Lab - 11: Database design

Team Members:
Jay Sanjaybhai Patel (jy451478@dal.ca)
Kenil Kevadiya (kn486501@dal.ca)

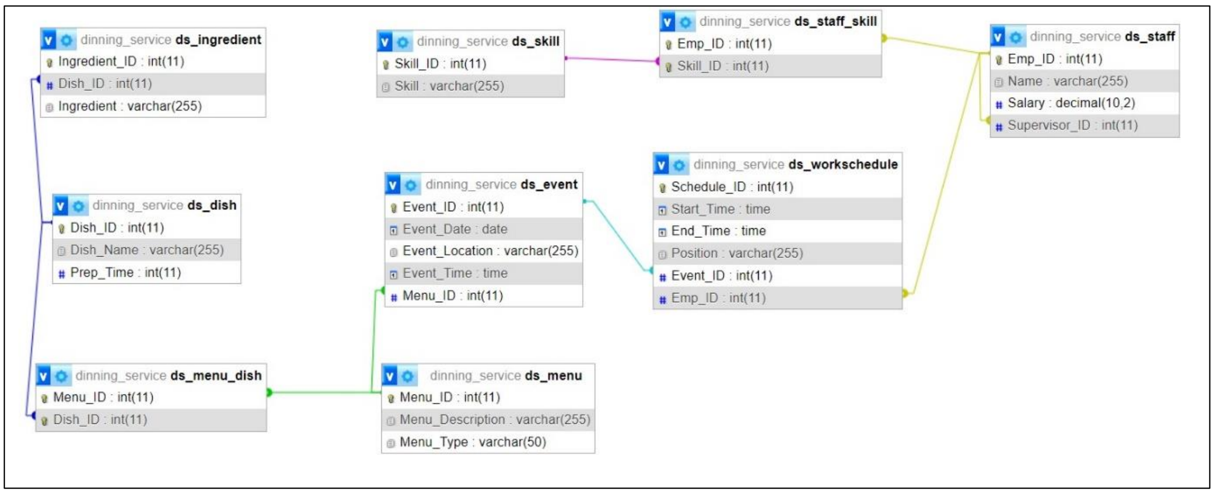
Date:
April 03, 2024

Subject:
Software Development Concepts

Professor:
Michael McAllister

Part 1

```
CREATE TABLE ds_menu (
  Menu_ID INT AUTO_INCREMENT PRIMARY KEY,
  Menu_Description VARCHAR(255) NOT NULL,
  Menu_Type VARCHAR(50) NOT NULL
);

CREATE TABLE ds_dish (
  Dish_ID INT AUTO_INCREMENT PRIMARY KEY,
  Dish_Name VARCHAR(255) NOT NULL,
  Prep_Time INT NOT NULL
);

CREATE TABLE ds_ingredient (
  Ingredient_ID INT AUTO_INCREMENT PRIMARY KEY,
  Dish_ID INT NOT NULL,
  Ingredient VARCHAR(255) NOT NULL,
  FOREIGN KEY (Dish_ID) REFERENCES ds_dish(Dish_ID)
);

CREATE TABLE ds_menu_dish (
  Menu_ID INT NOT NULL,
  Dish_ID INT NOT NULL,
  PRIMARY KEY (Menu_ID, Dish_ID),
  FOREIGN KEY (Menu_ID) REFERENCES ds_menu(Menu_ID),
  FOREIGN KEY (Dish_ID) REFERENCES ds_dish(Dish_ID)
);

CREATE TABLE ds_event (
  Event_ID INT AUTO_INCREMENT PRIMARY KEY,
  Event_Date DATE NOT NULL,
  Event_Location VARCHAR(255) NOT NULL,
  Event_Time TIME NOT NULL,
  Menu_ID INT NOT NULL,
  FOREIGN KEY (Menu_ID) REFERENCES ds_menu(Menu_ID)
);

CREATE TABLE ds_staff (
  Emp_ID INT AUTO_INCREMENT PRIMARY KEY,
  Name VARCHAR(255) NOT NULL,
  Salary DECIMAL(10, 2) NOT NULL,
  Supervisor_ID INT,
  FOREIGN KEY (Supervisor_ID) REFERENCES ds_staff(Emp_ID)
);

CREATE TABLE ds_skill (
  Skill_ID INT AUTO_INCREMENT PRIMARY KEY,
  Skill VARCHAR(255) NOT NULL
);

CREATE TABLE ds_staff_skill (
  Emp_ID INT NOT NULL,
  Skill_ID INT NOT NULL,
  PRIMARY KEY (Emp_ID, Skill_ID),
  FOREIGN KEY (Emp_ID) REFERENCES ds_staff(Emp_ID),
  FOREIGN KEY (Skill_ID) REFERENCES ds_skill(Skill_ID)
);

CREATE TABLE ds_workschedule (
  Schedule_ID INT AUTO_INCREMENT PRIMARY KEY,
  Start_Time TIME NOT NULL,
  End_Time TIME NOT NULL,
  Position VARCHAR(255) NOT NULL,
  Event_ID INT NOT NULL,
  Emp_ID INT NOT NULL,
  FOREIGN KEY (Event_ID) REFERENCES ds_event(Event_ID),
  FOREIGN KEY (Emp_ID) REFERENCES ds_staff(Emp_ID)
);
```



