Performance Assessment: Data Management - Applications - VHT2

Andrea Hayes

Western Governors University

Data Management - Applications – C170

May 30th, 2022

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:

A1a. Assign *each* attribute from the 1NF table into the correct 2NF table.

A1b. 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).

Nora's Bagel Bin Database Blueprints

Second Normal Form (2NF)

BAGEL ORDER	
PK	Bagel Order ID
	Order Date
	First Name
	Last Name
	Address 1
	Address 2
	City
	State
	Zip
	Mobile Phone
	Delivery Fee
	Special Notes

1:M

BAGEL ORDER LINE ITEM	
PK / FK	Bagel Order ID
PK / FK	Bagel ID
	Bagel Quantity

1:1

BAGEL	
PK	Bagel ID
	Bagel Name
	Bagel Description
	Bagel Price

A1c. Explain how you assigned attributes to the 2NF tables and determined the cardinality of the relationships between your 2NF tables.

To move the 1NF table in Nora's Bagel Bin Database Blueprints to 2NF form, I first separated the data into three tables: 'BAGEL ORDER', 'BAGEL ORDER LINE ITEM', and 'BAGEL'. In the 'BAGEL ORDER' table, all the attributes that depend solely on the *Bagel Order ID* were implemented and the *Bagel Order ID* was made into the primary key. The order date, name, address, city, state, zip, mobile phone, delivery fee, and special notes, depend solely on the *Bagel Order ID* and do *not* depend on the *Bagel ID*.

In the 'BAGEL' table, I included only the attributes that depend solely on the *Bagel ID*: the bagel name, description, and price. These attributes are not dependent upon the *Bagel Order ID*.

Lastly, for the associative entity, the 'BAGEL ORDER LINE ITEM' table was created, with both the *Bagel Order ID* and the *Bagel ID* listed as the PK / FK that links the other two tables. The only attribute included in this table is *Bagel Quantity*, because you cannot determine *Bagel Quantity* with the *Bagel Order ID* or the *Bagel ID* alone. You cannot determine the *Bagel Quantity* without being able to reference the data in the bagel order form, which is included in the 'BAGEL ORDER' table. Additionally, you cannot determine the *Bagel Quantity* without the *Bagel ID* in the 'BAGEL TABLE', because there are various types of bagels, and you wouldn't know which type of bagel the quantity was referencing.

For the cardinality of the two tables, I deduced the minimum and maximum number of attributes for each entity that could logically be related to the corresponding entity.

For the cardinality relationship between the 'BAGEL ORDER' and the 'BAGEL ORDER LINE ITEM':

- **One Bagel Order contains one to many Bagel Order Line Items.**
- **One to many Bagel Order Line Item is contained by one and only one Bagel Order.**
- **This results in a cardinality of 1:M.**

For the cardinality relationship between the 'BAGEL ORDER LINE ITEM' and the 'BAGEL ID':

- **One Bagel Order Line Item can include one and only one Bagel ID. (**One line item cannot include multiple Bagel IDs).
- **One Bagel ID can be included on one and only one Bagel Order Line Item**. (One Bagel ID cannot be listed on many Bagel Order Line Items within the order form).
- **This results in a cardinality of 1:1.**

A2. Complete the third normal form (3NF) section of the attached "Nora's Bagel Bin Database Blueprints" document by doing the following:

A2a. 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.

