Logical and Physical Model

Data Storage Paradigms, IV1351

Group 50

2025-01-03

Project members:
[Allan Al Saleh, Allan2@kth.se]
[Mostafa Faik, mfaik@kth.se]
[Derfesh Mariush, derfesh@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.

Introduction

For the previous part of the assignment, we had to create a conceptual model for the musical academy Soundgood. In this assignment our goal is to transition this model to a hybrid between a logical and physical model and create a database from it.

Literature Study

To gain the foundational knowledge for this task, we relied on these resources:

- Course videos(Logical and physical model): These taught the basics of logical and physical models, in addition to providing guidance on converting the conceptual model to an amalgam of both models. They also brought up fundamental concepts such as tables, relations, domains, primary and foreign keys.
- Textbooks: We referred to the book *Fundamentals of Database Systems* (7th edition by Elmasri and Navathe), particularly chapters 5 and 8. This boosted our understanding of relational models and relational algebra. Additionally, we read chapter 6 for specific details of data types in SQL and different constraint methods such as triggers.

Method

1. Diagram Editor and Tools Used:

The logical and physical model was designed using `Astah Professional`, which supports `Crow's Foot notation`. The database schema was implemented and tested in `PostgreSQL`. Additionally, `pgAdmin` was used for database management, exporting SQL scripts and data generation for the inserted script data, which was facilitated by the online tool `Generatedata.`

2. Procedure to Create the Logical and Physical Model:

The following steps were taken:

- Step 1: Analyze and translate the conceptual model from task 1 into a logical model with enough physical aspects to allow for database creation. During this step:
  - All entities from the CM were assigned into tables.
  - All attributes with the cardinality *0..1* or *1..1* were assigned to their respective tables.
  - New tables were created for attributes with a higher cardinality.
  - Data types for attributes were determined based on their purpose (e.g., VARCHAR for strings, INT for numerical values and TIMESTAMP for dates).

  - Constraints were added to enforce rules, such as sibling relationships, maximum rental quotas and pricing validity periods.
  - All tables were assigned primary keys (PKs), and foreign key (FK) relationships were clearly defined.
  - PKs of the strong end were used as foreign key (FK) in the weak end, for one-to-one and one-to-many relations.
  - Surrogate PKs were used in the weak end if it had meaning without the strong end, otherwise the FKs were assigned as PKs.
  - FKs constraints were considered, such as default (no action), allow (set FK to NULL) and Delete CASCADE.
  - A cross-reference tables were created for each many-to-many relation. Additionally PKs/FKs were added, FKs constraints were considered and additonal columns were added if needed.
  - Ensured that the logical model is normalized and alligns with the 3NF and avoiding redundancy. This included re-evaluating entity attributes and their dependencies.

- Step 2: Validate the logical model against the business requirements to ensure it fully supports all operations, such as lesson scheduling, payments, and rentals. Changes were made to the model when necessary, such as adding the `lesson_price` table for flexible pricing.
- Step 3: Create the database schema using SQL scripts generated in PostgreSQL. The schema strictly adhered to the logical model.
- Step 4: Populate the database with sample data using the insertion script generated from the online tool. Test queries were run to validate the schemas functionality and data integrity.

Result

1. Logical and Physical Model:

The final database model is created in the `Crow's Foot notation`. It consists of key tables such as student, instructor, lesson, instrument and related tables. Key relationships, such as `student_lesson` (tracking lesson enrollments) and rental (tracking instrument rentals), are clearly defined.

- The model includes fields like `valid_from` and `valid_to` in the `lesson_price` table to track historical prices, ensuring pricing consistency for past lessons.
- The availability table maps instructors' teaching availability, while the sibling table tracks relationships for applying sibling discounts.

- Constraints such as a maximum of two rentals per student and lesson capacity (minimum and maximum, for group and ensemble lessons) are enforced within the model.

