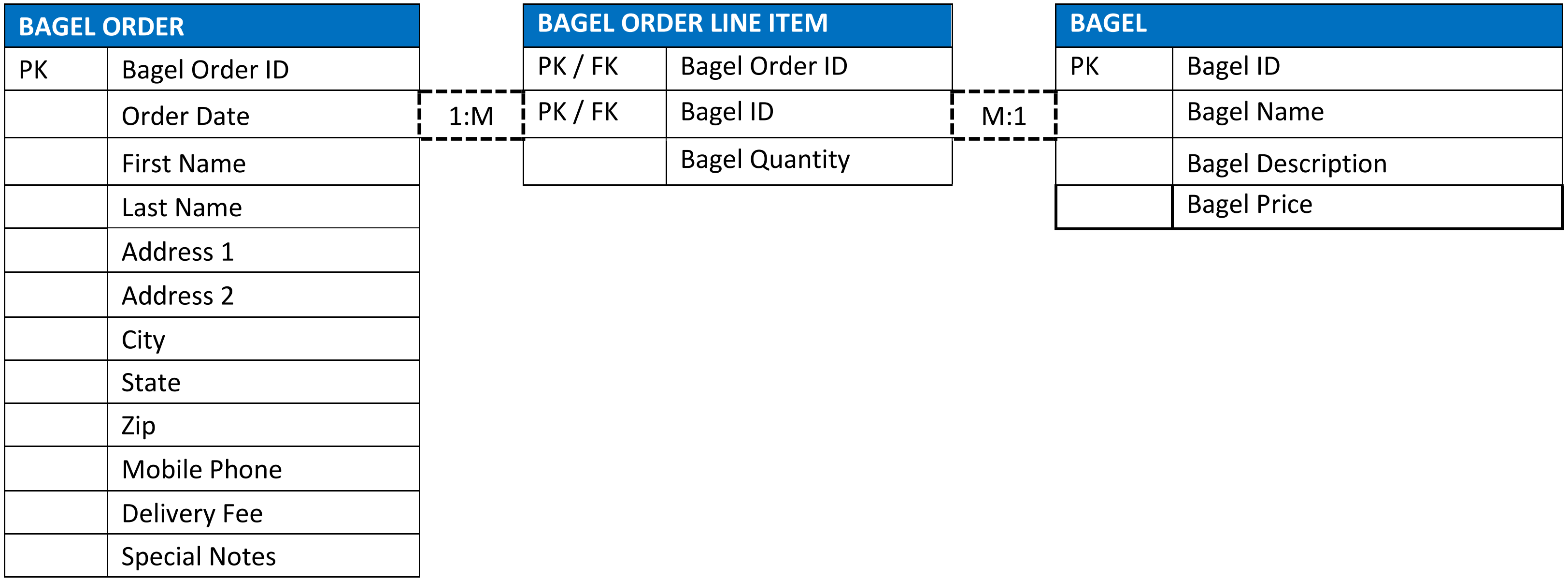
Nora’s Bagel Bin Database ER Diagram and Normalization

1st Normal Form

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

Using the Bagel Order Form Template, a 1st Normal form was created for the relation (table), grouping repeated attributes and maintaining single values in each cell.

2nd Normal Form



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

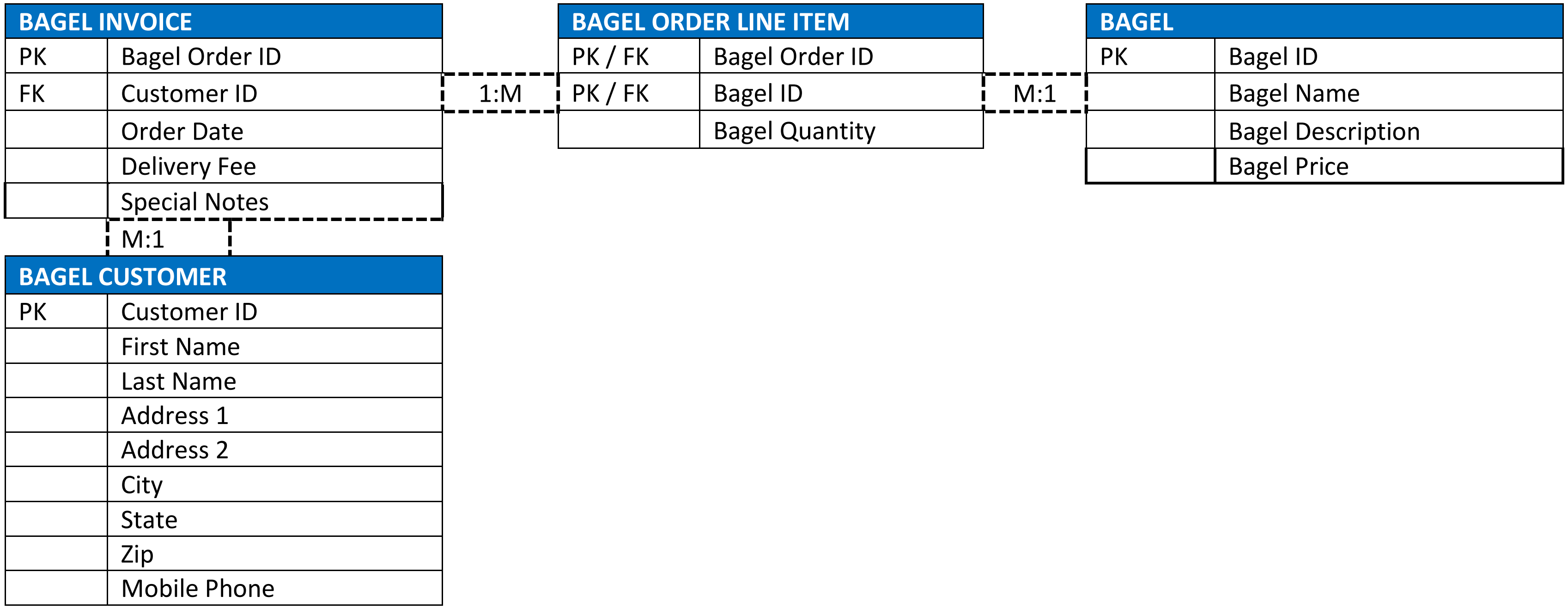
BAGEL ORDER LINE ITEM	
PK / FK	Bagel Order ID
PK / FK	Bagel ID
	Bagel Quantity

