|  |
| --- |
| **Overview of the Lab** |
| In this lab we preview a few advanced topics, namely star schemas, big data, embedded SQL, and database connectivity.  From a technical perspective, together, we will learn:   * to identify and understand the major parts of a dimensional star schema. * to use the ROLLUP extension to GROUP BY to query a star schema. * to identify and understand embedded SQL. * to simulate horizontal and vertical fragmentation and defragmentation for distributed databases. |

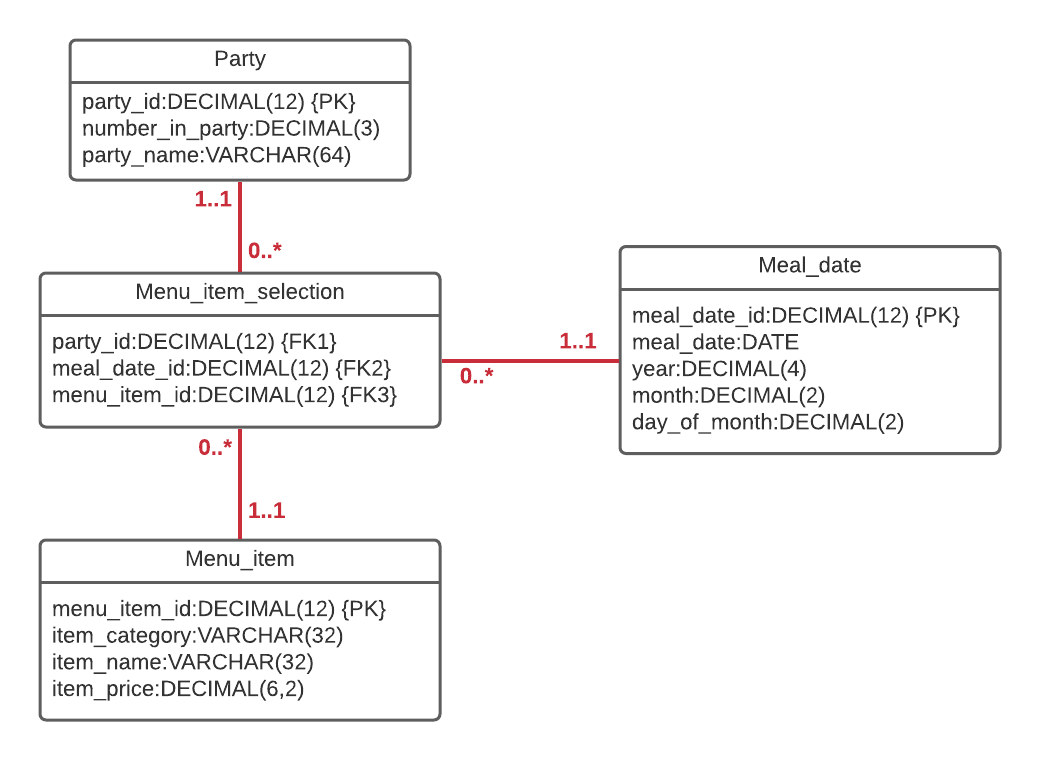
|  |
| --- |
| **Lab 6 Explanations Reminder** |
| As a reminder, it is important to read through the Lab 6 Explanation document to successfully complete this lab, available in the assignment inbox alongside this lab. The explanation document illustrates how to correctly execute each SQL construct step-by-step, and explains important theoretical and practical details. |

|  |
| --- |
| **Other Reminders** |
| * The examples in this lab will execute in modern versions of Oracle, Microsoft SQL Server, and PostgreSQL as is. * The screenshots in this lab display execution of SQL in the default SQL clients supported in the course – Oracle SQL Developer, SQL Server Management Studio, and pgAdmin – but your screenshots may vary somewhat as different version of these clients are released. * Don’t forget to commit your changes if you work on the lab in different sittings, using the “COMMIT” command, so that you do not lose your work. |

**Section One – Dimensional Modeling**

**Section Background**

Imagine that an organization uses a data mart to record restaurants’ transactions worldwide, and offer services such as reports and performance analysis to help optimize restaurants’ profitability. To capture the information, the organization provides an application the integrates with the restaurants’ point of sales systems, which then transmits the data back to the organization in near real time. This information is stored in the star schema below.



This star schema represents groups of people (parties) eating meals at restaurants. The schema is grained to each individual menu item selection for each meal. The schema is incomplete in that it does not represent all significant information that would be included for a complete picture. The following business rules help capture the restaurants’ workings.

* Parties of one or more people eat meals at restaurants.
* The restaurants ask each party to give a name associated with the party, in addition to the number of people in the party, when the party arrives (or when the party makes the reservation).
* Each party selects one or more items from the menu for their meal; the same item might be selected multiple times at the same meal if different people want to eat the same item.
* Every menu item has a category (such as “Entrée”, “Side”, “Dessert”, and so on).
* There are many restaurants, and each restaurant has their own name, location, and address.
* A waitperson serves a party.