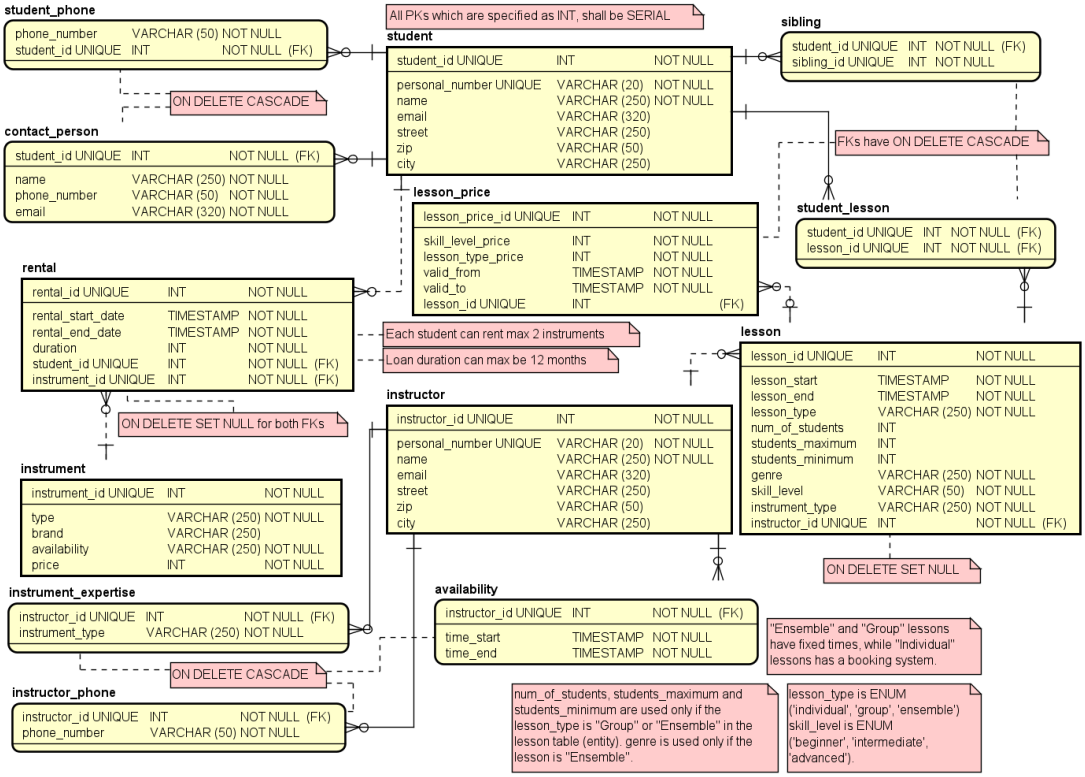

Figure 1: Logical and Physical Model diagram

2. Git Repository Link:

SQL scripts for creating and populating the database are available in the following Git repository:

https://gits-15.sys.kth.se/derfesh/IV1351-HT24-Data-Storage-Paradigms-Soundgood

3. Brief Explanation of SQL Scripts:

- database.sql: This script defines the schema, including tables, constraints, primary and foreign keys, and relationships between tables.
- data.sql: This script inserts values and test data into the database, including students, instructors, lessons and `lesson_price`. The data reflects various scenarios like group lessons and instrument rentals.

Discussion

Difficulties:

One of the struggles we had with this assignment was understand how much of the payment we needed to manage. At first we thought that we might need to save exactly how much the student needs to pay or the salary of the teacher. After a while, we figured out that doing so would be considered as storing redundant data. The reason for this is that you can run queries on the database and from the information, figure out the costs.

We had discussion around some of the constraints, specifically the UNIQUE constraint. We were uncertain how it exactly worked. After an hour long discussion we subsequently came to an agreement that the UNIQUE constraint meant that the tupples in a certain table could not have the same combination of values in the UNIQUE columns.

Criteria Analysis:

The Logical and Physical Model fulfills *Soundgood Music School's* requirements by making the essential business operations like student enrollment, lesson scheduling, instrument rentals etc.

1. Naming conventions:

- Naming conventions were strictly followed throughout the model. All tables and columns have clear, descriptive names that convey their purpose. For instance, tables like `student_lesson` and `lesson_price` are self-explanatory, while columns such as `valid_from` and `valid_to` indicate the managing of historical data.

