

**Performance Assessment: Data Management - Applications - VHT2**

Andrea Hayes

Western Governors University

Data Management - Applications – C170

May 30<sup>th</sup>, 2022

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

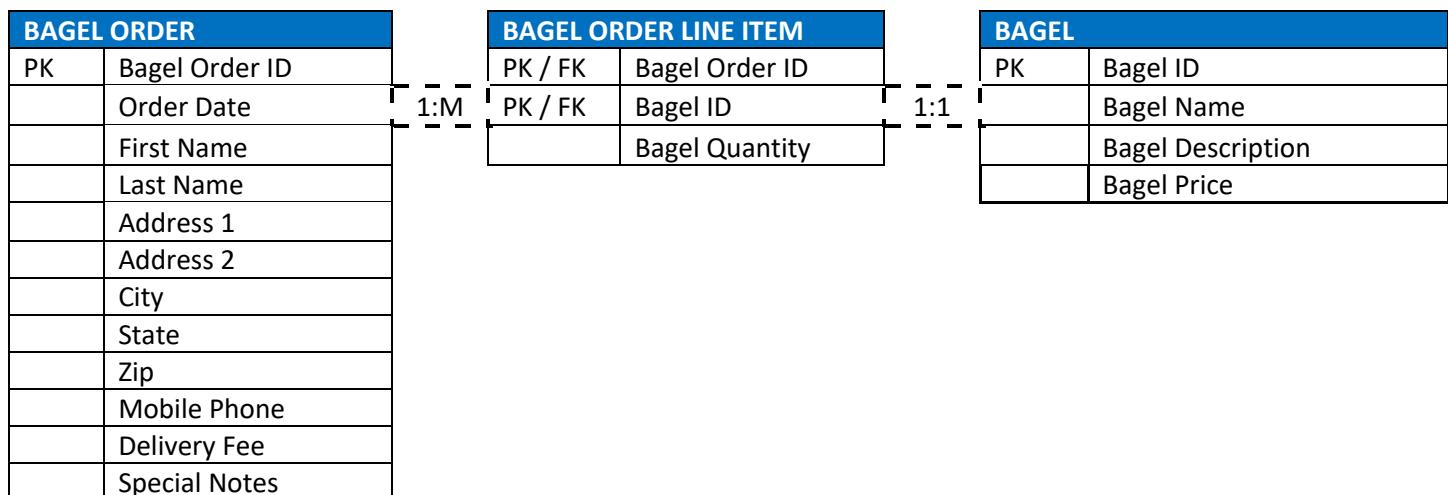
**1. Complete the second normal form (2NF) section of the attached “Nora’s Bagel Bin Database Blueprints” document by doing the following:**