Database design created using XAMPP

Part 2

In the first figure, only one table is used to store the entire dataset, and according to our analysis, it is not suitable for this data. Moreover, the first figure is in the first normal form.

Advantages:

- Easy to read (Simplicity): Anyone can read the data, and since it has only one table, it will be easy for them to understand.
- Easy to access as it has only one table: Complex queries like joins are not required to fetch any information.

Disadvantages:

- Redundant data: As the first normal form (1NF) of the database only contains atomic values, there is redundancy in the data. For example, student names often have both a first name and last name, resulting in redundant data because the surname information is repeated for each row.
- Difficult to update data: Insertion, update, and deletion anomalies are more likely to occur, making it difficult to maintain data integrity. For instance, we can't add a student until they enroll in at least one course. Additionally, if there are changes in office locations for certain subjects, many rows need to be updated. Lastly, if there are no students enrolled in a particular course, we can't retrieve details of that course using this structure.
- Unnecessary retrieval of data and inefficient query: On one hand, if we want unique student names, and on the other hand, if we want unique office keys, in both cases we have to scan a similar number of rows.

Reason for choosing this database:

- This database structure is suitable only for limited volumes of data. Otherwise, it becomes very difficult to scale using this approach.

The second figure is in a higher normal form compared to the first figure. Here, we have less redundant data, and updating data is easier.

Advantages:

- Updating, inserting, and deleting records become easier and less error-prone.
- Queries can be optimized to access only the necessary tables, resulting in faster query execution times.
- Provides a flexible foundation for database growth and scalability.

Disadvantages:

- Redundant Data: In the studentSupplementals table, we have SupplementalCourses for each student. If, for example, StudentID = 1 takes 3 courses, then we have to add all three course IDs into one column, which is not a good database design. Now, if 10 more students take the same three courses, we have redundant data in all 11 rows.
- Loss of Data: If there are no students in one particular course, then we can't get any information regarding that course.
- No supervisor exists if there are no students in the thesis program.

Reason for choosing this database:

- If we have a medium to large database and we are not too concerned about the supervisor and courses that do not exist without students, then we can use this design.
- It has better segregation than the first figure.

The third figure is in the highest normal form and has no disadvantages, as it handles only one piece of information in one particular table, ensuring optimal database design.

