Seminar 1 Report

Data Storage Paradigms, IV1351

Viktor Danielsson, Armin Eghtesadi

November 19, 2025

Project members:
[Viktor Danielsson, vdani@kth.se]
[Armin Eghtesadi, armineg@kth.se]

Declaration:

By submitting this assignment, it is hereby declared that all group members listed above have contributed to the solution. It is also declared that all project members fully understand all parts of the final solution and can explain it upon request.

It is furthermore declared that the solution below is a contribution by the project members only, and specifically that no part of the solution has been copied from any other source (except for lecture slides at the course IV1351), no part of the solution has been provided by someone not listed as a project member above, and no part of the solution has been generated by a system.

1 Introduction

The task at hand is to design and create a database capable of describing a universities course system. There has to exist a course layout that describes the course code, course name, how many points the course is worth, the minimum and maximum amount of students. A course instance is needed to track the total amount of students registered and in which period the course is given in for example P1, P2, P3, P4. Each course has many different teaching activites such as lectures, labs, tutorials and seminars. Every teaching activity requires a diffrerent amount of time for the teacher to prepare (multiplication factor) which has to be included in the database. Every course has a varying amount of time for each teaching activity which also has to be considered in the database. The last things are the different departments with their manager (who is also an employee) and the teachers/employees themselves. An employee or teacher works at a department and must have contact details, salary, teaching activites based on skill set, job title and their supervisor/manager. Each teacher can be involved in many teaching activites for multiple different course instances and each course instance can have multiple teachers involved at once. However, a teacher can only be in up to four course instances in a single period.

The final database has to be able to handle all the affermentioned details from the university and be flexible enough to add more teaching activites later on. It must also be possible to add different course versions of the same course layout to to have a 15hp course aswell as a 7.5hp for the same course.

A university customer would have to specify which operations the database must handle but for a general solution the operations are defined as follows:

- A teacher is not allowed to be overloaded with work during a period.
- Since costs are computed for all activities given during the particular period, the database must contain information about who gave which activity for which course instance.
- It must be possible to add more teaching activities.
- It must be possible to change salary for teachers.
- A course must be able to be planned based on the hours needed for teaching activities.
- When a teaching activity has been made a teacher must be able to be allocated to said activity.

2 Literature Study

Some of the knowledge gathered to complete the task was:

- Relational data model: What we are making is a relational database built on the relational data model. It stores its data in a collection of tables all of which contain columns to describe what the table contains. The rows describe an instance of the table. The relationship between tables describes how they are related and what type of cardinality they have.
- Cardinality: A relationship between two tables must express how many of such tables can be associated with the other table. One-to-one would indicate that the their only exists a single table related to another single table. However Many-to-many would instead indicate multiple tables can have a single relationship to multiple other tables. Cardinality is also necessary to define how many of a certain attribute is allowed in a signle instance, can we have a null amount, one to many, zero to many and so on.

- Attribute Domain: For each column or attribute there has to be an assigned domain to define all the values that would be allowed to fill that column for each instance of the table.
- Atomic values: Our database strives to use atomic values in our columns. That means that each value is in its smallest part and not able to be divided into any more smaller parts. So our domain can be atomic if it only allows for a single integer in its column.
- Normalization: A database should not store loads of redundant data, it is space inefficient and requires more computation when you perform operations on the database. So normalization is design decisition that formally defines how normalized the data is in the database. The first normal form 1NF is when every single attribute is atomic. NF2 is the state of the model where every attribute is fully dependent on the entire primary key and 3NF has removed all the transitive functional dependencies in a lossless manner (no information was lost).
- Primary Key: A primary key is a specific attribute choosen specifically to identify each tuple in the relation, or alternativly each row in a table.
- Foreign key: The foreign key is a value in a different table that has to match one of the values in the primary key of that other table. This allows for references between tables.

