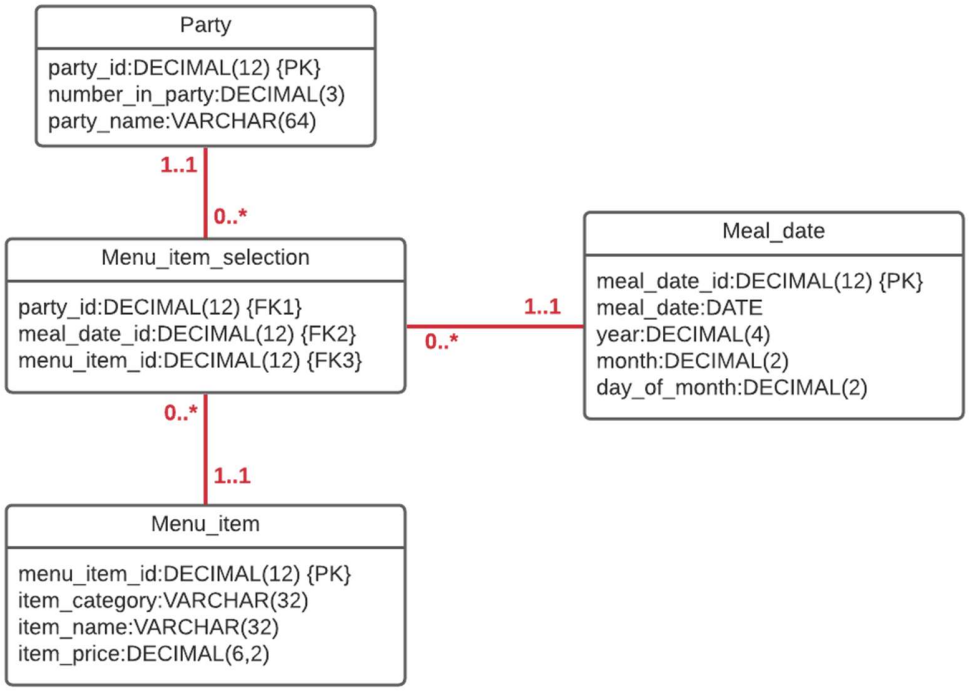
Section One – Dimensional Modeling

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.
    a. Identify the fact table and explain what event it represents.
    The event in the menu-item-selection star scheme is the selection of a menu item (table: Menu_item_selection) and represents each individual menu item selection for each meal by the party/group.

b. Identify the dimension tables and explain what event participant it represents.
In the menu-item-selection schema, there are three dimensions drawn in the entity-relationship model: Party, Menu_item, and Meal_date. The Party represents the group of people selecting menu items. The Menu_item is the item being selected. And the Meal_date is the abstract participant representing the date on which the menu item selection occurred. There are also two other dimensions that are not drawn in the entity-relationship model but are mentioned in the text below the entity-relationship model: Restaurant and Waitperson.
The restaurant represents the place where the party of people selects menu items. And the waitperson serves the menu items to the parties in the restaurants.

c. Identify a hierarchy that exists in one of the dimension tables and explain what it represents.
One hierarchy is found in the Meal_date dimension table: year >> month >> day_of_month. The attribute year is the highest in this hierarchy and can be thought of as a container that contains the attribute month; and the attribute month contains the attribute day_of_month; so year >> month >> day_of_month attributes can be defined as a hierarchy. It can also be said that a year defines the context for the month, and that the month defines the context for a day_of_month. Furthermore, each year has many months, each month has many day_of_months, so they satisfy the one-to-many relationship that must exist between attributes that follow each other in the hierarchy.

2. *Adding a Dimension* – Next, identify and add in a dimension that is missing by completing the following.
    a. Review the business rules in the section introduction, identify a dimension that is missing, and explain.
    The dimension restaurant is missing. In the star scheme without a restaurant, it is unclear where the parties order (and eat) the menu items.

    b. Explain what attributes and hierarchies this dimension would reasonably contain.
    The dimension table Restaurant has the attribute restaurant_id (primary key) to uniquely identify each restaurant, which is then referenced in the fact table to connect the Restaurant dimension table to the fact table Menu_item_selection (the event). Likewise, the Restaurant table will have the attributes name (of the restaurant), street1 (street1 is the combination of street name and street number), city, state, and

postal_code, so that the participants like parties can find the restaurant. A hierarchy in the Restaurant dimension is state >> city. The state gives the context where the city is located. There is also a necessary one-to-many relationship: every state has several cities; every city belongs to one state.

