

# **Data Management – Applications – C170**

**By**

**Brandon Jones**

This document contains my submission for the C170 performative assessment. I have included the instructions, step by step and have included my solutions. Images have been attached where applicable, otherwise my text solution has been denoted by red text color.

## **Scenario**

You are a database designer and developer who has been hired by two local businesses, Nora's Bagel Bin and Jaunty Coffee Co., to build databases to help them manage their businesses. First, you will design a normalized physical database model to store data for Nora's Bagel Bin's ordering system. Then, you will use an existing database design document for Jaunty Coffee Co. to create its database. Once the tables have been built, you will load them with sample data and create a view and an index to protect and improve query performance. Finally, you will create both a simple query and a more complex table joins query to produce meaningful reports from the newly created database.

## Nora's Bagel Bin Normalized Physical Database Model

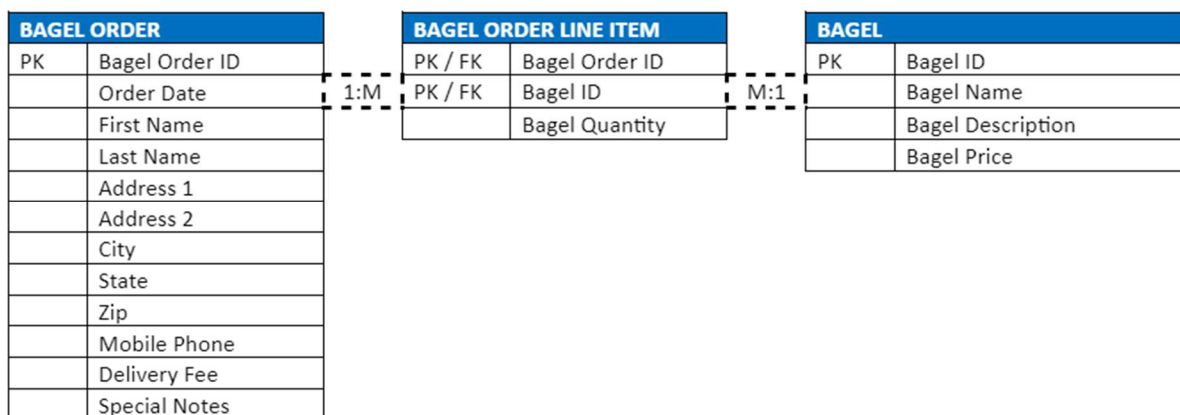
### A. Construct a normalized physical database model to represent the ordering process for Nora's Bagel Bin by doing the following:

#### 1. Complete the second normal form (2NF) section of the attached "Nora's Bagel Bin Database Blueprints" document by doing the following:

Assign each attribute from the 1NF table into the correct 2NF table and describe the relationship between the two pairs of 2NF tables by indicating their cardinality in each of the dotted cells: one-to-one (1:1), one-to-many (1:M), many-to-one (M:1), or many-to-many (M:M).

Explain how you assigned attributes to the 2NF tables and determined the cardinality of the relationships between your 2NF tables.

