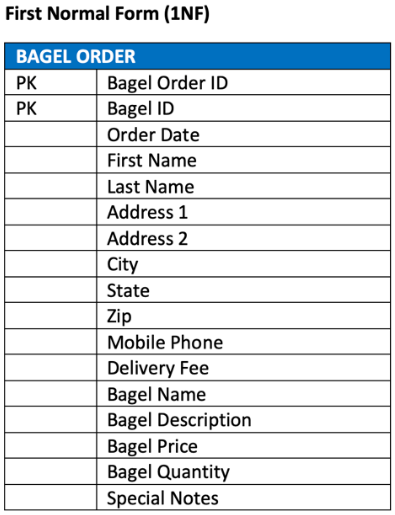
Max Sealey

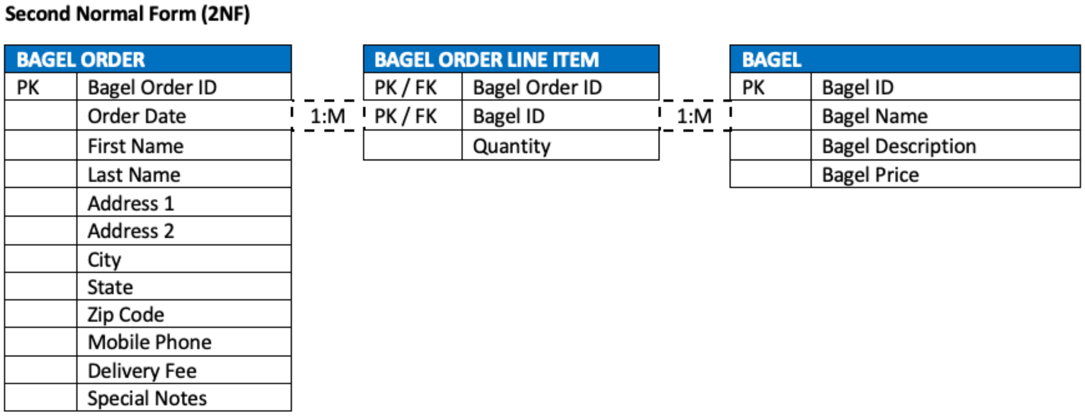
Part A: Nora’s Bagel Bin



First Normal Form (1NF)

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

All the non-key columns in the normalized 2NF diagram (below) are now functionally dependent on a whole primary key, not a composite primary key, as in the 1NF diagram (above). The 1NF ERD was partitioned into three tables. The first (Bagel Order) contains information specific to the order, including the Order ID (simple primary key) and date, the customer’s information (contact info and delivery address), and information necessary to order fulfillment (the fee, special notes). The third table (Bagel) contains information about each bagel item, so it doesn’t change based on the other two tables. Splitting the two tables up reduces a lot of redundancy from the 1NF ERD. The second table (Bagel Order Line Item) refers to both tables with foreign keys, using the Bagel Order ID and the Bagel ID, and specifies the quantity of each bagel item in the order.



Second Normal Form (2NF)

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

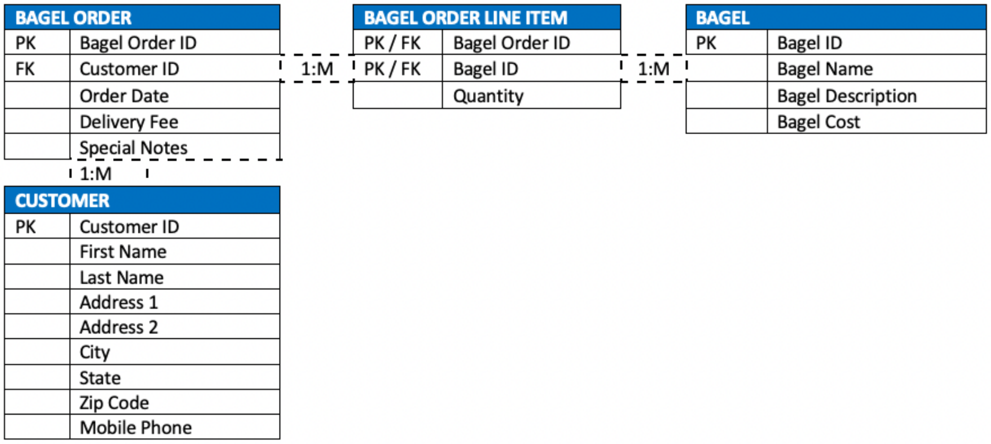
BAGEL ORDER LINE ITEM	
PK / FK	Bagel Order ID
PK / FK	Bagel ID
	Quantity

BAGEL	
PK	Bagel ID
	Bagel Name
	Bagel Description
	Bagel Price

I determined that the relationship between the Bagel Order table and the Bagel Order Line Item table was one-to-many. Each order can contain many lines, each with a different type of bagel, but each line can only relate to one Bagel Order ID.