BAGEL	
PK	Bagel ID
	Bagel Name
	Bagel Description
	Bagel Price

After 1NF was completed, 2NF was achieved by making sure all non-key attributes (columns) were functionally dependent on the *entire* primary key (PK). Two new relations were created (Bagel Order Line Item and Bagel) with their respective non-key attributes to conform to 2NF standards. The Bagel relation contains all attributes that strictly describe bagels and not the order ID, so were separated from the Bagel Order relation. Bagel Quantity is the only attribute that is dependent on the entire composite primary key, so were separated in the Bagel Order Line Item relation. The rest of the attributes are dependent on the Bagel Order ID only, so were left in the original Bagel Order relation.

Cardinality between the Bagel Order entity and Bagel Order Line Item entity can be explained as one order can have many order line items, while each order line item instance can be on only one order (1 to Many). An Order Line Item can only have 1 type of bagel, while each bagel type can appear on multiple line items (Many to One).

3rd Normal Form



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

1:M

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

M:1

BAGEL	
PK	Bagel ID
	Bagel Name
	Bagel Description
	Bagel Price

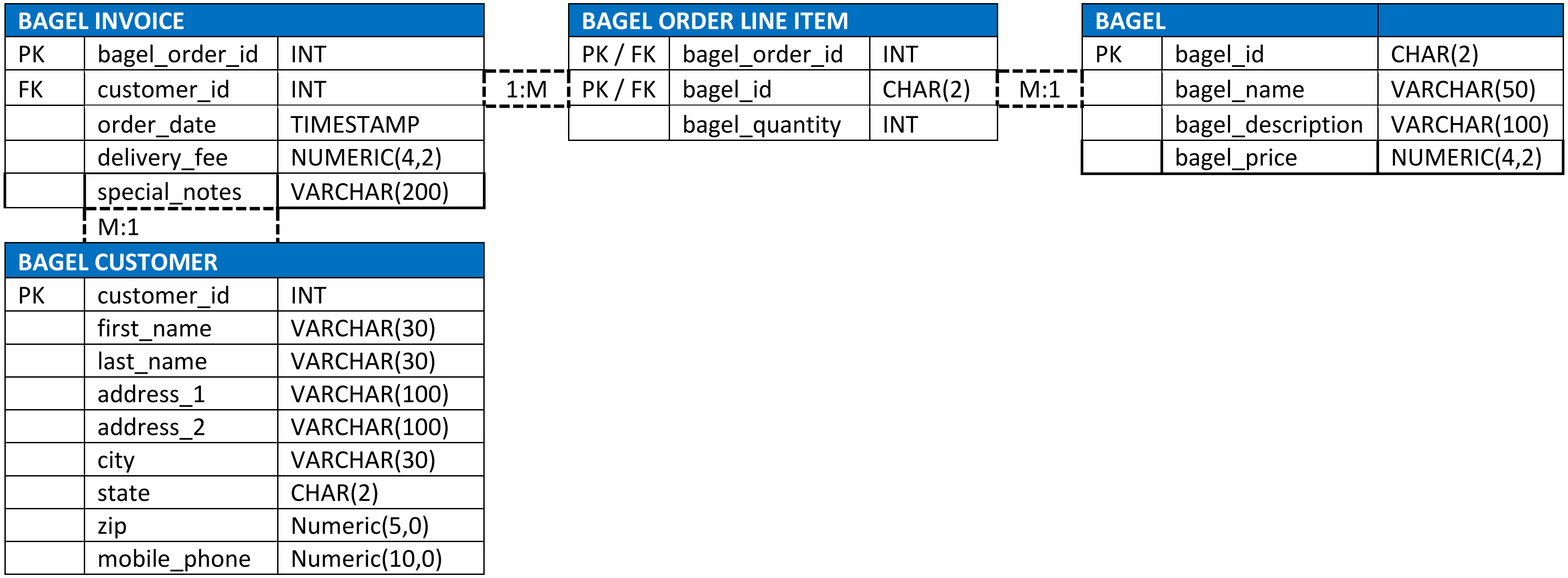
M:1

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

Once 2NF is done, 3NF was performed to eliminate transitive dependencies (non-key attributes with dependencies on other non-key attributes). A new attribute (Customer ID) was created as a foreign key in the renamed Bagel Order (Bagel Invoice) and used as a PK in Bagel Customer to tie all customer attributes to the new PK.

Cardinality remained the same for the old relations, but for the new relations a bagel invoice can only have one customer, while a customer can have many invoices (many to one).