3 Method

The method of creating a database starts by creating a miniworld of reality that describes how different entities interact and exchange information Then the conceptual model is transformed into a logical and physical model which describes how the miniworld would be represented in a database. The eleven steps necessary to formulate the correct logical and physical model were:

Two trivial steps of creating tables and rows based on the conceptual model.

Then assigning the domain for each attribute.

”UNIQUE” attributes will be marked in the model.

Define primary keys for each table which is definied as a strong entity in the model. Typically surrogate key.

Then define the relationships between tables, one-to-many, one-to-one, many-to-many

Generate pk and fk for multivalued attribute tables

Normalize the model

Final step of making the model is verifying all the operations.

After each step is completed the model is complete and the tool used to make the model Astah professional is able to generate a rough SQL dump. Finalizing the database is done by adding UNIQUE to the correct attributes and adding foreign key constraints ON DELETE.

When the schema has been defined a process of adding data has to be added and then some SQL operations can be performed on the model to see that all the data that should be accesable is.

4 Result

Generating a simple logical model with just tables and attributes directly from the conceptual model generated the basis for the finished logical model. A finalized version of the entire logical model can be seen in figure 1

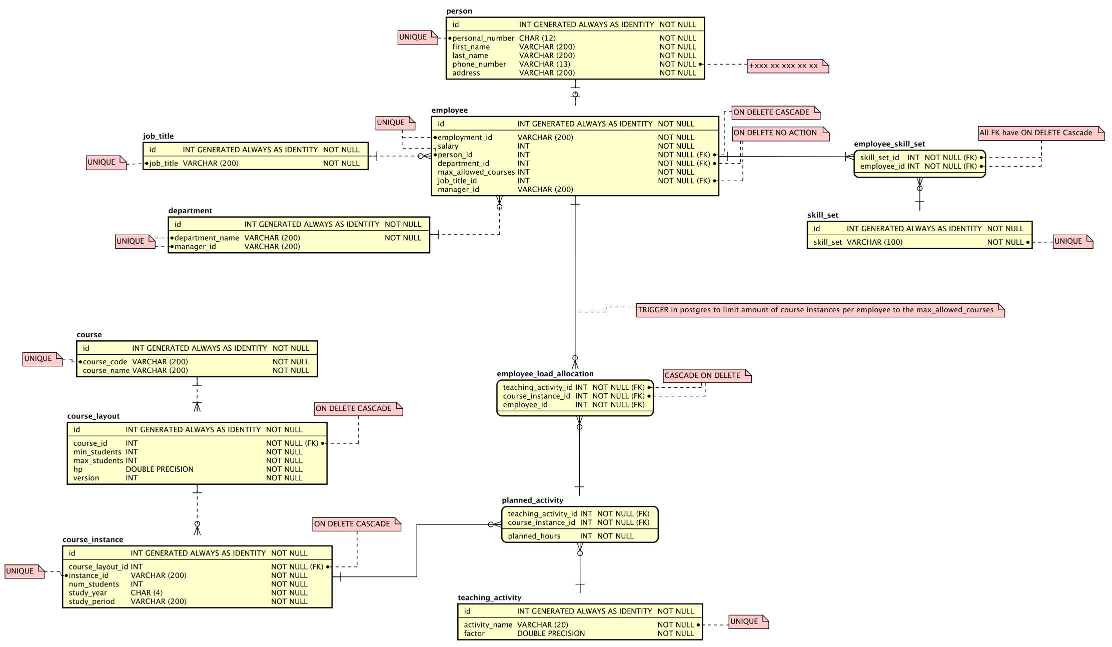
person
id INT GENERATED ALWAYS AS IDENTITY NOT NULL
personal_number CHAR (12) NOT NULL
first_name VARCHAR (200) NOT NULL
last_name VARCHAR (200) NOT NULL
phone_number VARCHAR (13) NOT NULL
address VARCHAR (200) NOT NULL
