Introduction to database design

RG 19.1, 19.2, 19.4, 19.5, 19.6, 19.7, 19.9

Rasmus Pagh

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.

Redundancy in a relation

- Redundant (“unnecessary”) information:
  Same fact is repeated in several tuples.

- **Example**: Instance of

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

  where the length of a movie is repeated several times (once for each `starName`).

- Obvious problem:
  Uses more memory than is necessary.

"Anomalies" caused by redundancy

- **Update anomaly**. 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 anomaly**. Deleting a tuple (recording some fact) may delete another fact from the database.
  (E.g., information on a movie in the Movies relation.)
- **Insertion anomaly**. The "dual" of deletion anomalies.

Normalization theory

- Principled approach to avoiding (or at least being aware of) anomalies in a database design.
- Captures situations where unrelated facts are placed in a single relation.
- Decompose (split) to avoid anomalies:

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

becomes

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

Problem session

- We have a running database with table

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

  and want to change the schema to

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

- What are the keys of the different tables?
- How should we fill the tables `Movies1` and `Movies2`?

Recombining relations

- Decomposed relations must contain the same information as the original relation.
- **Idea**: Compute original relation by a ”join” query that combines tuples where foreign key value = key value.
- **Example**: In SQL, compute Movies as:

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

A “key” concept

- A *candidate key* for a relation is a set K of its attributes that satisfy:
  - **Uniqueness:** The values of the attribute(s) in K uniquely identify a tuple.
  - **Minimality:** The uniqueness property goes away if we remove any attribute from K.
- If uniqueness is satisfied the attributes are said to form a *superkey*.
- **Example**: For `Movies`,
  - `{Title,year,starName}` is a candidate key.
  - `{Title,year,starName,length}` is a superkey.
  - `{Title, year}` is not a key.

Candidate vs primary key

- Important: Candidate key is defined with respect to what data can ***possibly*** occur, 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.
  - There could be several candidate keys to choose from.
  - For normalization, it is irrelevant which key is chosen as primary key.

Example

- `Person(id,cpr,name,address)`
- Candidate keys: `{id}`,`{cpr}`
- Superkeys: `{id}`,`{cpr}`,`{id,cpr}`, `{id,name}`,`{id,address}`, `{cpr,name}`,`{cpr,address}`, `{id,name,address}`, `{cpr,name,address}`,`{id,cpr,name}`, `{id,cpr,address}`, `{id,cpr,name,address}`.
- Not superkey: `{name}`,`{address}`,`{name,address}`

Functional dependency game

- Consider this game:
  - I look at some tuple in a relation R, and tell you the value of attribute A.
  - You look at R and win if you can guess the value of attribute B.
- Consider playing on these relations:

R_1

a	b
4	2
9	3
4	-2
1	-1

R_2

a	b
4	16
9	81
2	4
1	1

Functional dependency (FD)

- We say that A (functionally) determines B, written A→B, if the value of B is ***always*** determined by the value(s) of A (for ***any*** possible relation).
- **Examples**:
  - `cpr → name` in `Person(cpr,name)`
  - `title year → length` in `Movie`
- **Non-example**:
  - `title year → starName` does not hold for `Movie`

What FDs to expect?

- If A is a candidate key for a relation then clearly A→B for any attribute B.
- Similarly if $\{A_1,A_2\}$ forms a superkey we have A_1A_2→B for any B, etc.
- FDs with a (super)key on the left, and FDs such as B→B are **unavoidable**.

Boyce-Codd Normal Form (BCNF)

- A relation is in BCNF if all functional dependencies among its attributes are unavoidable.
- **Example**: `Movies` has the FD
  `title year → length`
  where `{title,year}` is not a superkey.
  - This means that `Movies` is not in BCNF.
- The anomalies we saw in `Movies` are in fact *caused* by the above FD!
  - requires us to store the same movie length again and again.

Decomposing into BCNF

- Suppose relation R is not in BCNF.
  Then there is an FD $A_1A_2 \ldots A_n \rightarrow B_1B_2 \ldots B_m$ that is not unavoidable.
- To eliminate the FD we split R into two relations:
  - R1 with all attributes of R except $B_1B_2 \ldots B_m$.
  - R2 with attributes $A_1A_2 \ldots A_n \rightarrow B_1B_2 \ldots B_m$. Note that $A_1A_2 \ldots A_n$ is a superkey of R2, so a join recovers the original relation R.
- This process is repeated until all relations are in BCNF.