The relationship between the Bagel Order Line Item table and the Bagel table was also determined to be one-to-many, since each line can only specify one type of bagel, but many bagels can be included in each line.

Third Normal Form (3NF)



BAGEL ORDER	
PK	Bagel Order ID
FK	Customer ID
	Order Date
	Delivery Fee
	Special Notes

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

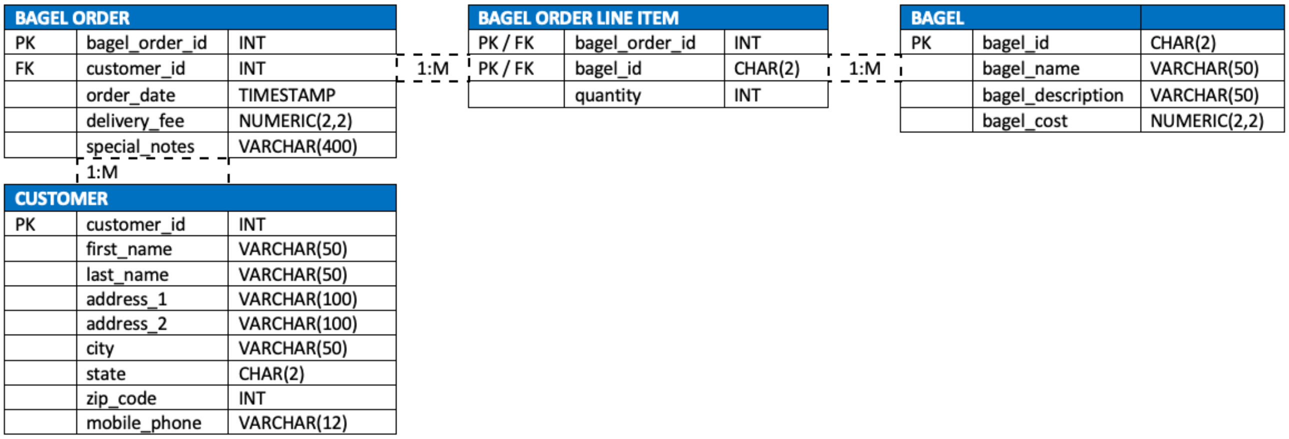
BAGEL	
PK	Bagel ID
	Bagel Name
	Bagel Description
	Bagel Cost

CUSTOMER	
PK	Customer ID
	First Name
	Last Name
	Address 1
	Address 2
	City
	State
	Zip Code
	Mobile Phone

I assigned each attribute from the 2NF ERD into a new 3NF ERD (above), provided each table with a suitable name, and added a ‘Customer ID’ field to the Bagel Order table as a foreign key, connecting it with the primary key of the newly created Customer table. By separating the customer information from the bagel order information, I eliminated the remaining transitive functional dependency. Now a customer could place multiple bagel orders without redundancy. All the attributes in the Customer table are items that are specific to a single customer instance, as identified by the Customer ID.

The cardinality between the Bagel Order, Bagel Order Line Item, and Bagel tables remains the same as in the 2NF ERD. I found that the relationship between the Bagel Order table and the Customer table is one-to-many, since while a customer instance could place several orders, each order can only have one customer placing it.

Final Physical Database Model



BAGEL ORDER		
PK	bagel_order_id	INT
FK	customer_id	INT
	order_date	TIMESTAMP
	delivery_fee	NUMERIC(2,2)
	special_notes	VARCHAR(400)

BAGEL ORDER LINE ITEM		
PK / FK	bagel_order_id	INT
PK / FK	bagel_id	CHAR(2)
	quantity	INT

BAGEL		
PK	bagel_id	CHAR(2)
	bagel_name	VARCHAR(50)
	bagel_description	VARCHAR(50)
	bagel_cost	NUMERIC(2,2)

