

Section One – Advanced Topics

Step 1 – Identifying Essential Parts

a. Identify the fact table and explain what event it represents.

A fact table in the star schema is a table that represents the event itself, it usually takes on the name of the event. As the star schema in our example represents groups of people (parties) eating meals at restaurants, we can refer to it as representing a Meal_item_selection event, and the fact table in our star schema as the Meal_item_selection.

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

A dimension table in the star schema represents anything that participates in the event itself, real or abstract, in the event. It usually takes on the name of the participant. For the schema in our example, there are three dimensions - Party, Menu_item and Meal_date. The Party represents the groups of people eating meals at restaurants. The Menu_item represents the food item that the Party selects. The Meal_date is the abstract participant representing the date that the party is requesting the meal.

c. Identify a hierarchy that exists in one of the dimension tables and explain what it represents.

A hierarchy defines a hierarchical relationship between specific attributes inside of a dimension so it is always inside of one dimension. One hierarchy is found in the Meal_date dimension, the year, month, day_of_month. For example, a year contains months, and a month contains days, so we can define year-month-day attributes as a hierarchy. While the day of the month alone isn't useful; it's important to know what month the day is in. Likewise, a month alone isn't that useful; it's important to know what year the month is in.

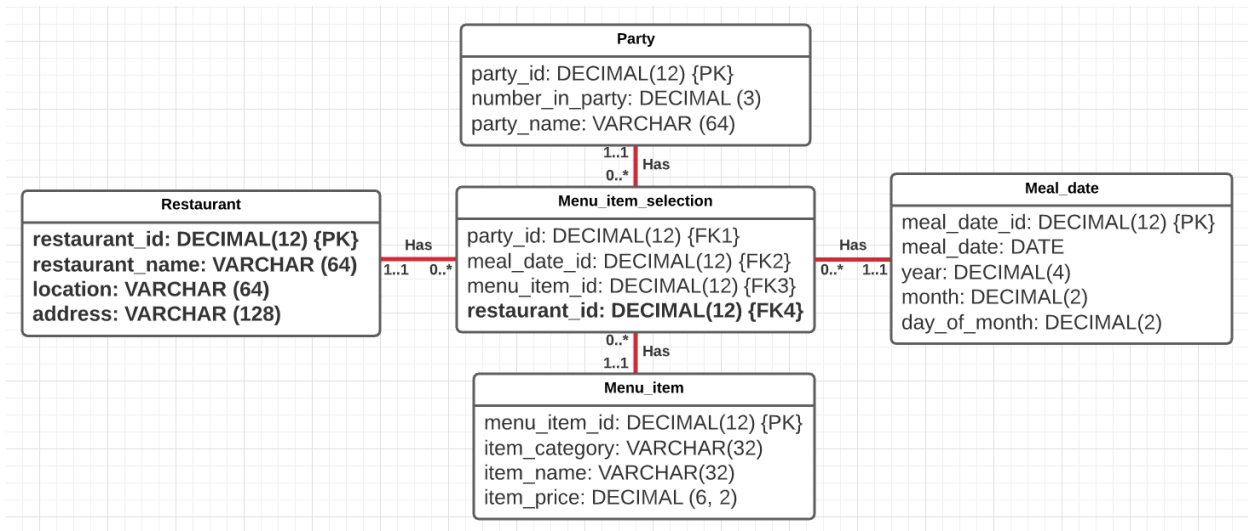
There is always a one-to-many relationship between an attribute higher in the hierarchy and a lower one. For example, there are 12 months in one year, and up to 31 days in each month. We could carry this hierarchy further with more advanced analysis. For example, we can aggregate Meal_date by year, by year and month, or by year and month and day.

Step 2 – Adding a Dimension

a. Review the business rules in the section introduction, identify a dimension that is missing, and explain.

Looking through the business rules, I identified that the dimension that is missing is 'There are many restaurants, and each restaurant has their own name, location, and address.' I can add another dimension - Restaurant to the schema which captures the name, the location and the address of the restaurant. I have also added a foreign key restaurant_id to the fact table Party.

The screenshot below is the schema after the change with the changes bolded.



b. Explain what attributes and hierarchies this dimension would reasonably contain.

