

# Seminar 1 Report

Data Storage Paradigms, IV1351

Alexander Tashkinov

[2026-01-06]

**Project member(s):**

Alexander Tashkinov, atas@kth.se

**GitHub repository link:**

[REPO\_LINK]

## 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 goal of Task 1 was to design and implement a relational database based on the project description. The work focused on translating the conceptual requirements into a logical/physical model using crow's foot notation, and then implementing the schema in PostgreSQL using SQL. A key part of the task was to keep the model normalized (3NF) and to enforce data integrity using primary keys, foreign keys, and appropriate constraints.

## 2 Method

The database design started by identifying the entities, attributes, and relationships described in the assignment. After the entities were identified, the relationships were refined by deciding cardinalities and whether a relationship should be represented directly or through an associative table. During this step, special care was taken to ensure that many-to-many relationships were resolved into separate tables with composite primary keys, and that all non-key attributes depended only on the key.

The logical/physical model was produced in Astah using crow's foot notation. The physical design decisions were then translated into SQL and implemented in PostgreSQL. The implementation was split into two scripts: `create.sql` for creating tables, constraints and indexes,

and `insert.sql` for inserting sample data. The scripts were executed in `psql` to verify that the schema builds cleanly from scratch and that inserts succeed without violating constraints.

### 3 Result

Figure 1 shows the final logical/physical model. The implemented schema contains tables for people and employees, course layouts and course instances, teaching activities, planned activities, and allocations. Planned hours are stored per course instance and activity, and allocated hours are stored per teacher, course instance, and activity.

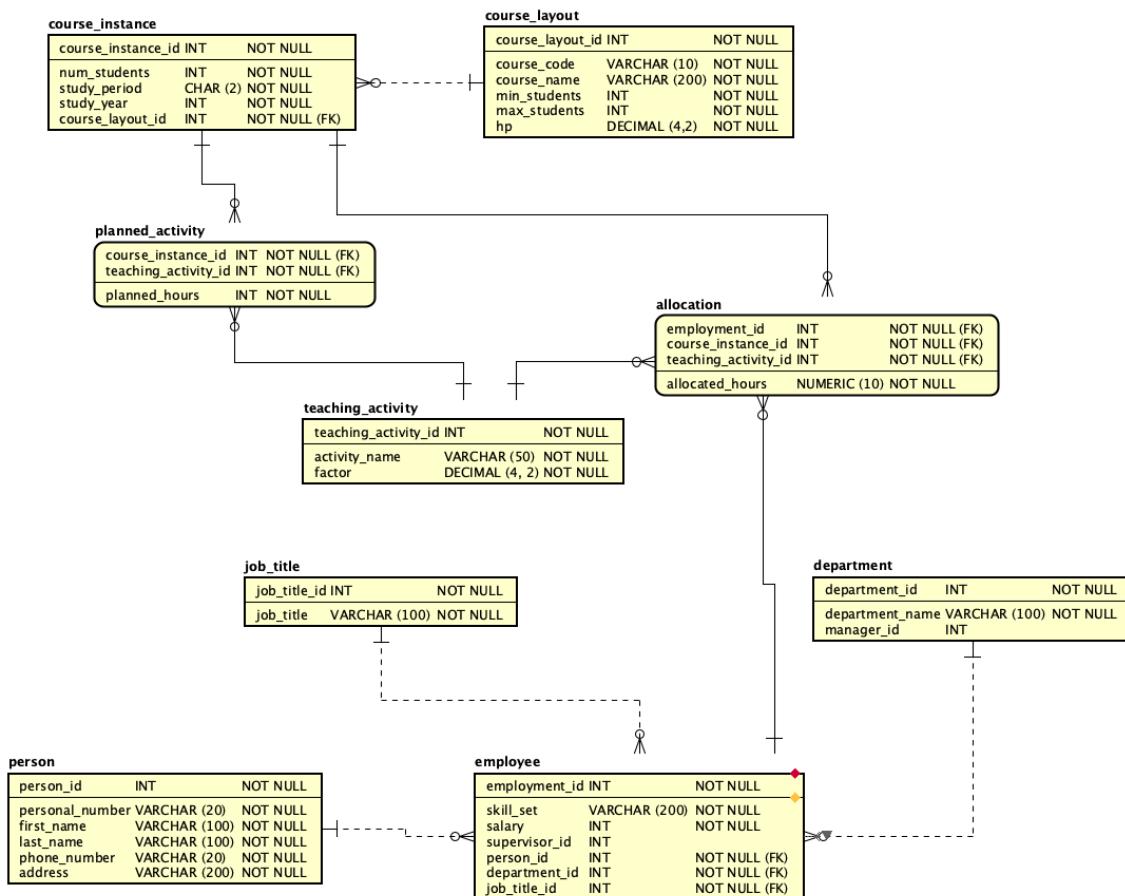


Figure 1: Logical/Physical model (crow's foot).

The schema is implemented with explicit constraints to enforce data quality. For example, study periods are restricted to the allowed values (P1–P4), hours and salary are constrained to non-negative values, and referential integrity is ensured via foreign keys. Composite primary keys are used in the associative tables `planned_activity` and `allocation` to guarantee uniqueness of each combination.

#### 3.1 Database build and verification

The following terminal excerpt shows that the database scripts execute successfully and that tables are created as expected.

```

COMMIT
[iv1351=# \dt
      List of tables
 Schema |       Name        | Type | Owner
-----+----------------+-----+-----
 public | allocation    | table | postgres
 public | course_instance | table | postgres
 public | course_layout  | table | postgres
 public | department    | table | postgres
 public | employee      | table | postgres
 public | job_title     | table | postgres
 public | person         | table | postgres
 public | planned_activity | table | postgres
 public | teaching_activity | table | postgres
(9 rows)

```

Figure 2: Example terminal output after running `create.sql` and `insert.sql`.

## 4 Discussion

The resulting design aims to follow Third Normal Form by separating concepts into dedicated tables and by avoiding duplicated attributes across entities. The associative tables `planned_activity` and `allocation` prevent many-to-many relationships from producing redundancy, while still allowing the database to store detailed information per activity.

The choice of constraints improves reliability by catching incorrect data early. For example, restricting `study_period` to valid values prevents inconsistent records, and foreign keys prevent allocating hours to non-existing employees or course instances. Indexes were added on columns commonly used for joins and filtering, which is useful both for correctness (supporting typical access patterns) and for later analytical queries in Task 2.