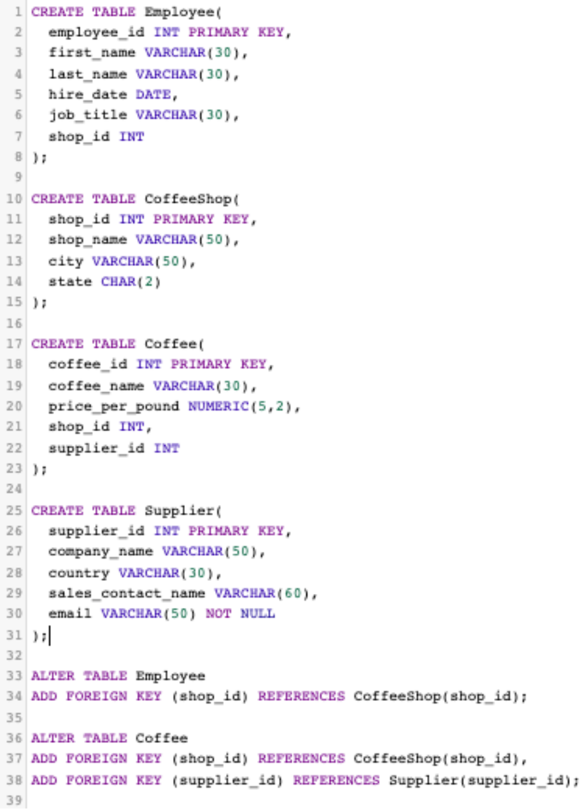
CUSTOMER		
PK	customer_id	INT
	first_name	VARCHAR(50)
	last_name	VARCHAR(50)
	address_1	VARCHAR(100)
	address_2	VARCHAR(100)
	city	VARCHAR(50)
	state	CHAR(2)
	zip_code	INT
	mobile_phone	VARCHAR(12)

The Final Physical Database Model (above) has all of the information from the 3NF diagram, but has renamed the attributes so that they are usable database characters and specified a data type for each one.

Part B: Jaunty Coffee Co.

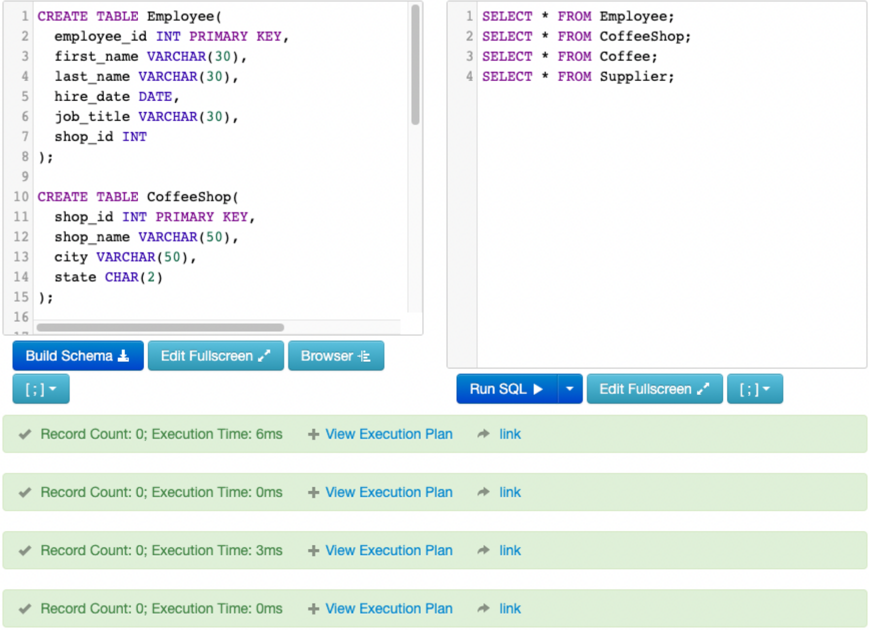
1. **Develop SQL code to create *each* table as specified in the attached "Jaunty Coffee Co. ERD"**

The SQL code I wrote to create the tables specified in the given ERD:



```
CREATE TABLE Employee(
  employee_id INT PRIMARY KEY,
  first_name VARCHAR(30),
  last_name VARCHAR(30),
  hire_date DATE,
  job_title VARCHAR(30),
  shop_id INT
);

CREATE TABLE CoffeeShop(
  shop_id INT PRIMARY KEY,
  shop_name VARCHAR(50),
  city VARCHAR(50),
  state CHAR(2)
);

CREATE TABLE Coffee(
  coffee_id INT PRIMARY KEY,
  coffee_name VARCHAR(30),
  price_per_pound NUMERIC(5,2),
  shop_id INT,
  supplier_id INT
);

CREATE TABLE Supplier(
  supplier_id INT PRIMARY KEY,
  company_name VARCHAR(50),
  country VARCHAR(30),
  sales_contact_name VARCHAR(60),
  email VARCHAR(50) NOT NULL
);

ALTER TABLE Employee
ADD FOREIGN KEY (shop_id) REFERENCES CoffeeShop(shop_id);

ALTER TABLE Coffee
ADD FOREIGN KEY (shop_id) REFERENCES CoffeeShop(shop_id),
ADD FOREIGN KEY (supplier_id) REFERENCES Supplier(supplier_id);
```