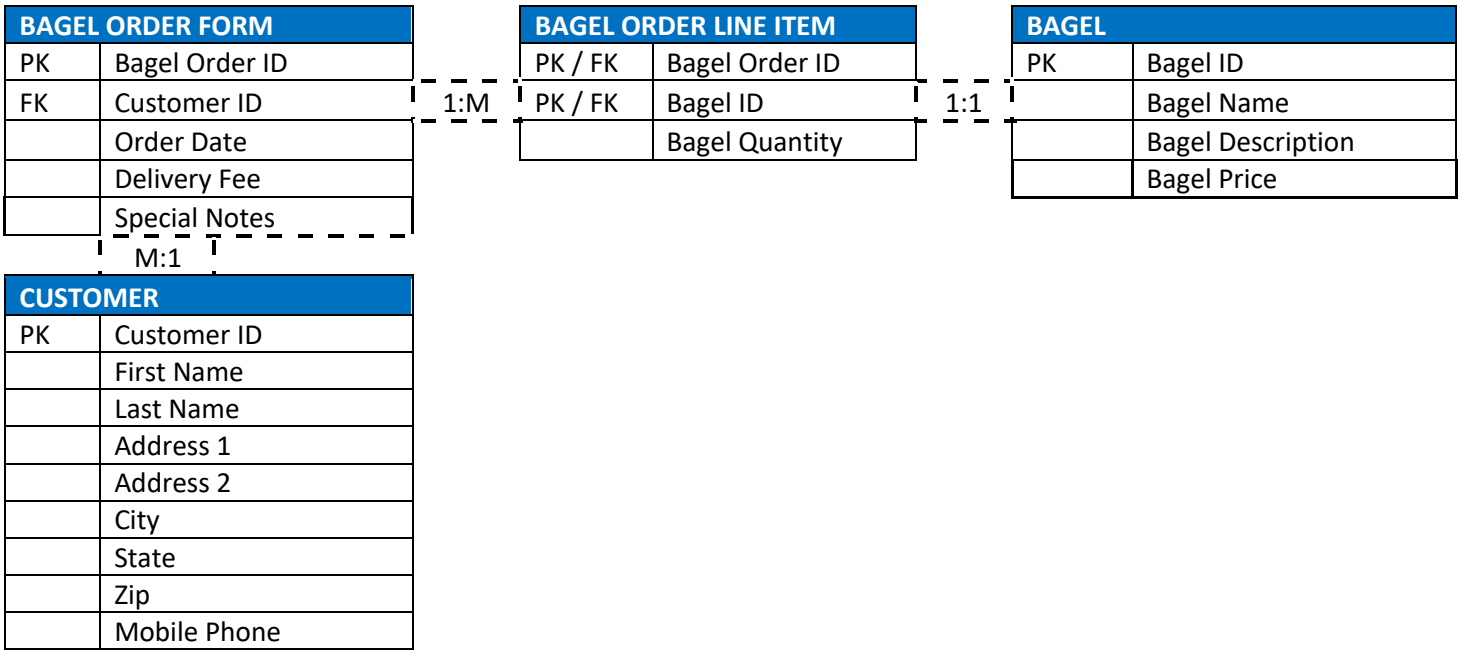
A2b. Provide *each* 3NF table with a name that reflects its contents.

A2c. 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.

A2d. 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).

Nora's Bagel Bin Database Blueprints

Third Normal Form (3NF)



A2e. Explain how you assigned attributes to the 3NF tables and determined the cardinality of the relationships between your 3NF tables.

To move the table from 2NF to 3NF, it was first necessary to determine which of the remaining attributes might still potentially repeat. I determined that any attributes from the 'BAGEL ORDER' table associated with the customer should be removed and placed in their own table, so to avoid redundancy within the database. To illustrate how this might occur, one customer may place multiple orders, which would lead to the customer attributes being stored multiple times.

For the new table, I named it 'CUSTOMER' and I created a new attribute *Customer ID*. This new attribute serves to link the 'BAGEL ORDER FORM' and 'CUSTOMER' table together by operating as the primary key for 'CUSTOMER' and the foreign key for 'BAGEL ORDER FORM'.

For the cardinality relationship between 'BAGEL ORDER FORM' and 'CUSTOMER':

- **One Bagel Order Form can name one and only one customer.**
- **One Customer can be named on one-to-many Bagel Order Forms.**
- **This results in a cardinality of M:1.**