**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)



**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.

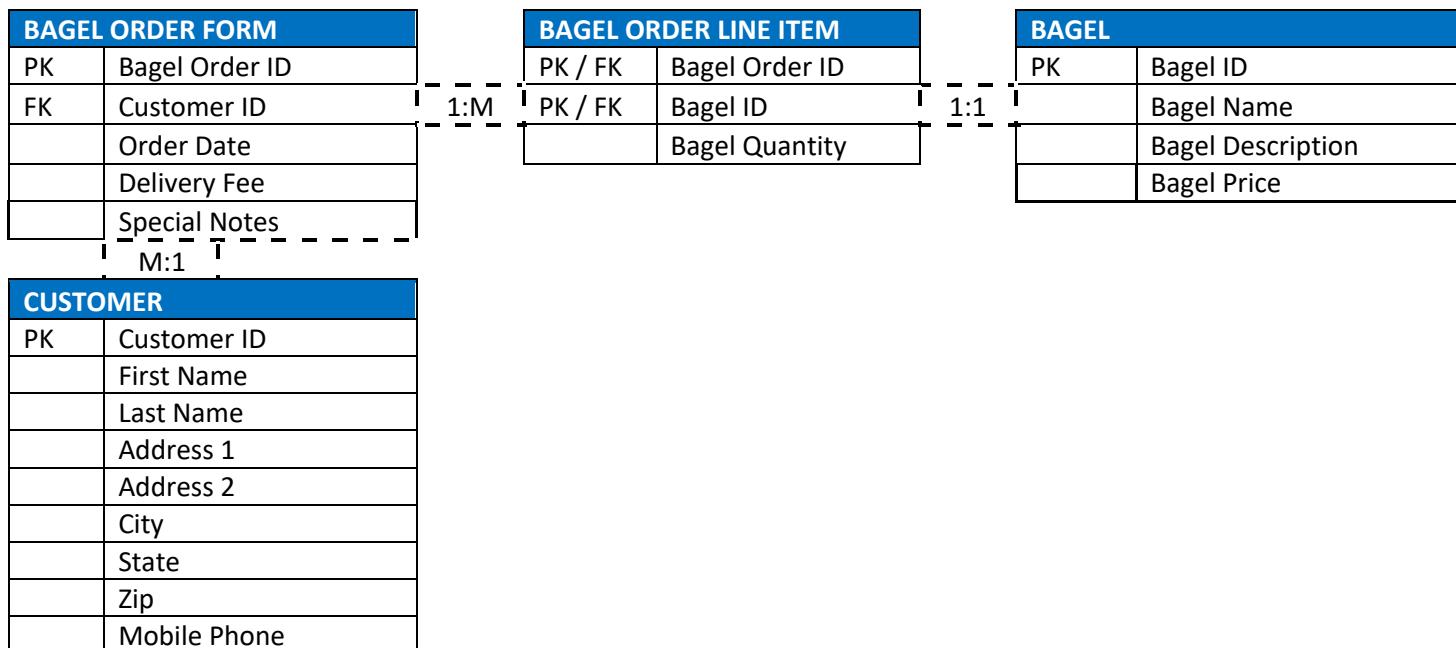
**A2b.** Provide *each* 3NF table with a name that reflects its contents.

**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.**

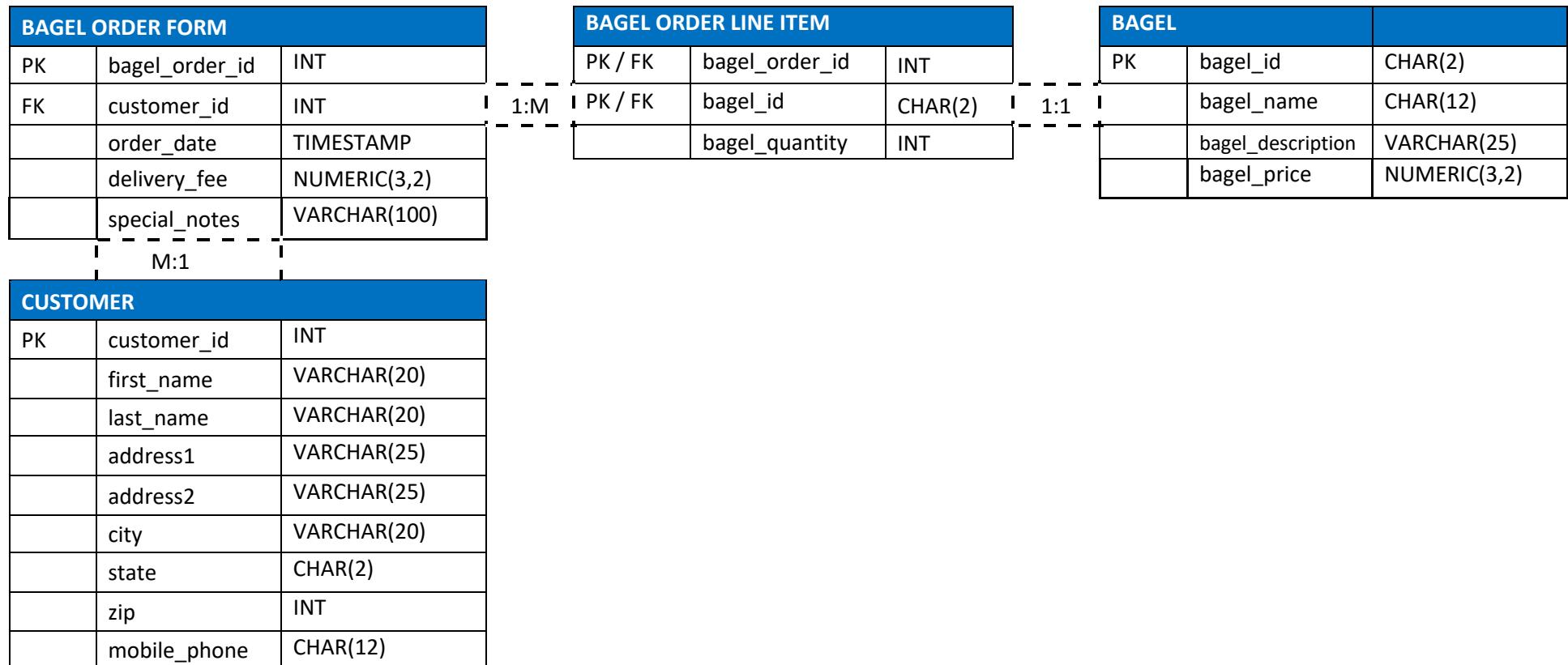
**A3. Complete the "Final Physical Database Model" section of the attached “Nora’s Bagel Bin Database Blueprints” document by doing the following:**