With the new Restaurant dimension, the restaurant becomes another participant with the Party event, which makes sense because a party needs to be held at a restaurant with a physical address in our case. We can now track the name, the location and the address of the restaurant. Notice that the Menu_item_selection fact table now includes a restaurant_id foreign key to track which restaurant holds the event.

As far as hierarchies in Restaurant, location-address is a hierarchy. The location is at the top of the hierarchy, and there are many possible addresses for each location. For instance, there are many McDonald's in New York and each McDonald has its own address. Similar rules apply to the Restaurant dimension in our schema that there are many addresses for restaurants at each location, so there is a one-to-many relationship between location and address. The standalone attribute restaurant_name indicates the name of each restaurant.

c. 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.

```

CREATE TABLE Restaurant(
restaurant_id DECIMAL(12),
restaurant_name VARCHAR(64),
location VARCHAR(64),
address VARCHAR(128),
PRIMARY KEY (restaurant_id));

ALTER TABLE Menu_item_selection
ADD restaurant_id DECIMAL(12),
ADD CONSTRAINT restaurant_id_fk
FOREIGN KEY (restaurant_id)
REFERENCES Restaurant(restaurant_id);
    
```

Step 3 – Adding a Measure







- a. As there are no measures in the schema, identify a useful one that could be added, and explain what it measures.

Since the measure in a fact table represents a measurement of something of value when the event occurs, and there are no measures in our schema currently. I identified that it would be useful to add a 'satisfaction' as the numeric measure in the fact table. This could be useful to determine the satisfaction for the groups of people (parties) eating meals at restaurants. Given a scaling question, 1 being the lowest, 5 being the highest, each party can rate their overall dining experience at the selected restaurant, and the data would be useful for the organization to analyze the performance of the restaurant and make necessary improvements.

- b. In SQL, add the measure to the fact table.

```
48 ALTER TABLE Menu_item_selection
49 ADD satisfaction DECIMAL(1) CHECK (satisfaction BETWEEN 1 AND 5);
50
51 SELECT * From Menu_item_selection;
52
```

Data Output Explain Messages Notifications

	party_id numeric (12) 	meal_date_id numeric (12) 	menu_item_id numeric (12) 	restaurant_id numeric (12) 	satisfaction numeric (1) 
------------------------------------------------------------------------------------	-------------------------------------------------------------------------------------------------------------------	-----------------------------------------------------------------------------------------------------------------------	-----------------------------------------------------------------------------------------------------------------------	--------------------------------------------------------------------------------------------------------------------------	------------------------------------------------------------------------------------------------------------------------

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

```

CREATE SEQUENCE party_seq START WITH 1;
CREATE SEQUENCE restaurant_seq START WITH 1;
CREATE SEQUENCE date_seq START WITH 1;
CREATE SEQUENCE menu_seq START WITH 1;

CREATE OR REPLACE FUNCTION AddInformation(
    party_id IN DECIMAL, restaurant_id IN DECIMAL, number_in_party IN DECIMAL, party_name IN VARCHAR,
    satisfaction IN DECIMAL, restaurant_name IN VARCHAR, location IN VARCHAR, address IN VARCHAR,
    meal_date IN DATE, year IN DECIMAL, month IN DECIMAL, day_of_month IN DECIMAL
    item_category IN VARCHAR, item_name IN VARCHAR, item_price IN DECIMAL)
RETURNS VOID
AS
$proc$
BEGIN
    INSERT INTO Restaurant(restaurant_id, restaurant_name, location, address)
    VALUES(nextval('restaurant_seq'), restaurant_name, location, address);

    INSERT INTO Party(party_id, number_in_party, party_name)
    VALUES(nextval('party_seq'), number_in_party, party_name);

    INSERT INTO Meal_date(meal_date_id, meal_date, year, month, day_of_month)
    VALUES(nextval('date_seq'), meal_date, year, month, day_of_month);

    INSERT INTO Menu_item(menu_item_id, item_category, item_name, item_price)
    VALUES(nextval('menu_seq'), item_category, item_name, item_price);

    INSERT INTO Menu_item_selection(party_id, meal_date_id, menu_item_id, restaurant_id, satisfaction)
    VALUES(currval('party_seq'), currval('date_seq'), currval('menu_seq'), currval('restaurant_seq'), satisfaction);
END;
$proc$ LANGUAGE plpgsql

