EDAF75
Database Technology

Lecture 7: Database Normalization

Christian.Soderberg@cs.lth.se

February 10, 2025

Lecture plan

- Session 1-2: SQL: simple queries, grouping, joining
- Session 3-4: Database modeling, SQL for inserting, updating, and deleting
- Session 5: Connecting to SQL databases from Python (`pysqlite`) or Java (`JDBC`)
- Session 6: REST services (Python/Java)
- Session 7 (today): *Normalization*, a formal method of removing redundancies
- Session 8 (Thursday): More normalization

From the previous lecture

- Make sure the port you're using isn't already used by some other process
- We can use a cool trick to see all the SQL statements our service executes by adding:

```
db = sqlite3.connect("colleges.sqlite")
db.set_trace_callback(print)
```

Example

A travel agency saves information about their customers and their flights. They use one big table (spreadsheet) with the following columns:

- social security number (`ssn`)
- name (`name`)
- passport id (`pp-id`)
- flight number (`flight-no`)
- departure airport (`dep`), and arrival airport (`arr`)
- flight date (`date`)
- aircraft type (`model`, always the same for a given flight)
- maximum number of passengers (`no-pass`)

ssn	name	pp-id	flight-no	dep	arr	date	model	no-pass
910101-0123	Ann Alm	42254	SAS505	CPH	LHR	2018-02-18	B737	126
910101-0123	Ann Alm	42254	BA189	LHR	EWR	2018-02-18	B77W	365
910101-1234	Bo Ek	13151	SAS505	CPH	LHR	2018-02-20	B737	126
...	...	...	...	...	...	...	...	...

What's problematic with this table?

Redundancy

- Whenever we have the same information in several places, we have a *redundancy*
- For many complex systems (such as our bodies!), redundancy means we get more resilience when things fail — so redundancy can be very useful
- In databases, redundancies are problematic – they're the root of various kinds of *anomalies*

First normal form

- We want a database with our friends, and want to save:
  - names
  - phone numbers
  - contexts (i.e., keep track of where we meet a friend)
- How about:

name	phone	context
Liv	[0708-111222, 0708-823123]	[school, basketball, judo]
Adam	[0708-222111]	[school, basketball]

- The "*First normal form*" (1NF) of relational algebra doesn't allow us to use arrays (although some RDBMS actually have them...)

Removing redundancy

- Assuming Liv has two phone numbers, and belongs to three contexts, how many rows would she occupy in our table if we abide by 1NF and keep phone and `context` in the same table?
- Do we need all those rows, can't we just skip some of the rows as long as each phone number and each context is still associated with her somewhere in the table?

name	phone	context
Liv	0708-111222	school
Liv	0708-111222	basketball
Liv	0708-111222	judo
Liv	0708-823123	school
Liv	0708-823123	basketball
Liv	0708-823123	judo
Adam	0708-222111	school
Adam	0708-222111	basketball

Update anomaly

- Given a relation `friends`:

```
friends(name, phone, context)
```

with values:

name	phone	context
Liv	0708-123654	school
Liv	0708-123654	basketball
Liv	0708-111222	judo
Liv	0708-823123	school
Liv	0708-823123	basketball
Liv	0708-823123	judo
...	...	...

- We must change phone numbers in several rows when a friend with more than one context changes numbers
- If we forget to update one of the rows, the data in our table will be *inconsistent* (as in the case with Liv's phone number above)
- This is called an *update anomaly*

Insertion anomaly

- Given a relation `teachers`:

  `teachers(id, name, email, course_code)`

  with values:

id	name	email	course_code
101	Christian	christian@cs	edaf75
201	Emma	emma@cs	edag05
305	Liv	liv@cs	edaf55

- Even if Adam works at the department, and have an `id`, a `name`, and an `email`, we have nowhere to store it until he teaches his first course
- This is called an *insertion anomaly*

Deletion anomaly

- Given a relation `teachers`:

  `teachers(id, name, email, course_code)`

  with values:

id	name	email	course_code
101	Christian	christian@cs	edaf75
201	Emma	emma@cs	edag05
305	Liv	liv@cs	edaf55

- When Christian stops teaching edaf75, all his info disappears (`id`, `name`, and `email`), unless he also teaches another course
- This is called a *deletion anomaly*