The DDL to create the tables in the schema is listed below.

DROP TABLE Menu\_item\_selection;

DROP TABLE Party;

DROP TABLE Meal\_date;

DROP TABLE Menu\_item;

CREATE TABLE Party (

party\_id DECIMAL(12) NOT NULL PRIMARY KEY,

number\_in\_party DECIMAL(3) NOT NULL,

party\_name VARCHAR(64));

CREATE TABLE Meal\_date (

meal\_date\_id DECIMAL(12) NOT NULL PRIMARY KEY,

meal\_date DATE NOT NULL,

year DECIMAL(4) NOT NULL,

month DECIMAL(2) NOT NULL,

day\_of\_month DECIMAL(2) NOT NULL);

CREATE TABLE Menu\_item (

menu\_item\_id DECIMAL(12) NOT NULL PRIMARY KEY,

item\_category VARCHAR(32) NOT NULL,

item\_name VARCHAR(32) NOT NULL,

item\_price DECIMAL(6,2));

CREATE TABLE Menu\_item\_selection (

party\_id DECIMAL(12) NOT NULL,

meal\_date\_id DECIMAL(12) NOT NULL,

menu\_item\_id DECIMAL(12) NOT NULL,

FOREIGN KEY (party\_id) REFERENCES Party(party\_id),

FOREIGN KEY (meal\_date\_id) REFERENCES Meal\_date(meal\_date\_id),

FOREIGN KEY (menu\_item\_id) REFERENCES Menu\_item(menu\_item\_id));

**Section Steps**

1. *Identifying Essential Parts –* First, identify different parts of the star schema by completing the following.
   1. Identify the fact table and explain what event it represents.

The event in the given star schema is *menu selection*, which represents the menu items selected by each party and the date on which the party will be arriving. The fact table is Menu\_item\_selection table.

* 1. Identify the dimension tables and explain what event participant it represents.

There are three dimensions for the given schema: *Party, Meal Date,* and *Menu Item*. A party represents a group of people that will be attending the meal selection event. The menu item is a list of items available on the menu from which the party can choose their meal. The meal date is an abstract entity that represents the date on which the meal will be served.

* 1. Identify a hierarchy that exists in one of the dimension tables and explain what is represents.

One hierarchy from the Menu\_item dimension is item category – item name. If the item name is “applesauce”, it can be served both as a side and as a dessert. A person may order applesauce thinking it is a side dish but ends up receiving applesauce as a big bowl of dessert. The item category defines the context. There are many item names in each item category, so there exists one-to-many relationship.

Another hierarchy is year-month-dayofmonth in meal date dimension. A year has many months, and each month has many days, so there exists one-to-many relationship between them. The year defines the context for month and day. The month defines context for the day. To know someone’s age, their birth year is the context; to know when Christmas is each year, month is the context for the day.

1. *Adding a Dimension –* Next, identify and add in a dimension that is missing by completing the following.
   1. Review the business rules in the section introduction, identify a dimension that is missing, and explain.

Restaurant is a dimension that is missing in the given schema. While reviewing business rules, the following rule denotes Restaurant is missing in the schema.

“There are many restaurants, and each restaurant has their own name, location, and address.”

As every dimension has a grain, it is better to add Restaurant as another dimension in the schema.

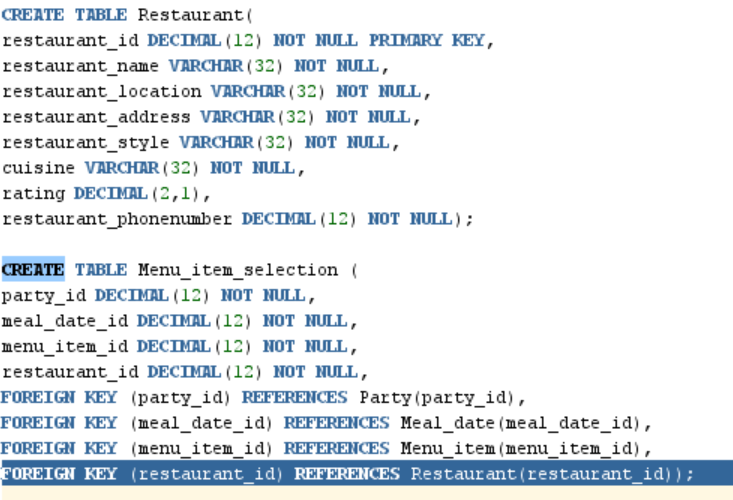
* 1. Explain what attributes and hierarchies this dimension would reasonably contain.

The business rule, “There are many restaurants, and each restaurant has their own name, location, and address.”, explicitly presents name, location, and address as some of the attributes. Additional attributes that can be added to Restaurant dimension are id, restaurant\_style, cusine, rating, and phone\_number.

One hierarchy in this dimension is location – address. For example, “Hamilton Town Center” is a location with many restaurants each having their own addresses. There can be many addresses associated with one location, so there exists one-to-many relationship.