```

```

START TRANSACTION;
DO
$$BEGIN
EXECUTE AddInformation(nextval('party_seq'), nextval('restaurant_seq'), 2, 'Amy', 5, 'Yummy Town', 'Miami', '123 main street, FL 33101');
EXECUTE AddInformation(nextval('party_seq'), nextval('restaurant_seq'), 1, 'Bryan', 3, 'Ramen House', 'Pittsburgh', '123 main street, PA 15201');
EXECUTE AddInformation(nextval('party_seq'), nextval('restaurant_seq'), 3, 'Charles', 5, 'Pizza Lover', 'New York', '123 main street, NY, 10001');
EXECUTE AddInformation(nextval('party_seq'), nextval('restaurant_seq'), 2, 'David', 4, 'Happy Food', 'New York', '123 main street, NY, 10001');
EXECUTE AddInformation(nextval('party_seq'), nextval('restaurant_seq'), 4, 'Edison', 3, 'Food Town', 'Los Angeles', '123 main street, CA 90001');
EXECUTE AddInformation(nextval('party_seq'), nextval('restaurant_seq'), 2, 'Frank', 2, 'Food Mall', 'El Paso', '123 main street, TX 79936');
EXECUTE AddInformation(nextval('party_seq'), nextval('restaurant_seq'), 3, 'Giorgio', 1, 'Yummy Happy', 'Los Angeles', '123 main street, CA 90001');
EXECUTE AddInformation(nextval('party_seq'), nextval('restaurant_seq'), 3, 'Helen', 5, 'Chicken Lover', 'New York', '123 main street, NY, 10001');
EXECUTE AddInformation(nextval('party_seq'), nextval('restaurant_seq'), 5, 'Ivy', 1, 'Burger Place', 'Los Angeles', '123 main street, CA 90001');
EXECUTE AddInformation(nextval('party_seq'), nextval('restaurant_seq'), 4, 'Jason', 3, 'Delicious', 'Chicago', '123 main street, IL 60629');
EXECUTE AddInformation(nextval('party_seq'), nextval('restaurant_seq'), 2, 'Kevin', 4, 'Happy Meal', 'Miami', '123 main street, FL 33101');
EXECUTE AddInformation(nextval('party_seq'), nextval('restaurant_seq'), 3, 'Lisa', 5, 'Food Lover', 'New York', '123 main street, NY, 10001');
EXECUTE AddInformation(nextval('party_seq'), nextval('restaurant_seq'), 1, 'Mary', 1, 'Food House', 'Savannah', '123 main street, GA 31301');
EXECUTE AddInformation(nextval('party_seq'), nextval('restaurant_seq'), 5, 'Nancy', 2, 'Yummy Food', 'Portland', '123 main street, OR 97001');
EXECUTE AddInformation(nextval('party_seq'), nextval('restaurant_seq'), 2, 'Peter', 3, 'Hot Dog Stand', 'Chicago', '123 main street, IL 60601');
END$$;
COMMIT TRANSACTION;

```

d. 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.

```

SELECT Restaurant.location, Restaurant.restaurant_name, SUM(satisfaction) AS total_satisfaction
FROM Menu_item_selection
JOIN Restaurant ON Restaurant.restaurant_id = Menu_item_selection.restaurant_id
GROUP BY ROLLUP(Restaurant.restaurant_name), Restaurant.location
ORDER BY total_satisfaction DESC;

```

	location character varying (64)	restaurant_name character varying (64)	total_satisfaction numeric
1	New York	[null]	19
2	Miami	[null]	9
3	Chicago	[null]	6
4	Los Angeles	[null]	5
5	New York	Chicken Lover	5
6	New York	Food Lover	5
7	New York	Pizza Lover	5

The ROLLUP extension to GROUP BY adds subtotals and a grand total to the results. With the ROLLUP clause at the code above, there would be new rows that give a total satisfaction score for each location, and here we make the restaurant name optional. Without the ROLLUP, we would not have the grand total of the total satisfaction scores that add up for each location. I ordered by the most total satisfaction number, and as we can see the location that received the most total satisfaction is New York receiving 19 scores.