c. Add the dimension into the schema by creating the dimension table in SQL along with is attributes, and adding a foreign key to the fact table.

```
26 CREATE TABLE Restaurant (
27 restaurant_id DECIMAL(12) NOT NULL PRIMARY KEY,
28 name VARCHAR(64) NOT NULL,
29 street1 VARCHAR(64) NOT NULL,
30 city VARCHAR(64) NOT NULL,
31 state VARCHAR(64) NOT NULL,
32 postal_code VARCHAR(64) NOT NULL);
33
34 CREATE TABLE Menu_item_selection (
35 party_id DECIMAL(12) NOT NULL,
36 meal_date_id DECIMAL(12) NOT NULL,
37 menu_item_id DECIMAL(12) NOT NULL,
38 restaurant_id DECIMAL(12) NOT NULL,
39 FOREIGN KEY (party_id) REFERENCES Party(party_id),
40 FOREIGN KEY (meal_date_id) REFERENCES Meal_date(meal_date_id),
41 FOREIGN KEY (menu_item_id) REFERENCES Menu_item(menu_item_id),
42 FOREIGN KEY (restaurant_id) REFERENCES Restaurant(restaurant_id));
```

121 %

Messages

```
Commands completed successfully.
```

3. *Adding a Measure* – Next, identify and make use of a useful measure by completing the following.
    a. As there are no measures in the schema, identify a useful one that could be added, and explain what it measures.
    A useful measure is how many menu items were selected in the menu selection; this measure will be called total_quantity. The measure total_quantity in the fact table is useful to calculate the total revenue for a menu item selection: Menu_item_selection.total_quantity multiplied by Menu_item.item_price. Also, total_quantity can be used to calculate the total revenue of the entire restaurant. And if cost information is included in the scheme, then the restaurant's profit can be calculated, too.

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

```
34 CREATE TABLE Menu_item_selection (
35 party_id DECIMAL(12) NOT NULL,
36 meal_date_id DECIMAL(12) NOT NULL,
37 menu_item_id DECIMAL(12) NOT NULL,
38 restaurant_id DECIMAL(12) NOT NULL,
39 total_quantity INT NOT NULL,
40 FOREIGN KEY (party_id) REFERENCES Party(party_id),
41 FOREIGN KEY (meal_date_id) REFERENCES Meal_date(meal_date_id),
42 FOREIGN KEY (menu_item_id) REFERENCES Menu_item(menu_item_id),
43 FOREIGN KEY (restaurant_id) REFERENCES Restaurant(restaurant_id));
```

121 %

Messages

```
Commands completed successfully.
```

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.

```
50 CREATE SEQUENCE party_id_seq START WITH 1;
51 CREATE SEQUENCE meal_date_id_seq START WITH 1;
52 CREATE SEQUENCE menu_item_id_seq START WITH 1;
53 CREATE SEQUENCE restaurant_id_seq START WITH 1;
54
55 INSERT INTO Party(party_id, number_in_party, party_name)
56 VALUES(NEXT VALUE FOR party_id_seq, 5, 'Party A');
57 INSERT INTO Party(party_id, number_in_party, party_name)
58 VALUES(NEXT VALUE FOR party_id_seq, 10, 'Party B');
59 INSERT INTO Party(party_id, number_in_party, party_name)
60 VALUES(NEXT VALUE FOR party_id_seq, 15, 'Party C');
61 INSERT INTO Party(party_id, number_in_party, party_name)
62 VALUES(NEXT VALUE FOR party_id_seq, 20, 'Party D');
63 INSERT INTO Party(party_id, number_in_party, party_name)
64 VALUES(NEXT VALUE FOR party_id_seq, 25, 'Party E');
```