* 1. Add the dimension into the schema by creating the dimension table in SQL along with its attributes and adding a foreign key to the fact table.

The *Restaurant* dimension table has been created and its primary key *resatuarant\_id* has been added as a foreign key in the fact table, Menu\_item\_selection.



1. *Adding a Measure –* Next, identify and make use of a useful measure by completing the following.
   1. As there are no measures in the schema, identify a useful one that could be added, and explain what it measures.

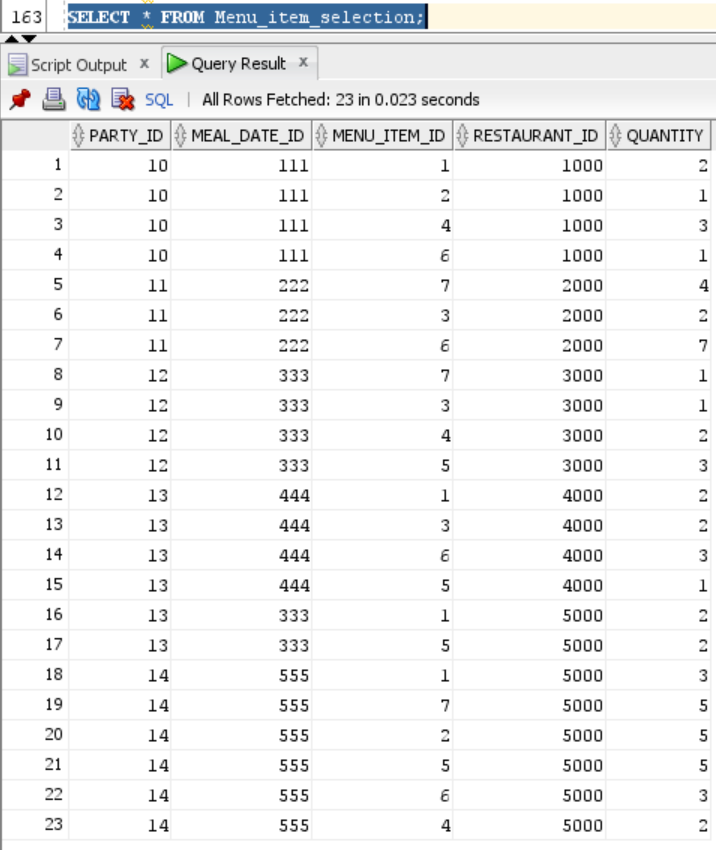
Quantity is a useful measure that can be used to check which menu item gets the most number of orders. This information provides information on the menu items that are liked the most and is likely to receive more orders as many parties may select these items.

* 1. In SQL, add the measure to the fact table.

ALTER TABLE Menu\_item\_selection

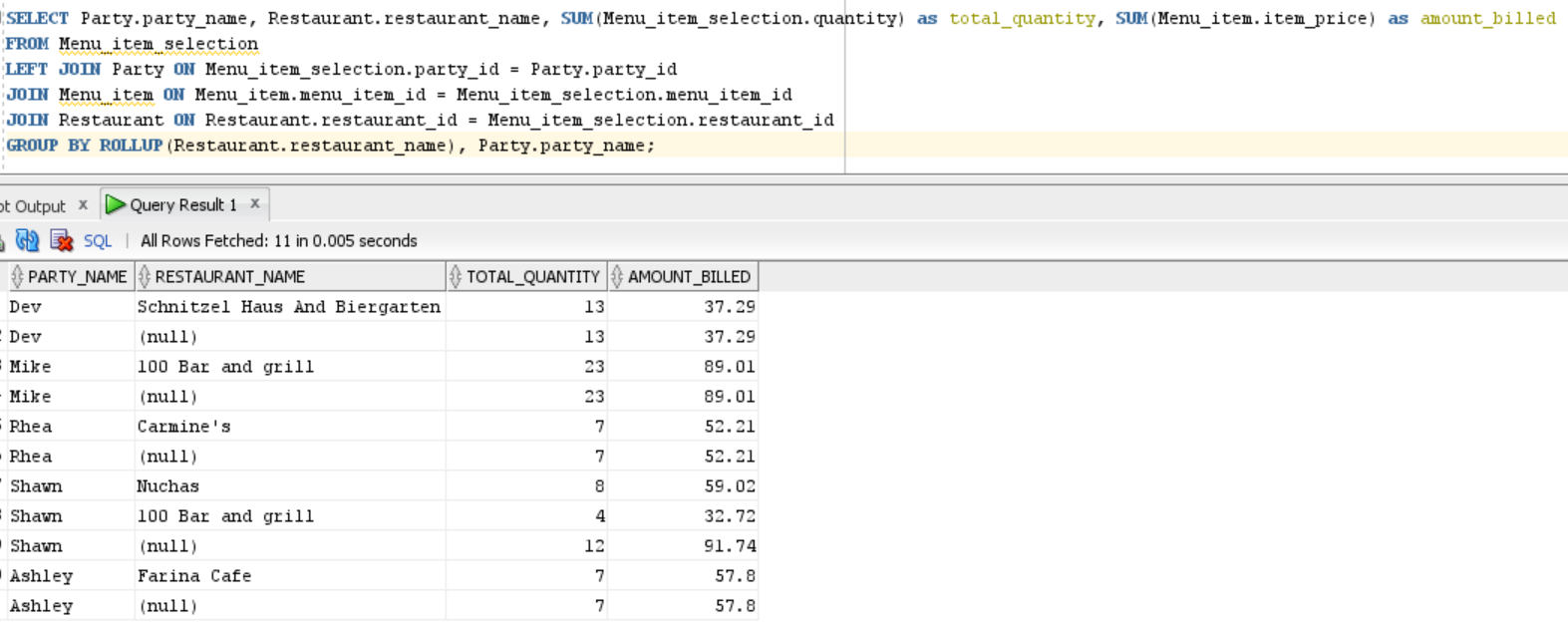
ADD quantity DECIMAL(3) NOT NULL;

