
Introduction to Database Design, fall 2011
IT University of Copenhagen

Normalization

Rasmus Pagh

Based on KBL sections 6.1-6.8 (except p. 203–207m), 6.9 (until “Multivalued dependencies”), 6.11, and 6.12.

— Today's lecture —

- Anomalies in relations.
- Functional dependencies.
- Normal forms: Boyce-Codd normal form, 3rd normal form, and a little bit on higher normal forms.

— 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 —

Selection by example:

```
SELECT DISTINCT title, year
FROM Movie
```

Join by example:

```
SELECT *
FROM Movie, Director
WHERE Movie.director = Director.id
```

— Redundancy in a relation —

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

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.

Example: Instance of

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

where the length of a movie is repeated several times.

— 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.)
- **Insertion anomalies.** The “dual” of deletion anomalies.

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.

— Anomalies and good design —

As we will see, anomalies are a sign that we tried to encode several, unrelated types of facts into a single relation.

Thus, normalization can help *improving the design* such that the unrelatedness of these facts are captured by the database schema (and E-R diagram).

— 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)
```


— 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 DISTINCT 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 need the decomposed relations to contain the same information as the original relation. In particular, we must be able to recombine them to recover the original relation. (*The lossless join property.*)

If decomposition is done properly, recombining can be done by **joining** the relations on attributes of the same name (this is called a **natural join**).

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