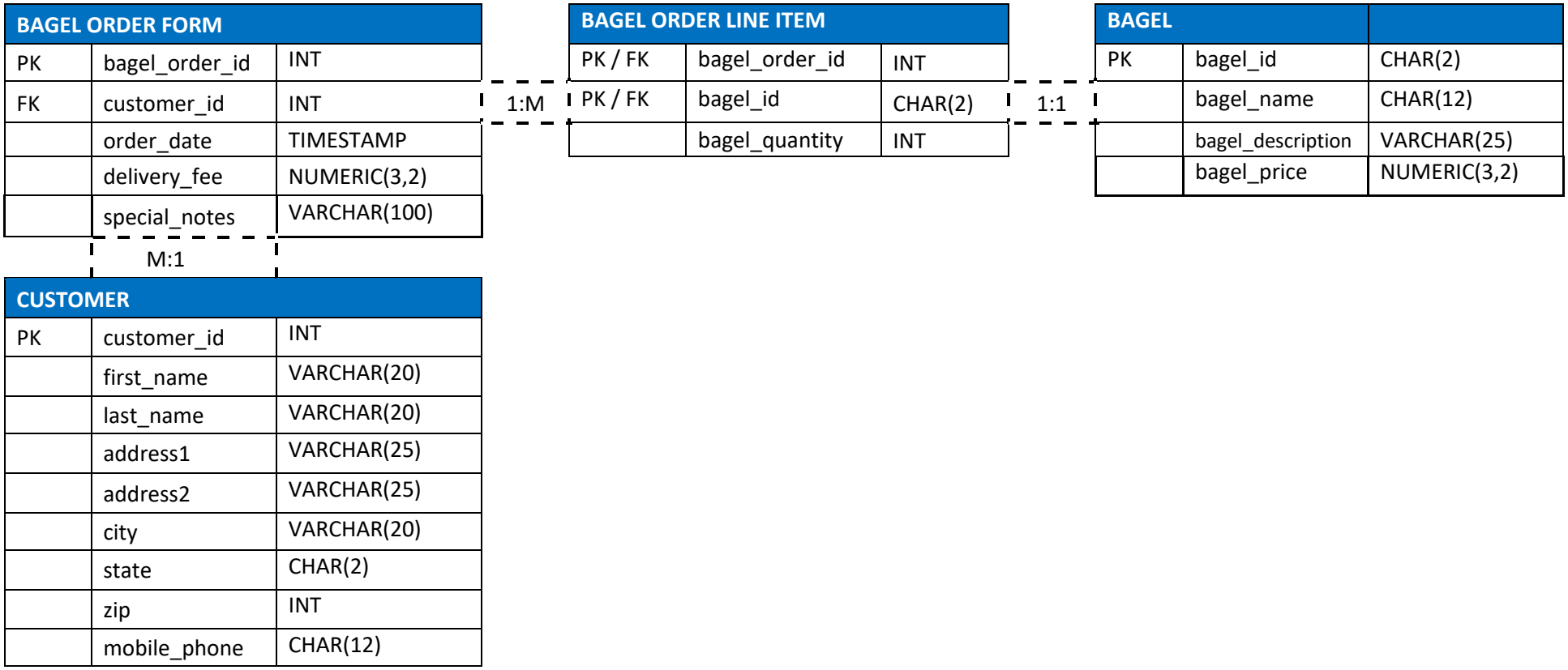
A3. Complete the "Final Physical Database Model" section of the attached "Nora's Bagel Bin Database Blueprints" document by doing the following:

A3a. Copy the table names and cardinality information from your 3NF diagram into the "Final Physical Database Model" and rename the attributes.

A3b. 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.