**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.**

The screenshot shows a MySQL Workbench interface with the following details:

- Database:** MySQL v8.0
- Actions:** Run, Save, Load Example, Collaborate
- User:** Sign in, Have any feedback?
- Schema SQL:** A large block of SQL code defining four tables: coffee\_shop, employee, supplier, and coffee. The code includes primary key and foreign key constraints.
- Execution Result:** A green box indicates "Query successfully executed in 53ms".
- Bottom Left:** Text to DDL button.
- Bottom Right:** Navigation icons.

```
1 CREATE TABLE coffee_shop
2 (
3     shop_id          int          NOT NULL,
4     shop_name        varchar(50)  NOT NULL,
5     city             varchar(50)  NOT NULL,
6     state            char(2)      NOT NULL,
7     PRIMARY KEY (shop_id)
8 );
9
10 CREATE TABLE employee
11 (
12     employee_id      int          NOT NULL,
13     first_name       varchar(30)  NOT NULL,
14     last_name        varchar(30)  NOT NULL,
15     hire_date        date         NOT NULL,
16     job_title        varchar(30)  NOT NULL,
17     shop_id          int          NOT NULL,
18     PRIMARY KEY (employee_id),
19     FOREIGN KEY (shop_id) REFERENCES coffee_shop (shop_id)
20 );
21
22 CREATE TABLE supplier
23 (
24     supplier_id      int          NOT NULL,
25     company_name     varchar(50)  NOT NULL,
26     country          varchar(30)  NOT NULL,
27     sales_contact_name varchar(60) NOT NULL,
28     email             varchar(50)  NOT NULL,
29     PRIMARY KEY (supplier_id)
30 );
31
32 CREATE TABLE coffee
33 (
34     coffee_id        int          NOT NULL,
35     shop_id          int          NOT NULL,
36     supplier_id      int          NOT NULL,
37     coffee_name      varchar(30)  NOT NULL,
38     price_per_pound   numeric(5,2) NOT NULL,
39     PRIMARY KEY (coffee_id),
40     FOREIGN KEY (shop_id) REFERENCES coffee_shop (shop_id),
41     FOREIGN KEY (supplier_id) REFERENCES supplier (supplier_id)
42 );
43
```

**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.**

Fiddle Title

50 characters remaining.

Fiddle Description

300 characters remaining.

Private Fiddle **PRO**

This setting cannot be modified after saving the fiddle.