121 %

Messages

```
(1 row affected)

(1 row affected)

(1 row affected)

(1 row affected)

(1 row affected)
```

```
INSERT INTO Meal_date(meal_date_id, meal_date, [year], [month], day_of_month)
 VALUES(NEXT VALUE FOR meal_date_id_seq, '01/01/2022', 2022, 1, 1);
INSERT INTO Meal_date(meal_date_id, meal_date, [year], [month], day_of_month)
 VALUES(NEXT VALUE FOR meal_date_id_seq, '01/02/2022', 2022, 1, 2);
INSERT INTO Meal_date(meal_date_id, meal_date, [year], [month], day_of_month)
 VALUES(NEXT VALUE FOR meal_date_id_seq, '01/03/2022', 2022, 1, 3);
INSERT INTO Meal_date(meal_date_id, meal_date, [year], [month], day_of_month)
 VALUES(NEXT VALUE FOR meal_date_id_seq, '01/04/2022', 2022, 1, 4);
INSERT INTO Meal_date(meal_date_id, meal_date, [year], [month], day_of_month)
 VALUES(NEXT VALUE FOR meal_date_id_seq, '01/05/2022', 2022, 1, 5);
INSERT INTO Meal_date(meal_date_id, meal_date, [year], [month], day_of_month)
 VALUES(NEXT VALUE FOR meal_date_id_seq, '01/06/2022', 2022, 1, 6);
INSERT INTO Meal_date(meal_date_id, meal_date, [year], [month], day_of_month)
 VALUES(NEXT VALUE FOR meal_date_id_seq, '01/07/2022', 2022, 1, 7);
INSERT INTO Meal_date(meal_date_id, meal_date, [year], [month], day_of_month)
 VALUES(NEXT VALUE FOR meal_date_id_seq, '01/08/2022', 2022, 1, 8);
INSERT INTO Meal_date(meal_date_id, meal_date, [year], [month], day_of_month)
 VALUES(NEXT VALUE FOR meal_date_id_seq, '01/09/2022', 2022, 1, 9);
INSERT INTO Meal_date(meal_date_id, meal_date, [year], [month], day_of_month)
 VALUES(NEXT VALUE FOR meal_date_id_seq, '01/10/2022', 2022, 1, 10);
INSERT INTO Meal_date(meal_date_id, meal_date, [year], [month], day_of_month)
 VALUES(NEXT VALUE FOR meal_date_id_seq, '01/11/2022', 2022, 1, 11);
INSERT INTO Meal_date(meal_date_id, meal_date, [year], [month], day_of_month)
 VALUES(NEXT VALUE FOR meal_date_id_seq, '01/12/2022', 2022, 1, 12);
INSERT INTO Meal_date(meal_date_id, meal_date, [year], [month], day_of_month)
 VALUES(NEXT VALUE FOR meal_date_id_seq, '01/13/2022', 2022, 1, 13);
INSERT INTO Meal_date(meal_date_id, meal_date, [year], [month], day_of_month)
 VALUES(NEXT VALUE FOR meal_date_id_seq, '01/14/2022', 2022, 1, 14);
INSERT INTO Meal_date(meal_date_id, meal_date, [year], [month], day_of_month)
 VALUES(NEXT VALUE FOR meal_date_id_seq, '01/15/2022', 2022, 1, 15);

121 %
Messages

(1 row affected)

(1 row affected)

(1 row affected)

(1 row affected)
```

```
INSERT INTO Menu_item(menu_item_id, item_category, item_name, item_price)
 VALUES(NEXT VALUE FOR menu_item_id_seq, 'Entree', 'Item 1', 24.99);
INSERT INTO Menu_item(menu_item_id, item_category, item_name, item_price)
 VALUES(NEXT VALUE FOR menu_item_id_seq, 'Entree', 'Item 2', 36.99);
INSERT INTO Menu_item(menu_item_id, item_category, item_name, item_price)
 VALUES(NEXT VALUE FOR menu_item_id_seq, 'Side', 'Item 3', 4.99);
INSERT INTO Menu_item(menu_item_id, item_category, item_name, item_price)
 VALUES(NEXT VALUE FOR menu_item_id_seq, 'Side', 'Item 4', 5.99);
INSERT INTO Menu_item(menu_item_id, item_category, item_name, item_price)
 VALUES(NEXT VALUE FOR menu_item_id_seq, 'Dessert', 'Item 5', 6.99);
INSERT INTO Menu_item(menu_item_id, item_category, item_name, item_price)
 VALUES(NEXT VALUE FOR menu_item_id_seq, 'Dessert', 'Item 6', 7.99);

INSERT INTO Restaurant(restaurant_id, name, street1, city, state, postal_code)
 VALUES(NEXT VALUE FOR restaurant_id_seq, 'Restaurant 1', '2368 Clover Drive', 'Salida', 'Colorado', '81201');
INSERT INTO Restaurant(restaurant_id, name, street1, city, state, postal_code)
 VALUES(NEXT VALUE FOR restaurant_id_seq, 'Restaurant 2', '4249 Gorby Lane', 'Barlow', 'Mississippi', '39083');

121 %
Messages

(1 row affected)

(1 row affected)

(1 row affected)

(1 row affected)

(1 row affected)
```

```
115 INSERT INTO Menu_item_selection(party_id, meal_date_id, menu_item_id, restaurant_id, total_quantity)
116  VALUES(1, 1, 1, 1, 10);
117 INSERT INTO Menu_item_selection(party_id, meal_date_id, menu_item_id, restaurant_id, total_quantity)
118  VALUES(2, 2, 2, 1, 20);
119 INSERT INTO Menu_item_selection(party_id, meal_date_id, menu_item_id, restaurant_id, total_quantity)
120  VALUES(3, 3, 3, 1, 30);
121 INSERT INTO Menu_item_selection(party_id, meal_date_id, menu_item_id, restaurant_id, total_quantity)
122  VALUES(4, 4, 4, 2, 20);
123 INSERT INTO Menu_item_selection(party_id, meal_date_id, menu_item_id, restaurant_id, total_quantity)
124  VALUES(5, 5, 5, 2, 10);
125 INSERT INTO Menu_item_selection(party_id, meal_date_id, menu_item_id, restaurant_id, total_quantity)
126  VALUES(1, 6, 1, 1, 10);
127 INSERT INTO Menu_item_selection(party_id, meal_date_id, menu_item_id, restaurant_id, total_quantity)
128  VALUES(2, 7, 2, 1, 20);
129 INSERT INTO Menu_item_selection(party_id, meal_date_id, menu_item_id, restaurant_id, total_quantity)
130  VALUES(3, 8, 3, 1, 30);
131 INSERT INTO Menu_item_selection(party_id, meal_date_id, menu_item_id, restaurant_id, total_quantity)
132  VALUES(4, 9, 4, 2, 40);
133 INSERT INTO Menu_item_selection(party_id, meal_date_id, menu_item_id, restaurant_id, total_quantity)
134  VALUES(5, 10, 5, 2, 50);
135 INSERT INTO Menu_item_selection(party_id, meal_date_id, menu_item_id, restaurant_id, total_quantity)
136  VALUES(1, 11, 1, 1, 10);
137 INSERT INTO Menu_item_selection(party_id, meal_date_id, menu_item_id, restaurant_id, total_quantity)
138  VALUES(2, 12, 2, 1, 20);
139 INSERT INTO Menu_item_selection(party_id, meal_date_id, menu_item_id, restaurant_id, total_quantity)
140  VALUES(3, 13, 3, 1, 30);
141 INSERT INTO Menu_item_selection(party_id, meal_date_id, menu_item_id, restaurant_id, total_quantity)
142  VALUES(4, 14, 4, 2, 40);
143 INSERT INTO Menu_item_selection(party_id, meal_date_id, menu_item_id, restaurant_id, total_quantity)
144  VALUES(5, 15, 5, 2, 50);
```