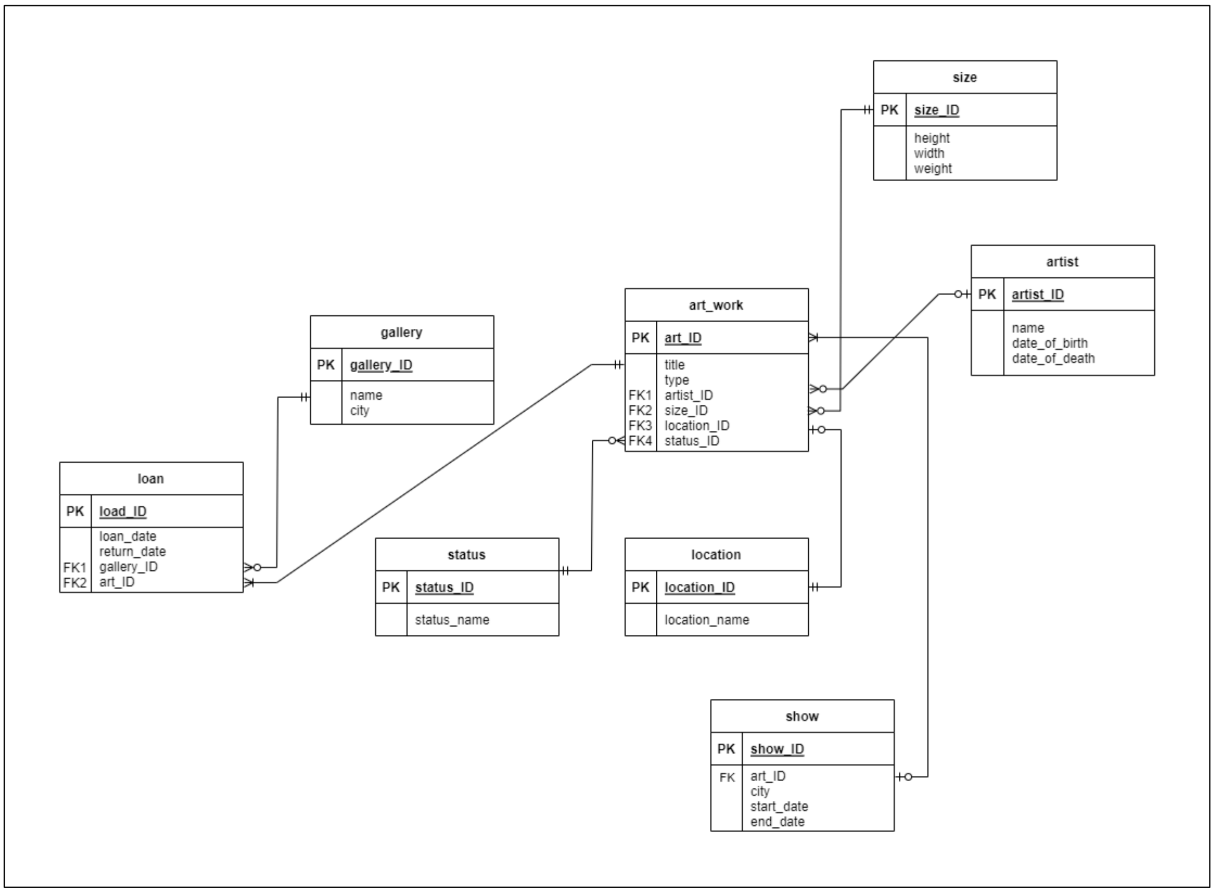
Advantages:

- No chance of data redundancy, ensuring data integrity and consistency.
- Free from update, delete, and insert anomalies, maintaining database reliability.
- Efficient JOIN operations for seamless data retrieval and analysis.
- Efficient and logical schema leads to better understanding and maintenance of the database structure.
- Easily scalable and delivers high performance using optimized queries, enhancing overall system efficiency.

Reason for choosing this database:

- If we have a large database, we must follow a normalized form, which helps minimize data loss while maximizing performance. Moreover, we can easily modify or add tables in this form, maintaining database consistency.

Part 3



Created using drow.io