EDAF75
Database Technology

Lecture 3

`Christian.Soderberg@cs.lth.se`

January 28, 2020

Administration

- We need course representatives
- You can now sign up for the labs (see web site)
- If you want a QA-session on Friday, please sign up on Moodle
- This week we'll continue with SQL queries, and look at *database modeling*
- Lab 1 is next week, it's an exercise in SQL queries
- Lab 2 is the week after next, and deals with modeling (which we'll talk about this week)

Today

- We'll first continue with some SQL
- Then we'll talk about how to make a model of a system, and then turn that model into a database

Short recapitulation

- A *relational database* is a collection of one or more *tables*, where each table has a fixed set of *columns* and a variying number of rows – *all cells contain primitive values*
- Simple queries (`SELECT-FROM-WHERE`)
- Set operations (`UNION`, `INTERSECT`, `EXCEPT`)
- Simple functions and aggregate functions
- Grouping (`GROUP BY-HAVING`)
- Subqueries
- Joins (`CROSS`-, `INNER`-, `LEFT OUTER`-, ...)

Today

- Views: ’virtual’ tables which enables us to simplify queries (in SQLite it’s essentially a stored SELECT-statement)
- Common Table Expressions (CTE): a view which exists only for the duration of a single query
- Database modeling

Modeling

- To design a database, we’ll start out with what’s called an Entity/Relation Model (E/R Model)
- There are many ’standards’ for drawing E/R diagrams, we’ll use UML class diagrams – it’s becoming increasingly popular for database modeling

Elements of an E/R Model

- *Entity Sets*: these are the ’objects’ of our model, they correspond to classes in a traditional object oriented model
- *Attributes*: properties of our objects – must be primitive values (see the next slide)
- *Relationships*: associations between our entity sets
- We will typically convert entity sets to tables (*relations*), and attributes to columns in our tables – relationships will be dealt with according to their multiplicities

Primitive values in our models

- integers: `INT`, `INTEGER`, ...
- real numbers: `REAL`, `DECIMAL(w,d)`, ...
- strings: `TEXT`, `CHAR(n)`, `VARCHAR(n)`
- dates: `DATE`, `TIME`, `TIMESTAMP`, ...
- blobs (binary largs objects): `BLOB` (only in some databases)

UML class diagrams

- We'll use UML classes in approixmately the same way as you may have seen in earlier courses, with some caveats:
  - There will be no methods in our classes
  - All our attributes will be primitive and public
  - We won't bother much with aggregates and compositions, we'll use plain associations instead
  - We'll be very careful to mark multiplicities everywhere
  - We will be preoccupied by what constitutes a *key* for each entity set

Example

Exercise: *Create a simple model for students taking courses at LTH*

Example

Exercise: *Create a model for students taking courses at LTH, be a little bit more elaborate this time*

Keys

- A *superkey* is a set of attributes for which there are no tuples (rows) that have the same values for the attributes of the set
- A *key* is a *minimal* set of attributes which uniquely identifies each row in an entity set
- Sometimes there are several candidate keys, when we model our database we pick one of them and call it our *primary key*
- We need keys to find values easily, and to keep track of related rows in other tables

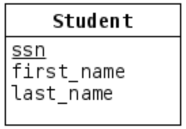
Natural keys and invented keys

- Sometimes keys occur naturally in the problem domain, we call them *natural keys* or *business keys*
- Some entity sets have no natural keys, we call them *weak entity sets*
- Sometimes we invent keys by introducing some artificial attribute, these keys are called *invented keys*, *surrogate keys*, *synthetic keys*, ...

Natural keys and invented keys

- Adding an invented key to a ”weak” entity set makes it a regular entity set
- Whether to use an invented key or not is often a question of simplicity vs effeciency:
  - Without an invented key we sometimes get an unwieldy key (several attributes)
  - With an invented key our tables and queries only need a single key column, *but finding the key may require additional joins*
- If an attribute might change over time, it’s not a good choice for a key – it would require us to update all tables which uses the old value

UML class diagrams – classes

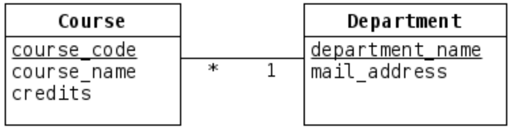


Student

ssn
first_name
last_name

- The entity/class name in singular
- Only one box (since we have no methods)
- We will underline keys

UML class diagrams – associations



Course

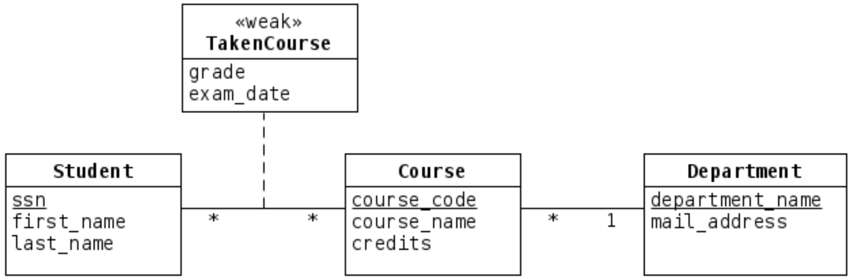
course_code
course_name
credits

Department

department_name
mail_address

- We always mark multiplicity on our associations
- We use associations instead of attributes whenever the value is a reference to another object

UML class diagrams – association classes



- Sometimes the association between two entity sets contains data itself
- We use an *association class* to capture that data
- Often we can use either an association class between two entity sets, or another entity set ’between’ them

Translating E/R models to databases

- For each entity set (class) we create a relation (table) with the same attributes (columns) as the entity set
- For relationships (associations), what we do depend on their multiplicity:
  - ’1 – 1’-associations often means that the two entity sets can be merged into one entity set
  - ’$*$ – 1’-associations will be turned into *foreign keys* in the tables on the $*$-side
  - ’$*$ – $*$’-associations will be turned into relations (tables)
- Inheritance and other multiplicities require more consideration (we’ll look at it next time)

Foreign keys

- A foreign key is an attribute which refers to a key in another relation
- If we declare a column to be a `FOREIGN KEY`, our database will check that there is a corresponding value in the referenced table