**Upgrade to PRO**

50% OFF for Early Adopters

Show Keyboard Shortcuts

**Schema SQL**

```

19 FOREIGN KEY (shop_id) REFERENCES coffee_shop (shop_id);
20 );
21
22 CREATE TABLE supplier
23 (
24   supplier_id      int      NOT NULL,
25   company_name    varchar(50) NOT NULL,
26   country         varchar(30) NOT NULL,
27   sales_contact_name varchar(60) NOT NULL,
28   email            varchar(50) NOT NULL,
29   PRIMARY KEY (supplier_id)
30 );
31
32 CREATE TABLE coffee
33 (
34   coffee_id        int      NOT NULL,
35   shop_id          int      NOT NULL,
36   supplier_id     int      NOT NULL,
37   coffee_name     varchar(30) NOT NULL,
38   price_per_pound numeric(5,2) NOT NULL,
39   PRIMARY KEY (coffee_id),
40   FOREIGN KEY (shop_id) REFERENCES coffee_shop (shop_id),
41   FOREIGN KEY (supplier_id) REFERENCES supplier (supplier_id)
42 );
43
44 INSERT INTO coffee_shop
45 VALUES (123, 'Elevate Coffee Company', 'Phoenix', 'AZ');
46

```

**Text to DDL**

**Query SQL**

```

1 SELECT * 
2 FROM coffee_shop;
3
4 SELECT*
5 FROM employee;
6
7 SELECT*
8 FROM supplier;
9
10 SELECT*
11 FROM coffee;

```



Flatfile is the world's most advanced CSV data importer. Ready-to-go SDKs in Angular, React, Vue, and more.  
ads via Carbon

**Results**

Query #1 **Execution time: 0ms**

shop_id	shop_name	city	state
123	Elevate Coffee Company	Phoenix	AZ
124	Lux Coffee	Dallas	TX
125	Blue Tiger Coffee	Los Angeles	CA

Query #2 **Execution time: 1ms**

employee_id	first_name	last_name	hire_date	job_title	shop_id
11	Amos	Delgado	2015-12-12	Manager	125
51	Larry	Nolan	2012-03-16	Manager	123
89	Sarah	Ellis	2017-05-05	Manager	124

**Results**

Query #3 **Execution time: 0ms**

supplier_id	company_name	country	sales_contact_name	email
34	Code Brew	USA	Michelle Reese	michellereese@gmail.com
89	Southwest Refreshments	USA	Richard Weller	richardweller@gmail.com
97	The Human Bean	USA	Jayde Midas	jaydemidas@gmail.com