The SQL commands I used to test the code and the database server's response:



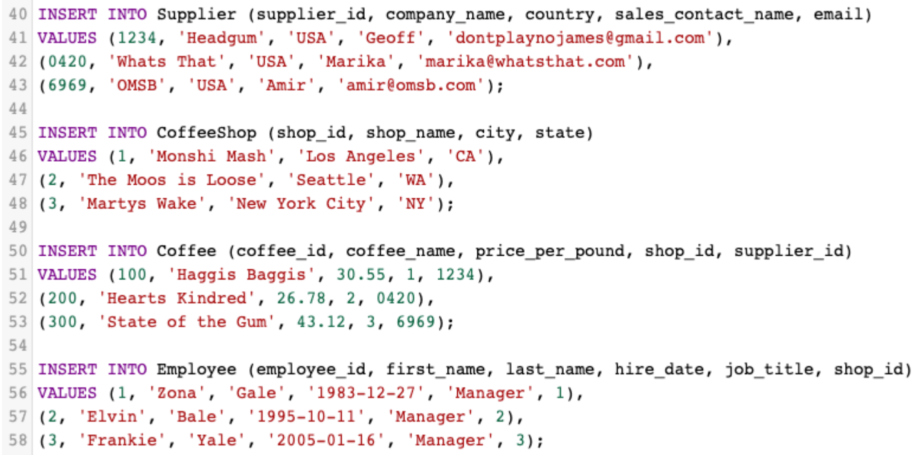
```
CREATE TABLE Employee(
  employee_id INT PRIMARY KEY,
  first_name VARCHAR(30),
  last_name VARCHAR(30),
  hire_date DATE,
  job_title VARCHAR(30),
  shop_id INT
);

CREATE TABLE CoffeeShop(
  shop_id INT PRIMARY KEY,
  shop_name VARCHAR(50),
  city VARCHAR(50),
  state CHAR(2)
);
```

```
SELECT * FROM Employee;
SELECT * FROM CoffeeShop;
SELECT * FROM Coffee;
SELECT * FROM Supplier;
```

```
✔ Record Count: 0; Execution Time: 6ms    + View Execution Plan    link

✔ Record Count: 0; Execution Time: 0ms    + View Execution Plan    link

✔ Record Count: 0; Execution Time: 3ms    + View Execution Plan    link

✔ Record Count: 0; Execution Time: 0ms    + View Execution Plan    link
```

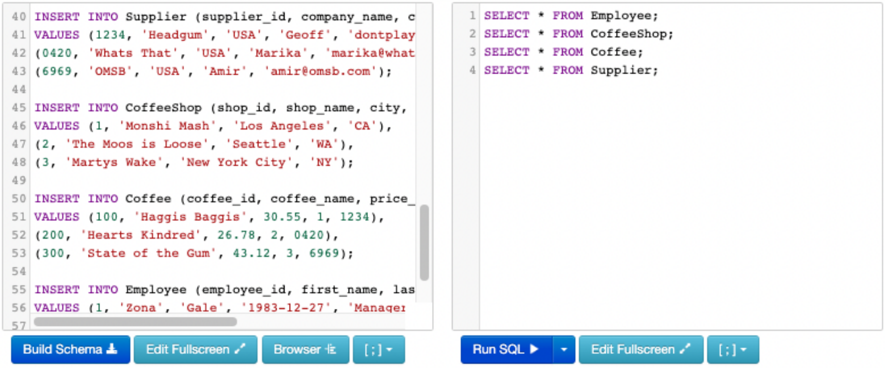
2. Develop SQL code to populate *each* table in the database design document

The SQL code I wrote to populate each table with three rows of data:



```sql
INSERT INTO Supplier (supplier_id, company_name, country, sales_contact_name, email)
VALUES (1234, 'Headgum', 'USA', 'Geoff', 'dontplaynojames@gmail.com'),
(0420, 'Whats That', 'USA', 'Marika', 'marika@whatsthat.com'),
(6969, 'OMSB', 'USA', 'Amir', 'amir@omsb.com');

INSERT INTO CoffeeShop (shop_id, shop_name, city, state)
VALUES (1, 'Monshi Mash', 'Los Angeles', 'CA'),
(2, 'The Moos is Loose', 'Seattle', 'WA'),
(3, 'Martys Wake', 'New York City', 'NY');

INSERT INTO Coffee (coffee_id, coffee_name, price_per_pound, shop_id, supplier_id)
VALUES (100, 'Haggis Baggis', 30.55, 1, 1234),
(200, 'Hearts Kindred', 26.78, 2, 0420),
(300, 'State of the Gum', 43.12, 3, 6969);

INSERT INTO Employee (employee_id, first_name, last_name, hire_date, job_title, shop_id)
VALUES (1, 'Zona', 'Gale', '1983-12-27', 'Manager', 1),
(2, 'Elvin', 'Bale', '1995-10-11', 'Manager', 2),
(3, 'Frankie', 'Yale', '2005-01-16', 'Manager', 3);
```