* 1. In SQL, insert 15 rows of data into the fact table, along with the corresponding dimension rows. Make sure the data has some variety.



* 1. Write a query that uses the ROLLUP extension to GROUP BY, along with an aggregate function on the measure, to analyze some important aspect of the business. Explain what the results mean.

The following query analyzes the total number of items ordered (Quantity) by each party at a restaurant and their bill total. The result shows how many menu items each party ordered and how much each party was billed at each restaurant they visited. The ROLLUP(Restaurant.restaurant\_name) in the GROUP BY clause instructs SQL to display usual results output by GROUP BY, and output rows where the restaurant is not considered. The result also outputs the total items (Quantity) ordered by each party and the total amount billed to each party, regardless of restaurant.



**Section Two – Advanced Topics**

**Section Background**

*Embedded SQL* is SQL code embedded in a programming language. Embedding SQL allows an application to execute SQL as needed to store or retrieve data in the database. In this section, you explore some sample embedded SQL in a Java application.

A *distributed database* is one that uses multiple database instances that coordinate with one another to provide a complete logical view of a database schema. Each instance manages *fragments*, which are subsets of tables that together comprise complete logical tables. *Horizontal fragments* divide the table by row; *vertical fragments* divide the table by column; *mixed fragments* divide the table by row and column.

In this section, you explore some distributed database concepts by simulating fragments and communication between distributed database instances, through use of SQL.

**Section Steps**

1. *Understanding Embedded SQL –* Imagine the organization uses the following Java code on the star schema.

**String connectionUrl =**

**"jdbc:sqlserver://ip\_address:1433;"**

**+ "database=MyDB;"**

**+ "user=MyUser;"**

**+ "password=ABC123;";**

**Connection connection = DriverManager.*getConnection(connectionUrl);***

**Statement statement = connection.createStatement();**

**String sql =**

**"SELECT Party.party\_name, Menu\_item.item\_category, Menu\_item.item\_name "**

**"FROM Menu\_item\_selection " +**

**"JOIN Meal\_date ON Meal\_date.meal\_date\_id = Menu\_item\_selection.meal\_date\_id " +**

**"JOIN Menu\_item ON Menu\_item.menu\_item\_id = Menu\_item\_selection.menu\_item\_id " +**

**"JOIN Party ON Party.party\_id = Menu\_item\_selection.party\_id " +**

**"WHERE Meal\_date.meal\_date = CAST('01-APR-2021' AS DATE) " +**

**"ORDER BY Party.party\_id ";**

**ResultSet results = statement.executeQuery(sql);**

**while (results.next()) {**

**String party\_name = results.getString(1);**

**String item\_category = results.getString(2);**

**String item\_name = results.getString(3);**

**System.out.println("Party " + party\_name + " bought " + item\_name + " of type " + item\_category + ".");**

**}**

Java Code on Star Schema

1. 1. Identify and list out the embedded SQL query in this program, then explain what kind of results the SQL query obtains.

The following lists the embedded SQL query in the program:

**String sql =**

**"SELECT Party.party\_name, Menu\_item.item\_category, Menu\_item.item\_name "**

**"FROM Menu\_item\_selection " +**

**"JOIN Meal\_date ON Meal\_date.meal\_date\_id = Menu\_item\_selection.meal\_date\_id " +**

**"JOIN Menu\_item ON Menu\_item.menu\_item\_id = Menu\_item\_selection.menu\_item\_id " +**

**"JOIN Party ON Party.party\_id = Menu\_item\_selection.party\_id " +**

**"WHERE Meal\_date.meal\_date = CAST('01-APR-2021' AS DATE) " +**

**"ORDER BY Party.party\_id ";**

In Java, the + operator is used for concatenating strings. The concatenated string is one full query.

The result will display the name of the party, the item category, and the name of the item for the meal date booked for **01-APR-2021**. The result is ordered by the party\_id, so it displays the list with party\_id in the ascending order.