Query #4 **Execution time: 11ms**

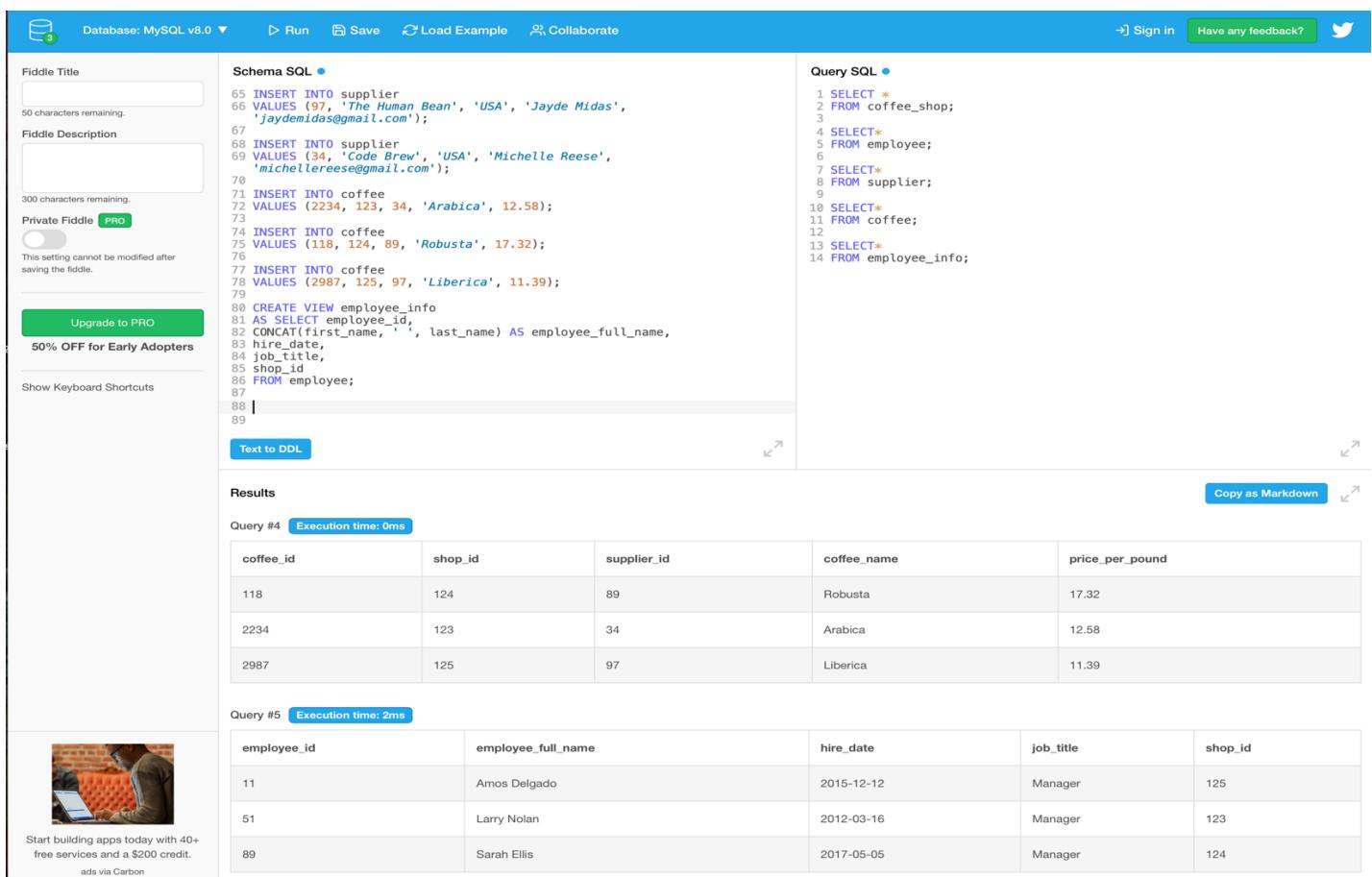
coffee_id	shop_id	supplier_id	coffee_name	price_per_pound
118	124	89	Robusta	17.32
2234	123	34	Arabica	12.58
2987	125	97	Liberica	11.39

### 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.



The screenshot shows the MySQL Fiddle interface. On the left, there are fields for 'Fiddle Title' and 'Fiddle Description'. Below these are buttons for 'Private Fiddle' (PRO), 'Upgrade to PRO', and a '50% OFF for Early Adopters' offer. A 'Show Keyboard Shortcuts' link is also present. On the right, the 'Schema SQL' and 'Query SQL' panes are visible. The Schema SQL pane contains DDL statements for creating tables and inserting data into 'supplier', 'coffee', and 'employee'. The Query SQL pane contains the SQL code for creating the 'employee\_info' view. Below these panes, the 'Results' section displays the output of two queries. Query #4 shows data from the 'coffee' table, and Query #5 shows data from the newly created 'employee\_info' view. The results are presented in tabular formats with columns like 'coffee\_id', 'shop\_id', 'supplier\_id', 'coffee\_name', 'price\_per\_pound', 'employee\_id', 'employee\_full\_name', 'hire\_date', 'job\_title', and 'shop\_id'. At the bottom left, there is a small image of a person working on a laptop and some promotional text about building apps with MySQL.

```
Schema SQL
65 INSERT INTO supplier
66 VALUES (97, 'The Human Bean', 'USA', 'Jayde Midas',
67 , 'jaydemidas@gmail.com');
68 INSERT INTO supplier
69 VALUES (34, 'Code Brew', 'USA', 'Michelle Reese',
70 , 'michellereese@gmail.com');
71 INSERT INTO coffee
72 VALUES (2234, 123, 34, 'Arabica', 12.58);
73 INSERT INTO coffee
74 VALUES (118, 124, 89, 'Robusta', 17.32);
75 INSERT INTO coffee
76 VALUES (2987, 125, 97, 'Liberica', 11.39);
77 CREATE VIEW employee_info
78 AS SELECT employee_id,
79 CONCAT(first_name, ' ', last_name) AS employee_full_name,
80 hire_date,
81 job_title,
82 shop_id
83 FROM employee;
84
85
86
87
88
89

Query SQL
1 SELECT *
2 FROM coffee_shop;
3
4 SELECT*
5 FROM employee;
6
7 SELECT*
8 FROM supplier;
9
10 SELECT*
11 FROM coffee;
12
13 SELECT*
14 FROM employee_info;

Results
Query #4 Execution time: 0ms
coffee_id shop_id supplier_id coffee_name price_per_pound
118 124 89 Robusta 17.32
2234 123 34 Arabica 12.58
2987 125 97 Liberica 11.39

Query #5 Execution time: 2ms
employee_id employee_full_name hire_date job_title shop_id
11 Amos Delgado 2015-12-12 Manager 125
51 Larry Nolan 2012-03-16 Manager 123
89 Sarah Ellis 2017-05-05 Manager 124
```