BNCF decomposition example

- The relation

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

  has the FD `title year → length`, so we decompose it into

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

- **Claim**: The relations `Movies1` and `Movies2` are in BCNF, so this finishes the BCNF decomposition.

Arguing that a relation is in BCNF

- Requires domain knowledge about the possible data:
  - What are the candidate keys?
  - What are the FDs?
- Systematic approach:
  - Consider every maximal set of attributes K that leaves out at least one attribute from each candidate key.
  - For each attribute B in K, consider whether the following FD holds: $K \setminus \{B\} \rightarrow B$.
- No such FD found $\Rightarrow$ relation is in BCNF.

Arguing that a relation is in BCNF

- Example relation:
  `Movies1(title,year,length,filmType,studioName)`
  The only candidate key is `{title,year}`.
- Case 1.
  - `K={year,length,filmType,studioName}.`
  - FD `length filmType studioName → year`?
  - FD `year filmType studioName → length`?
  - …
- Case 2.
  - `K={title,length,filmType,studioName}`
  - FD `length filmType studioName → title`?
  - FD `title filmType studioName → length`?
  - …

Problem session

- Consider a relation containing an inventory record:
  `Inventory(part,WareHouse,quantity,WHaddress)`

- Consider the following (you will need to make assumptions to answer):
  - What are the candidate keys of the relation?
  - What are the avoidable functional dependencies?
  - Perform a decomposition into BCNF.

Interrelation dependencies

- Consider `Bookings(title,theater,city)`:
  - `theater` → `city` (`theater` is not key)
  - `title city` → `theater` (`city` is not key)
- BCNF decomposition:

  `Bookings1(theater,city)`

  `Bookings2(theater,title)`.
- Relation instances separately legal:

theater	*city*
Guild	Menlo Park
Park	Menlo Park

theater	*title*
Guild	The net
Park	The net

Interrelation dependencies

Dependencies between allowed tuples in the two relations.
No key constraint can ensure that the FD `title city → theater` holds.

`Bookings1(theater,city)`

`Bookings2(theater,title).`

- Relation instances separately legal:

theater	*city*
Guild	Menlo Park
Park	Menlo Park

theater	*title*
Guild	The net
Park	The net

Third normal form

- The problem arose because we split the attributes of a candidate key among several relations.
- Third normal form: Eliminate avoidable FDs, *except* those that would result in a candidate key being split.
- In other words, it allows any FD
  $A_1A_2 \ldots A_n \rightarrow B_1B_2 \ldots B_m$
  where at least one of $B_1B_2 \ldots B_m$ is part of a candidate key.

Second 3NF example

- `HasAccount(AccountNumber,ClientId,OfficeId)`
- Functional dependencies:
  - `ClientId OfficeId → AccountNumber`
  - `AccountNumber → OfficeId`
- **Claim**: Is in 3NF, but not BCNF (why?).
- Can be decomposed losslessly:
  - `AcctOffice(AccountNumber,OfficeId)`
  - `AcctClient(AccountNumber,ClientId)`

Other normal forms

- First and second normal forms: Historical importance only, see book.
- Fourth normal form:
  - Eliminates certain “blatant” anomalies that are not caught by FDs.
  - For any sane schema same as BCNF.
- Fifth normal form:
  - Performs decomposition into 3 or more relations, even when decomposition into 2 relations is not possible without information loss.
- 5NF ⇒ 4NF ⇒ BCNF ⇒ 3NF ⇒ 2NF

How to use normal forms

- May be seen as *guidelines* for designing a good relation schema.
- In some cases there is a trade-off, e.g. between avoiding anomalies and:
  - Being able to check constraints
  - Efficiency of query evaluation (more on this later in course).

Course goal

After the course the students should be able to:

- find functional dependencies in a relation and perform decomposition to eliminate unwanted dependencies.

Next steps...

- Exercises from 12.30 as usual.
- Will start by a TA presentation of some exercises from last week (<15 min.)
- Several problems from past exams on normalization
  - practice makes the master!
- Next week: Large case study including E-R modeling, relational modeling, and normalization.