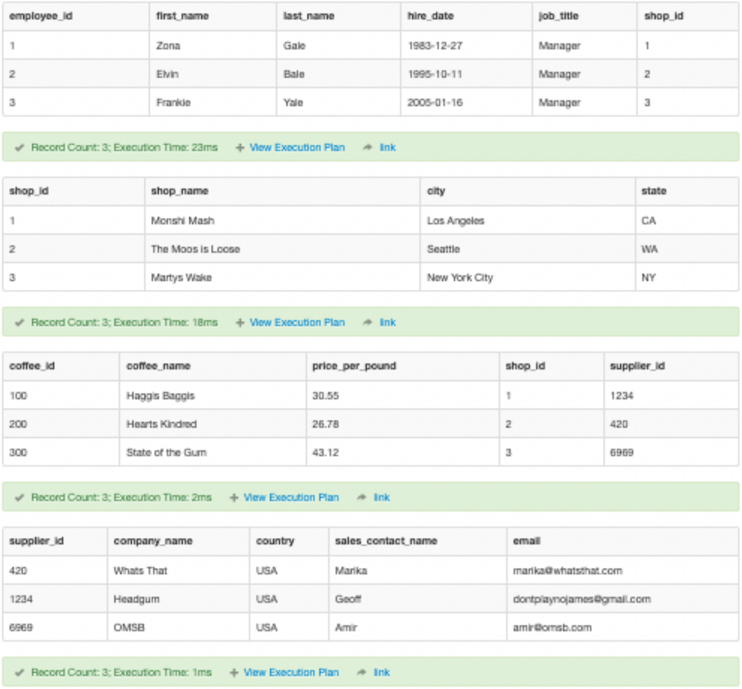
The SQL commands I used to test the code:



```sql
INSERT INTO Supplier (supplier_id, company_name, c
VALUES (1234, 'Headgum', 'USA', 'Geoff', 'dontplay
(0420, 'Whats That', 'USA', 'Marika', 'marika@what
(6969, 'OMSB', 'USA', 'Amir', 'amir@omsb.com');

INSERT INTO CoffeeShop (shop_id, shop_name, city,
VALUES (1, 'Monshi Mash', 'Los Angeles', 'CA'),
(2, 'The Moos is Loose', 'Seattle', 'WA'),
(3, 'Martys Wake', 'New York City', 'NY');

INSERT INTO Coffee (coffee_id, coffee_name, price_
VALUES (100, 'Haggis Baggis', 30.55, 1, 1234),
(200, 'Hearts Kindred', 26.78, 2, 0420),
(300, 'State of the Gum', 43.12, 3, 6969);

INSERT INTO Employee (employee_id, first_name, las
VALUES (1, 'Zona', 'Gale', '1983-12-27', 'Manager
```

```sql
SELECT * FROM Employee;
SELECT * FROM CoffeeShop;
SELECT * FROM Coffee;
SELECT * FROM Supplier;
```

The database server's response:



employee_id	first_name	last_name	hire_date	job_title	shop_id
1	Zona	Gale	1983-12-27	Manager	1
2	Elvin	Bale	1995-10-11	Manager	2
3	Frankie	Yale	2005-01-16	Manager	3

✓ Record Count: 3; Execution Time: 23ms + View Execution Plan link

shop_id	shop_name	city	state
1	Monshi Mash	Los Angeles	CA
2	The Moos is Loose	Seattle	WA
3	Martys Wake	New York City	NY

✓ Record Count: 3; Execution Time: 18ms + View Execution Plan link

coffee_id	coffee_name	price_per_pound	shop_id	supplier_id
100	Haggis Baggis	30.55	1	1234
200	Hearts Kindred	26.78	2	420
300	State of the Gum	43.12	3	6969

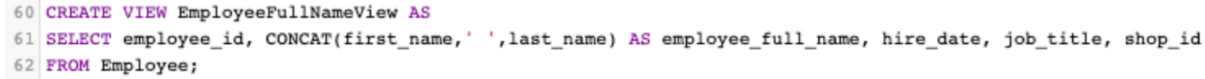
✓ Record Count: 3; Execution Time: 2ms + View Execution Plan link

supplier_id	company_name	country	sales_contact_name	email
420	Whats That	USA	Marika	marika@whatsthat.com
1234	Headgum	USA	Geoff	dontplaynojames@gmail.com
6969	OMSB	USA	Amir	amir@omsb.com