**ResultSet results = statement.executeQuery(sql);**

**statement.executeQuery(sql)** functionexecutes the SQL query and stores the results

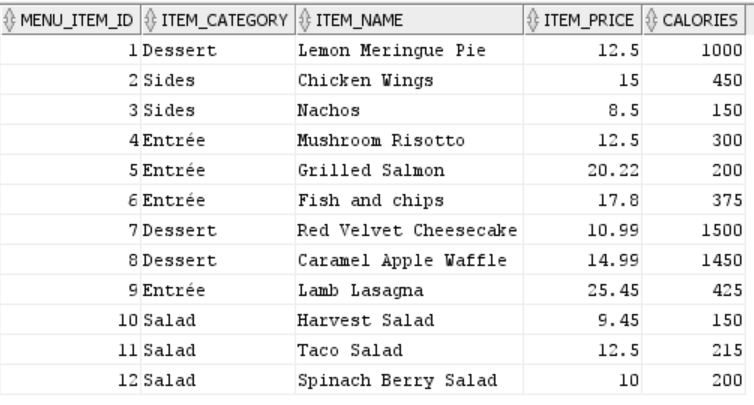
into the **results** variable.

The while loop loops through all the results and prints them out to the screen. The **results.getString()** function is used to retrieve value in each column as while loop iterates through each row.

* 1. What is the purpose of embedding this SQL into the program, as opposed to manually typing the SQL into a SQL client? Explain.

Embedded SQL is a set of SQL statements contained within an application programming language. Embedded SQL allows an application to accomplish a sequence of tasks that can be accomplished using traditional SQL statements. The applications operate the database and manipulate data using one of the host languages when SQL statements are embedded. If the host language is C++, it is possible to access a database from an application written in C++ by placing the SQL statements within the C++ code. The SQL statements remain the same regardless of the host language.

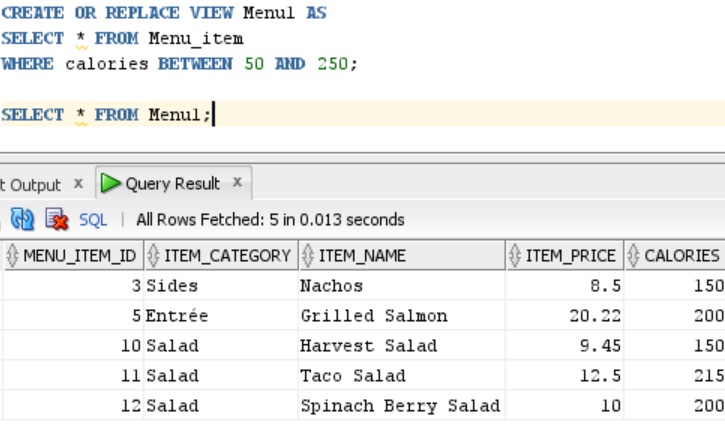
1. *Simulating Horizontal Fragmentation –* In this step, you simulate horizontal fragmentation and defragmentation. Complete the following substeps.   
     
   a. Create a table that has at least twelve rows and five columns. Make sure the table has a primary key.



b. Create three views (using the CREATE VIEW command) that simulate three horizonal fragments based upon some reasonable criteria. Show each view’s contents.

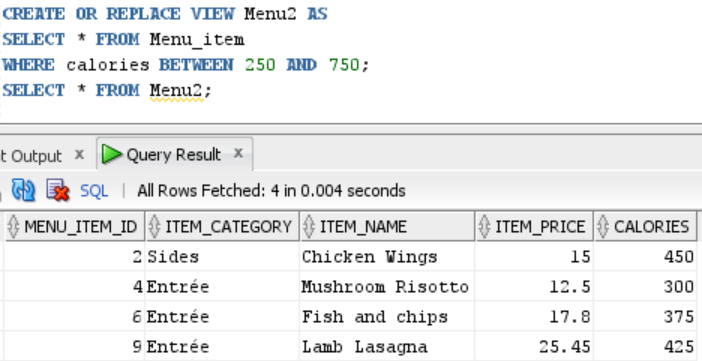
VIEW NO.1

This view fragments rows that fulfill the criteria in the WHERE clause. All the menu items with calories between 50 and 250 are displayed.



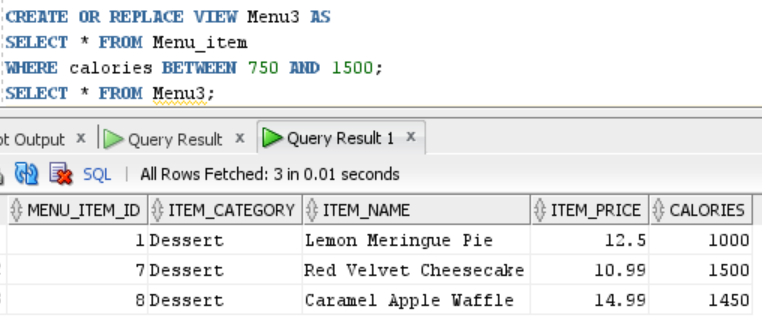
VIEW NO.2

This view fragments rows that fulfill the criteria in the WHERE clause. All the menu items with calories between 250 and 750 are displayed.