121 %

Messages

```
(1 row affected)

(1 row affected)

(1 row affected)

(1 row affected)

(1 row affected)
```

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.

```
154 SELECT name, party_name, SUM(total_quantity) AS total_quantity
155  FROM Menu_item_selection
156  JOIN Party ON Party.party_id = Menu_item_selection.party_id
157  JOIN Restaurant ON Restaurant.restaurant_id = Menu_item_selection.restaurant_id
158  GROUP BY ROLLUP(party_name), Restaurant.restaurant_id, name
159  ORDER BY name, party_name;
```

121 %

Results | Messages

	name	party_name	total_quantity
1	Restaurant 1	NULL	180
2	Restaurant 1	Party A	30
3	Restaurant 1	Party B	60
4	Restaurant 1	Party C	90
5	Restaurant 2	NULL	210
6	Restaurant 2	Party D	100
7	Restaurant 2	Party E	110

For each of the restaurants, the results table gives the parties that ate in the restaurant and the associated total quantity of menu items selected.

By using ROLLUP on party_name followed by the restaurant_id in the GROUP BY statement, the total quantity of menu items selected for each restaurant is displayed in the total_quantity column; see row numbers 1 and 5; the row in which the total quantity per restaurant is located is indicated by the NULL value in the party_name column. In restaurant 1, a total of 180 menu items were selected across the parties and in restaurant 2, 210 menu items were selected across the parties. Thus, 30 more quantity items were selected in restaurant 2 than in restaurant 1.

