Databases

TDA357/DIT621– LP3 2023

Lecture 4

Ana Bove

(much of the material is based on material from
both Thomas Hallgren and Jonas Duregård)

January 23th 2023

Recall Last Lecture

- Local definitions;
- Views (vs materialized views);
- Set operations (unions, intersections, excepts);
- Databases with several tables;
- Database queries involving several tables;
- Cross products and inner/left outer/right outer/full outer/ natural joins;
- In and not in, Exists and not exists;
- Dealing with and avoiding empty values.

Overview of Today's Lecture

- Foreign keys;
- More about consistency:
  - Policies on referencial contraints;
  - Assertions;
- Summary of **SQL**;
- Summary of relational schemas;
- **SQL** Example.

Our Running Database

- Relational schema:

  Countries (<u>name</u>, abbr, capital, area, population, continent, currency)
  Unique abbr
  population $\geqslant 0$

  Currencies (<u>code</u>, name, value)

- **SQL**:

```
CREATE TABLE Countries (
  name TEXT PRIMARY KEY,
  abbr CHAR(2) NOT NULL UNIQUE,
  capital TEXT NOT NULL,
  area FLOAT NOT NULL,
  population INT NOT NULL CHECK (population >= 0),
  continent CHAR(2),
  currency CHAR(3) );
```

```
CREATE TABLE Currencies (
  code CHAR(3) PRIMARY KEY,
  name TEXT,
  value FLOAT );
```

Foreign Keys

Back to our Countries and Currencies tables...

Our Currencies table defines the valid currency codes.

So only currency codes from the Currencies table should be allowed in the Countries table.

Is this the case in our database?

```
SELECT DISTINCT currency AS code
FROM Countries
WHERE currency NOT IN (SELECT code FROM Currencies);
```

Adding Foreign Keys

- **SQL**:

```
ALTER TABLE Countries
ADD FOREIGN KEY (currency) REFERENCES Currencies (code);
```

Note:

- Will not work if there are a currencies in Countries whose code are not in Currencies;
- Foreign keys can only refer to unique values (primary/secondary keys).

- Relation schema:

Countries (<u>name</u>, abbr, capital, area, population, continent, currency)
currency $\rightarrow$ Currencies.code
Unique abbr
population $\geqslant 0$

Adding the Missing Values for Foreing Keys to Be Defined

- Directly:

```
INSERT INTO Currencies VALUES ('NOK', 'Norwegian Krone', 0.96);
```

- Using a query:

```
INSERT INTO Currencies
   (SELECT DISTINCT currency AS code
    FROM Countries
    WHERE currency NOT IN (SELECT code FROM Currencies) );
```

We only added the code, the other attributes are empty so far.

```
UPDATE Currencies
SET name = 'Norwegian Krone', value = 0.96
WHERE code = 'NOK';
```

Alternative Foreign Keys Definition

```
CREATE TABLE Currencies (
   code CHAR(3) PRIMARY KEY,
   ... );
```

```
CREATE TABLE Countries (
   name TEXT PRIMARY KEY,
   ... ,
   currency CHAR(3) REFENCES Currencies (code) );
```

If we reference a primary key we can omit the name of the attribute:

```
CREATE TABLE Countries (
   name TEXT,
   ... ,
   currency CHAR(3),
   PRIMARY KEY (name),
   FOREIGN KEY (currency) REFENCES Currencies );
```

Compound Foreign Keys

Foreign keys can also be compound.

(Observe the order in the reference, can you explain why?)

- **SQL**:

```
CREATE TABLE Cities (
   name TEXT,
   country TEXT,
   PRIMARY KEY (name, country) );
```

```
CREATE TABLE CountryCapitals (
   name TEXT PRIMARY KEY,
   capital TEXT,
   FOREIGN KEY (capital, name) REFERENCES Cities );
```

- Relational schema:

Cities (<u>name</u>, <u>country</u>)
CountryCapitals (<u>name</u>, capital)
  (capital, name) $\rightarrow$ Cities.(name, country)

Mutual References

What if the country in Cities must be one of those countries in CountryCapitals?

Having mutal refences when defining the tables is not allowed.

Instead we need to add the extra foreign key later.

```
ALTER TABLE Cities
ADD FOREIGN KEY (country) REFERENCES CountryCapitals;
```

Now the tables reference to each other.

Working with Mutually Referenced Tables

- Inserting values:

```
INSERT INTO CountryCapitals VALUES ('Sweden', NULL);

INSERT INTO Cities VALUES ('Stockholm', 'Sweden');

UPDATE CountryCapitals SET capital = 'Stockholm'
WHERE name = 'Sweden';
```

NULL in a foreign key does not require the existence of a value in the referenced table!

- Deleting the tables (exchanging the names of the tables will also do):

```
DROP TABLE Cities CASCADE;

DROP TABLE CountryCapitals;
```

CASCADE will also drop the dependency.