##### Second Normal Form (2NF)



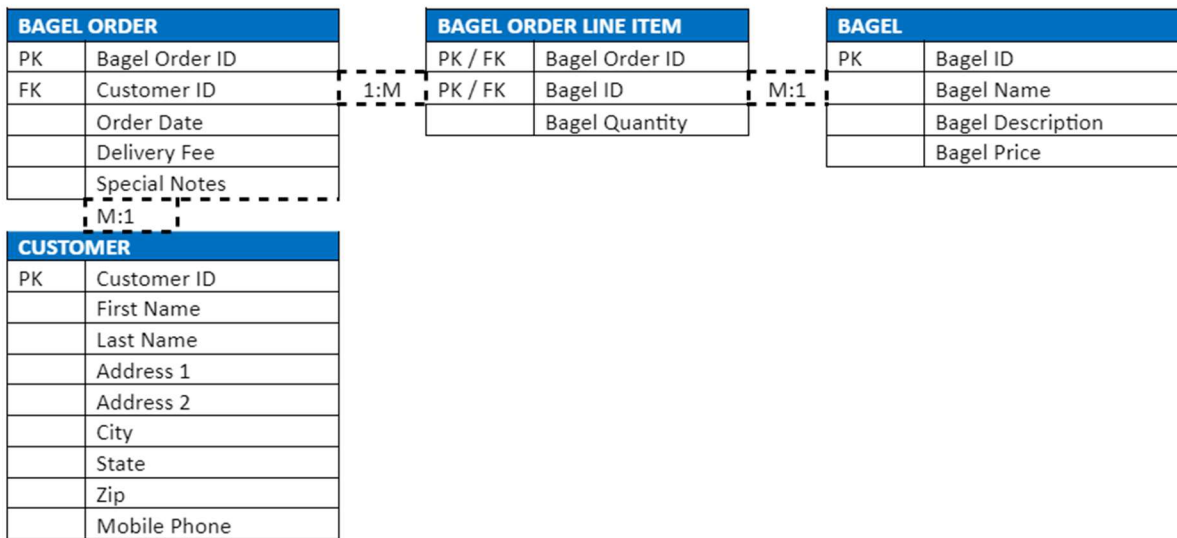
I assigned each attribute by starting with adding every attribute that only applied to bagel to the bagel 2NF table. Since we have a line item 2NF table, I knew that our Bagel Quantity would belong to this table instead of the Bagel Order table. The last step then, was adding each attribute that applied to the bagel order to the Bagel Order 2NF table. A bagel order can have multiple line items and one bagel can exist in multiple line items, however only one bagel can exist in each line item. For this reason, the relationship between bagel order and line item is a one to many, and the relationship between a line item and a bagel is a many to one.

#### 2. Complete the third normal form (3NF) section of the attached "Nora's Bagel Bin Database Blueprints" document by doing the following:

- Assign each attribute from your 2NF "Bagel Order" table into one of the new 3NF tables. Copy all other information from your 2NF diagram into the 3NF diagram.
- Provide each 3NF table with a name that reflects its contents.

- c. Create a new field that will be used as a key linking the two 3NF tables you named in part A2b. Ensure that your primary key (PK) and foreign key (FK) fields are in the correct locations in the 3NF diagram.
- d. Describe the relationships between the 3NF tables by indicating their cardinality in each of the dotted cells: one-to-one (1:1), one-to-many (1:M), many-to-one (M:1), or many-to-many (M:M).
- e. Explain how you assigned attributes to the 3NF tables and determined the cardinality of the relationships between your 3NF tables.