VIEW NO.3

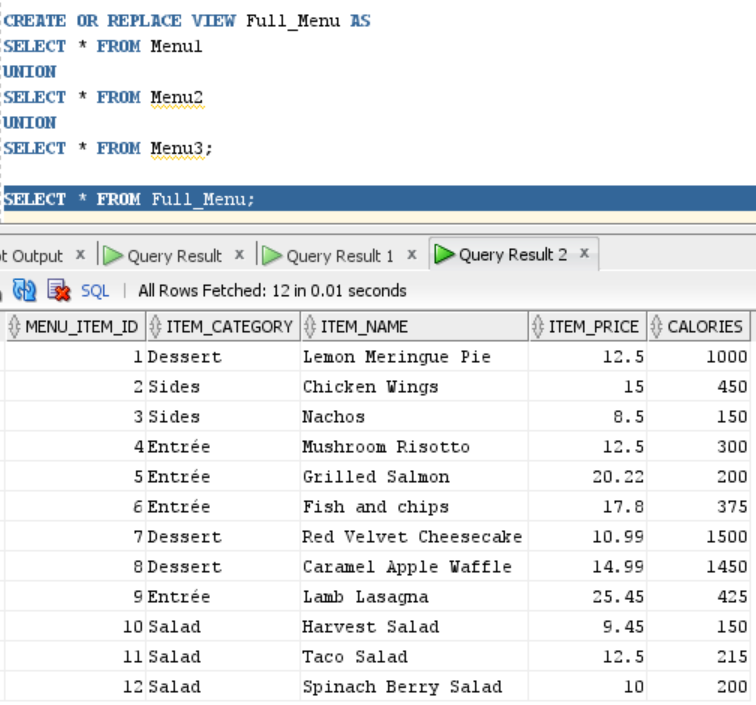
This view fragments rows that fulfill the criteria in the WHERE clause. All the menu items with calories between 750 and 1500 are displayed.



The first view has 5 rows, the second has 4 rows, and the third has 3 rows. Together, they add up to 12 rows present in the original table.

c. To simulate defragmentation, write and execute a query that combines the views to re-create the original table.

The views are combined using UNION and the defragmented table is loaded into a single view called Full\_Menu.

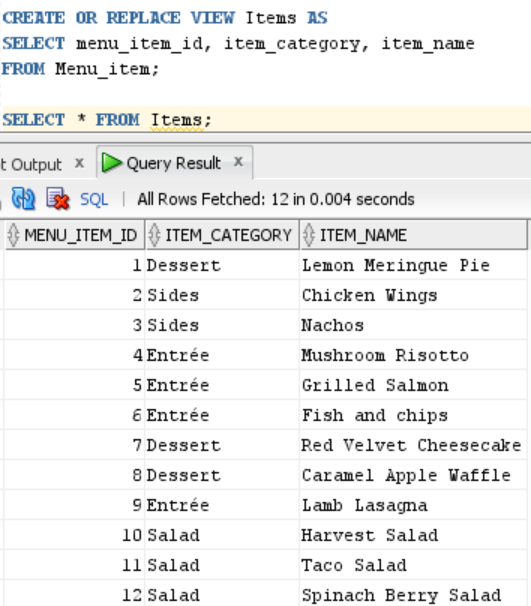


1. *Simulating Vertical Fragmentation –* In this step, you simulate vertical fragmentation and defragmentation. Complete the following substeps.  
   a. Starting with the same logical table as in #5, create two views (using the CREATE VIEW command) that simulate two vertical fragments based upon some reasonable column separation. Show each view’s contents.

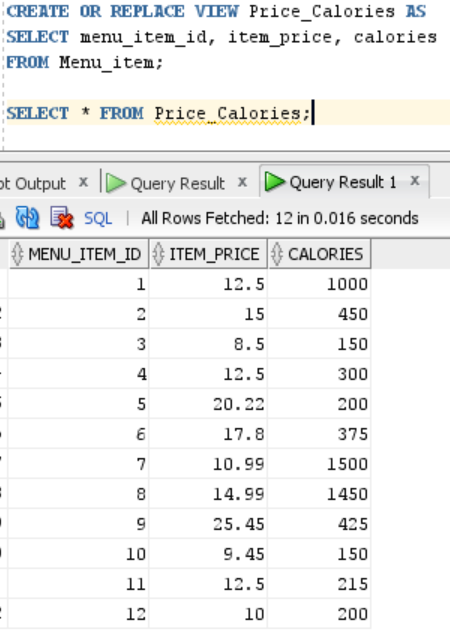
VIEW#1 *Items* fragment lists menu\_item\_id with item categories and item names.

VIEW#2 *Price\_Calories* fragment lists prices and calories for each item in the menu\_item\_id.