**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.**

The screenshot shows the MySQL Fiddle interface. On the left, there are input fields for 'Fiddle Title' and 'Fiddle Description'. A 'Private Fiddle' button is set to 'PRO'. A 'Upgrade to PRO' button and a '50% OFF for Early Adopters' offer are also visible. Below these are 'Show Keyboard Shortcuts' and a SonarLint advertisement. The central area contains two tabs: 'Schema SQL' and 'Query SQL'. The 'Schema SQL' tab displays several INSERT statements into the 'supplier' and 'coffee' tables. The 'Query SQL' tab contains a single digit '1'. At the bottom, there's a 'Results' section stating 'There are no results to be displayed.' and a 'Copy as Markdown' button.

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

Database: MySQL v8.0 ▾ Run Save Load Example Collaborate → Sign in Have any feedback? Twitter

Fiddle Title

50 characters remaining.

Fiddle Description

300 characters remaining.

Private Fiddle PRO

This setting cannot be modified after saving the fiddle.

Upgrade to PRO

50% OFF for Early Adopters

Show Keyboard Shortcuts

sonarlint | Code Quality & Security IDE Extension

Catch Bugs and Vulnerabilities as you code, directly in the IDE. Download SonarLint for free today!

ads via Carbon

Schema SQL

```
64 INSERT INTO supplier  
65 VALUES (97, 'The Human Bean', 'USA', 'Jayde Midas', 'jaydemidas@gmail.com');  
67  
68 INSERT INTO supplier  
69 VALUES (34, 'Code Brew', 'USA', 'Michelle Reese', 'michellereese@gmail.com');  
70  
71 INSERT INTO coffee  
72 VALUES (2234, 123, 34, 'Arabica', 12.58);  
73  
74 INSERT INTO coffee  
75 VALUES (118, 124, 89, 'Robusta', 17.32);  
76  
77 INSERT INTO coffee  
78 VALUES (2987, 125, 97, 'Liberica', 11.39);  
79  
80 CREATE VIEW employee_info  
81 AS SELECT employee_id,  
82 CONCAT(first_name, ' ', last_name) AS employee_full_name,  
83 hire_date,  
84 job_title,  
85 shop_id  
86 FROM employee;  
87  
88 CREATE INDEX coffee_index  
89 ON coffee (coffee_name);  
90
```

Query SQL

```
1
```

Results

There are no results to be displayed.

Copy as Markdown

- 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.**

The screenshot shows the DB Fiddle interface with the following details:

- Schema SQL:**

```
1 CREATE TABLE coffee_shop
2 (
3   shop_id          int      NOT NULL,
4   shop_name        varchar(50) NOT NULL,
5   city             varchar(50) NOT NULL,
6   state            char(2)  NOT NULL,
7   PRIMARY KEY (shop_id)
8 );
9
10 CREATE TABLE employee
11 (
12   employee_id     int      NOT NULL,
13   first_name      varchar(30) NOT NULL,
14   last_name       varchar(30) NOT NULL,
15   hire_date       date    NOT NULL,
16   job_title       varchar(30) NOT NULL,
17   shop_id         int      NOT NULL,
18   PRIMARY KEY (employee_id),
19   FOREIGN KEY (shop_id) REFERENCES coffee_shop (shop_id)
20 );
21
22 CREATE TABLE supplier
23 (
24   supplier_id     int      NOT NULL,
25   company_name    varchar(50) NOT NULL,
26   country          varchar(30) NOT NULL,
27   sales_contact_name varchar(60) NOT NULL,
28   email            varchar(50) NOT NULL,
```
- Query SQL:**

```
1 SELECT*
2 FROM employee
3 WHERE hire_date BETWEEN '2014-01-01' AND '2017-12-15';
```
- Results:**

