Bakery Database

CSBP 3287 - Spring 2023 Semester Project

Eric Hanley

Project Requirements

- ☑ Multiple Table
- ☑ Relationships between table items (foreign keys)
- ☑ Show SQL statements (and any accompanying code) for all table creation, insertion of initial data, updates, and queries.
- ☑ Table Creation
- ☑ Constraints
- ☑ Indexes
- ☑ Triggers
- ☑ Queries
- ☑ Joins between tables
- ☑ Grouping Results
- ☑ Updates (show triggers being executed)
- ☑ Deleting items that are foreign keys in other tables (show triggers being executed)

Create a SQLite DB

In [1]:
```
import sqlite3
import csv
from sqlite3 import Error

def create_conn(db_file):
    """ create a db connection to a SQLite db """
    conn = None
    try:
        conn = sqlite3.connect(db_file)
        print(sqlite3.version)
    except Error as e:
        print(e)
    finally:
        if conn:
            conn.close()
```

In [2]:
```
create_conn('bakery.db')
```

```
2.6.0
```

In [3]:
```
%load_ext sql
%sql sqlite:///bakery.db
```

Drop All Tables and Triggers For Clean Build

In [4]:
```
%%sql
drop table if exists ingredient_metadata;
drop table if exists ingredient_inventory;
drop table if exists supplier;
drop table if exists product_metadata;
drop table if exists product_inventory;
drop table if exists bill_of_materials;
drop table if exists product_order;
drop table if exists supply_order;
drop table if exists customer;
drop trigger if exists update_ingredient_inventory_insert_trigger;
drop trigger if exists update_product_inventory_insert_trigger;
drop trigger if exists delete_supplier_ingredients;
```

```
 * sqlite:///bakery.db
Done.
Done.
Done.
Done.
Done.
Done.
Done.
Done.
Done.
Done.
Done.
Done.
```

Out[4]:
```
[]
```

In [5]:
```
### Create Tables
```

In [6]:
```
%%sql
CREATE TABLE ingredient_metadata(
    ingredient_ID int PRIMARY KEY,
    supplier_ID int NOT NULL,
    name varchar(50) NOT NULL,
    category varchar(50) NOT NULL,
    description varchar(100) NOT NULL,
    FOREIGN KEY (supplier_ID)
        REFERENCES supplier (supplier_ID)
);

CREATE TABLE ingredient_inventory(
    ingredient_category varchar(50) PRIMARY KEY,
    quantity real NOT NULL,
    unit varchar(50) NOT NULL,
    CHECK (quantity >= 0),
    CHECK (unit IN ('g', 'mL')),
    FOREIGN KEY (ingredient_category)
        REFERENCES ingredient_metadata (category)
);

CREATE TABLE supplier(
    supplier_ID int PRIMARY KEY,
    name varchar(50) NOT NULL,
    address varchar(100) NOT NULL,
```

```
    city varchar(50) NOT NULL,
    state varchar(2) NOT NULL,
    zip_code varchar(5) NOT NULL
);

CREATE TABLE product_metadata(
    product_ID int PRIMARY KEY,
    name varchar(50) NOT NULL,
    description varchar(100) NOT NULL,
    unit_price real NOT NULL
);

CREATE TABLE product_inventory(
    product_ID int PRIMARY_KEY,
    quantity int NOT NULL
);

CREATE TABLE bill_of_materials(
    bom_ID int NOT NULL,
    product_ID int NOT NULL,
    line int NOT NULL,
    ingredient_category varchar(50) NOT NULL,
    quantity real NOT NULL,
    unit varchar(50) NOT NULL,
    CHECK (unit IN ('g', 'mL'))
    FOREIGN KEY (ingredient_category)
        REFERENCES ingredient_metadata (category)
    FOREIGN KEY (product_ID)
        REFERENCES product_metadata (product_ID)
    PRIMARY KEY (bom_ID, line)
);

CREATE TABLE product_order(
    order_ID int NOT NULL,
    line int NOT NULL,
    order_date int NOT NULL,
    due_date int NOT NULL,
    product_ID int NOT NULL,
    quantity int NOT NULL,
    customer_ID int NOT NULL,
    FOREIGN KEY (product_ID)
        REFERENCES product_metadata (product_ID)
    FOREIGN KEY (customer_ID)
        REFERENCES customer (customer_ID)
    PRIMARY KEY (order_ID, line)
);

CREATE TABLE supply_order(
    order_ID int NOT NULL,
    line int NOT NULL,
    supplier_ID int NOT NULL,
    ingredient_ID int NOT NULL,
    quantity real NOT NULL,
    unit varchar(50) NOT NULL,
    FOREIGN KEY (supplier_ID)
        REFERENCES supplier (supplier_ID)
    FOREIGN KEY (ingredient_ID)
        REFERENCES ingredient_metadata (ingredient_ID)
    PRIMARY KEY (order_ID, line)
    CHECK (unit IN ('g', 'mL'))

);

CREATE TABLE customer(
    customer_ID int PRIMARY_KEY,
    name varchar(50) NOT NULL,
    address varchar(100) NOT NULL,
    city varchar(50) NOT NULL,
    state varchar(2) NOT NULL,
    zip_code varchar(5) NOT NULL
);
```

```
 * sqlite:///bakery.db
Done.
Done.
Done.
Done.
Done.
Done.
Done.
Done.
Done.
Done.
```

Out[6]: []