✓ Record Count: 3; Execution Time: 1ms + View Execution Plan link

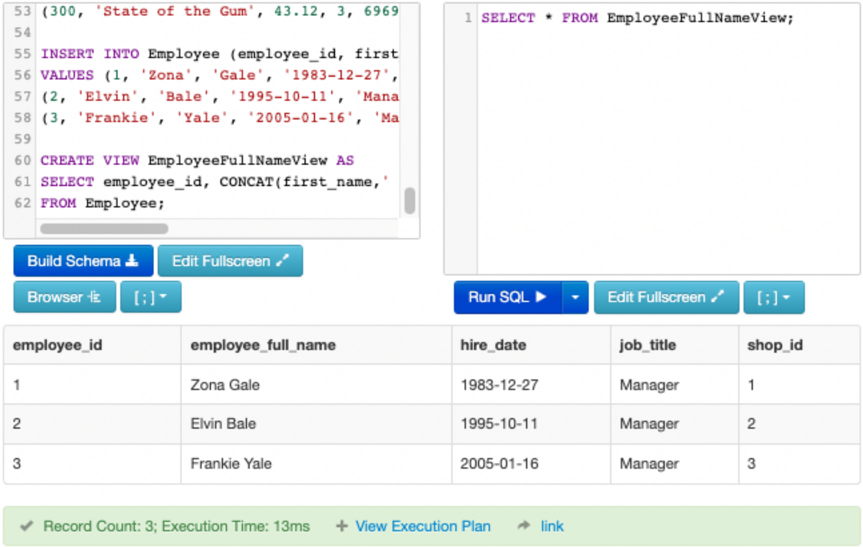
3. Develop SQL code to create a view

The SQL code I wrote to create a view



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

The SQL commands I used to test the code and the database server's response:



```
(300, 'State of the Gum', 43.12, 3, 6969

INSERT INTO Employee (employee_id, first
VALUES (1, 'Zona', 'Gale', '1983-12-27',
(2, 'Elvin', 'Bale', '1995-10-11', 'Mana
(3, 'Frankie', 'Yale', '2005-01-16', 'Ma

CREATE VIEW EmployeeFullNameView AS
SELECT employee_id, CONCAT(first_name,'
FROM Employee;
```

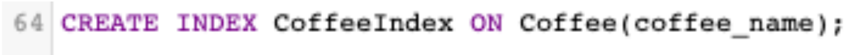
```
SELECT * FROM EmployeeFullNameView;
```

employee_id	employee_full_name	hire_date	job_title	shop_id
1	Zona Gale	1983-12-27	Manager	1
2	Elvin Bale	1995-10-11	Manager	2
3	Frankie Yale	2005-01-16	Manager	3

✓ Record Count: 3; Execution Time: 13ms

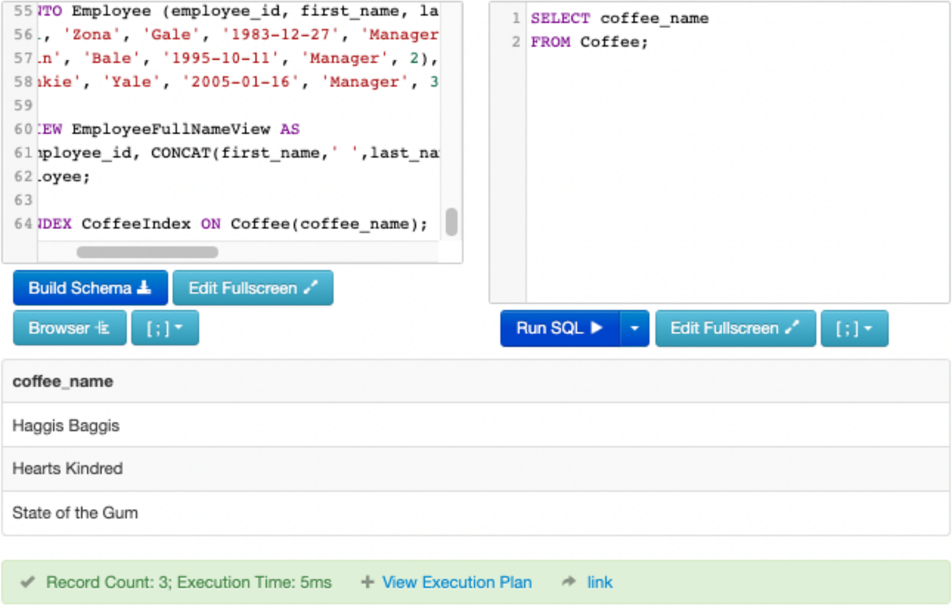
4. Develop SQL code to create an index on the coffee_name field