2. Crow Foot Notation:

- Crow's Foot Notation used in (ER) diagrams to represent database schema. It helps in visualizing the relationships between different entities in a database system.
- Entities and Attributes: Entities has been added according to the requirements for (Soundgood Music) Attributes match the purpose for their entities and take care about the cardinalties which has many shapes such as, one to one or one to many and so on.

3. Normalization and 3NF Compliance:

- Third Normal Form (3NF). There are no transitive dependencies among non-prime attributes within any table. To be in 3NF, a schema must not have transitive dependencies where non-prime attributes depend on other non-prime attributes. Each attribute's dependency is either on the primary key or the whole of a composite key (for junction tables like student lesson).

4. Completeness of Tables and Columns:

The provided tables cover the basic needs for Soundgood music school system included student registration, lesson management, instrument rental, and instructor data. However, expanding the database to include more additional tables could be efficient if they match the School requirements. But for the current functions for those tables which are exist in ER diagram are satisfied and corresponded with the current requirements.

Every column required for storing relevant data is present. For example:

- The instrument table captures instrument details, while rental tracks transactions with constraints enforcing a maximum of two rentals per student.
- Constraints such as lesson capacity is implemented through attributes like `num_of_students`, `students_maximum and student_minimum`.

5. Primary Key Selection:

Primary keys were chosen to ensure uniqueness and efficient indexing. Composite keys, such as those in the `student_lesson` table (combining `student_id` and `lesson_id`), are used where needed to reflect natural relationships. All primary keys are unique and enforce data integrity.

6. Relationship and Multiplicities:

All relationships are relevant and correctly reflect the business processes at Soundgood. For example:

- The rental table connects student and instrument with the correct cardinality.
- Relationships such as student to `student_lesson` and lesson to `lesson_price` are clearly defined and support all required operations.

7. Business Rules and Constraints:

Constraints not visible in the diagram are clearly described in the implementation:

- Each student can rent up two instruments (maximum), enforced through SQL constraints.
- Sibling discounts are applied via the sibling table.
- Historical lesson prices are managed through the `lesson_price` table, ensuring accurate billing even if prices change.

8. ENUMs and Constants:

- All values which only have constant inputs, such as class type and the difficulty are ENUM's both in the model and database.
- It might be preferable to have the data type be VARCHAR, as the school can add or remove difficulty levels more freely.

Storing All Data in the Database:

- Advantages:
  - All data is centralized, making it easy to query and maintain.
  - Business rules, like rental limits and sibling discounts, are enforced at the database level, reducing the risk of errors.
- Disadvantages:
  - The database become slightly more complex to design and maintain, requiring careful attention to constraints and rules.
  - Some constraints, like sibling relationships, may increase the complexity of data entry.

Handling Historical Prices:

The `lesson_price` table stores multiple versions of lesson prices with `valid_from` and `valid_to` timestamps to manage historical prices.

- Advantages:
  - Accurate billing is ensured by maintaining historical prices, even if prices change before payment is made.
  - Flexibility is added, as prices can be updated without overwriting previous records.
- Disadvantages:
  - Querying current or past prices requires slightly more complex SQL queries to determine which price is valid at a given time.
  - Storage requirements increases slightly due to storing multiple price records for each lesson type and level.

Changes and Improvements:

Feedback we received from Leif:

When we first sent this assignment we received some feedback which we improved upon:

- 1. You actually came to the wrong conclusion about the UNIQUE constraint. Even if you specify multiple columns in the same table as UNIQUE, it still only means that the values in each column is unique within that column, not that combinations of columns are unique. To specify that combinations of columns are unique, the syntax is UNIQUE (column1, column2).
- 2. I appreciate your layout of the model, it's very easy to read the diagram.
- 3. I can't see that you store the values for max concurrent rentals (2) and max rental period (12 months) anywhere. You have to do that to get the higher grade points. If the values are stored and I just missed them, you can explain how you stored them instead of changing the database.
- 4. You've stored lesson prices in a very strange way. First, one lesson can have many different prices, which is not correct. Second, the same price must be stored again for all lessons of the same type and level.
- 5. If the column "availability" in the table "instrument" means how many instruments are available to rent, the type should be int. Also, there should be some record of individual instruments, the company will probably want to know where each specific instrument is (who has rented it, if anyone), not just how many are left in stock.

