Database Design

The purpose of this exercise is to practice designing and implementing a database from scratch based on an initial set of requirements.

Learning Objectives

After completing this exercise, students will understand:

- How to create a normalized database based on a set of requirements.
- The differences between One-to-Many and Many-to-Many relationships, as well as how to define them.
- How to implement associative entities to appropriately resolve Many-to-Many relationships.
- How to use Data Definition Language (DDL) to create tables and relationships.

Evaluation Criteria & Functional Requirements

- All of the DDL and DML statements run as expected.
- All tables and relationships necessary to satisfy the requirements are defined.
- Design is normalized appropriately.
- The correct data types are used.
- Necessary constraints have been defined.
- Code is clean, concise, and readable.
- Your database script runs correctly, and your database is created as a result of running the script.
- The correct number of rows are represented in each of the tables based on the requirements.

Implement a database to hold the contents of a simple Project Organizer.

Create a database called **project_organizer** and a connection to the data base in dbVisualizer.

The Project Organizer tracks which department and individual employees are working on a given project.

Each Employee has the following attributes:

- Employee Number
- Job Title
- Last Name
- First Name
- Gender
- Date of Birth
- Date of Hire
- Department the Employee Works For

Each Department has the following attributes:

- Department Number
- Name
- Has zero-to-many Employees

Each Project has the following attributes:

- Project Number
- Name
- Start Date
- Has zero-to-many Employees

Requirements

1. Create a database called **project_organizer** and a connection to the data base in dbVisualizer.
2. All tables are required to have a primary key.
3. Populate the tables with data for at least four projects, three departments, and eight employees.
4. Make sure each project has at least one employee assigned to it, and each department has at least two employees.

Design Tips

There are several questions you should ask yourself:

- Is a natural key available for the primary key, or will you need a surrogate?
- What data type should be used for each column?
- Which attributes are required? Which are optional?
- Are there additional constraints such as length, data format, or restricted values for any of the columns?
- What table columns will the user likely search?
- Is there an official list of valid job titles?

Don't forget to normalize.

Getting Started

- Create a database called **project_organizer** and a connection to the data base in dbVisualizer.
- Get a pen and paper or a whiteboard and a whiteboard marker.
- Write down the list of entities you believe should be represented in the database.
- Draw lines between the entities to represent the relationships between them.
- Create a file called `projects.sql` in dbVisualizer for your **project_organizer** database.
- Inside of the file, take the database you designed in the first two steps, and begin adding the DDL statements to implement the design.
- Add DML statements to populate the tables with the required data.
- Create your database by running the script you created.

Tips and Tricks

- While you could start by writing your DDL statements directly, it is typically better to create a diagram first. Database diagrams make it easier to visualize relationships between tables, and typically, using a whiteboard or pen and paper is cheaper and more efficient than writing code first.
- There are many varying levels of normalization a database design can achieve. How might you know if your database is normalized enough?
- Remember: Many-to-Many relationships require an additional entity, often referred to as an associative entity, to correctly define the relationship. If you didn't do this, which table needs to carry the foreign key reference?

- How do you know which table actually needs to hold the reference to the other table? The "crow's foot" is often a good indication as to which table the reference column belongs in. For instance, consider an artist and a song. For simplicity purposes, assume that a song belongs to one artist. That being said, you might say that "A Song must belong to one Artist, and an Artist may have many songs." The talons of the crow's foot would point at the Song table. This is a good indication that a column should be created on the Song table that would allow you to specify the Artist the Song belongs to.
- In the previous example, how might the design be impacted if you decide to allow a Song to belong to multiple Artists?