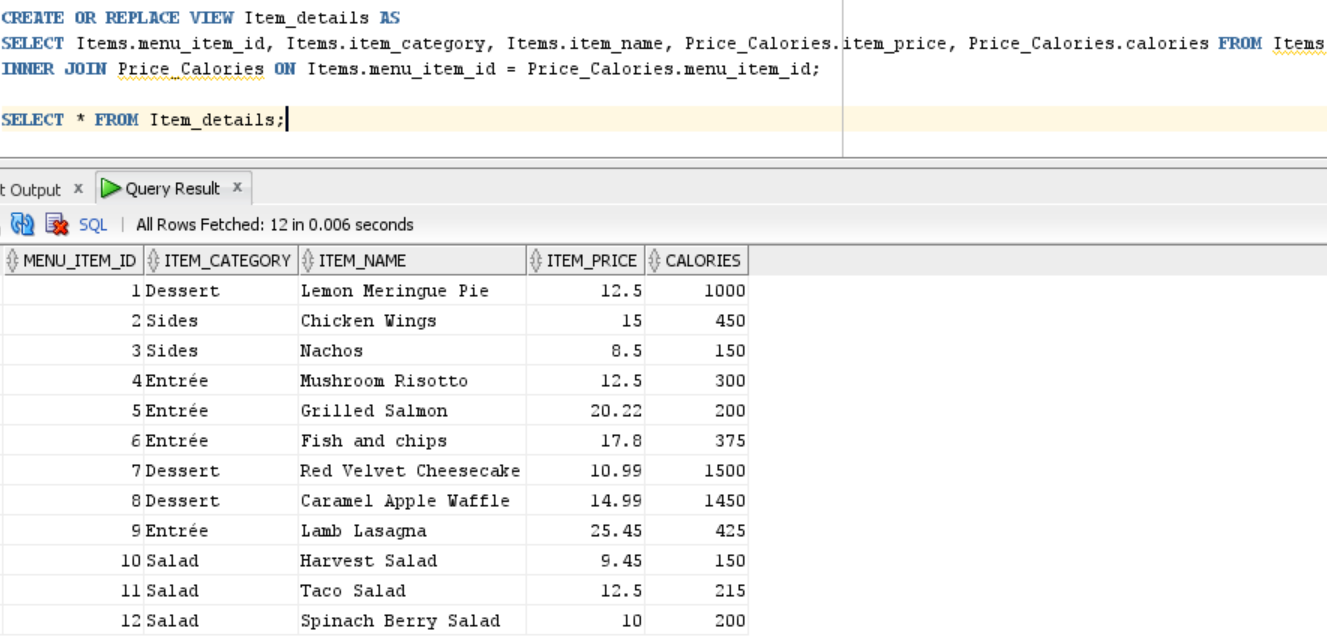
VIEW#1 *Items* fragment



VIEW#2 *Price\_Calories* fragment



b. To simulate defragmentation, write and execute a query that combines the views to re-create the original table.



# Evaluation

Your lab will be reviewed by your facilitator or instructor with the criteria outlined in the table below. Note that the grading process:

* involves the grader assigning an appropriate letter grade to each criterion.
* uses the following letter-to-number grade mapping – A+=100,A=96,A-=92,B+=88,B=85,B-=82,C+=88,C=85,C-=82,D=67,F=0.
* provides an overall grade for the submission based upon the grade and weight assigned to each criterion.
* allows the grader to apply additional deductions or adjustments as appropriate for the submission.
* applies equally to every student in the course.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Criterion** | **A** | **B** | **C** | **D** | **F** |
| **Problem 1 (20%)** | The fact table, dimension tables, and dimension hierarchy have been accurately identified. The explanations supporting the identifications are excellent. | The fact table, dimension tables, and dimension hierarchy have been identified mostly accurately. The explanations supporting the identifications are good. | The fact table, dimension tables, and dimension hierarchy have been identified somewhat accurately. The explanations supporting the identifications are satisfactory. | The fact table, dimension tables, and dimension hierarchy identifications are mostly inaccurate. The explanations supporting the identifications are minimal. | The answer for problem 1 may be missing, or the identifications are entirely inaccurate. Explanations supporting the identifications are missing. |
| **Problem 2: Identification (10%)** | A missing dimension table, along with its attributes and hierarchies, have been accurately identified, based upon the business rules and/or reasonable assumptions. The explanations supporting the identifications are excellent. | A missing dimension table, along with its attributes and hierarchies, have been identified mostly accurately, based upon the business rules and/or reasonable assumptions. The explanations supporting the identifications are good. | A missing dimension table, along with its attributes and hierarchies, have been identified somewhat accurately, based upon the business rules and/or reasonable assumptions. The explanations supporting the identifications are satisfactory. | The missing dimension table identified, along with its attributes and hierarchies, are mostly inaccurate, based upon the business rules and/or reasonable assumptions. The explanations supporting the identifications are minimal. | The answer for problem 2 may be missing, or the identification is entirely inaccurate. Explanations supporting the identifications are missing. |
| **Problem 2: SQL (10%)** | The SQL to create the dimension table and reference it with the fact table is syntactically correct, and entirely reflects the design described in parts A and B. The SQL could be executed in a modern relational database without modification. | The SQL to create the dimension table and reference it with the fact table is mostly syntactically correct, and mostly reflects the design described in parts A and B. Minor adjustments to the SQLare necessary to execute it in a modern relational database or to correctly reflect the design. | The SQL to create the dimension table and reference it with the fact table is somewhat syntactically correct, and partially reflects the design described in parts A and B. Some adjustments to the SQL may be necessary to execute the it in a modern relational database or to correclty reflect the design. | The SQL to create the dimension table and reference it with the fact table is mostly syntactically incorrect, and minimally reflects the design described in parts A and B. Major adjustments to the SQL are necessary to execute it in a modern relational database or to correctly reflect the design. | The answer for problem 2 may be missing. The SQL to create the dimension table and reference it with the fact table is entirely syntactically incorrect, and does not reflect the design described in parts A and B. A complete rewrite is necessary to execute the SQL in a modern relational database and to correctly reflect the design. |
| **Problem 3: Identification (10%)** | A missing measure has been accurately identified, based upon the business rules and/or reasonable assumptions. The explanations supporting the identification are excellent. | A missing measure has been identified mostly accurately, based upon the business rules and/or reasonable assumptions. The explanations supporting the identification are good. | A missing measure has been identified somewhat accurately, based upon the business rules and/or reasonable assumptions. The explanations supporting the identification are satisfactory. | The missing measure identified is mostly inaccurate, based upon the business rules and/or reasonable assumptions. The explanations supporting the identification are minimal. | The answer for problem 3 may be missing, or the identification is entirely inaccurate. Explanations supporting the identification are missing. |
| **Problem 3: SQL (10%)** | The SQL to create the measure and rows is accurate. The query analyzes a useful aspect of the business, and the construction of the SQL, including its use of the ROLLUP extension, is entirely sound. | The SQL to create the measure and rows is mostly accurate. The query analyzes a useful aspect of the business, and the construction of the SQL, including its use of the ROLLUP extension, is mostly sound. | The SQL to create the measure and rows is somewhat accurate. The query analyzes a somewhat useful aspect of the business, and the construction of the SQL, including its use of the ROLLUP extension, is somewhat sound. | The SQL to create the measure and rows is mostly inaccurate. The query analyzes an aspect of the business that is not particularly useful, and the construction of the SQL, including its use of the ROLLUP extension, is mostly unsound. | The answer for problem 3 may be missing. The SQL to create the measure and rows is entirely inaccurate. The query analyzes an aspect of the business that is not useful, and the construction of the SQL, including its use of the ROLLUP extension, is unsound. |
| **Problem 4 (10%)** | The explanations for all parts of this problem are entirely accurate, are well supported, are thorough, and very clear. | The explanations for all parts of this problem are mostly accurate, are supported, are fairly thorough, and clear. | The explanations for all parts of this problem are somewhat accurate, are somewhat supported, and are somewhat clear. | The explanations for all parts of this problem are mostly inaccurate, are minimally supported, and are mostly unclear. | The answer for problem 4 may be missing. The explanations for all parts of this problem are entirely inaccurate, are not supported, and are unclear. |
| **#5 Quality (15%)** | Entirely complete and accurate fragments have been defined as views for #5b. The query for #5c completely and accurately reconstructs the original table. | Mostly complete and accurate fragments have been defined as views for #5b. The query for #5c mostly reconstructs the original table. | Somewhat complete and accurate fragments have been defined as views for #5b. The query for #5c somewhat reconstructs the original table. | Mostly incomplete and inaccurate fragments have been defined as views for #5b. The reconstruction of the original table with the query for #5c is mostly inaccurate. | Answers for #5 may be missing, or views and queries may not have been used to answer the question. Incomplete and inaccurate fragments have been defined for #5b. The query for #5c does not reconstruct the original table accurately. |
| **#6 Quality (15%)** | Entirely complete and accurate fragments have been defined as views for #6a. The query for #6b completely and accurately reconstructs the original table. | Mostly complete and accurate fragments have been defined as views for #6a. The query for #6b mostly reconstructs the original table. | Somewhat complete and accurate fragments have been defined as views for #6a. The query for #6b somewhat reconstructs the original table. | Mostly incomplete and inaccurate fragments have been defined as views for #6a. The reconstruction of the original table with the query for #6b is mostly inaccurate. | Answers for #6 may be missing, or views and queries may not have been used to answer the question. Incomplete and inaccurate fragments have been defined for #6a. The query for #6b does not reconstruct the original table accurately. |

Use the **Ask the Teaching Team Discussion Forum** if you have any questions regarding how to approach this lab. Make sure to include your name in the filename and submit it in the *Assignments* section of the course.