Nora's Bagel Bin Database Blueprints

Final Physical Database Model



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

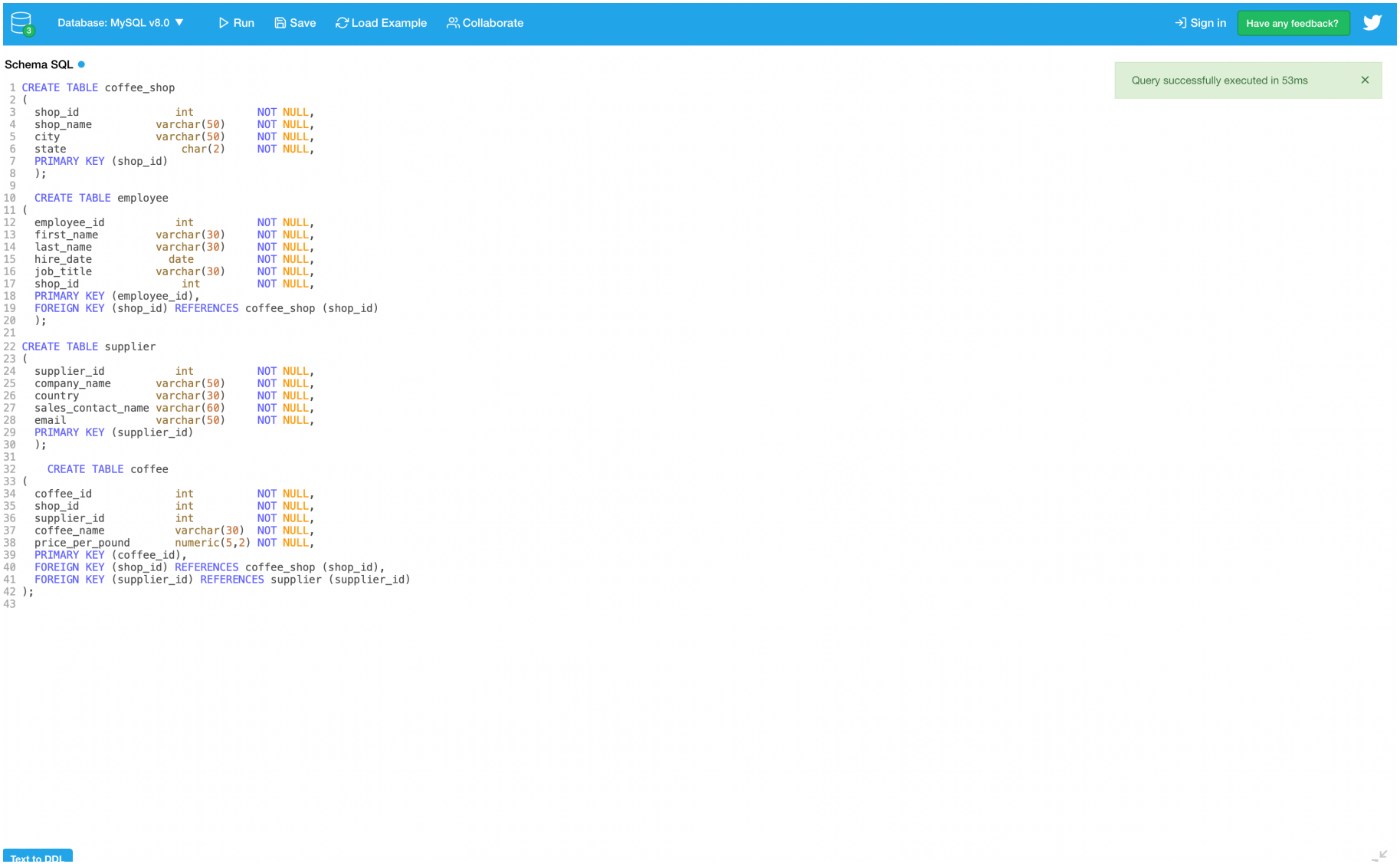
B1. Develop SQL code to create *each* table as specified in the attached "Jaunty Coffee Co. ERD" by doing the following:

B1a. Provide the SQL code you wrote to create *all* the tables.

```
CREATE TABLE coffee_shop
(
 shop_id          int          NOT NULL,
 shop_name        varchar(50)  NOT NULL,
 city             varchar(50)  NOT NULL,
 state            char(2)      NOT NULL,
 PRIMARY KEY (shop_id)
 );

CREATE TABLE employee
(
 employee_id      int          NOT NULL,
 first_name       varchar(30)  NOT NULL,
 last_name        varchar(30)  NOT NULL,
 hire_date        date         NOT NULL,
 job_title        varchar(30)  NOT NULL,
 shop_id          int          NOT NULL,
 PRIMARY KEY (employee_id),
 FOREIGN KEY (shop_id) REFERENCES coffee_shop (shop_id)
 );

CREATE TABLE supplier
(
 supplier_id      int          NOT NULL,
```

```
  company_name       varchar(50)     NOT NULL,
  country            varchar(30)     NOT NULL,
  sales_contact_name varchar(60)     NOT NULL,
  email              varchar(50)     NOT NULL,
  PRIMARY KEY (supplier_id)
  );

CREATE TABLE coffee
(
  coffee_id          int         NOT NULL,
  shop_id            int         NOT NULL,
  supplier_id        int         NOT NULL,
  coffee_name        varchar(30)  NOT NULL,
  price_per_pound    numeric(5,2) NOT NULL,
  PRIMARY KEY (coffee_id),
  FOREIGN KEY (shop_id) REFERENCES coffee_shop (shop_id),
  FOREIGN KEY (supplier_id) REFERENCES supplier (supplier_id)
);
```

B1b. Demonstrate that you tested your code by providing a screenshot showing your SQL commands and the database server's response.