Functional dependencies

Definition: Functional dependency

If $\{A_1, A_2, \ldots, A_n\}$ and $\{B_1, B_2, \ldots, B_m\}$ are sets of attributes of a relation R, and we always can determine $B_1, B_2, \ldots, B_m$ from $A_1, A_2, \ldots, A_n$, then we write

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

and call it a *functional dependency* (FD)

- In the relation

  `friends(ssn, name, address)`

  we have the functional dependence

  `ssn` → `name address`
- If Y is a subset of X, then we say $X \rightarrow Y$ is a *trivial functional dependency*
- *Functional dependencies* can cause redundancies, if not handled with care

Functional dependencies

- Given

  `movies(title, year, length, category, studio_name, star)`

  we have the following FD's (assuming movie titles are unique each year):
  - `title year` → `length`
  - `title year` → `category`
  - `title year` → `studio_name`
- These FD's can be written more compactly as:

  `title year` → `length category studio_name`

Functional dependencies

- Given

  `movies(title, year, length, category, studio_name, star)`

  – is

  `title year → star`

  a functional dependency?
- No, there can be many stars in a movie, so just giving the name and year of a movie doesn't identify a specific star
- Observe that FD's are about the semantics of a scheme, not just it's current values (so the 'FD' above wouldn't be a FD even if our table happened to only contain movies with exactly one star in each).

Example

A travel agency saves information about their customers and their flights. They use one big table (spreadsheet) with the following columns:

- social security number (`ssn`)
- name (`name`)
- passport id (`pp-id`)
- flight number (`flight-no`)
- departure airport (`dep`), and arrival airport (`arr`)
- flight date (`date`)
- aircraft type (`model`, always the same for a given flight)
- maximum number of passengers (`no-pass`)

ssn	name	pp-id	flight-no	dep	arr	date	model	no-pass
910101-0123	Ann Alm	42254	SAS505	CPH	LHR	2018-02-18	B737	126
910101-0123	Ann Alm	42254	BA189	LHR	EWR	2018-02-18	B77W	365
910101-1234	Bo Ek	13151	SAS505	CPH	LHR	2018-02-20	B737	126
...	...	...	...	...	...	...	...	...

Which are the functional dependencies of this table?

Keys

Using the concept of functional dependencies, we can now define what a *key* is formally:

Definition: Key

A set of one or more attributes $A_1, A_2, \dots, A_n$ is a *key* for a relation R if, and only if:

(a) $A_1, A_2, \dots, A_n$ functionally determines all other attributes of R

(b) No proper subset of $A_1, A_2, \dots, A_n$ functionally determines all other attributes of R

- (a) means that $A_1, A_2, \dots, A_n$ is a *superkey* for R
- (b) means a key has to be 'minimal' (i.e., *it has no superfluous attributes*)
- A relation can have many keys, in that case we choose one and call it our *primary key*

Simple rules for FD's

- Transitivity:

$$(A \rightarrow B) \wedge (B \rightarrow C) \Rightarrow A \rightarrow C$$

- Splitting and combining (just a notational convenience):

$$(A \rightarrow B) \wedge (A \rightarrow C) \Leftrightarrow A \rightarrow B \ \ C$$

- Trivial:

$$A \ \ B \rightarrow A$$

Closure of a Set of Attributes

Definition: Closure of Set

The *closure* of a set of attributes, $\{A_1, A_2, \ldots, A_n\}$, under a set S of FD's, is the growing set B of attributes which can be generated by repeatedly applying the FD's of S, starting with $B = \{A_1, A_2, \ldots, A_n\}$.
The closure of $\{A_1, A_2, \ldots, A_n\}$ is written as $\{A_1, A_2, \ldots, A_n\}^+$

Exercise

Exercise

We have the relation $R(A, B, C, D, E, F)$ and the FD's:

FD_1: $A \; B \rightarrow C$

FD_2: $B \; C \rightarrow A \; D$

FD_3: $D \rightarrow E$

FD_4: $C \; F \rightarrow B$

Compute $\{A, B\}^+$

Exercise

Exercise

We have the relation $R(A, B, C, D, E, F)$ and the FD's:

FD_1: $A \; B \rightarrow C$

FD_2: $B \; C \rightarrow A \; D$

FD_3: $D \rightarrow E$

FD_4: $C \; F \rightarrow B$

What are the keys of this relation?