**Third Normal Form (3NF)**

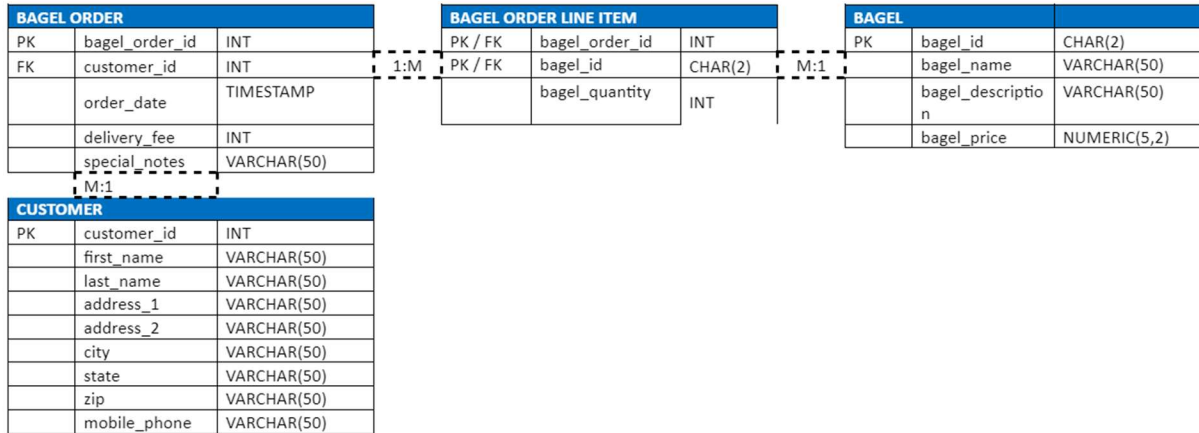


Since in the 2NF mockup, our bagel order table had repeating data that did not pertain to the bagel order, I took all of the repeating customer data and placed that into a new table, named Customer and added a Customer ID that acted as a primary key for Customer and a foreign key for Bagel Order. The cardinality of the pre-existing tables is the same as the last example, however we do have a new relationship between Bagel Order and customer. This relationship is a many to one, as a one customer can have many bagel orders, however each bagel order must have only one customer.

**3. Complete the "Final Physical Database Model" section of the attached “Nora’s Bagel Bin Database Blueprints” document by doing the following:**

- Copy the table names and cardinality information from your 3NF diagram into the “Final Physical Database Model” and rename the attributes.
- Assign one of the following five data types to each attribute in your 3NF tables: CHAR(), VARCHAR(), TIMESTAMP, INTEGER, or NUMERIC(). Each data type must be used at least once.

**Final Physical Database Model**



## B. Create a database using the attached "Jaunty Coffee Co. ERD" by doing the following:

1. Develop SQL code to create *each* table as specified in the attached "Jaunty Coffee Co. ERD" by doing the following:
  - a. Provide the SQL code you wrote to create *all* the tables.
  - b. Demonstrate that you tested your code by providing a screenshot showing your SQL commands and the database server's response.

The screenshot shows the SQL Fiddle interface with the following SQL code in the editor:

```
1 CREATE TABLE COFFEE_SHOP (  
2   shop_id INTEGER PRIMARY KEY AUTO_INCREMENT,  
3   shop_name VARCHAR(50),  
4   city VARCHAR(50),  
5   state CHAR(2)  
6 );  
7  
8 CREATE TABLE EMPLOYEE (  
9   employee_id INT PRIMARY KEY AUTO_INCREMENT,  
10  first_name VARCHAR(50),  
11  last_name VARCHAR(50),  
12  hire_date DATE,  
13  job_title VARCHAR(50),  
14  shop_id INTEGER,  
15  FOREIGN KEY (shop_id) REFERENCES COFFEE_SHOP(shop_id)  
16 );  
17  
18 CREATE TABLE SUPPLIER (  
19  supplier_id INT AUTO_INCREMENT,  
20  company_name VARCHAR(50),  
21  country VARCHAR(50),  
22  sales_contact_name VARCHAR(80),  
23  email VARCHAR(50),  
24  PRIMARY KEY (supplier_id)  
25 );  
26  
27 CREATE TABLE COFFEE (  
28  coffee_id INT AUTO_INCREMENT,  
29  shop_id INT,  
30  coffee_name VARCHAR(50),  
31  price_per_pound NUMERIC(5,2),  
32  PRIMARY KEY (coffee_id),  
33  FOREIGN KEY (shop_id) REFERENCES COFFEE_SHOP(shop_id),  
34  FOREIGN KEY (supplier_id) REFERENCES SUPPLIER(supplier_id)  
35 );
```

The interface includes buttons for "Build Schema", "Edit Fullscreen", "Browser", and "Run SQL". A status bar at the bottom indicates "Schema Ready".

2. Develop SQL code to populate *each* table in the database design document by doing the following:
  - a. Provide the SQL code you wrote to populate the tables with *at least three* rows of data in *each* table.
  - b. Demonstrate that you tested your code by providing a screenshot showing your SQL commands and the database server's response.