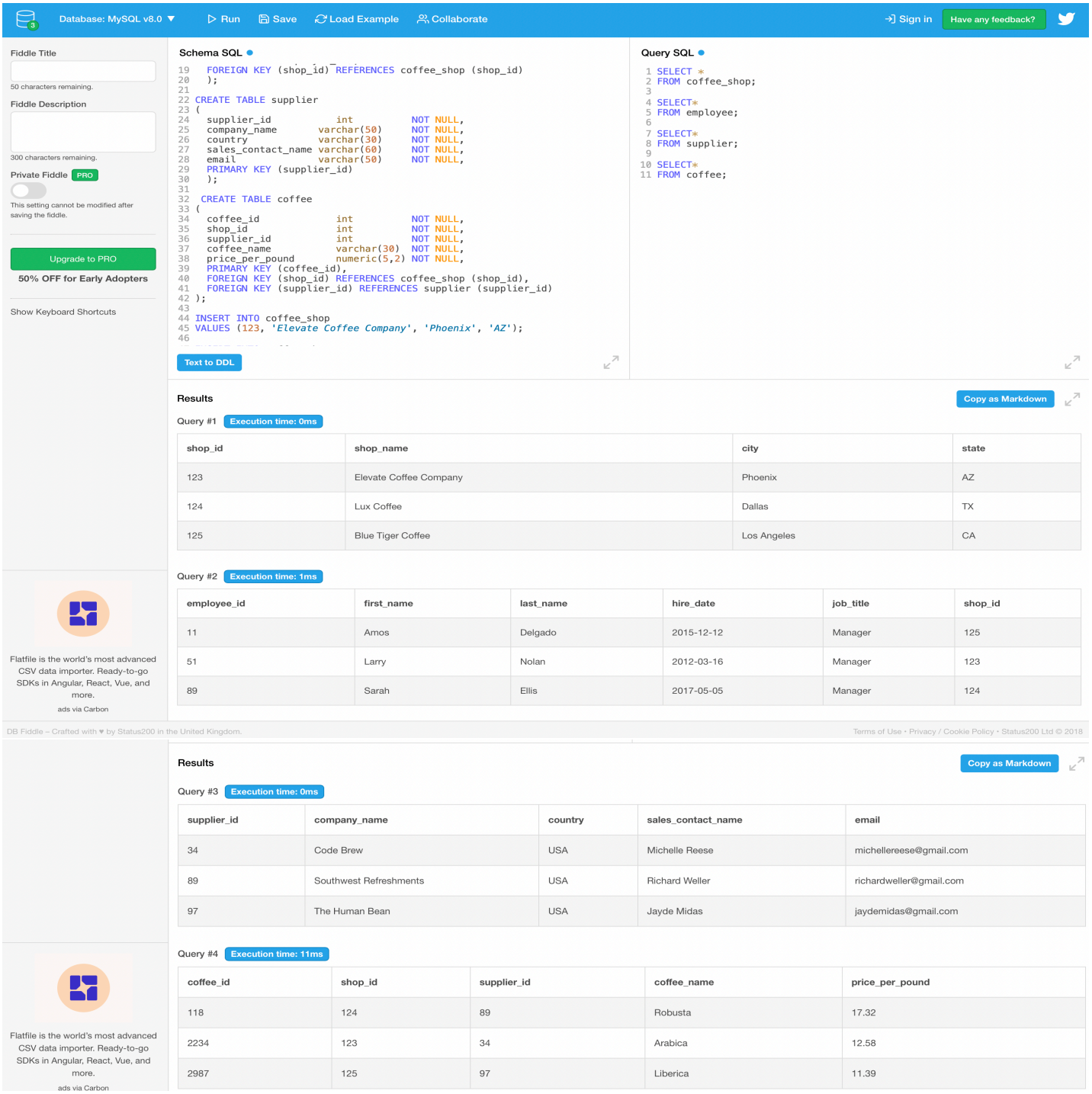
B2. Develop SQL code to populate *each* table in the database design document by doing the following:

B2a. Provide the SQL code you wrote to populate the tables with *at least* three rows of data in *each* table.

```
INSERT INTO coffee_shop
VALUES (123, 'Elevate Coffee Company', 'Phoenix', 'AZ');

INSERT INTO coffee_shop
VALUES (124, 'Lux Coffee', 'Dallas', 'TX');

INSERT INTO coffee_shop
VALUES (125, 'Blue Tiger Coffee', 'Los Angeles', 'CA');

INSERT INTO employee
VALUES (51, 'Larry', 'Nolan', '2012-03-16', 'Manager', 123);

INSERT INTO employee
VALUES (89, 'Sarah', 'Ellis', '2017-05-05', 'Manager', 124);

INSERT INTO employee
VALUES (11, 'Amos', 'Delgado', '2015-12-12', 'Manager', 125);

INSERT INTO supplier
VALUES (89, 'Southwest Refreshments', 'USA', 'Richard Weller',
  'richardweller@gmail.com');

INSERT INTO supplier
VALUES (97, 'The Human Bean', 'USA', 'Jayde Midas', 'jaydemidas@gmail.com');
```

```
INSERT INTO supplier
VALUES (34, 'Code Brew', 'USA', 'Michelle Reese', 'michellereese@gmail.com');

INSERT INTO coffee
VALUES (2234, 123, 34, 'Arabica', 12.58);

INSERT INTO coffee
VALUES (118, 124, 89, 'Robusta', 17.32);

INSERT INTO coffee
VALUES (2987, 125, 97, 'Liberica', 11.39);
```

B2b. Demonstrate that you tested your code by providing a screenshot showing your SQL commands and the database server's response.

