes called **composite**.

Recombining requires a superkey

Suppose we decompose a relation R into two relations R_1 and R_2 with common attributes $B_1, B_2, \ldots, B_m$. What is required to be able to recover R?

[Figure 3.28 shown on slide]

Criterion for being able to recombine:

$\{B_1, B_2, \ldots, B_m\}$ must be a superkey for R_1 or R_2.

Next: Functional dependencies

Functional dependencies cause anomalies

When values of attribute B can be derived from the attributes $A_1, \ldots, A_n$ we say that B is **functionally dependent** on $A_1, \ldots, A_n$. This is written as follows:

$$A_1 A_2 \ldots A_n \quad \rightarrow \quad B$$

`[Figure 3.16 shown on slide]`

Example: `Movies` has the functional dependency (FD)

$$title \quad year \quad \rightarrow \quad length$$

but *not* the FD

$$title \quad year \quad \rightarrow \quad starName$$

This is in fact *the very reason* for the anomalies we saw!

Unavoidable functional dependency

Functional dependency on a superkey

Whenever we see the attribute values of some (super)key $\{A_1, \ldots, A_n\}$, we can uniquely identify the tuple from which the values come.

In particular, we can determine the value of any other attribute B in the relation, so we unavoidably have the FD

$$A_1 A_2 \ldots A_n \quad \rightarrow \quad B$$

Trivial functional dependency

Also, we can always determine the value of attribute A_i from the value of attribute A_i. So we unavoidably have the FD

$$A_1 A_2 \ldots A_n \quad \rightarrow \quad A_i$$

Problem session (10 minutes)

Consider a relation containing an inventory record:

`Inventory(part, warehouse, quantity, warehouse-address)`

- What are the keys of the relation?
- What are the avoidable functional dependencies?
- Can you suggest a way of decomposing the relation to eliminate the avoidable functional dependencies?

Next: Boyce-Codd normal form (BCNF)

Boyce-Codd normal form (BCNF)

A **normal form** is a criterion on a relation schema.

> A relation is in **Boyce-Codd normal form** (BCNF) if there are only unavoidable functional dependencies among its attributes.

Example: `Movies` has the functional dependency

$$title \quad year \quad \rightarrow \quad length$$

which is *not* unavoidable because it is nontrivial and {*title*, *year*} is not a (super)key. Thus, `Movies` is not in BCNF.

Examples of relations in BCNF

The relations of our decomposition:

`Movies1(title, year, length, filmType, studioName)`

`Movies2(title, year, starName)`

are in BCNF. The only nontrivial nonreducible FDs are

$$\texttt{Movies1:} \quad title \quad year \quad \rightarrow \quad length$$

$$\texttt{Movies1:} \quad title \quad year \quad \rightarrow \quad filmType$$

$$\texttt{Movies1:} \quad title \quad year \quad \rightarrow \quad studioName$$

$$\texttt{Movies2:} \quad title \quad year \quad \rightarrow \quad starName$$

and they are unavoidable since $\{$*title*, *year*$\}$ is a key for both relations.

Writing functional dependencies

Reducing FDs: Whenever we can reduce the number of attributes when writing an FD we do so. For example, `Movies1` has the FDs

$$title \quad year \quad filmType \quad \rightarrow \quad length$$

$$title \quad year \quad studioName \quad \rightarrow \quad length$$

which can both be reduced to

$$title \quad year \quad \rightarrow \quad length$$

Combining FDs: Whenever several FDs have the same left hand side we combine them. For example, the three FDs we saw for `Movies` can be written succinctly as:

$$title \quad year \quad \rightarrow \quad length \quad filmType \quad studioName$$

Decomposing a relation into BCNF

Suppose we have a relation R which is not in BCNF. Then there is an FD

$$A_1 A_2 \ldots A_n \quad \rightarrow \quad B_1 B_2 \ldots B_m$$

which is not unavoidable.

To eliminate the FD we split R into two relations: `[Figure 3.24]`

- One with all attributes of R except $B_1, B_2, \ldots, B_m$.
- One with attributes $A_1, A_2, \ldots, A_n, B_1, B_2, \ldots, B_m$.

If any of the resulting relations is not in BCNF, the process is repeated.

Note: $A_1, A_2, \ldots, A_n$ is a superkey for the second relation – therefore we can recover R as the natural join of the two relations.