Projecting Relations

- We normalize relations by splitting them in two or more relations, we do that by removing some attributes from the original relation – this is called *projection*
- Some FD's may depend (transitively) on attributes which are removed from the original relation, but their semantics don't change, so all FD's which can be derived from the original FD's will still be valid

Exercise

Exercise

We have the relation $R_1(A, B, C, D)$ and the FD's:

FD_1: $A \to B$

FD_2: $B \to C$

FD_3: $C \to D$

What FD's will hold in the new relation $R_2(A, C, D)$?

Normalization

- *Normalization* is a formal method to:
  - discover redundancies, and
  - decompose a relation into two or more relations in order to remove redundancies and improve data integrity
- There are several *Normal Forms* (1st, 2nd, 3rd, Boyce-Codd, 4th, …)
- We will focus on Boyce-Codd (BCNF), since it means we have no redundancies based on functional dependencies

Attributes, dependencies and normal forms

- An attribute which is part of a key is called a *prime attribute*
- An attribut which is *not* part of a key is called a *non-prime attribute* (duh!)
- *Partial dependency*: a non-prime attribute which depends on a proper subset of a key (can only happen if we have a composite key)
- *Transitive dependency*: a non-prime attribute which depends on another non-prime attribute

The first three and a half normal forms

- **1NF**: Only primitive values, and unique rows
- **2NF**: 1NF, and no partial dependencies
- **3NF**: 2NF, and no transitive dependencies
- **BCNF** (3.5NF): 3NF, and no reduncancies based on functional dependencies

BCNF: Boyce-Codd Normal Form (3.5NF)

Definition: BCNF

A relation R is in BCNF if, and only if, the left hand side of every non-trivial FD is a superkey for R.

- Observe that this means there are no FD's which determines only parts of a row (they all determine the whole row), so we have no redundancies based on FD's

Exercise

Exercise

We have the relation

```
movies(title, year, length, category, star)
```

and the FD's:

FD_1: `title year star` → `length category`

FD_2: `title year` → `length category`

Is this in BCNF?

Decomposition into BCNF

Algorithm: BCNF-decomposition

(1) We have a relation R which is not in BCNF

(2) Pick a FD which makes R break BCNF:

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

(3) Create one new relation with all the attributes of the FD (i.e., all the A's and B's)

(4) Create another new relation with all attributes in R, *except* those in the right hand side of the FD (i.e., all the B's) – observe that all the A's should be in this new relation

(5) Remove the old BCNF-violating relation (R), and repeat from step (1), if necessary

Decomposition into BCNF

- The first relation, from step (3), tells us how to 'look up' the values of $B_1\, B_2\, \ldots\, B_m$ given the 'partial' key $A_1\, A_2\, \ldots\, A_n$ (it's not actually called a *partial key*, but note that it by definition is not a superkey)
- In the second relation we keep the 'partial' key (i.e., the left hand side of the FD, $A_1\, A_2\, \ldots\, A_n$), but remove the values we now can look up in our 'lookup table' ($B_1\, B_2\, \ldots\, B_m$)
- We get the dependents from the 'lookup table' using an equi-join, using the left hand side of the FD ($A_1\, A_2\, \ldots\, A_n$) in the join predicate

Exercise

Exercise

We have the relation

movies(title, **year**, length, category, star)

and the FD's:

FD$_1$: `title year star` → `length category`

FD$_2$: `title year` → `length category`

Decompose into BCNF.

Exercise

Exercise

We have the relation

movies(title, **year**, length, category, studio_name, studio_address)

with the only key {`title`, `year`}.
Decompose into BCNF.

Causes for non-BCNF

- Converting a many-to-many association into an attribute in one relationship (such as `star` in the example above)
- Putting transitive dependencies in a table (such as `studio_name` → `studio_address` in the example above)
- If we translate our models into relations the way we saw last week, we'll avoid these pitfalls

Exercise

Exercise

(a) Make a model of movies and studios, and translate it into relations

(b) Make a model of movies and movie stars, and translate it into relations