Final Physical Database Model



BAGEL INVOICE		
PK	bagel_order_id	INT
FK	customer_id	INT
	order_date	TIMESTAMP
	delivery_fee	NUMERIC(4,2)
	special_notes	VARCHAR(200)

M:1

BAGEL CUSTOMER		
PK	customer_id	INT
	first_name	VARCHAR(30)
	last_name	VARCHAR(30)
	address_1	VARCHAR(100)
	address_2	VARCHAR(100)
	city	VARCHAR(30)
	state	CHAR(2)
	zip	Numeric(5,0)
	mobile_phone	Numeric(10,0)

1:M

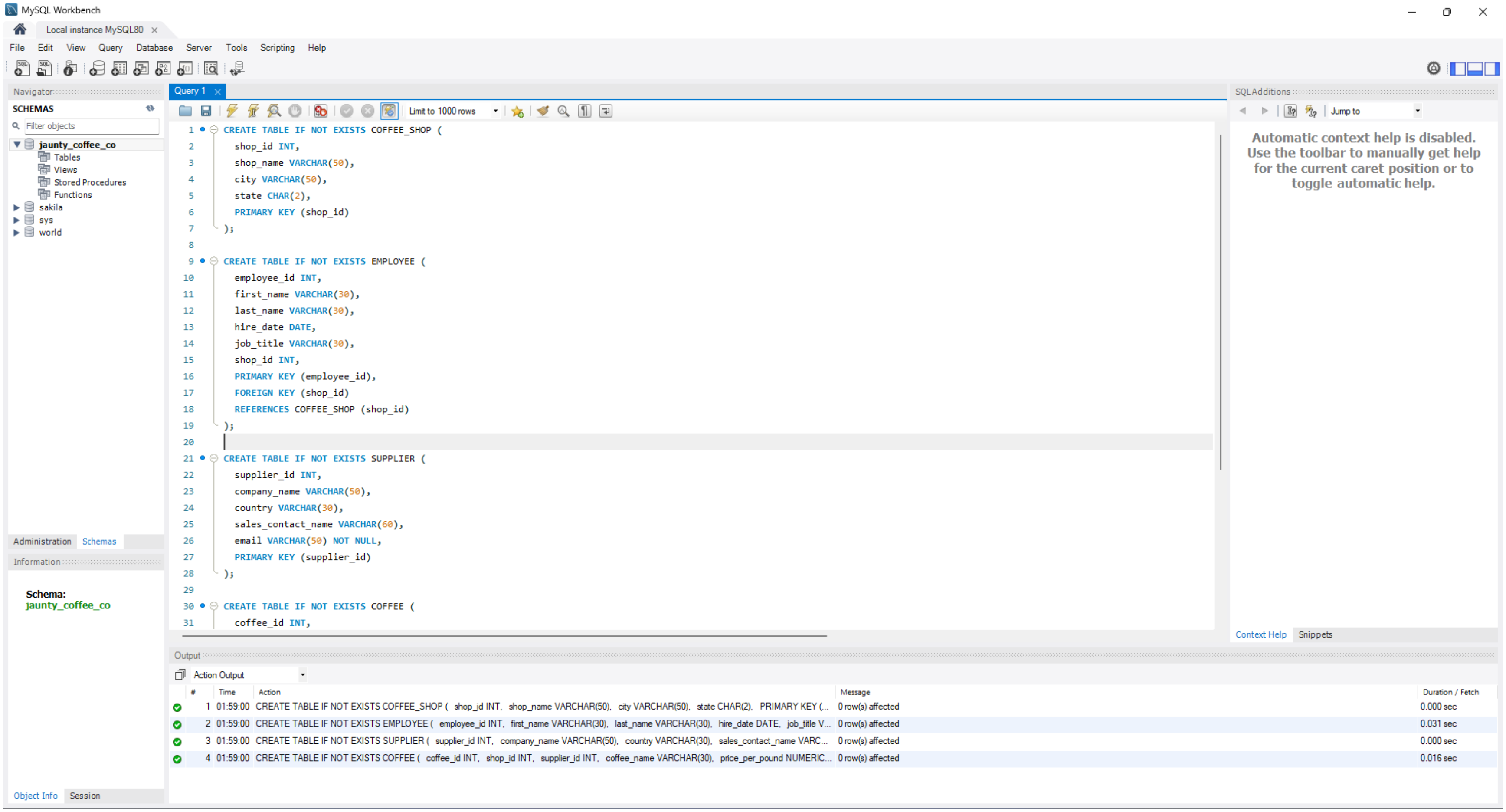
BAGEL ORDER LINE ITEM		
PK / FK	bagel_order_id	INT
PK / FK	bagel_id	CHAR(2)
	bagel_quantity	INT

M:1

BAGEL		
PK	bagel_id	CHAR(2)
	bagel_name	VARCHAR(50)
	bagel_description	VARCHAR(100)
	bagel_price	NUMERIC(4,2)

All attributes were renamed to conform to database naming conventions (no spaces) and data types were described for each attribute. Some string (CHAR) and number (Numeric) data types were defined for attributes that are known to have a fixed length.