BCNF decomposition example

Recall the relation `Movies` with schema

`Movies(title, year, length, filmType, studioName, starName)`

It has the following FD, which is not unavoidable:

$$title \quad year \quad \rightarrow \quad length \quad filmType \quad studioName$$

Thus the decomposition yields the following relations (both in BCNF):

`Movies1(title, year, length, filmType, studioName)`

`Movies2(title, year, starName)`

Problem session (5 minutes)

Consider the following relation with information on movie studios:

`MovieStudio(title, year, length, filmType, studioName, studioAddr)`

Argue that the relation is not in BCNF, and find a decomposition into BCNF.

Reasoning about FDs

For relations with many attributes, it may be difficult to discover all functional dependencies. A tool that can be used to find all FDs that follow from the FDs already known are the following rules:

- **The augmentation rule.** If $A_1 A_2 \ldots A_n \quad \rightarrow \quad B_1 B_2 \ldots B_m$ then for any set of attributes $C_1, \ldots, C_k$:

$$A_1 A_2 \ldots A_n C_1 \ldots C_k \quad \rightarrow \quad B_1 B_2 \ldots B_m C_1 \ldots C_k$$

- **The transitive rule.** If $A_1 A_2 \ldots A_n \quad \rightarrow \quad B_1 B_2 \ldots B_m$ and $B_1 B_2 \ldots B_m \quad \rightarrow \quad C_1, \ldots, C_k$ then

$$A_1 A_2 \ldots A_n \quad \rightarrow \quad C_1 C_2 \ldots C_k$$

Example of reasoning about FDs

Suppose that we add to `Movies` the attribute `studioAddr`:

`Movies(title, year, length, filmType, studioName, starName)`

This relation would have the FDs:

$$title \quad year \quad \rightarrow \quad studioName$$

$$studioName \quad \rightarrow \quad studioAddr$$

Using the transitive rule we derive the new FD:

$$title \quad year \quad \rightarrow \quad studioAddr$$

Next: 3rd normal form

Interrelation dependencies

Consider the relation with schema `Bookings(title,theater,city)`

Under certain assumptions, it has the FD $theater \rightarrow city$, but $theater$ is not a superkey. The BCNF decomposition yields relation schemas `Bookings1(theater,city)` and `Bookings2(theater,title)`.

These schemas and their FDs allow, e.g., the relation instances:

theater	*city*
Guild	Menlo Park
Park	Menlo Park

theater	*title*
Guild	The net
Park	The net

which violate the presumed FD $title\ city \rightarrow theater$.

Thus, there are implicit dependencies between values in different relations. *We cannot check FDs separately in each relation.*

Splitting keys

As we just saw, decomposition can result in a relational database schema where a functional dependency "disappeared".

The problem in the previous example arose because we decomposed according to the FD $theater \rightarrow city$, where $city$ is part of a key for the `Bookings` relation. Thus we ended up splitting the key $\{city, theater\}$.

This problem of FDs that are not preserved **never** arises if we do not decompose in this case.

Third normal form

We have motivated the following normal form which never splits a keys of the original relation:

> A relation is in **3rd normal form** (3NF) if any functional dependency among its attributes is either unavoidable, or has a member of some key on the right hand side.

In words: A relation is in 3NF if there are no unavoidable functional dependencies among non-key attributes.

When to stop decomposition at 3NF?

Whether it is a good idea to stop decomposition when third normal form is reached depends on the specific scenario.

- Mostly, 3NF and BCNF coincide, so there is nothing to consider.
- If not, the redundancy in tuples in 3NF should be weighed against the fact that some FD is difficult to check in BCNF.

Next: Attribute value redundancy (not in book)

Attribute value redundancy

A kind of redundancy, which is different from functional dependence, is redundancy in attribute values.

Example: The string `Star Wars` was repeated many times to designate the movie. Longer strings would make the problem even more obvious.

This kind of redundancy could also cause update anomalies.

Example: Suppose the working title `Lucky Luke Skywalker` had to be changed in the whole database to `Star Wars`.

Reducing attribute value redundancy

A way of reducing this redundancy is to use (or introduce) a short key value for each string, and put strings in a separate relation like

```
MovieNames(key,name)
```

Whether this is a good idea depends on the number of occurrences and other factors such as the need for efficiency.

Most important points in this lecture

As a minimum, you should after this week:

- Understand the significance of normalization.
- Be able to determine whether a relation is in Boyce Codd normal form or 3rd normal form.
- Be able to split a relation in several relations to achieve any of the normal forms.
- Know how to recombine normalized relations in SQL.

Next time

Next week we finish the study of normalization:

- 4th normal form
- Some observations on normalization.

Then we will go through a number of cases of database design, including

- E/R design
- Conversion to relation schemas
- Normalization