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

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

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 + ".");
}
```

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

```
"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 ";
```

The embedded SQL query can be executed via Java, however if the SQL query is to be executed without Java, for example, in Microsoft SQL Server, then the quotation marks and plus signs must be removed; then the SQL query looks like below:

```
190 SELECT Party.party_name, Menu_item.item_category, Menu_item.item_name
191 FROM   Menu_item_selection
192 JOIN   Meal_date ON Meal_date.meal_date_id = Menu_item_selection.meal_date_id
193 JOIN   Menu_item ON Menu_item.menu_item_id = Menu_item_selection.menu_item_id
194 JOIN   Party ON Party.party_id = Menu_item_selection.party_id
195 WHERE  Meal_date.meal_date = CAST('01-APR-2021' AS DATE)
196 ORDER BY Party.party_id;
```

The SQL query obtains the party names, ordered item categories, and the associated item names – for those rows that have the meal_date 04/01/2021. The results table obtained can look like this:

Results | Messages

	party_name	item_category	item_name
1	Party A	Entree	Item 1
2	Party A	Entree	Item 1
3	Party B	Entree	Item 2
4	Party C	Side	Item 3
5	Party D	Side	Item 4

b. What is the purpose of embedding this SQL into the program, as opposed to manually typing the SQL into a SQL client? Explain.
One purpose of embedding SQL in a program is that SQL can be generated and used programmatically. For example, a program can obtain values from APIs and embed these values into SQL queries and execute the SQL queries. Another purpose is to encapsulate SQL. Here, encapsulation means that the user of a program that encapsulates SQL, depending on the level of encapsulation, has to know little to no SQL to use SQL, because the user, for example, uses a program such as Microsoft Power BI that is used to create data visualizations by selecting databases, values, and diagram types, without having to write any SQL queries. Another purpose of embedding is that the

entire functionality of high-level programming language can enhance the SQL use cases: When SQL is embedded in Java, Python or C++, object-oriented programs can be built using thousands of open-source packages, and these programs contain functionality that SQL itself cannot provide.

5. *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.

```
210 INSERT INTO Person(person_id, first_name, last_name, height_inches, age)
211  VALUES(NEXT VALUE FOR person_id_seq, 'Abby', 'Aoe', 60, 78);
212 INSERT INTO Person(person_id, first_name, last_name, height_inches, age)
213  VALUES(NEXT VALUE FOR person_id_seq, 'Brian', 'Boe', 63, 51);
214 INSERT INTO Person(person_id, first_name, last_name, height_inches, age)
215  VALUES(NEXT VALUE FOR person_id_seq, 'Charlie', 'Coe', 67, 20);
216 INSERT INTO Person(person_id, first_name, last_name, height_inches, age)
217  VALUES(NEXT VALUE FOR person_id_seq, 'Doe', 'Doe', 69, 31);
218 INSERT INTO Person(person_id, first_name, last_name, height_inches, age)
219  VALUES(NEXT VALUE FOR person_id_seq, 'Elliott', 'Eoe', 72, 21);
220 INSERT INTO Person(person_id, first_name, last_name, height_inches, age)
221  VALUES(NEXT VALUE FOR person_id_seq, 'Finley', 'Foe', 70, 34);
222 INSERT INTO Person(person_id, first_name, last_name, height_inches, age)
223  VALUES(NEXT VALUE FOR person_id_seq, 'Graham', 'Goe', 68, 43);
224 INSERT INTO Person(person_id, first_name, last_name, height_inches, age)
225  VALUES(NEXT VALUE FOR person_id_seq, 'Hudson', 'Hoe', 74, 53);
226 INSERT INTO Person(person_id, first_name, last_name, height_inches, age)
227  VALUES(NEXT VALUE FOR person_id_seq, 'Isabella', 'Ioe', 64, 61);
228 INSERT INTO Person(person_id, first_name, last_name, height_inches, age)
229  VALUES(NEXT VALUE FOR person_id_seq, 'Jessica', 'Joe', 63, 53);
230 INSERT INTO Person(person_id, first_name, last_name, height_inches, age)
231  VALUES(NEXT VALUE FOR person_id_seq, 'Katie', 'Koe', 70, 48);
232 INSERT INTO Person(person_id, first_name, last_name, height_inches, age)
233  VALUES(NEXT VALUE FOR person_id_seq, 'Liam', 'Loe', 64, 22);
234
235  SELECT * FROM Person;
```

121 %

Results | Messages

	person_id	first_name	last_name	height_inches	age
1	1	Abby	Aoe	60	78
2	2	Brian	Boe	63	51
3	3	Charlie	Coe	67	20
4	4	Doe	Doe	69	31
5	5	Elliott	Eoe	72	21
6	6	Finley	Foe	70	34
7	7	Graham	Goe	68	43
8	8	Hudson	Hoe	74	53
9	9	Isabella	Ioe	64	61
10	10	Jessica	Joe	63	53
11	11	Katie	Koe	70	48
12	12	Liam	Loe	64	22

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