employee_id	first_name	last_name	hire_date	job_title	shop_id
11	Amos	Delgado	2015-12-12	Manager	125
89	Sarah	Ellis	2017-05-05	Manager	124

**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.**

The screenshot shows the MySQL Workbench interface. On the left, the Schema pane displays the database schema with three tables: coffee\_shop, supplier, and coffee. The coffee\_shop table has columns shop\_id (PK), shop\_name, company\_name, country, sales\_contact\_name, email, and phone. The supplier table has columns supplier\_id (PK), supplier\_name, coffee\_name, and price\_per\_pound. The coffee table has columns coffee\_id (PK), shop\_id (FK), and supplier\_id (FK). Data is inserted into the coffee\_shop table with values 123, 124, and 125. The supplier table has entries for 'Elevate Coffee Company' and 'The Human Bean'. The coffee table has entries for 'Code Brew', 'Arabica', 'Liberica', and 'Southwest Refreshments'. The right pane, titled 'Query SQL', contains the provided SQL query. Below it, the 'Results' pane shows the execution time and the query results, which are identical to the ones shown in the Schema pane.

```
Schema SQL •  
18 PRIMARY KEY (employee_id),  
19 FOREIGN KEY (shop_id) REFERENCES coffee_shop (shop_id)  
);  
20  
21  
22 CREATE TABLE supplier  
23 {  
24     supplier_id      int      NOT NULL,  
25     company_name    varchar(50)  NOT NULL,  
26     country        varchar(30)  NOT NULL,  
27     sales_contact_name varchar(60) NOT NULL,  
28     email           varchar(50)  NOT NULL,  
29     PRIMARY KEY (supplier_id)  
30 };  
31  
32 CREATE TABLE coffee  
33 {  
34     coffee_id       int      NOT NULL,  
35     shop_id         int      NOT NULL,  
36     supplier_id    int      NOT NULL,  
37     coffee_name    varchar(30) NOT NULL,  
38     price_per_pound numeric(5,2) NOT NULL,  
39     PRIMARY KEY (coffee_id),  
40     FOREIGN KEY (shop_id) REFERENCES coffee_shop (shop_id),  
41     FOREIGN KEY (supplier_id) REFERENCES supplier (supplier_id)  
42 };  
43  
44 INSERT INTO coffee_shop  
45 VALUES (123, 'Elevate Coffee Company', 'Phoenix', 'AZ');  
46  
47 INSERT INTO coffee_shop  
48 VALUES (124, 'Lux Coffee', 'Dallas', 'TX');  
49  
50 INSERT INTO coffee_shop  
51 VALUES (125, 'Blue Tiger Coffee', 'Los Angeles', 'CA');
```

```
Query SQL •  
1 SELECT shop_name, company_name, coffee_name  
2 FROM coffee_shop  
3 CROSS JOIN supplier  
4 CROSS JOIN coffee;
```

shop_name	company_name	coffee_name
Elevate Coffee Company	Code Brew	Arabica
Lux Coffee	Code Brew	Arabica
Blue Tiger Coffee	Code Brew	Arabica
Elevate Coffee Company	Southwest Refreshments	Arabica
Lux Coffee	Southwest Refreshments	Arabica
Blue Tiger Coffee	Southwest Refreshments	Arabica
Elevate Coffee Company	The Human Bean	Arabica
Lux Coffee	The Human Bean	Arabica
Blue Tiger Coffee	The Human Bean	Arabica
Elevate Coffee Company	Code Brew	Liberica
Lux Coffee	Code Brew	Liberica