Jaunty Coffee Co. Database Creation in MySQL

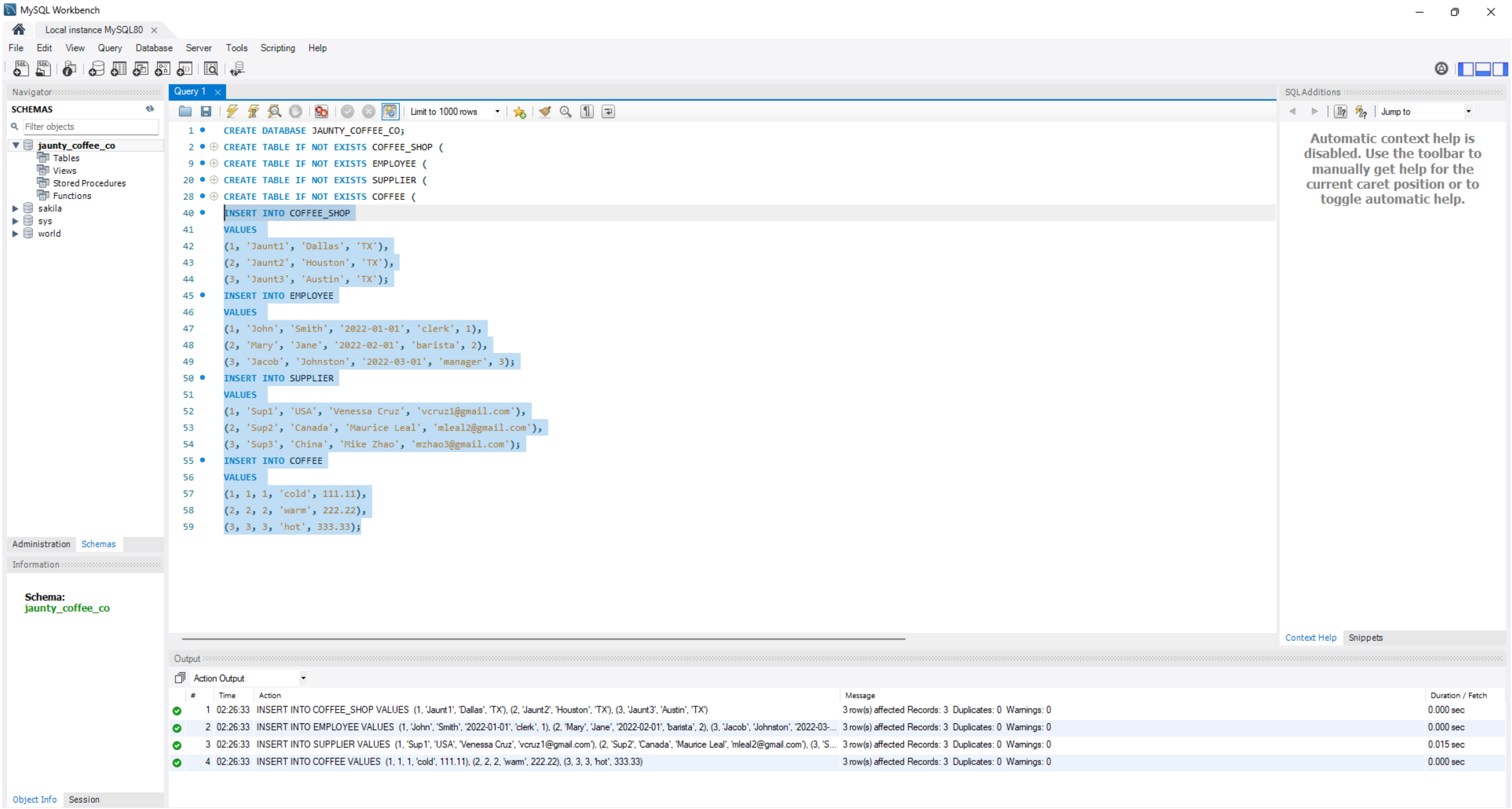


Schema: jaunty_coffee_co

```
CREATE TABLE IF NOT EXISTS COFFEE_SHOP (
    shop_id INT,
    shop_name VARCHAR(50),
    city VARCHAR(50),
    state CHAR(2),
    PRIMARY KEY (shop_id)
);

CREATE TABLE IF NOT EXISTS EMPLOYEE (
    employee_id INT,
    first_name VARCHAR(30),
    last_name VARCHAR(30),
    hire_date DATE,
    job_title VARCHAR(30),
    shop_id INT,
    PRIMARY KEY (employee_id),
    FOREIGN KEY (shop_id)
    REFERENCES COFFEE_SHOP (shop_id)
);

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

CREATE TABLE IF NOT EXISTS COFFEE (
    coffee_id INT,
```

```
   #  Time      Action                                                                                                                  Message              Duration / Fetch
✓  1  01:59:00  CREATE TABLE IF NOT EXISTS COFFEE_SHOP (  shop_id INT,  shop_name VARCHAR(50),  city VARCHAR(50),  state CHAR(2),  PRIMARY KEY (...  0 row(s) affected    0.000 sec
✓  2  01:59:00  CREATE TABLE IF NOT EXISTS EMPLOYEE (  employee_id INT,  first_name VARCHAR(30),  last_name VARCHAR(30),  hire_date DATE,  job_title V...  0 row(s) affected    0.031 sec
✓  3  01:59:00  CREATE TABLE IF NOT EXISTS SUPPLIER (  supplier_id INT,  company_name VARCHAR(50),  country VARCHAR(30),  sales_contact_name VARC...  0 row(s) affected    0.000 sec
✓  4  01:59:00  CREATE TABLE IF NOT EXISTS COFFEE (  coffee_id INT,  shop_id INT,  supplier_id INT,  coffee_name VARCHAR(30),  price_per_pound NUMERIC...  0 row(s) affected    0.016 sec
```

All tables were created as specified in the Entity Relationship Diagram (ERD) and added to the jaunty_coffee_co database.