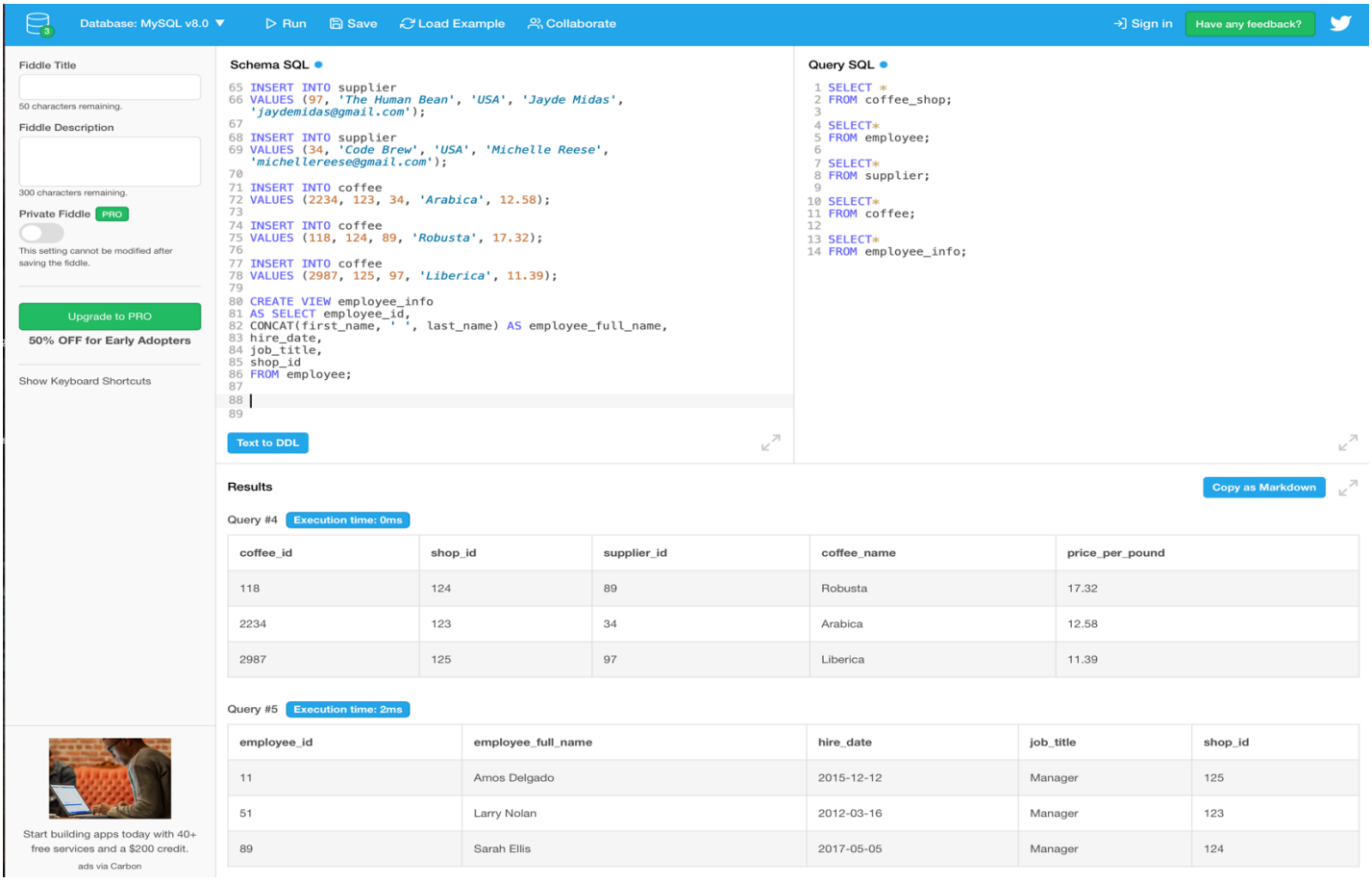


3. Develop SQL code to create a view by doing the following:

3a. 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.

```
CREATE VIEW employee_info
AS SELECT employee_id,
CONCAT(first_name, ' ', last_name) AS employee_full_name,
hire_date,
job_title,
shop_id
FROM employee;
```

3b. Demonstrate that you tested your code by providing a screenshot showing your SQL commands and the database server's response.