UNIQUE
+xxx xx xxx xx xx
employee
id INT GENERATED ALWAYS AS IDENTITY NOT NULL
employment_id VARCHAR (200) NOT NULL
salary INT NOT NULL
person_id INT NOT NULL (FK)
department_id INT NOT NULL (FK)
max_allowed_courses INT NOT NULL
job_title_id INT NOT NULL (FK)
manager_id VARCHAR (200)
UNIQUE
ON DELETE CASCADE
ON DELETE NO ACTION
job_title
id INT GENERATED ALWAYS AS IDENTITY NOT NULL
job_title VARCHAR (200) NOT NULL
UNIQUE
employee_skill_set
skill_set_id INT NOT NULL (FK)
employee_id INT NOT NULL (FK)
All FK have ON DELETE Cascade
skill_set
id INT GENERATED ALWAYS AS IDENTITY NOT NULL
skill_set VARCHAR (100) NOT NULL
UNIQUE
department
id INT GENERATED ALWAYS AS IDENTITY NOT NULL
department_name VARCHAR (200) NOT NULL
manager_id VARCHAR (200)
UNIQUE
TRIGGER in postgres to limit amount of course instances per employee to the max_allowed_courses
course
id INT GENERATED ALWAYS AS IDENTITY NOT NULL
course_code VARCHAR (200) NOT NULL
course_name VARCHAR (200) NOT NULL
UNIQUE
course_layout
id INT GENERATED ALWAYS AS IDENTITY NOT NULL
course_id INT NOT NULL (FK)
min_students INT NOT NULL
max_students INT NOT NULL
hp DOUBLE PRECISION NOT NULL
version INT NOT NULL
ON DELETE CASCADE
employee_load_allocation
teaching_activity_id INT NOT NULL (FK)
course_instance_id INT NOT NULL (FK)
employee_id INT NOT NULL (FK)
CASCADE ON DELETE
planned_activity
teaching_activity_id INT NOT NULL (FK)
course_instance_id INT NOT NULL (FK)
planned_hours INT NOT NULL
course_instance
id INT GENERATED ALWAYS AS IDENTITY NOT NULL
course_layout_id INT NOT NULL (FK)
instance_id VARCHAR (200) NOT NULL
num_students INT NOT NULL
study_year CHAR (4) NOT NULL
study_period VARCHAR (200) NOT NULL
UNIQUE
ON DELETE CASCADE
teaching_activity
id INT GENERATED ALWAYS AS IDENTITY NOT NULL
activity_name VARCHAR (20) NOT NULL
factor DOUBLE PRECISION NOT NULL
UNIQUE