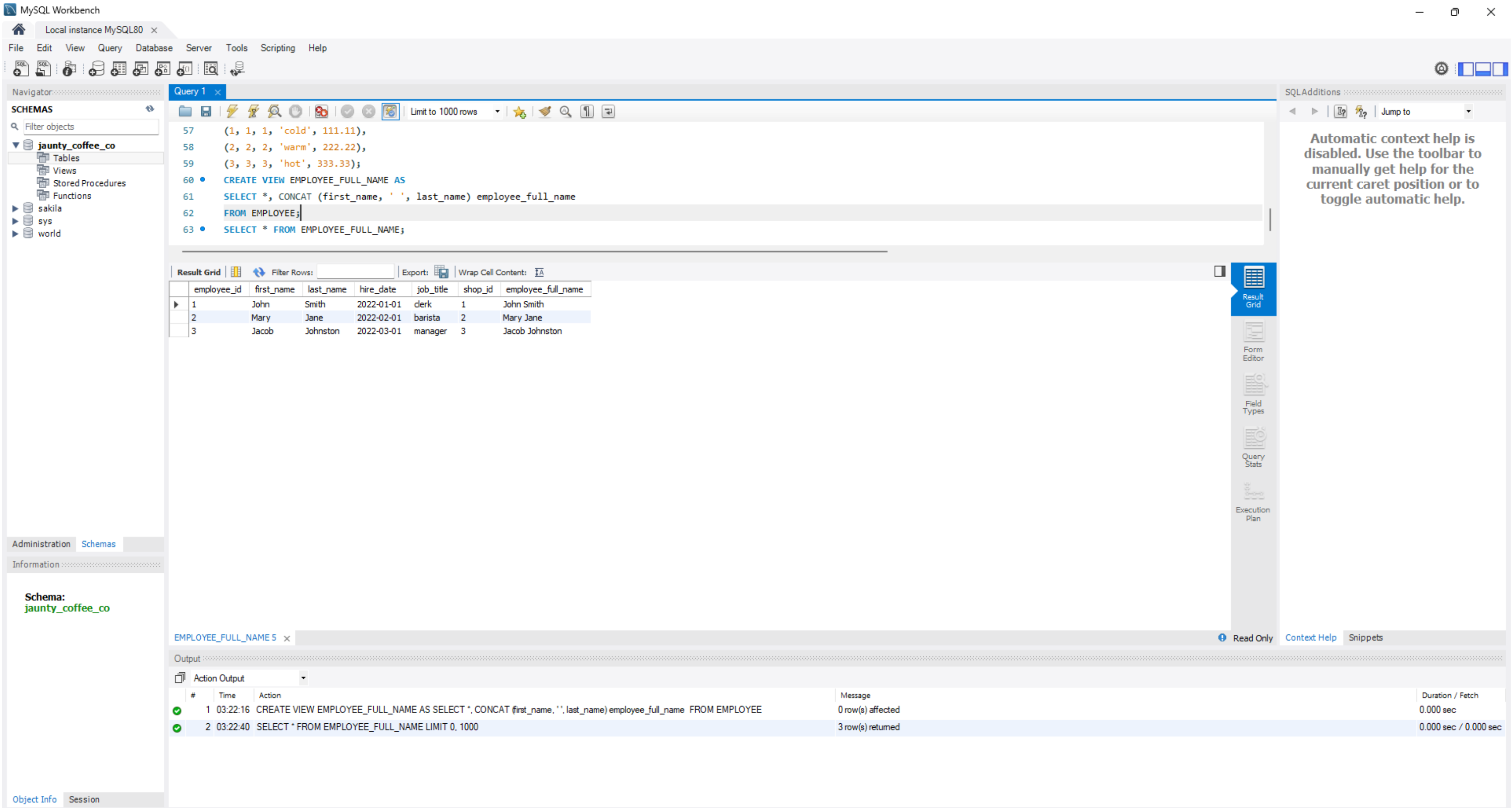
Schema:
jaunty_coffee_co

Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help.

```sql
CREATE DATABASE JAUNTY_COFFEE_CO;
CREATE TABLE IF NOT EXISTS COFFEE_SHOP (
CREATE TABLE IF NOT EXISTS EMPLOYEE (
CREATE TABLE IF NOT EXISTS SUPPLIER (
CREATE TABLE IF NOT EXISTS COFFEE (
INSERT INTO COFFEE_SHOP
VALUES
(1, 'Jaunt1', 'Dallas', 'TX'),
(2, 'Jaunt2', 'Houston', 'TX'),
(3, 'Jaunt3', 'Austin', 'TX');
INSERT INTO EMPLOYEE
VALUES
(1, 'John', 'Smith', '2022-01-01', 'clerk', 1),
(2, 'Mary', 'Jane', '2022-02-01', 'barista', 2),
(3, 'Jacob', 'Johnston', '2022-03-01', 'manager', 3);
INSERT INTO SUPPLIER
VALUES
(1, 'Sup1', 'USA', 'Venessa Cruz', 'vcruz1@gmail.com'),
(2, 'Sup2', 'Canada', 'Maurice Leal', 'mleal2@gmail.com'),
(3, 'Sup3', 'China', 'Mike Zhao', 'mzhao3@gmail.com');
INSERT INTO COFFEE
VALUES
(1, 1, 1, 'cold', 111.11),
(2, 2, 2, 'warm', 222.22),
(3, 3, 3, 'hot', 333.33);
```

#	Time	Action	Message	Duration / Fetch
1	02:26:33	INSERT INTO COFFEE_SHOP VALUES (1, 'Jaunt1', 'Dallas', 'TX'), (2, 'Jaunt2', 'Houston', 'TX'), (3, 'Jaunt3', 'Austin', 'TX')	3 row(s) affected Records: 3 Duplicates: 0 Warnings: 0	0.000 sec
2	02:26:33	INSERT INTO EMPLOYEE VALUES (1, 'John', 'Smith', '2022-01-01', 'clerk', 1), (2, 'Mary', 'Jane', '2022-02-01', 'barista', 2), (3, 'Jacob', 'Johnston', '2022-03-...	3 row(s) affected Records: 3 Duplicates: 0 Warnings: 0	0.000 sec
3	02:26:33	INSERT INTO SUPPLIER VALUES (1, 'Sup1', 'USA', 'Venessa Cruz', 'vcruz1@gmail.com'), (2, 'Sup2', 'Canada', 'Maurice Leal', 'mleal2@gmail.com'), (3, 'S...	3 row(s) affected Records: 3 Duplicates: 0 Warnings: 0	0.015 sec
4	02:26:33	INSERT INTO COFFEE VALUES (1, 1, 1, 'cold', 111.11), (2, 2, 2, 'warm', 222.22), (3, 3, 3, 'hot', 333.33)	3 row(s) affected Records: 3 Duplicates: 0 Warnings: 0	0.000 sec

3 rows were added to each table with random values.