The screenshot shows the SQL Fiddle interface with the following SQL code in the editor:

```
36 INSERT INTO COFFEE_SHOP (shop_name, city, state) VALUES ("Brewland", "Vukon", "OK");  
37 INSERT INTO COFFEE_SHOP (shop_name, city, state) VALUES ("Don't", "Eik City", "OK");  
38  
39 INSERT INTO EMPLOYEE (first_name, last_name, hire_date, job_title, shop_id) VALUES ("John", "Alasaster", "2000-01-01", "Barista", "1");  
40 INSERT INTO EMPLOYEE (first_name, last_name, hire_date, job_title, shop_id) VALUES ("Adrian", "Delore", "2007-03-22", "Manager", "3");  
41 INSERT INTO EMPLOYEE (first_name, last_name, hire_date, job_title, shop_id) VALUES ("Jacob", "Daugherty", "2001-10-05", "Cashier", "2");  
42  
43 INSERT INTO SUPPLIER (company_name, country, sales_contact_name, email) VALUES ("Blast Coffee", "Norway", "Donald Burch", "sales@blastcoffee.com");  
44 INSERT INTO SUPPLIER (company_name, country, sales_contact_name, email) VALUES ("Noisleep", "United Kingdom", "Frank", "sales@noisleep.com");  
45 INSERT INTO SUPPLIER (company_name, country, sales_contact_name, email) VALUES ("Duke", "United States", "Alice Thomas", "sales@duke.com");  
46  
47 INSERT INTO COFFEE (shop_id, supplier_id, coffee_name, price_per_pound) VALUES ("1", "2", "adrenaline", "13.40");  
48 INSERT INTO COFFEE (shop_id, supplier_id, coffee_name, price_per_pound) VALUES ("2", "3", "Auker", "23.80");  
49 INSERT INTO COFFEE (shop_id, supplier_id, coffee_name, price_per_pound) VALUES ("2", "2", "adrenaline", "13.40");
```

The interface includes buttons for "Build Schema", "Edit Fullscreen", "Browser", and "Run SQL". A status bar at the bottom indicates "Schema Ready".

3. Develop SQL code to create a view by doing the following:
  - a. Provide the SQL code you wrote to create your view. The view should show *all* of the information from the "Employee" table but concatenate *each* employee's first and last name, formatted with a space between the first and last name, into a new attribute called `employee_full_name`.
  - b. Demonstrate that you tested your code by providing a screenshot showing your SQL commands and the database server's response.

The screenshot shows the SQL Fiddle interface with the following SQL code in the editor:

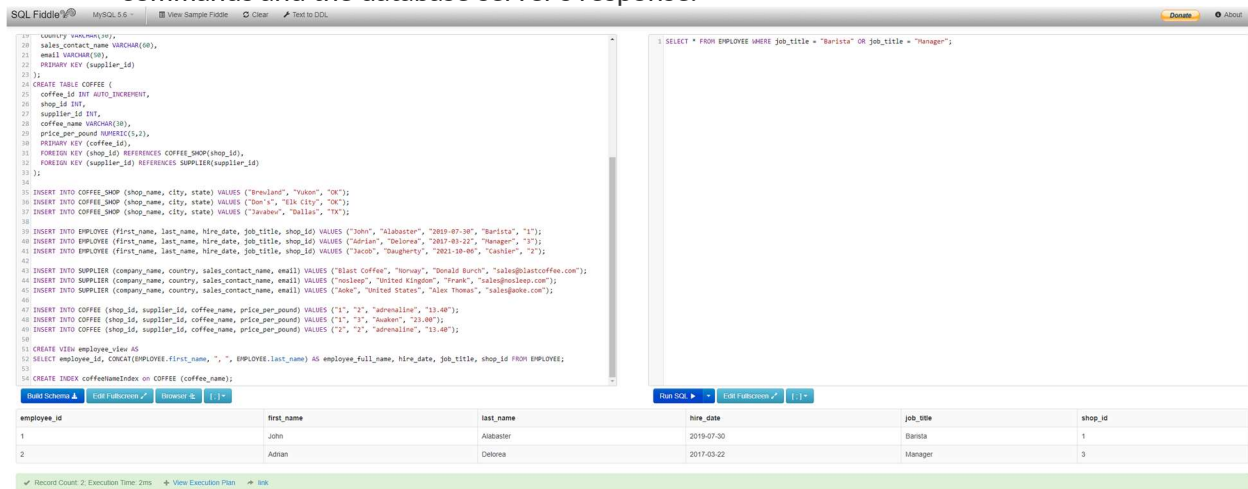
```
51 CREATE VIEW employee_view AS  
52 SELECT employee_id, CONCAT(EMPLOYEE.first_name, " ", EMPLOYEE.last_name) AS employee_full_name, hire_date, job_title, shop_id FROM EMPLOYEE;
```

The interface includes buttons for "Build Schema", "Edit Fullscreen", "Browser", and "Run SQL". A status bar at the bottom indicates "Schema Ready".

