# Lab– SQL Programming Views, Stored Procedures, Functions

## Overview

In this lab, we will practice using SQL program units (views, stored procedures, and functions) to create the external data model. We will use the following concepts from class:

* Views.
* Stored Procedures
* Functions
* The Execute command to run a stored procedure

### Learning Objectives

Upon completion of this learning unit you should be able to:

* Describe views, functions, stored procedures, and triggers.
* Explain the importance of procedural language constructs in the SQL environment.
* Demonstrate how to solve business problems with SQL programming.
* Describe the many advantages of the external data model.

### Lab Goals

This lab consists of 3 parts:

1. The first part will set the stage for the lab, giving you the background you need to understand the external portion of the data model.
2. In the second part, you will mainly respond by following along and typing in the provided SQL statements.
3. In part three, you will have to solve problem using SQL based on the provided requirements. Much of the details are left for you to figure out on your own. ☺

Once again, you will have to hand in pieces from parts 2 and 3 on this week’s learning assessment.

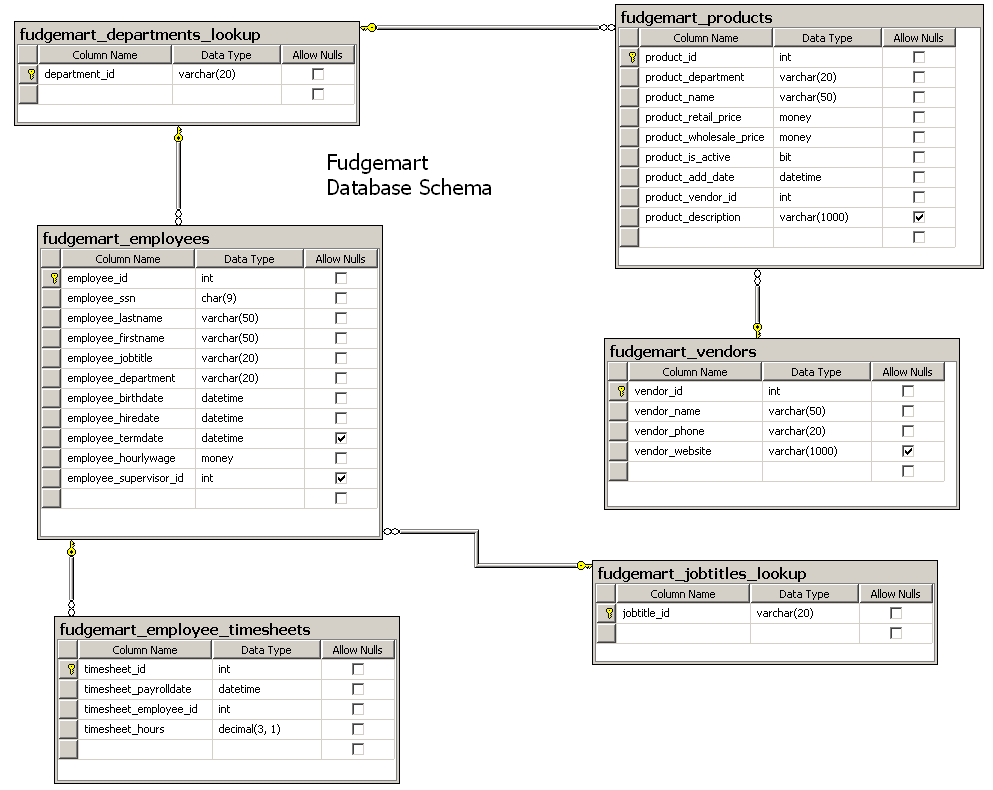
### What you will need to begin

1. Connect to your SQL Server instance.

## Part 1: The Fudgemart Database Schema Review

Throughout the semester we will use several different case studies to help reinforce the concepts we learn in class. One of the recurring case-studies we will use in class and the labs is the Fudgemart database. This database supports the business operations of a fictitious mega-store retailer and retailer called Fudgemart. The Fudgemart database supports all aspects of the business from human resources and payroll to sales transactions, and e-commerce. In each lab we will add new database objects and data to the Fudgemart schema.

### 1A: Fudgemart Internal Model (Schema)

By now, you’ve seen this diagram a few times and should be quite familiar with it. The reason we call this an ***internal model*** is because it represents how we actually store data for Fudgemart. There’s one critical element the internal model *doesn’t show us* - how our applications use the Fudgemart schema to store and retrieve data. This is the responsibility of the **external model** and the topic of today’s lab.