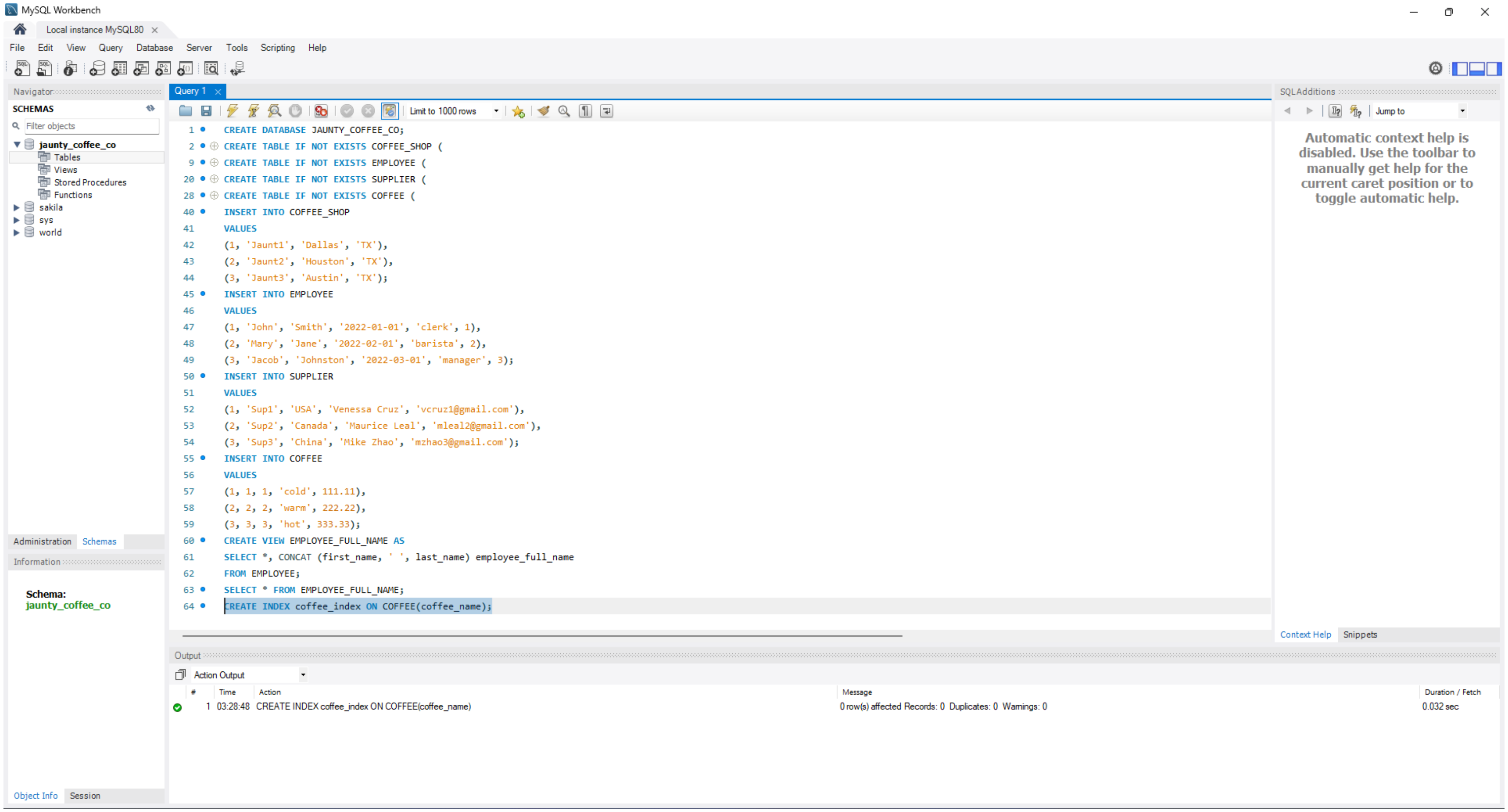
Schema: jaunty_coffee_co

```sql
(1, 1, 1, 'cold', 111.11),
(2, 2, 2, 'warm', 222.22),
(3, 3, 3, 'hot', 333.33);
CREATE VIEW EMPLOYEE_FULL_NAME AS
SELECT *, CONCAT (first_name, ' ', last_name) employee_full_name
FROM EMPLOYEE;
SELECT * FROM EMPLOYEE_FULL_NAME;
```

employee_id	first_name	last_name	hire_date	job_title	shop_id	employee_full_name
1	John	Smith	2022-01-01	clerk	1	John Smith
2	Mary	Jane	2022-02-01	barista	2	Mary Jane
3	Jacob	Johnston	2022-03-01	manager	3	Jacob Johnston

#	Time	Action	Message	Duration / Fetch
1	03:22:16	CREATE VIEW EMPLOYEE_FULL_NAME AS SELECT *, CONCAT (first_name, ' ', last_name) employee_full_name FROM EMPLOYEE	0 row(s) affected	0.000 sec
2	03:22:40	SELECT * FROM EMPLOYEE_FULL_NAME LIMIT 0, 1000	3 row(s) returned	0.000 sec / 0.000 sec

First and Last name were concatenated with a space into a new column, then added to a view of the EMPLOYEE table.



