Today’s lecture

- Anomalies in relations.
- Functional dependencies.
- 1st, 2nd normal form – skipped in lecture.
- Boyce-Codd normal form (BCNF).
- 3rd normal form.
- 4th normal form.
- Attribute value redundancy.
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 (type)
  - Relationship (type)

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.

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.)

- **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.
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).

---

**Anomalies and good design**

---

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)

into two relation schemas

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

---

Movie(title, year, length, filmType, studioName)
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 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.

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.
**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:

$$A_1A_2\ldots A_n \rightarrow B$$

**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 \( \{A_1, \ldots, A_n\} \), 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

\[
A_1 A_2 \ldots A_n \rightarrow B
\]

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

\[
A_1 A_2 \ldots A_n \rightarrow A_i
\]

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

\[
title \ year \ \rightarrow \ length
\]

which is *not* unavoidable because it is nontrivial and \{\textit{title}, \textit{year}\} is not a superkey. Thus, Movies is not in BCNF.

---

--- Examples of relations in BCNF ---

The relations of our decomposition:

Movies1(\textit{title}, \textit{year}, \textit{length}, \textit{filmType}, \textit{studioName})

Movies2(\textit{title}, \textit{year}, \textit{starName})

are in BCNF. The only nontrivial nonreducible FDs are (all in Movies1):

\[
title \ year \ \rightarrow \ length
\]

\[
title \ year \ \rightarrow \ \textit{filmType}
\]

\[
title \ year \ \rightarrow \ \textit{studioName}
\]

and they are unavoidable since \{\textit{title}, \textit{year}\} is a candidate 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

\[ \text{title year filmType} \rightarrow \text{length} \]
\[ \text{title year studioName} \rightarrow \text{length} \]

which can both be reduced to

\[ \text{title year} \rightarrow \text{length} \]

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:

\[ \text{title year} \rightarrow \text{length filmType studioName} \]

Decomposing a relation into BCNF

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

\[ A_1 A_2 \ldots A_n \rightarrow B_1 B_2 \ldots B_m \]

which is not unavoidable.

To eliminate the FD we split R into two relations:

- One with all attributes of R except \( B_1, B_2, \ldots, B_m \).
- 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.
Recall the relation Movies with schema
Movies(title, year, length, filmType, studioName, starName)

It has the following FD, which is not unavoidable:

\[ \text{title, year} \rightarrow \text{length, filmType, studioName} \]

Thus the decomposition yields the following relations (both in BCNF):
Movies1(title, year, length, filmType, studioName)
Movies2(title, year, starName)

---

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

Under certain assumptions, it has the FD \( \text{theater} \rightarrow \text{city} \), but \( \text{theater} \) is not a candidate key. The BCNF decomposition yields relation schemas Bookings1(theater, city) and Bookings2(theater, title).

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

<table>
<thead>
<tr>
<th>theater</th>
<th>city</th>
<th>theater</th>
<th>title</th>
</tr>
</thead>
<tbody>
<tr>
<td>Guild</td>
<td>Menlo Park</td>
<td>Guild</td>
<td>The net</td>
</tr>
<tr>
<td>Park</td>
<td>Menlo Park</td>
<td>Park</td>
<td>The net</td>
</tr>
</tbody>
</table>

which violate the presumed FD \( \text{title} \rightarrow \text{city} \rightarrow \text{theater} \).

Thus, there are implicit dependencies between values in different relations. \textit{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 problem in the previous example arose because we decomposed according to the FD \( \text{theater} \rightarrow \text{city} \), where \( \text{city} \) is part of a candidate key for the Bookings relation. Thus we ended up splitting the candidate key \{\( \text{city}, \text{theater} \)\}.

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 candidate 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 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.

In the terminology of MDM, 3NF eliminates all *transitive* functional dependencies.
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: Multivalued dependencies and 4th normal form.
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:

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

Curing it with NULL values?

