Building the Normalized Database

After completing the normalization task, where a messy, denormalized student dataset was transformed into Third Normal Form (3NF), the next step was to bring this schema to life in a real database system. I chose Supabase, a cloud-hosted PostgreSQL platform, for its developer-friendly tools, live SQL editor, and real-time capabilities. This task allowed me to apply database theory in practice and gave me insight into collaborative, production-level database design.

From Normalized Tables to Supabase Schema

The original dataset contained repeated and nested fields, such as multiple course names, teachers, and exam boards listed in the same row. To make this suitable for analysis and integration, I broke it down into six distinct tables during the normalization process:

- students
- courses
- teachers
- exam_boards
- course_details (each course is linked to one teacher and one board)
- student_courses (many-to-many relationship linking students and courses)

With the normalized model in place, I moved to Supabase and created the schema using SQL. Each table was manually defined using the platform's SQL editor. For example, the students table was created like this:

```
CREATE TABLE students (
    student_number INTEGER PRIMARY KEY,
    student_name   VARCHAR(100),
    exam_score     INTEGER,
    support        BOOLEAN,
    date_of_birth  DATE
);
```

Similarly, I created courses, teachers, and exam_boards with simple primary keys, and then defined course_details and student_courses to establish relationships. Here's how I ensured that each course linked to exactly one board and one teacher:

```
CREATE TABLE course_details (
    course_id  INTEGER PRIMARY KEY REFERENCES courses(course_id),
    board_id   INTEGER NOT NULL REFERENCES exam_boards(board_id),
    teacher_id INTEGER NOT NULL REFERENCES teachers(teacher_id)
);
```

And to link students with their enrolled courses:

```
CREATE TABLE student_courses (
    student_number INTEGER REFERENCES students(student_number),
    course_id      INTEGER REFERENCES courses(course_id),
    PRIMARY KEY (student_number, course_id)
);
```

This structure enforced referential integrity and avoided redundant data.

UML Diagram

The following diagram represents the final structure of the database:

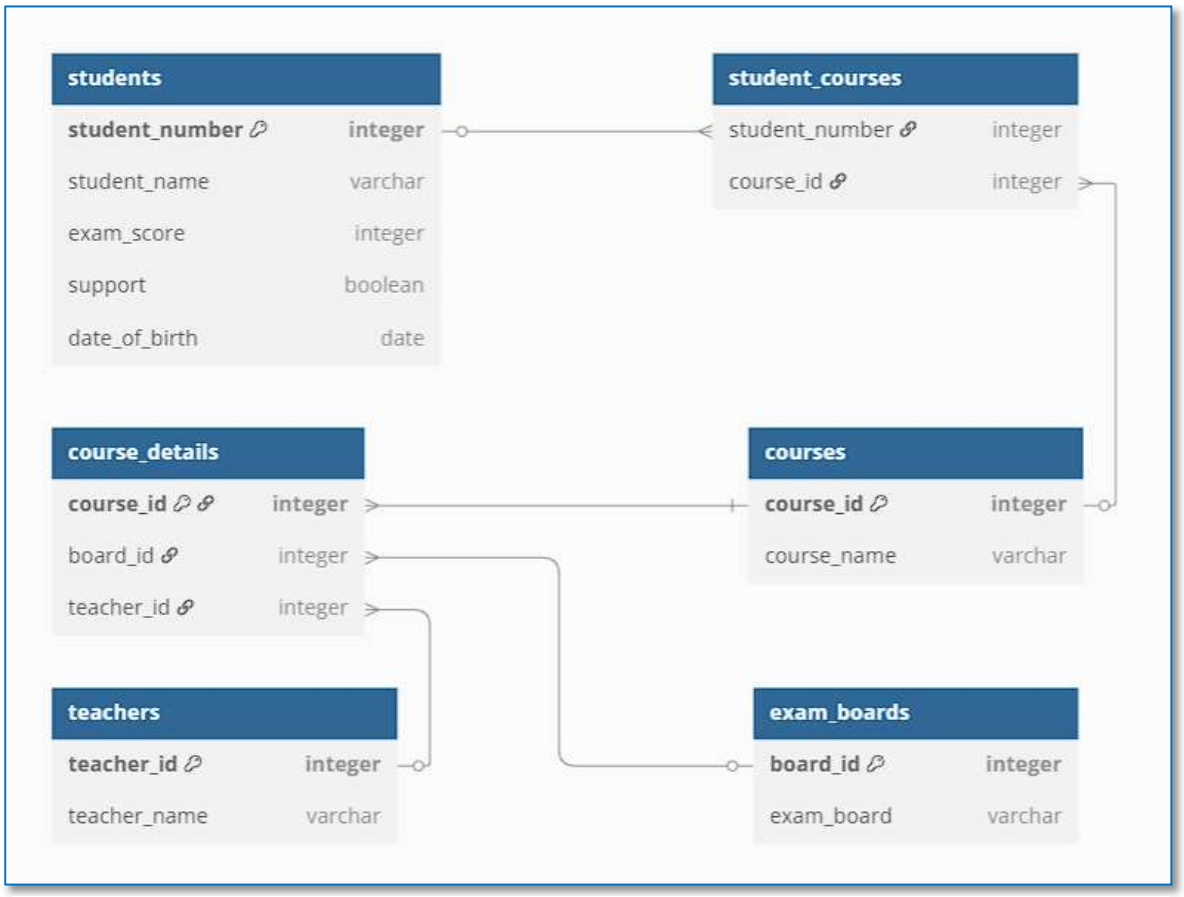

Figure 1 - Schema UML Diagram

Implementation Process on Supabase

Creating the Tables

Each table was defined using SQL CREATE TABLE statements. I carefully chose data types—for instance, using BOOLEAN for the support flag and DATE for birth dates—to ensure accuracy and consistency.

Inserting Data

I added sample data manually using INSERT INTO statements across all six tables. For example:

```
INSERT INTO teachers (teacher_id, teacher_name) VALUES (1, 'Mr Jones');
INSERT INTO courses (course_id, course_name) VALUES (1, 'Maths');
```

Testing Queries

I ran several queries to verify the integrity and analytical value of the schema. One key query brought together students, their courses, and the associated teacher and exam board:

```
SELECT s.student_name, c.course_name, eb.exam_board, t.teacher_name
FROM student_courses sc
JOIN students s ON sc.student_number = s.student_number
JOIN courses c ON sc.course_id = c.course_id
JOIN course_details cd ON cd.course_id = c.course_id
JOIN exam_boards eb ON cd.board_id = eb.board_id
JOIN teachers t ON cd.teacher_id = t.teacher_id;
```

Other queries included:

- Counting how many students were enrolled in each course
- Listing top-performing students per course using ORDER BY and LIMIT

Reflections on Learning Outcomes

This project challenged me to turn disorganized data into a clean, structured format that could support real analysis. I began by identifying patterns of redundancy and inconsistency, then broke the data down into well-defined, normalized tables.

Supabase made it straightforward to implement this structure in a live environment. Its PostgreSQL backbone allowed me to create tables, define constraints, and test for data integrity. For example, when I tried to insert a student-course link referencing a non-existent course, the system blocked it—confirming that referential integrity was working.

This wasn't just about building a database. Writing queries that pulled together student, course, teacher, and board data helped show how useful thoughtful design can be. Every part of the schema had a purpose, and the queries were simpler and more effective because of it. Documenting my process and building the schema from scratch strengthened my understanding of data wrangling and design. It gave me direct, hands-on experience with turning raw information into something usable and scalable.