The SQL code I wrote to create an index on the coffee_name field:



```
CREATE INDEX CoffeeIndex ON Coffee(coffee_name);
```

The SQL commands I used to test the code and the database server's response:



```
TO Employee (employee_id, first_name, la
., 'Zona', 'Gale', '1983-12-27', 'Manager
.n', 'Bale', '1995-10-11', 'Manager', 2),
kie', 'Yale', '2005-01-16', 'Manager', 3

EW EmployeeFullNameView AS
ployee_id, CONCAT(first_name,' ',last_na
.oyee;

DEX CoffeeIndex ON Coffee(coffee_name);
```

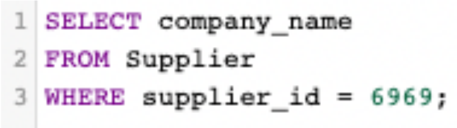
```
SELECT coffee_name
FROM Coffee;
```

coffee_name
Haggis Baggis
Hearts Kindred
State of the Gum

✓ Record Count: 3; Execution Time: 5ms

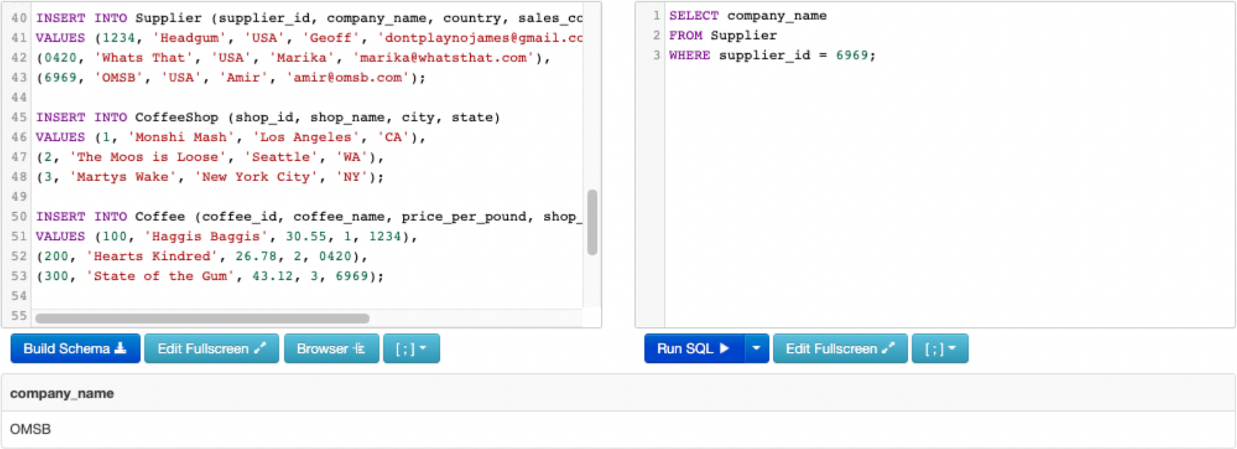
5. Develop SQL code to create an SFW (SELECT–FROM–WHERE) query for *any* of your tables or views

The SQL code I wrote to create a SFW query on the Supplier table:



```sql
SELECT company_name
FROM Supplier
WHERE supplier_id = 6969;
```

The SQL commands I used to test the code and the database server's response:



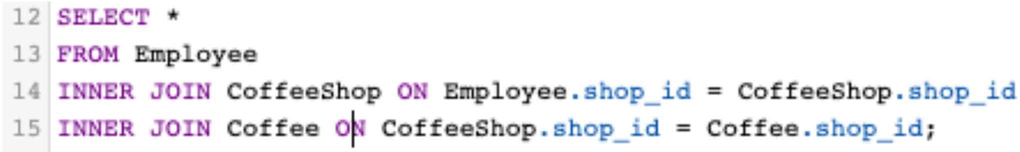
```sql
INSERT INTO Supplier (supplier_id, company_name, country, sales_cc
VALUES (1234, 'Headgum', 'USA', 'Geoff', 'dontplaynojames@gmail.cc
(0420, 'Whats That', 'USA', 'Marika', 'marika@whatsthat.com'),
(6969, 'OMSB', 'USA', 'Amir', 'amir@omsb.com');

INSERT INTO CoffeeShop (shop_id, shop_name, city, state)
VALUES (1, 'Monshi Mash', 'Los Angeles', 'CA'),
(2, 'The Moos is Loose', 'Seattle', 'WA'),
(3, 'Martys Wake', 'New York City', 'NY');

INSERT INTO Coffee (coffee_id, coffee_name, price_per_pound, shop_
VALUES (100, 'Haggis Baggis', 30.55, 1, 1234),
(200, 'Hearts Kindred', 26.78, 2, 0420),
(300, 'State of the Gum', 43.12, 3, 6969);
```

```sql
SELECT company_name
FROM Supplier
WHERE supplier_id = 6969;
```