Updates and Deletions on Referenced Values

What if Sweden becomes a republic and changes the name to Republic of Sweden?

By default, an attempt to delete/update (part of) a referenced row fails and nothing is changed!

By using ON UPDATE CASCADE, changes to the primary keys in a table are propagated to the references in the other table.

Similarly, when using ON DELETE CASCADE.

Updates and Deletions on Referenced Values (Cont.)

```
CREATE TABLE Cities (
   name TEXT,
   country TEXT,
   PRIMARY KEY (name, country) );
```

```
CREATE TABLE CountryCapitals (
   name TEXT PRIMARY KEY,
   capital TEXT,
   FOREIGN KEY (capital, name) REFERENCES Cities
   ON DELETE CASCADE ON UPDATE CASCADE );
```

```
ALTER TABLE Cities
ADD FOREIGN KEY (country) REFERENCES CountryCapitals
ON DELETE CASCADE ON UPDATE CASCADE;
```

Changes in the name of the country or of a capital will propagate to the other table.

Updates and Deletions on Referenced Values (Cont.)

There are other possibilities when changing/deleting referenced values:

- ON UPDATE | DELETE CASCADE: updates/deletes the row if referenced value is deleted;
- ON UPDATE | DELETE SET NULL: sets value to NULL if referenced value is updated/deleted;
- ON UPDATE | DELETE RESTRICT: disallows updates/deletions, causes an error!

Some of the options do not work when trying to change (part of) a primary key.

Note: Test the different options in examples!

Assertions

Part of **SQL** standar but not widely supported since they are usually very expensive to check (need to be check after each update).

In particular not supported in PostgeSQL or major DBMS.

They allow us to write conditions that should be globally true for the database.

Imagine a bank database with accounts:

```
CREATE ASSERTION minimum_balance AS
  CHECK ( (SELECT SUM(balance) FROM Accounts) >= 1000000);
```

Note: We will use triggers instead for this kind of constraints.

Primary Keys vs Unique Constraints vs. Foreign Keys

Primary Keys:

- A table can only have one primary key;
- Their value uniquely identifies the data;
- The primary key could though be compound (consists of several attributes)!

Unique Constraints: Data that must be unique but is not part of the primary key is marked with a UNIQUE constraint.

Foreign Keys

- A table can have multiple foreign keys;
- Each key can consists of one or more attributes;
- Each foreign key constraint is checked independently of each other;
- Foreign keys need to reference unique attributes!

So Far: **SQL**

So far we know how to:

- Create, delete and alter tables in **SQL**;
- Define the attributes and their types;
- Set constraints:
  - Primary and unique/secondary constraints to disallow duplicates;
  - Reference contraints to ensure values exist in another table;
  - Particular constraints on the values of and between attributes;
- Insert, delete and update values in the tables;
- Deal with deletes and updates of references values;
- Query one or more tables.

Summary **SQL** Queries

Full query:

```
SELECT <columns/expressions>
FROM <tables/subqueries/JOIN>
WHERE <conditions on rows>
GROUP BY <columns>
HAVING <conditions on groups>
ORDER BY <columns/expressions> [ASC/DESC]
LIMIT n;
```

Set operations:

```
<query1> [UNION/INTERSECT/EXCEPT] <query2>
```

Even variants with ALL.

Expressions: built from columns, constants (0,'hello',...), operators ($+, -, ...$), functions (aggregates, COALESCE, ...).

Conditions: can use columns, constants, AND/OR/NOT, IN, EXISTS, IS NULL, $<, >, <=, >=, =$, !=, ...

So Far: Relational Schemas

- Relational schemas are a compact way to describe a database;
- Contain:
  - Name of the relations and their attributes;
  - State primary key;
  - Declare uniqueness, references and other constraints;
- They can be easily/mechanically translated into **SQL**;
- Recall types are missing from the schemas!

Exercise: The US electoral system

This exercise relates to how the US electoral system works and the US election in 2020.

a) Make a table with a column for state names (or abbreviations), columns for Biden and Trump votes respectively and a column for number of electors.
b) Then create a view called StateResults that shows for each state: the name of the state, the name of the winning candidate (Biden or Trump) and the number of electors.
c) Finally make a query that shows the total number of electoral votes of both candidates (the result should have two rows for the two candidates).

Hint: Start by writing a query showing only the states Biden wins, along with a column containing the text value 'Biden'. You're more than half-way done!

Hint: Use the view to create the winner-query.

Overview of Next Lecture

- Entity-relationship (ER) model:
  - Entities and attributes;
  - Many-to-many relationships;
  - Many-to-exactly-one relationships;
  - Many-to-at-most-one relationships.
  - Multiway relationships;
  - Self-relationships;
  - Weak entities;
  - ISA relationships;
- (ER-example/exercise.)

Reading:

Book: chapter 4.1–4.8

Notes: chapter 3