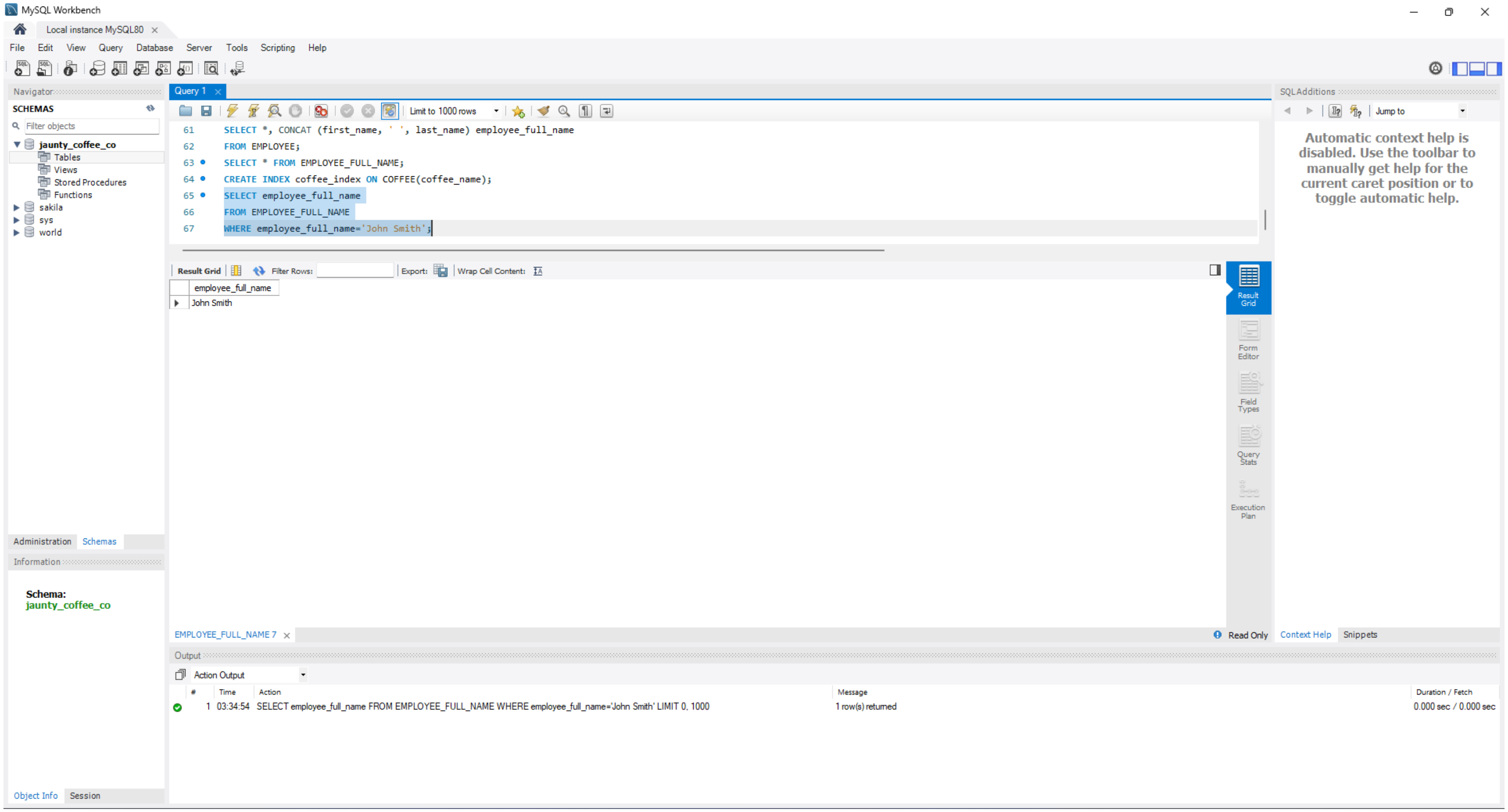
```sql
CREATE DATABASE JAUNTY_COFFEE_CO;
CREATE TABLE IF NOT EXISTS COFFEE_SHOP (
CREATE TABLE IF NOT EXISTS EMPLOYEE (
CREATE TABLE IF NOT EXISTS SUPPLIER (
CREATE TABLE IF NOT EXISTS COFFEE (
INSERT INTO COFFEE_SHOP
VALUES
(1, 'Jaunt1', 'Dallas', 'TX'),
(2, 'Jaunt2', 'Houston', 'TX'),
(3, 'Jaunt3', 'Austin', 'TX');
INSERT INTO EMPLOYEE
VALUES
(1, 'John', 'Smith', '2022-01-01', 'clerk', 1),
(2, 'Mary', 'Jane', '2022-02-01', 'barista', 2),
(3, 'Jacob', 'Johnston', '2022-03-01', 'manager', 3);
INSERT INTO SUPPLIER
VALUES
(1, 'Sup1', 'USA', 'Venessa Cruz', 'vcruz1@gmail.com'),
(2, 'Sup2', 'Canada', 'Maurice Leal', 'mleal2@gmail.com'),
(3, 'Sup3', 'China', 'Mike Zhao', 'mzhao3@gmail.com');
INSERT INTO COFFEE
VALUES
(1, 1, 1, 'cold', 111.11),
(2, 2, 2, 'warm', 222.22),
(3, 3, 3, 'hot', 333.33);
CREATE VIEW EMPLOYEE_FULL_NAME AS
SELECT *, CONCAT (first_name, ' ', last_name) employee_full_name
FROM EMPLOYEE;
SELECT * FROM EMPLOYEE_FULL_NAME;
CREATE INDEX coffee_index ON COFFEE(coffee_name);
```

Schema: jaunty_coffee_co

Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help.

```
#   Time      Action                                          Message                                                        Duration / Fetch
1   03:28:48  CREATE INDEX coffee_index ON COFFEE(coffee_name)  0 row(s) affected Records: 0  Duplicates: 0  Warnings: 0   0.032 sec
```

An index named coffee_index was created on the column coffee_name in table COFFEE.