```
SELECT *
FROM Movies1, Movies2
WHERE Movies1.title = Movies2.title AND
Movies1.year = Movies2.year
```

Candidate keys of a relation

A **candidate 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:

- {Title, year, starName} is a candidate key for the Movies relation.
- {Title, year, starName, length} is a superkey, but not a candidate key, for the Movies relation.
- {Title, year} does not satisfy uniqueness for Movies.

— Candidate keys vs primary keys —

Note that the concept of a candidate key is defined with respect to the relation (schema), and **not** with respect to any particular *instance* of the relation.

The primary key of a relation in a DBMS should be a candidate key, but there could be several candidate keys to choose from. When talking about normalization, it is irrelevant which key is chosen as primary key.

Next: Functional dependencies and normal forms

— Functional dependencies cause anomalies —

When values of attribute B can be derived (by an all-knowing entity) from the attributes A_1, \dots, A_n we say that B is **functionally dependent** on A_1, \dots, A_n . This is written as follows:

$$A_1 A_2 \dots A_n \rightarrow B$$

Example: Movies has the functional dependency (FD)

$$title \ year \rightarrow \ length$$

but *not* the FD

$$title \ year \rightarrow \ starName$$

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

— Unavoidable functional dependency —

Functional dependency on a candidate key

If the attributes of some candidate key is included in $\{A_1, \dots, A_n\}$, these attributes 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

$$A_1 A_2 \dots A_n \rightarrow B$$

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

$$A_1 A_2 \dots A_n \rightarrow A_i$$

— First and second normal form —

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

A relation schema is in **first normal form** (1NF) if it specifies that all tuples contain the same number of **atomic** attribute values.

In pure relational DBMSs it is only possible to have 1NF relations schemas.

A relation schema is in **second normal form** (2NF) if it is in 1NF and there are no functional dependencies with a proper subset of a candidate key on the left hand side.

Example: Movies has the functional dependency

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

where $\{title, year\}$ is a subset of the candidate key $\{title, year, starname\}$.

Thus, Movies is not in 2NF.

— Boyce-Codd normal form (BCNF) —

A relation schema 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 superkey. 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 (all in Movies1):

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

and they are unavoidable since $\{title, year\}$ is a candidate key for Movies1.

— 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

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

which can both be reduced to

$$title \ year \rightarrow length$$

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:

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

— Decomposing a relation into BCNF —

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

$$A_1A_2 \dots A_n \rightarrow B_1B_2 \dots B_m$$

which is not unavoidable.

To eliminate the FD we split R into two relations:

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

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

Note: A_1, A_2, \dots, 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:

$$\textit{title} \ \textit{year} \ \rightarrow \ \textit{length} \ \textit{filmType} \ \textit{studioName}$$

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

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

`Movies2(title, year, starName)`

— BCNF decomposition example 2 —

Movies(title, year, length, filmType, studioName, starName)
could also have been decomposed by using the following FDs, one by one:

$$\textit{title year} \rightarrow \textit{length}$$
$$\textit{title year} \rightarrow \textit{filmType}$$
$$\textit{title year} \rightarrow \textit{studioName}$$

Then the decomposition yields the following relations (all in BCNF):

Movies1(title, year, length), Movies2(title, year, filmType),

Movies3(title, year, studioName), Movies4(title, year, starName)

To avoid too many relations, as in this example, you should generally use *maximal* FDs where it is not possible to add attributes on the right side.

— Problem session —

Consider a relation containing an inventory record:

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

- What are the candidate keys of the relation?
- What are the avoidable functional dependencies?
- Perform a decomposition into BCNF.

Next: 3rd normal form and dependency preservation

— Interrelation dependencies —

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

$theater \rightarrow city$ (and $theater$ is not a candidate key).

$title\ city \rightarrow theater$.

BCNF dec.: `Bookings1(theater, city)` `Bookings2(theater, title)`.

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

<i>theater</i>	<i>city</i>	<i>theater</i>	<i>title</i>
Guild	Menlo Park	Guild	The net
Park	Menlo Park	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 to see such a dependency.

— Splitting candidate keys —

As we just saw, decomposition can result in a relational database schema where a functional dependency “disappeared”. (The decomposition is not *dependency preserving*.)

The problem in the previous example arose because we decomposed according to the FD $theater \rightarrow city$, where $city$ is part of a candidate key for the Bookings relation. Thus we ended up splitting the candidate 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 that never splits a candidate keys of the original relation:

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

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

— Third normal form, another example —

HasAccount (Account-number, ClientId, OfficeId)

Functional dependencies:

ClientId OfficeId \rightarrow AccountNumber

AccountNumber \rightarrow OfficeId

Is in 3rd normal form, but not BCNF (why?). Can be decomposed losslessly:

AcctOffice (Account-number, OfficeId)

AcctClient (Account-number, ClientId)

— 3NF by schema synthesis —

A 3NF schema can also be found “directly” by *schema synthesis*.

Given a relation R and a set \mathcal{F} of FDs.

- Find a minimal set of FDs that *cover* (imply) all FDs in \mathcal{F} .
- Create a relation R_i for each FD (with the attributes of the FD).
- If needed, add a relation R_0 containing a *key* of R .

We will not go into *why* this procedure produces a 3NF schema.

— 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/maintain in BCNF.

Example:

In the Bookings example, we might want to make the DBMS check that to every title and city, there is at most one theater. For the BCNF decomposed relations, this would involve a query on Bookings1 for every change of Bookings2, and vice versa.

Next: Higher normal forms.

— Redundancy in BCNF relations —

Boyce-Codd normal form eliminates redundancy in each tuple, but may leave redundancy among tuples in a relation.

This typically happens if two many-many relationships (or in general: a combination of two types of facts) are represented in one relation.

Example:

<i>name</i>	<i>street</i>	<i>city</i>	<i>title</i>	<i>year</i>
C. Fisher	123 Maple St.	Hollywood	Star Wars	1977
C. Fisher	123 Maple St.	Hollywood	Empire Strikes Back	1980
C. Fisher	123 Maple St.	Hollywood	Return of the Jedi	1983
C. Fisher	5 Locust Ln.	Malibu	Star Wars	1977
C. Fisher	5 Locust Ln.	Malibu	Empire Strikes Back	1980
C. Fisher	5 Locust Ln.	Malibu	Return of the Jedi	1983

— Curing it with NULL values? —

Then what about something like one of these:

<i>name</i>	<i>street</i>	<i>city</i>	<i>title</i>	<i>year</i>
C. Fisher	123 Maple St.	Hollywood	NULL	NULL
C. Fisher	5 Locust Ln.	Malibu	NULL	NULL
C. Fisher	NULL	NULL	Star Wars	1977
C. Fisher	NULL	NULL	Empire Strikes Back	1980
C. Fisher	NULL	NULL	Return of the Jedi	1983

<i>name</i>	<i>street</i>	<i>city</i>	<i>title</i>	<i>year</i>
C. Fisher	123 Maple St.	Hollywood	Star Wars	1977
C. Fisher	5 Locust Ln.	Malibu	Empire Strikes Back	1980
C. Fisher	NULL	NULL	Return of the Jedi	1983

Decomposition

A better idea is to eliminate redundancy by decomposing StarsIn as follows:

<i>name</i>	<i>street</i>	<i>city</i>
C. Fisher	123 Maple St.	Hollywood
C. Fisher	5 Locust Ln.	Malibu

<i>name</i>	<i>title</i>	<i>year</i>
C. Fisher	Star Wars	1977
C. Fisher	Empire Strikes Back	1980
C. Fisher	Return of the Jedi	1983

— 4th and 5th normal form —

Roughly speaking, a relation is in 4th normal form if it cannot be meaningfully decomposed into two relations.

Example: StarsIn is not in 4th normal form, since it can be decomposed, as we have just shown.

Roughly speaking, a relation is in 5th normal form if it cannot be meaningfully decomposed into *some number* of relations.

— Relationship among normal forms —

Inclusion among normal forms:

Any relation in 5NF is also in 4NF.

Any relation in 4NF is also in BCNF.

Any relation in BCNF is also in 3NF.

Properties of normal forms:

A “higher” normal form has less redundancy, but may not preserve functional and multivalued dependencies.

— How should normal forms be used? —

The various normal forms may be seen as *guidelines* for designing a good relation schema. Some complexities that arise are:

- Should we split candidate keys, introducing dependencies between relations (in 3NF we do not)?
- What is the effect of decomposition on performance? (More on this later.)

If one chooses a relation schema with avoidable functional dependencies, one should be aware of the extra effort that is needed to avoid anomalies.

— Most important points in this lecture —

Your goals related to today's lecture should be to:

- 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.