4. Develop SQL code to create an index on the coffee\_name field by doing the following:
  - a. Provide the SQL code you wrote to create your index on the coffee\_name field from the “Coffee” table.
  - b. Demonstrate that you tested your code by providing a screenshot showing your SQL commands and the database server’s response.



5. Develop SQL code to create an SFW (SELECT-FROM-WHERE) query for *any* of your tables or views by doing the following:
  - a. Provide the SQL code you wrote to create your SFW query.
  - b. Demonstrate that you tested your code by providing a screenshot showing your SQL commands and the database server’s response.



6. Develop SQL code to create a query by doing the following:
  - a. Provide the SQL code you wrote to create your table joins query. The query should join together **three** different tables and include attributes from *all* three tables in its output.
  - b. Demonstrate that you tested your code by providing a screenshot showing your SQL commands and the database server’s response.

SOL FiddleMySQL 5.6View Sample FiddleClearText to DDLDonateAbout

```
1 CREATE TABLE COFFEE_SHOP (
2   shop_id INTEGER PRIMARY KEY AUTO_INCREMENT,
3   shop_name VARCHAR(50),
4   city VARCHAR(50),
5   state CHAR(2)
6 );
7
8 CREATE TABLE EMPLOYEE (
9   employee_id INT PRIMARY KEY AUTO_INCREMENT,
10  first_name VARCHAR(50),
11  last_name VARCHAR(50),
12  hire_date DATE,
13  job_title VARCHAR(50),
14  shop_id INTEGER,
15  FOREIGN KEY (shop_id) REFERENCES COFFEE_SHOP(shop_id)
16 );
17
18 CREATE TABLE SUPPLIER (
19   supplier_id INT AUTO_INCREMENT,
20   company_name VARCHAR(50),
21   country VARCHAR(50),
22   sales_contact_name VARCHAR(50),
23   email VARCHAR(50),
24   PRIMARY KEY (supplier_id)
25 );
26
27 CREATE TABLE COFFEE (
28   coffee_id INT AUTO_INCREMENT,
29   shop_id INT,
30   supplier_id INT,
31   coffee_name VARCHAR(50),
32   price_per_pound NUMERIC(5,2),
33   PRIMARY KEY (coffee_id),
34   FOREIGN KEY (shop_id) REFERENCES COFFEE_SHOP(shop_id),
35   FOREIGN KEY (supplier_id) REFERENCES SUPPLIER(supplier_id)
36 );
37
38 INSERT INTO COFFEE_SHOP (shop_name, city, state) VALUES ('Brewland', 'Tulsa', 'OK');
39
40 INSERT INTO EMPLOYEE (first_name, last_name, hire_date, job_title, shop_id) VALUES ('John', 'Doe', '2015-01-01', 'Barista', 1);
```

```
1 SELECT shop_name, coffee_name, company_name FROM COFFEE_SHOP
2 JOIN COFFEE ON COFFEE_SHOP.shop_id=COFFEE.shop_id
3 JOIN SUPPLIER ON COFFEE.supplier_id=SUPPLIER.supplier_id
```

Run SQLAdd FiddleFullscreen

shop_name	coffee_name	company_name
Brewland	adrenaline	nosleep
Brewland	Awaken	Acho
Doris	adrenaline	nosleep

Record Count: 3Execution Time: 6msView Execution PlanLink

Did this query solve the problem? If no, consider donating \$5 to help make sure SOL Fiddle will be here next time you need help with a database problem. Thank!