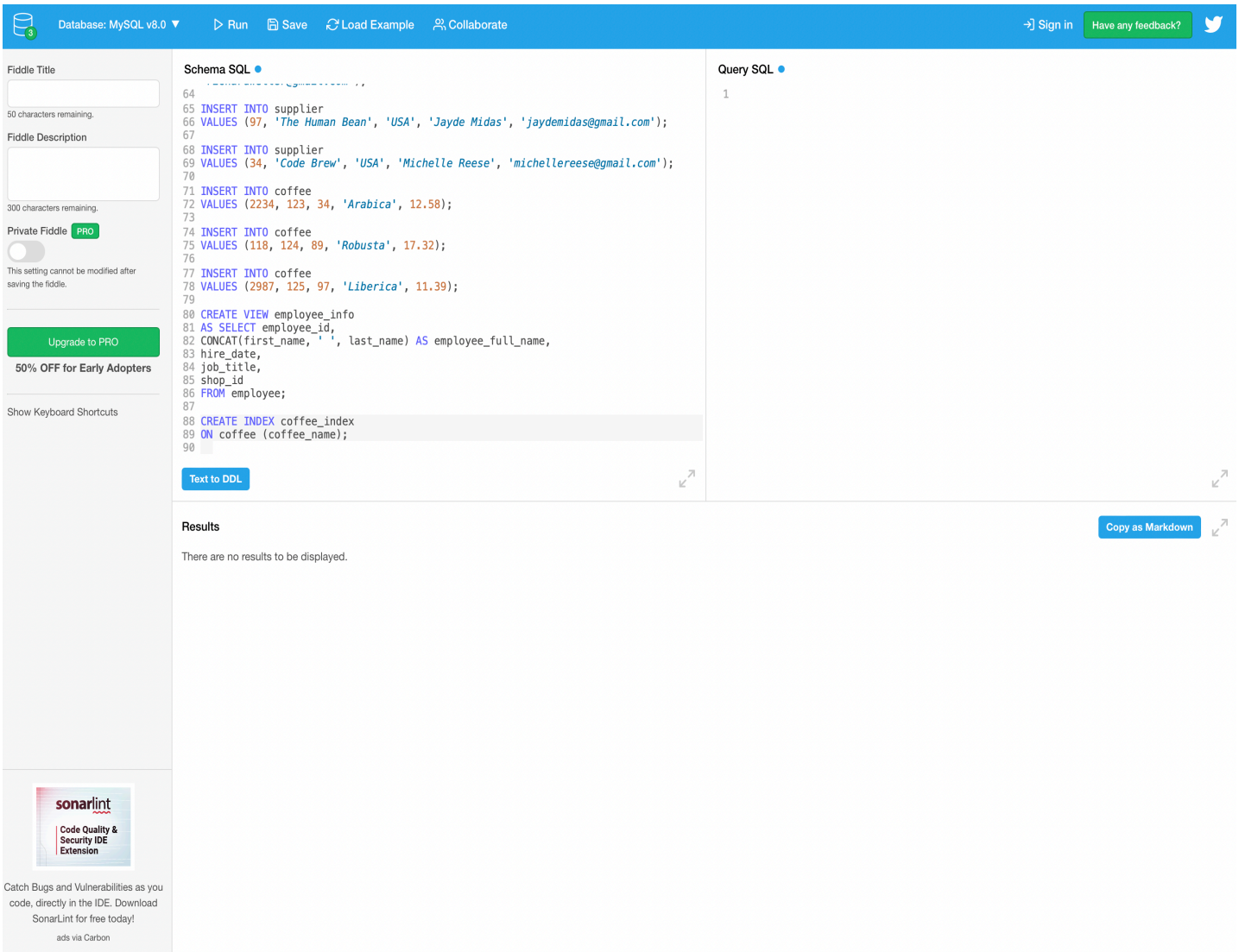


4. Develop SQL code to create an index on the coffee_name field by doing the following:

4a. Provide the SQL code you wrote to create your index on the coffee_name field from the "Coffee" table.

```
CREATE INDEX coffee_index
ON coffee (coffee_name);
```

4b. Demonstrate that you tested your code by providing a screenshot showing your SQL commands and the database server's response.