company_name
OMSB

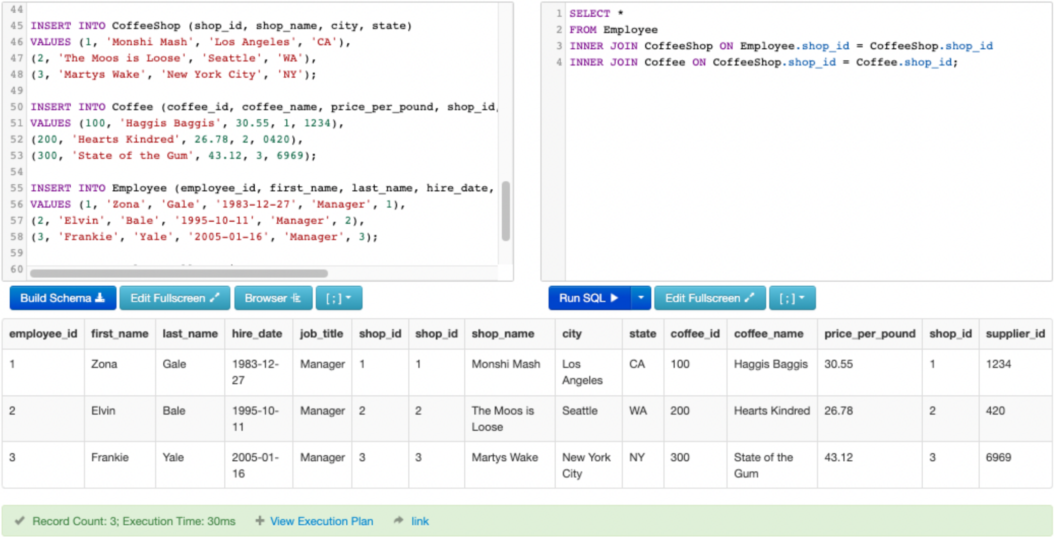
6. Develop SQL code to join three different tables and include attributes from all three

The SQL code I wrote to join the Employee, CoffeeShop, and Coffee tables:



```sql
SELECT *
FROM Employee
INNER JOIN CoffeeShop ON Employee.shop_id = CoffeeShop.shop_id
INNER JOIN Coffee ON CoffeeShop.shop_id = Coffee.shop_id;
```

The SQL commands I used to test the code and the database server's response:



```sql

INSERT INTO CoffeeShop (shop_id, shop_name, city, state)
VALUES (1, 'Monshi Mash', 'Los Angeles', 'CA'),
(2, 'The Moos is Loose', 'Seattle', 'WA'),
(3, 'Martys Wake', 'New York City', 'NY');

INSERT INTO Coffee (coffee_id, coffee_name, price_per_pound, shop_id,
VALUES (100, 'Haggis Baggis', 30.55, 1, 1234),
(200, 'Hearts Kindred', 26.78, 2, 0420),
(300, 'State of the Gum', 43.12, 3, 6969);

INSERT INTO Employee (employee_id, first_name, last_name, hire_date,
VALUES (1, 'Zona', 'Gale', '1983-12-27', 'Manager', 1),
(2, 'Elvin', 'Bale', '1995-10-11', 'Manager', 2),
(3, 'Frankie', 'Yale', '2005-01-16', 'Manager', 3);
```

```sql
SELECT *
FROM Employee
INNER JOIN CoffeeShop ON Employee.shop_id = CoffeeShop.shop_id
INNER JOIN Coffee ON CoffeeShop.shop_id = Coffee.shop_id;
```

employee_id	first_name	last_name	hire_date	job_title	shop_id	shop_id	shop_name	city	state	coffee_id	coffee_name	price_per_pound	shop_id	supplier_id
1	Zona	Gale	1983-12-27	Manager	1	1	Monshi Mash	Los Angeles	CA	100	Haggis Baggis	30.55	1	1234
2	Elvin	Bale	1995-10-11	Manager	2	2	The Moos is Loose	Seattle	WA	200	Hearts Kindred	26.78	2	420
3	Frankie	Yale	2005-01-16	Manager	3	3	Martys Wake	New York City	NY	300	State of the Gum	43.12	3	6969

✓ Record Count: 3; Execution Time: 30ms