An Exercise in Rudimentary Structured Query Language Application

Alex Wilson

[WillsonAlex@cityuniversity.edu](mailto:WillsonAlex@cityuniversity.edu)

City University Seattle

**Abstract**

Structured Query Language (SQL) is the means by which one can manage a pool of data, known as a database, by way of logical ‘queries’ to create, retrieve, organize and store data. The methodology by which this is done can range from very simple, to very complex.

**Keywords:** SQL, Joins, Aggregate, Views, Business

**1.** **SCENARIO**

The scenario for this paper will be creating and demonstrating management of a small database for a small company made up of four departments, each with a manager, and a total employee count of 12. The sequence of events for this project are as follows:

1. Establish the database
   1. Create a Database
   2. Switch to that database
   3. Create a Schema
   4. Create Tables within that Schema
   5. Populate those tables with data
2. Create Views
   1. Aggregate view
   2. Left Join View
3. Create Store Procedure
   1. Display all employees

**2. BACKGROUND**

**1. Establishing the Database**

Before conducting any coding, Team 2 worked up an excel workbook of the information to be included in the database (dB) to as a visual aid to establishing it. The small dB consisted of three tables falling under one schema: a Manager’s table, an Employees table, and an HR table. The tables’ primary keys/foreign keys were the employee ID column (empid).

**a. Create a Database**

The first and easiest step in create a database is to simply create the database, like so:

CREATE DATABASE teamTwo;

GO

This will create a database called ‘teamTwo’.

**b. Switch to that database**

Before you can do anything to this brand new database, you must ensure that you operating within that database. Most graphical user interfaces (GUI) for coding SQL will have a way to toggle between databases; but in order to make sure, it’s a good ‘best practice’ to run the command to ensure it:

USE teamTwo

GO

**c. Create a Schema**

Now that we’re operating under our database, we’ll need a Schema under which to create tables. The Schema in SQL acts as the architecture upon which all of the subsequent data presides in inter-relational fashion. Since we are establishing a small database for a company, we’ll simply call the Schema ‘company’:

CREATE SCHEMA company

GO

**d. Create Tables within that Schema**

If data is food, then tables are the …well, tables upon which the data rests. They are made up of rows and columns much like what you would see in any excel spreadsheet. The columns specify the category of the data, while the rows are different instances containing values that satisfy the categories specified in the columns.

For our database, we’ll create three tables: Employes, Managers and HR.

The queries for all three tables will look something like this:

CREATE TABLE company.Employees (

empid INT NOT NULL,

lastname NVARCHAR(20) NOT NULL,

firstname NVARCHAR(10) NOT NULL,

hiredate DATETIME NOT NULL,

mgrid INT NULL,

CONSTRAINT PK\_Employees PRIMARY KEY(empid),

CONSTRAINT FK\_Employees FOREIGN KEY(mgrid)

REFERENCES company.Employees(empid),

CONSTRAINT CHK\_hiredate CHECK(hiredate <= CURRENT\_TIMESTAMP)

);

The *Employees*table consisted of the all the employees ID, first name, last name, the ID number of their manager, the department they work in and their phone number.

The *Managers*table consisted of the employee ID of the managers, the first name of the managers, the last name and their respective departments.

The *HR*table consisted of consisted of the Employee ID (without the names), the zip code of each employee, their salary and the date they were hired.

The tables will relate to each other by way of something called the Primary Key. As you can see in the example above, we’ll be using the ‘empid’ column to relate all of the tables, since all of the tables have this column in common. As we create the tables, you can see that we also specify what data type each column can accept. eg. Integer, String, Date, etc.

**e. Populate those tables with data**

Now that we’ve created our database, schema and tables, we need to fill them with data. We do this by way of an insert statement. The syntax looks like this:

INSERT INTO company.Employees (empid, lastname, firstname, hiredate, mgrid) VALUES (1, N'Sky', N'Serena', '19900909 00:00:00:000', NULL);

After ‘INSERT’, we reference the schema and table we’re inserting into: ‘company.Employees’. Next, in the parenthesis, are the column names. Finally, after ‘VALUES’, also in parenthesis, are the actual values being input into the row. We’ll repeat this process twelve times for the Employee table for each of the 12 employees in the company. We’ll do a similar process to fill the data in the other tables.

**2. Create Views**

Once the tables have been established, a common practice with SQL is to create what is called a ‘View’. A view is a way to produce a virtual table from the data contained on the database’s tables. The date can be pulled from more than one table and can restrict the amount of data seen by someone invoking the view.

**a. Aggregate view**

An aggregate view is a view that utilizes an aggregate function, that is to say one or more of several function in SQL that aggregate (calculate) data together and return a single value. For our database, we’ll display all of our employees by their employee ID and their salaries, and then order them from high to low based on the Salary:

Create VIEW company.employeesRanking AS

SELECT empid,

salary,

RANK() OVER ( ORDER BY salary DESC ) AS emprank

FROM company.HR

**b. Left Join View**

Another version of a view that we can demonstrate uses more than one table. This view will display information from the ‘Employees’ and ‘Managers’ table, in order to show all employees names next to their respective managers’ names.

CREATE VIEW company.employeesmanagers AS

SELECT e.firstname AS empfname, e.lastname AS emplname, m.firstname AS manfname, m.lastname AS manlname

FROM company.Employees AS e

LEFT JOIN company.Managers AS m

ON e.mgrid = m.empid;

**3. Create Store Procedure**

Lastly, we’ll talk about Stored Procedures. A stored procedure is simply assigning a query (aka process aka procedure aka function) to a variable, much like establishing variables in many other coding languages. This is to help prevent having to type, potentially, hundreds of lines of code over and over again; instead you can simply execute the procedure by calling on whatever you’ve named it.

**a. Display all employees**

To demonstrate Stored Procedure, we’ll do a simple procedure to display all of the employees that work at the company ordered alphabetically by first name:

CREATE PROCEDURE SelectAllEmployees

AS

SELECT firstname, lastname FROM company.Employees

ORDER BY firstname;

alphabetically Now, anytime you would like this alphabetical list, you can simple execute the procedure with the ‘EXEC’ keyword:

EXEC SelectAllEmployees

**3.** **REFERENCES**

* Darmawikarta, D. (2014). [SQL for MySQL: A Beginner’s Tutorial](https://login.proxy.cityu.edu/sso/skillport?context=63328). Brainy Software Corp. (ISBN 9780980839678)
* W3schools.com