Then what about something like one of these:

<table>
<thead>
<tr>
<th>name</th>
<th>street</th>
<th>city</th>
<th>title</th>
<th>year</th>
</tr>
</thead>
<tbody>
<tr>
<td>C. Fisher</td>
<td>123 Maple St.</td>
<td>Hollywood</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>C. Fisher</td>
<td>5 Locust Ln.</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>C. Fisher</td>
<td>NULL</td>
<td>NULL</td>
<td>Empire Strikes Back</td>
<td>1980</td>
</tr>
<tr>
<td>C. Fisher</td>
<td>NULL</td>
<td>NULL</td>
<td>Return of the Jedi</td>
<td>1983</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>name</th>
<th>street</th>
<th>city</th>
<th>title</th>
<th>year</th>
</tr>
</thead>
<tbody>
<tr>
<td>C. Fisher</td>
<td>123 Maple St.</td>
<td>Hollywood</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>C. Fisher</td>
<td>5 Locust Ln.</td>
<td>Malibu</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>C. Fisher</td>
<td>NULL</td>
<td>NULL</td>
<td>Empire Strikes Back</td>
<td>1980</td>
</tr>
<tr>
<td>C. Fisher</td>
<td>NULL</td>
<td>NULL</td>
<td>Return of the Jedi</td>
<td>1983</td>
</tr>
</tbody>
</table>
--- Decomposition ---

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

<table>
<thead>
<tr>
<th>name</th>
<th>street</th>
<th>city</th>
</tr>
</thead>
<tbody>
<tr>
<td>C. Fisher</td>
<td>123 Maple St.</td>
<td>Hollywood</td>
</tr>
<tr>
<td>C. Fisher</td>
<td>5 Locust Ln.</td>
<td>Malibu</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>name</th>
<th>title</th>
<th>year</th>
</tr>
</thead>
<tbody>
<tr>
<td>C. Fisher</td>
<td>Star Wars</td>
<td>1977</td>
</tr>
<tr>
<td>C. Fisher</td>
<td>Empire Strikes Back</td>
<td>1980</td>
</tr>
<tr>
<td>C. Fisher</td>
<td>Return of the Jedi</td>
<td>1983</td>
</tr>
</tbody>
</table>

--- Multivalued dependencies ---

When we can decompose R into relations

\[ R_1(A_1, A_2, \ldots A_n, B_1, B_2, \ldots, B_m) \]
\[ R_2(A_1, A_2, \ldots A_n, C_1, C_2, \ldots, C_k) \]

(with no Bs among the Cs) then we say that there is a **multivalued dependency** (MVD) from the As to the Bs, written

\[ A_1 \ A_2 \ldots A_n \rightarrow\rightarrow \ B_1 \ B_2 \ldots B_m \]

**Example**: Since StarsIn can be decomposed into

StarsIn1(name, street, city) and StarsIn2(name, title, year)

it has the MVD name \(\rightarrow\rightarrow\) street city.
**Unavoidable and trivial MVDs**

If \( \{A_1, A_2, \ldots, A_n\} \) form a superkey, then for any \( B_1, B_2, \ldots, B_m \) we unavoidably have:

\[
A_1 \ A_2 \ldots \ A_n \rightarrow\!
\rightarrow B_1 \ B_2 \ldots B_m
\]

An MVD is said to be **trivial** if either

- One of the Bs is among the As, or
- All the attributes of R are among the As and Bs.

---

**4th normal form**

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

A relation is in **fourth normal form** (4NF) if any multivalued dependency among its attributes is either unavoidable or trivial.

**Example:** StarsIn has the MVD \( \text{name} \rightarrow \text{street} \ \text{city} \) which is nontrivial. Since \( \text{name} \) is not a superkey the relation is not in 4NF.
Decomposing a relation into 4NF

Suppose we have a relation $R$ which is not in 4NF. Then there is a nontrivial MVD

$$A_1A_2 \ldots A_n \rightarrow B_1B_2 \ldots B_m$$

which is not unavoidable.

To eliminate the MVD we split $R$ into two relations:

- 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 4NF, the process is repeated.

---

4NF decomposition example

Recall the relation $\text{StarsIn}$ with schema

$\text{StarsIn}(\text{name, street, city, title, year})$

It has the following nontrivial MVD, which is not unavoidable:

$$\text{name} \rightarrow \text{street city}$$

Thus the decomposition yields the following relations (both in 4NF):

$\text{StarsIn1}(\text{name, street, city})$

$\text{StarsIn2}(\text{name, title, year})$
Relationship among normal forms

Inclusion among normal forms:
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.)

Attribute value redundancy

Normalization does not remove all kinds of redundancy. An example of this 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

\[ \text{MovieNames(key,name)} \]

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.
- Understand in what situations multivalued dependencies arise, and what should be done about them.