### 1B: Fudgemart External Model

The external data model represents the interface applications will use to store and retrieve data in the database. It is an ***abstraction*** of the internal model; hiding the complexity from the developers who are tasked with building the user-interface for the database.

The following table outlines a sample subset of a typical external model which might be required for the Fudgemart database. Notice how the external model looks more like business processes than procedures or rules.

In the external model implementation, we use:

* **Stored Procedures** to represent the business logic or processes in our database, like ***Fill Out Timesheet***, or ***Give Department Raise***. Stored procedures insert, update, or delete data from multiple tables.
* **Views** to represent logical displays of information, like ***Vendors without products*** or ***Employees and their Managers***. Views are stored SELECT statements.
* **Functions** to perform custom calculations over our data, like ***Total Employee Hours Worked*** or ***Vendor Product Count***. Functions are like procedures, but they return one value, and can be used in SQL SELECT statements.

Here’s an Example of the Fudgemart External Model. Some of these portions we will implement in this lab:

|  |  |  |
| --- | --- | --- |
| Scope | Task (What it does) | SQL Object Name |
| Fudgemart Employees | Add new employee | p\_fudgemart\_hire\_employee |
|  | Give Raise to Department | p\_fudgemart\_give\_raise\_to\_department |
|  | Terminate employee | p\_fudgemart\_terminate\_employee |
|  | Total Employee Hours Worked | f\_fudgemart\_total\_hours\_worked |
|  | List of Employees and their Managers | v\_fudgemart\_employees\_and\_managers |
| Fudgemart Products | Add A new Product | p\_fudgemart\_add\_product |
|  | Increase Retail Price For Department | p\_fudgemart\_increase\_retail\_price\_for\_dept |
|  | Delete product | p\_fudgemart\_delete\_product |
|  | De-Activate product | p\_fudgemart\_deactivate\_product |
|  | Display active products | v\_fudgemart\_display\_active\_products |
| Fudgemart Vendors | Vendor Product Count | f\_fudgemart\_vendor\_product\_count |

**NOTE:** Do **NOT** attempt to create the external model at this time. This will be done in later portions of the lab.

## Part 3: On Your Own

In this part you will attempt to write SQL statements to create and then execute views, stored procedures and functions.

|  |
| --- |
| 3.a) Create an SQL view called **v\_fudgemart\_employee\_managers** which displays all the employee information for only those employees who supervise others. In other words if you’re not a supervisor, you should not appear in the list. **HINT 1:** You should try to write the SQL SELECT statement first, and when you think it’s correct execute it with the create view statement. Revisit the SUBSELECT from last week’s lab (the question where the employees hourly wage was greater than the average hourly wage). **Hint 2 (one word):** IN!  The correct SQL for the view logic should display output like this: |
| 3.b) Use **v\_fudgemart\_employee\_managers**  to write an SQL statement which displays name, title, department, and wage of all supervisors making more than $17/hr who are not the CEO. Sort by hourly wage, like this: |
| 3.c) Create a stored procedure titled **p\_fudgemart\_markup\_retail\_by\_department** which when given the following parameters:   * @dept varchar(20) * @amount money   The procedure updates the current retail price and add @amount for **all products** in that department @dept. |
| 3.d) Write SQL which uses the stored procedure you created in 3.c) to markup All **Clothing** by $2**.50** and all **Housewares** by $**3.75** **HINT:** you will need to execute the stored procedure *twice*.  Here’s a sample of the output so you can verify the stored procedure worked correctly. |
| 3.e) Create a stored procedure titled **p\_fudgemart\_deactivate\_product** which when given a product’s ID will then update that product so it is no longer active. |
| 3.f) Write an SQL script which uses the stored procedure you created in 3.e) to deactivate the ‘Slot Screwdriver’ and ‘Monkey-Wrench’ products you added in steps 2.b and 2.c.  **HINT:** Since the procedure accepts product\_id and not product\_name, you will need to first use a SELECT statement to retrieve the product\_id by searching the product’s name, and store that in a variable. |
| 3.g) Write an SQL SELECT statement **v\_fudgemart\_display\_active\_products** from 2.d to prove you deactivated the two products in 3.f) You can accomplish this by only showing the ‘Hardware’ department. Here’s my sample output… notice it’s missing ‘Slot Screwdriver’ and ‘Monkey-Wrench’ products: |
| 3.h) Write SQL to create a view called **v\_fudgemart\_vendors** which displays all rows and columns from fudgemart\_vendors and includes a column called vendor\_product\_count which calculates the number of products per vendor. Hint: think about aggregates from last week. |
| 3.i) Write an SQL SELECT statement that uses **v\_fudgemart\_vendors** to display all vendors supplying more than 10 products. Here’s sample output: |