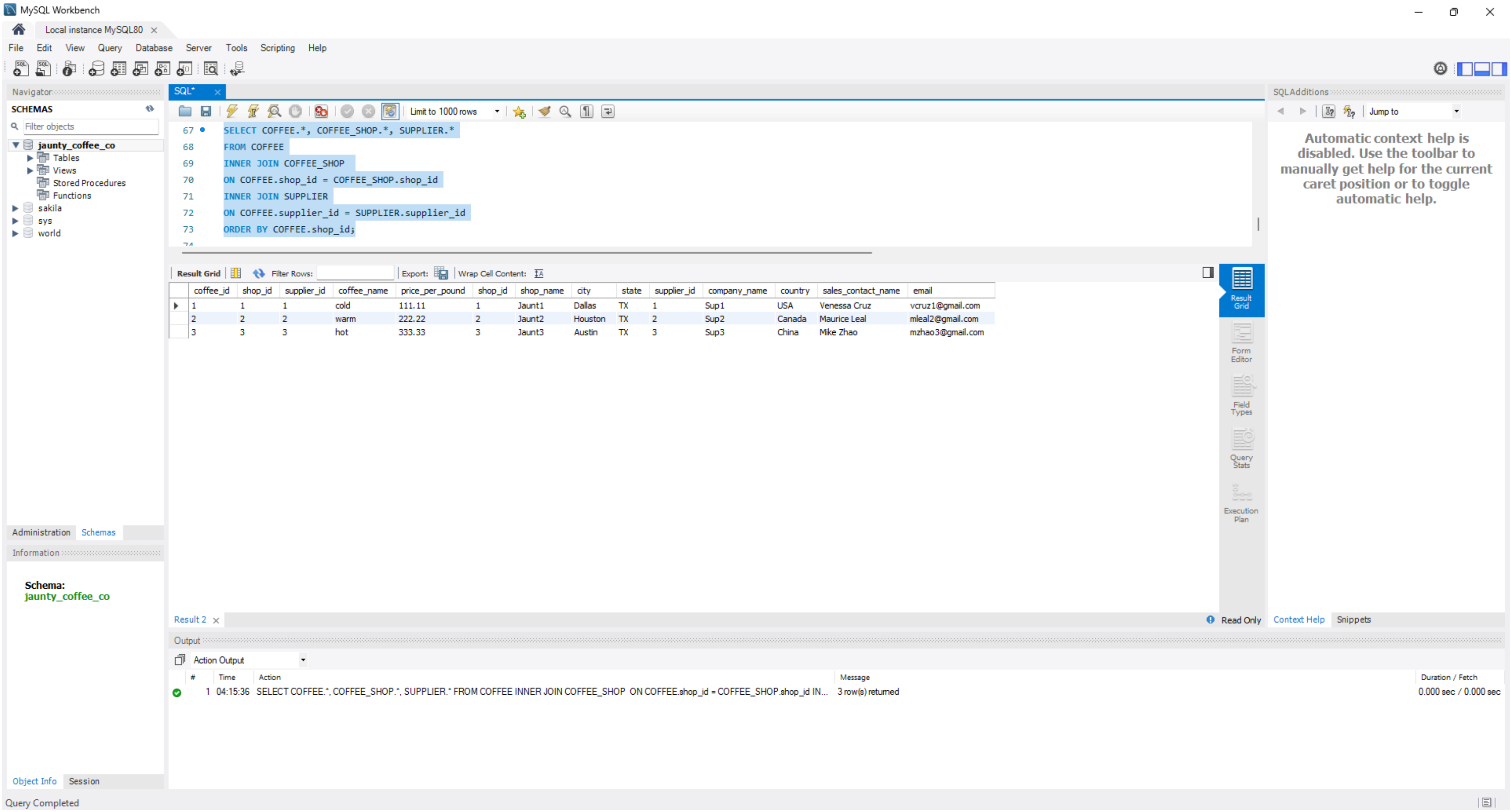
Schema:
jaunty_coffee_co

```
SELECT *, CONCAT (first_name, ' ', last_name) employee_full_name
FROM EMPLOYEE;
SELECT * FROM EMPLOYEE_FULL_NAME;
CREATE INDEX coffee_index ON COFFEE(coffee_name);
SELECT employee_full_name
FROM EMPLOYEE_FULL_NAME
WHERE employee_full_name='John Smith';
```

employee_full_name
John Smith

#	Time	Action	Message	Duration / Fetch
1	03:34:54	SELECT employee_full_name FROM EMPLOYEE_FULL_NAME WHERE employee_full_name='John Smith' LIMIT 0, 1000	1 row(s) returned	0.000 sec / 0.000 sec

To demonstrate a SELECT FROM WHERE (SFW) query, an employee full name match with ‘John Smith’ was selected from the EMPLOYEE_FULL_NAME view.



```sql
SELECT COFFEE.*, COFFEE_SHOP.*, SUPPLIER.*
FROM COFFEE
INNER JOIN COFFEE_SHOP
ON COFFEE.shop_id = COFFEE_SHOP.shop_id
INNER JOIN SUPPLIER
ON COFFEE.supplier_id = SUPPLIER.supplier_id
ORDER BY COFFEE.shop_id;
```

Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help.

coffee_id	shop_id	supplier_id	coffee_name	price_per_pound	shop_id	shop_name	city	state	supplier_id	company_name	country	sales_contact_name	email
1	1	1	cold	111.11	1	Jaunt1	Dallas	TX	1	Sup1	USA	Venessa Cruz	vcruz1@gmail.com
2	2	2	warm	222.22	2	Jaunt2	Houston	TX	2	Sup2	Canada	Maurice Leal	mleal2@gmail.com
3	3	3	hot	333.33	3	Jaunt3	Austin	TX	3	Sup3	China	Mike Zhao	mzhao3@gmail.com

#	Time	Action	Message	Duration / Fetch
1	04:15:36	SELECT COFFEE.*, COFFEE_SHOP.*, SUPPLIER.* FROM COFFEE INNER JOIN COFFEE_SHOP ON COFFEE.shop_id = COFFEE_SHOP.shop_id IN...	3 row(s) returned	0.000 sec / 0.000 sec

Query Completed

Inner join was used on the 2 tables COFFEE, COFFEE_SHOP, and SUPPLIER to display all attributes from all tables.