Step 4 – Understanding Embedded SQL

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

Java Code on 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 + ".");
}
```

The code started from connecting strings for the Java Database Connectivity connection, we are using SQL Server and connecting to the IP Address 1433. database=MyDB indicates that we are connecting to a database named MyDB. user=MyUser; password=ABC123; gives the account credentials authorized to connect to the instance and database. Then we are providing SQL query itself, and using the + operator concatenates multi-line strings into one long string in Java. Then the statement.executeQuery(sql) statement sends the SQL command to the database and executes it and stores the result into a results variable. Finally, the while loop retrieves the value, iterates through the rows and prints out the results to the screen as ("Party " + party_name + " bought " + item_name + " of type " + item_category + "."). The + operator concatenates the results as to a sentence.

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

Embedding SQL allows an application to execute SQL as needed to store or retrieve data in the database. Since users would need to navigate through the application's user interface and access its features, it is important for the application to be able to retrieve the relevant data from the database. As the process should be automatically executed when we need the data, and the database is large, the number of the database access requests is enormous; we can not manually type the SQL into a SQL client, it is not efficient and lacks features that the programming languages would be able to offer.

c. A Java program is being used. What kind of connectivity technology is used by this program (and Java programs in general) to connect to the database? Briefly explain how the technology works. You will need to perform some research to address this step.

JDBC stands for Java Database Connectivity, it is a Java-based data access technology. JDBC is a Java API to connect and execute the query with the database. JDBC offers a programming-level interface that handles the mechanics of Java applications communicating with a database or RDBMS. The connection is the session between java application and database.

There are 5 steps to connect any java application with the database using JDBC.

1. Register the Driver class: The forName() method of a Class class is used to register the driver class.
2. Create connection : The getConnection() method of DriverManager class is used to establish connection with the database.
3. Create statement : The createStatement() method of Connection interface is used to create statements. The object of the statement is responsible to execute queries with the database.
4. Execute queries: The executeQuery() method of Statement interface is used to execute queries to the database. This method returns the object of ResultSet that can be used to get all the records of a table.
5. Close connection: The close() method of Connection interface is used to close the connection.

Step 5 – Understanding Big Data

a. Explain why and when the business might move to a Big Data solution in terms of the volume, velocity, and variety of data.

Today, there is more than 100 terabytes of information, there is a need for quick data arrival, and there is a variety of complex data types we are storing, a Big Data solution is necessary.

- The volume of data refers to the total size of all of the data.
- The velocity of data refers to how quickly the data arrives.
- The variety refers to the number of types of data, and of particular concern are types beyond the basic types – characters, dates, times, and numbers.

We have to have enough space to store all of the data, and make sure that the relevant data we are retrieving for arrives quickly and can handle thousands of concurrent transactions across hundreds or thousands of computers, as well as the database supporting the complex types of data structure we need in the database.

b. What would cause the business to move to a document-oriented NoSQL database for its data storage, instead of a relational database with a star schema? Explain.

NoSQL stands for Not only Structured Query Language. When the business needs to store non-relational data, businesses move to a document-oriented NoSQL database for its data storage such as mongoDB and redis. NoSQL can be more flexible with structures in the database which attracts more businesses to move to a NoSQL database. For instance, NoSQL data structures don't need common fields and can store different types of data. SQL is more like an excel spreadsheet and null value can be dangerous. However, NoSQL like mongoDB is more like a JSON object with key value pairs that we can modify and not affect other structures. SQL likes structures and once defined, just stick to it. NoSQL is more flexible and adapts to the change when the things come up and data structure is not fully defined, like for start-up companies, they can one day sell toasters and tomorrow sell lessons. While NoSQL is not bound to a data structure, SQL is bound to the relationship. Also for scalability reasons, the business would choose to move to a NoSQL database. Once the database gets too huge, the computer would struggle and the database would become unstable in SQL. However, NoSQL manages data into smaller chunks of data which allows for a distributed system that spreads to a lot of computers. To sum up, SQL is more mature, it is a table structured, requires a schema and great with relationships but scales vertically while NoSQL is shiny and new, it is document structured, more flexible to changes, not that great with complex relationships but horizontally scalable.