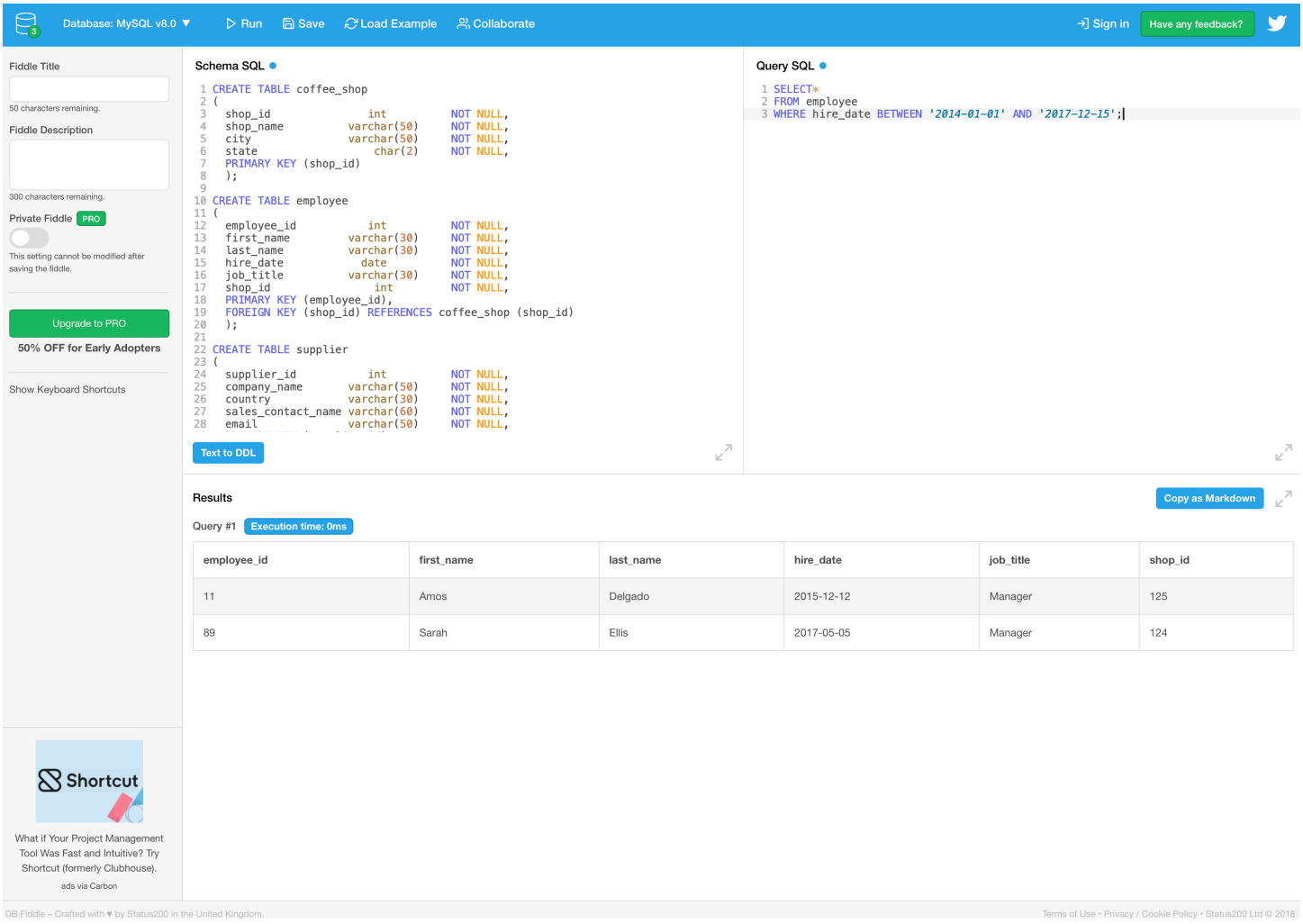


5. Develop SQL code to create an SFW (SELECT–FROM–WHERE) query for *any* of your tables or views by doing the following:

5a. Provide the SQL code you wrote to create your SFW query.

```
SELECT*

FROM employee

WHERE hire_date BETWEEN '2014-01-01' AND '2017-12-15';
```

5b. 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:

6a. 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.

```
SELECT shop_name, company_name, coffee_name

FROM coffee_shop

CROSS JOIN supplier

CROSS JOIN coffee;
```

6b. Demonstrate that you tested your code by providing a screenshot showing your SQL commands and the database server's response.