```
238 -- Defines views.
239 CREATE OR ALTER VIEW View_age_younger_30 AS
240 SELECT * FROM Person WHERE age < 30;
241
242 CREATE OR ALTER VIEW View_age_between_30_and_60 AS
243 SELECT * FROM Person WHERE age >= 30 AND age <= 60;
244
245 CREATE OR ALTER VIEW View_age_older_60 AS
246 SELECT * FROM Person WHERE age > 60;
```

121 %

Messages

```
Commands completed successfully.
```

```
248 -- Uses views.
249 SELECT * FROM View_age_younger_30;
```

121 %

Results | Messages

	person_id	first_name	last_name	height_inches	age
1	3	Charlie	Coe	67	20
2	5	Elliott	Eoe	72	21
3	12	Liam	Loe	64	22

```
250 SELECT * FROM View_age_between_30_and_60;
```

121 %

Results | Messages

	person_id	first_name	last_name	height_inches	age
1	2	Brian	Boe	63	51
2	4	Doe	Doe	69	31
3	6	Finley	Foe	70	34
4	7	Graham	Goe	68	43
5	8	Hudson	Hoe	74	53
6	10	Jessica	Joe	63	53
7	11	Katie	Koe	70	48

```
251 SELECT * FROM View_age_older_60;
```

121 %

Results | Messages

	person_id	first_name	last_name	height_inches	age
1	1	Abby	Aoe	60	78
2	9	Isabella	Ioe	64	61

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

```
254 SELECT * FROM View_age_younger_30
255 UNION
256 SELECT * FROM View_age_between_30_and_60
257 UNION
258 SELECT * FROM View_age_older_60;
```

121 %

Results | Messages

	person_id	first_name	last_name	height_inches	age
1	1	Abby	Aoe	60	78
2	2	Brian	Boe	63	51
3	3	Charlie	Coe	67	20
4	4	Doe	Doe	69	31
5	5	Elliott	Eoe	72	21
6	6	Finley	Foe	70	34
7	7	Graham	Goe	68	43
8	8	Hudson	Hoe	74	53
9	9	Isabella	Ioe	64	61
10	10	Jessica	Joe	63	53
11	11	Katie	Koe	70	48
12	12	Liam	Loe	64	22

6. *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.

```
261 -- Defines views.
262 CREATE OR ALTER VIEW View_names AS
263 SELECT person_id, first_name, last_name FROM Person;
264
265 CREATE OR ALTER VIEW View_heights_and_ages AS
266 SELECT person_id, height_inches, age FROM Person;
```

121 %

Messages

```
Commands completed successfully.
```

```
268 -- Uses views.
269 SELECT * FROM View_names;
```

121 %

Results | Messages

	person_id	first_name	last_name
1	1	Abby	Aoe
2	2	Brian	Boe
3	3	Charlie	Coe
4	4	Doe	Doe
5	5	Elliott	Eoe
6	6	Finley	Foe
7	7	Graham	Goe
8	8	Hudson	Hoe
9	9	Isabella	Ioe
10	10	Jessica	Joe
11	11	Katie	Koe
12	12	Liam	Loe

```
270 SELECT * FROM View_heights_and_ages;
```

121 %

Results | Messages

	person_id	height_inches	age
1	1	60	78
2	2	63	51
3	3	67	20
4	4	69	31
5	5	72	21
6	6	70	34
7	7	68	43
8	8	74	53
9	9	64	61
10	10	63	53
11	11	70	48
12	12	64	22

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

```
273 SELECT View_names.person_id, first_name, last_name, height_inches, age FROM View_names
274 JOIN View_heights_and_ages ON View_heights_and_ages.person_id = View_names.person_id;
```

121 %

Results | Messages

	person_id	first_name	last_name	height_inches	age
1	1	Abby	Aoe	60	78
2	2	Brian	Boe	63	51
3	3	Charlie	Coe	67	20
4	4	Doe	Doe	69	31
5	5	Elliott	Eoe	72	21
6	6	Finley	Foe	70	34
7	7	Graham	Goe	68	43
8	8	Hudson	Hoe	74	53
9	9	Isabella	Ioe	64	61
10	10	Jessica	Joe	63	53
11	11	Katie	Koe	70	48
12	12	Liam	Loe	64	22