Figure 1: A complete image of the logical and physical model

Assiging the domain to each attribute is a necessary step for the database to allow enough space for each column but also to apply the corresponding constraints which are associated within a domain. In the case of not knowing what the customer wants some liberty has to be taken because it is simply not possible to know exactly what domain a customer would want. The domain for the personal number is CHAR(12) because the amount of numbers in a swedish personal number is 12 charachters long and has to be unique. When the domain is not known VARCHAR(200) is used to give enough options for the customer without sacrificing loads of unused storage space

The same line of reasoning is used for employee where employment_id is unqiue to an employee and uses VARCHAR 200 and is a business logic id. What format the university would use is not known so a variety of formats is allowed by the database. Then an employee has a manager which is also an employee which is why it is allowed to be null. max_allowed_courses is an int, salary is an int. job_title of an employee is a foreign key to the table containing all job_titles.

Each attribute was chosen to be atomic and also relevant to how a university would use that value. most values used as an id is for business logic within the university and is not to be confused by the surrogate id which identifies the tuples.

Finding the strong entities was done by looking at the conceptual model and searching for an entity which is not dependant on another entity to exist. All non-identifying relations are tables with strong identifies such as person, employee, job_title, department, course, course_instance, teaching_activity.

After each table has been defined with its primary key it is then possible to begin assigning the correct foreign keys and their respective constraints. A foreign key should follow the 2NF and 3NF where the functional dependencies of are fully depend on their primary key and there are no transitive FD in the current solution. The constraints are mostly evaluated based on whether we want the delete to propagate through the entire database or not. If an employee is removed it should propagate through the entire database but if we delete a department we would not want to remove every employee from the department because they might just be reallocated to a new department.

After each table has been verified to be in 3NF the model is considered complete and can be seen in figure 1.

The SQL dump was generate by Astah by exporting the E-R diagram. This allowed us to quickly deploy a revised database and test it almost immediately. So if the new design was able to solve issues or not we could figure it out much faster than if we had to write a new dump by ourselves each time. The SQL dump is published on github but the code structure is similar to this for each table Finally some SQL queries were used to validate the operations define in the introduction. The first operation is not overloading a teacher. This would be solved by a trigger in the database which uses the max_allowed_courses but triggers are not necessary for this database. All of the operations are possible to perform. Finding which employee gave which activity for which course instance is recorded in the employee_load_allocation table. Teaching activity allows for more inserts and the employee salary is possible to update. Planned activity stores the hours for each activity in a course which fulfills that requirement. Employee_load_allocation has foreign keys for employees and teaching activites so the teacher will be linked to a certain activity.

```
CREATE TABLE employee (
 id INT GENERATED ALWAYS AS IDENTITY NOT NULL PRIMARY KEY,
 employment_id  VARCHAR(200) NOT NULL UNIQUE,
 salary INT NOT NULL,
 person_id  INT NOT NULL,
 department_id INT NOT NULL,
 max_allowed_courses INT NOT NULL,
 job_title_id INT NOT NULL,
 manager_id VARCHAR(200),

 FOREIGN KEY (person_id ) REFERENCES person (id) ON DELETE CASCADE,
 FOREIGN KEY (department_id) REFERENCES department (id) ON DELETE NO ACTION,
 FOREIGN KEY (job_title_id) REFERENCES job_title (id) ON DELETE NO ACTION
);
```

5 Discussion

In general each step has been revised over multiple iterations because the actions following a step might show flaws in the previous steps. One such flaw which showed up only after trying to add data was foreign keys which had ”circular” relationsships. Creating one table would first need a different table to be generated first but for that to happen the other table needs the current table, a catch-22 for generating data. These flaws were difficult to find directly in the model but obivous when the constraint violations are thrown in the databsase. These issues were prevelant in both the courses, employees with department but solved by having fewer NOT NULL constraints and less identifying relations.

The employee attribute skill set is tricky because multiple employees can share the same skill and we want to ensure the values are atomic. So multiple employees can have the same skill and a single employee has multiple skills. The many-to-many relation is solved with a cross-reference table.

For our many-to-many relations such as course instance and teaching activity and employee skillset we use the cross-reference table. It could be done with less tables such as tables with multivalued attributes but it is valuable to decrease the amount of duplicate data

When finding strong entites the department entity is perculiar because the manager_id is dependent on a employee existing. By allowing for a null manager we can define the department as a strong entity because it is no longer dependant on the employee table. This is allowed because null is not accounted for when checking the column for uniqueness. This was also a point where ”circular” dependencies showed up when generating data and having the department independent from employee allows us to create the department without an employee.

When considering the hp for a course there could be the same course with the same course id but with a different amount of hp. Our solution is to divide course and course layout in to two tables. The course only contains code and name with a surrogate key. Then layout has min and max students, hp, a version and a surrogate key as primary key. The relation between the two is non-identifying but one could argue that the layout should be identified by its course. Another way of solving it would be to combine the two tables and have the version as a primary key togehter with course_id but that solution was not used because the course_code and course_name are not related to version in the primary key.

6 Comments About the Course

The time invested to the project were six, five hour sessions so in total 30 hours to complete the project.