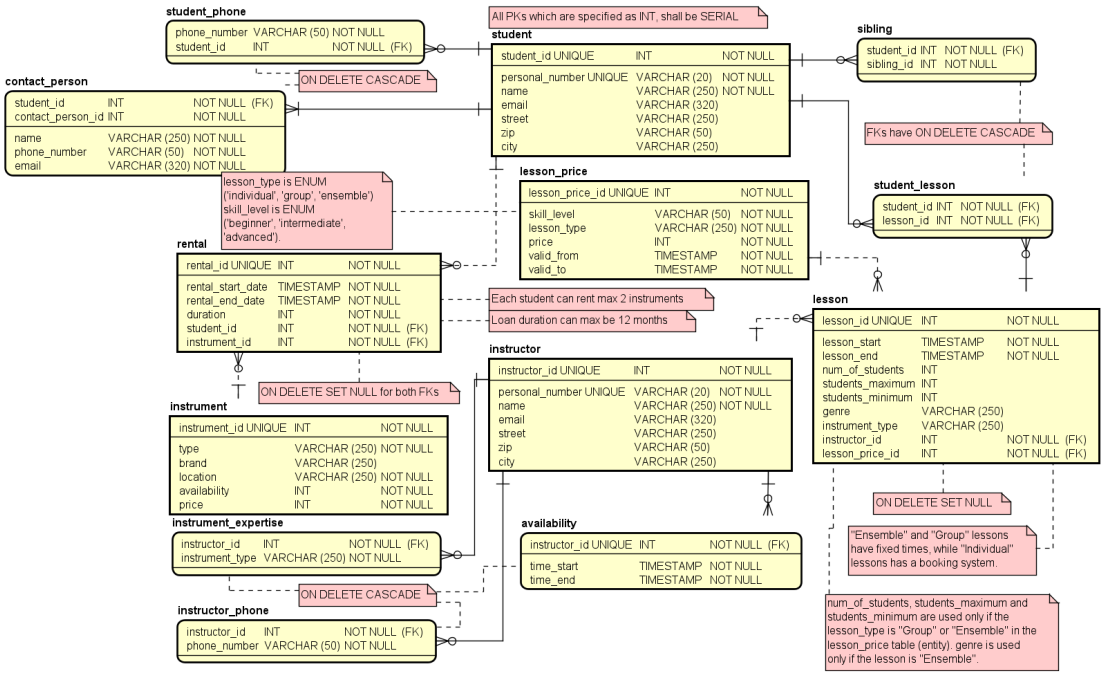

Figure 2: Updated Logical and Physical Model diagram

The changes we made to fix these problems were as follows:

- 1. We removed the UNIQUE constraint from the primary keys, as they were by definition UNIQUE. When you have two primary keys, they become a composite primary key, which means that the combined attributes are UNIQUE. Adding the UNIQUE constraint to the composite keys causes problems.
- 2. Thanks :)
- 3. This was not an issue, as we had limits in the form of comments in the model, and as constraints and triggers in the SQL database. (This bullet was approved by Leif).
- 4. We completely changed the attributes in the lesson_price table, as the structure before was inefficient and allowed for redundant data. We had individual prices for the skill level ("skill_level_price") and lesson type ("lesson_type_price"). In the new model we changed the "lesson_price" and "lesson" tables. We removed the columns "skill_level" and "lesson_type" from the lesson table and added them to the "lesson_price" table. We also removed the columns "lesson_type_price" and "skill_level_price" from the "lesson_price" table and added a new column "price", which contains a combined price for the skill_level and lesson_type in the lesson_price table.
- 5. We changed the attribute type from VARCHAR to INT and added a new attribute called "location", with the type VARCHAR, which can be used to describe where in the school the instrument can be found (if it is not rented at the moment). For example "Room A1, shelf 3, row 2" or whatever system they want to use.