Create Triggers

In [7]:
```
%%sql
/*
Create the trigger for INSERT operation on the supply_order table.
This trigger updates the quantity of an existing ingredient in inventory
or adds it to the inventory if it doesn't exist
*/
CREATE TRIGGER update_ingredient_inventory_insert_trigger
AFTER INSERT ON supply_order
BEGIN
    INSERT OR REPLACE INTO ingredient_inventory (ingredient_category, quantity, unit)
    SELECT
        (SELECT category FROM ingredient_metadata WHERE NEW.ingredient_ID = ingredient_metadata.ingredient_ID),
        COALESCE((SELECT quantity
                  FROM ingredient_inventory
                  WHERE ingredient_category = (
                      SELECT category
                      FROM ingredient_metadata
                      WHERE NEW.ingredient_ID == ingredient_metadata.ingredient_ID)
                 ), 0) + NEW.quantity,
        COALESCE((SELECT unit
                  FROM ingredient_inventory
                  WHERE (SELECT category
                         FROM ingredient_metadata
                         WHERE NEW.ingredient_ID = ingredient_metadata.ingredient_ID)
                 ), NEW.unit)
    ;
END;
```

```
/*
Create the trigger for INSERT operation on the product_order table.
This trigger updates the product inventory when an item is ordered
*/
CREATE TRIGGER update_product_inventory_insert_trigger
AFTER INSERT ON product_order
BEGIN
    INSERT OR REPLACE INTO product_inventory (product_ID, quantity)
    SELECT
        NEW.product_ID,
        COALESCE((SELECT quantity FROM product_inventory WHERE product_ID = NEW.product_ID), 0) - NEW.quantity
    ;
END;

/*
Create the trigger for DELETE operation on the supplier table.
This trigger removes all ingredients from a supplier if the supplier is deleted
*/
CREATE TRIGGER delete_supplier_ingredients
AFTER DELETE ON supplier
BEGIN
    DELETE FROM ingredient_metadata
    WHERE supplier_ID = OLD.supplier_ID;
END;

PRAGMA trigger_trace = ON;
```

```
 * sqlite:///bakery.db
Done.
Done.
Done.
Done.
```

Out[7]: []

Create Indices

In [8]:
```
%%sql
CREATE INDEX ingredient_inventory_quantity_index ON ingredient_inventory(quantity);
CREATE INDEX ingredient_inventory_ID_index ON ingredient_inventory(ingredient_category);
```

```
 * sqlite:///bakery.db
Done.
Done.
```

Out[8]: []

Populate Tables

In [9]:
```
# Helper function to insert example data from CSV files
def insert_from_csv(file_name, table_name):

    # Connect to the SQLite database
    conn = sqlite3.connect("bakery.db", timeout=30)
    cur = conn.cursor()


    # Open the CSV file
    with open(file_name, "r") as f:


        # Read the CSV data using csv.reader
        csv_data = csv.reader(f)

        # get field names from header row
        fields = next(csv_data)

        # build sql string
        ins_str = f'INSERT INTO {table_name} ({", ".join(fields)})\n VALUES ({", ".join(["?" for field in fields])})'
        print(ins_str)

        # Insert each row from the CSV into the supplier table
        for row in csv_data:
            cur.execute(ins_str, row)

    # Commit the transaction and close the connection
    conn.commit()
    conn.close()
    print(f'Table {table_name} populated and connection closed.\n')
```

In [10]:
```
ins_params = [('data/bakery_data/ingredients-Table 1.csv', 'ingredient_metadata'),
              ('data/bakery_data/bom-Table 1.csv', 'bill_of_materials'),
              ('data/bakery_data/customer-Table 1.csv', 'customer'),
              ('data/bakery_data/products-Table 1.csv', 'product_metadata'),
              ('data/bakery_data/supplier-Table 1.csv', 'supplier')]

for item in ins_params:
    insert_from_csv(item[0], item[1])
```

```
INSERT INTO ingredient_metadata (ingredient_ID, supplier_ID, name, category, description)
 VALUES (?, ?, ?, ?, ?)
Table ingredient_metadata populated and connection closed.

INSERT INTO bill_of_materials (bom_ID, product_ID, line, ingredient_category, quantity, unit)
 VALUES (?, ?, ?, ?, ?, ?)
Table bill_of_materials populated and connection closed.

INSERT INTO customer (customer_ID, name, address, city, state, zip_code)
 VALUES (?, ?, ?, ?, ?, ?)
Table customer populated and connection closed.

INSERT INTO product_metadata (product_ID, name, description, unit_price)
 VALUES (?, ?, ?, ?)
Table product_metadata populated and connection closed.

INSERT INTO supplier (supplier_ID, name, address, city, state, zip_code)
 VALUES (?, ?, ?, ?, ?, ?)
Table supplier populated and connection closed.
```

Demo Path

1. Customer places order (create order, show order, join for order/product/price/inventory)
2. Check materials inventory against order requirements (join inventory, group by ingred category)
3. Place a materials order to adjust for shortfall (show updated materials inventory)
    A. one order for item that is not already in inventory
    B. one order for item that already exists in inventory
4. Remove a supplier (too slow, price too high, etc) - show cascading FK delete in ingred metadata

Customer Places an Order

The following code block inserts a new customer order. It then queries that order and augments it by joining the customer and product_metadata tables to include customer names, product name, and total price. First we'll query the product inventory (currently empty) to confirm the order trigger updates the product inventory when a customer order is placed.

In [11]:
```
%%sql
-- Check product inventory table before order is placed
SELECT *
FROM product_inventory;
```

```
 * sqlite:///bakery.db
Done.
```

Out[11]:

product_ID	quantity

In [12]:
```
%%sql
-- Customer places order for three loaves of Rustic Sourdough and eight loaves of Dark Rye
INSERT INTO product_order (order_ID, line, order_date, due_date, product_ID, quantity, customer_ID)
VALUES
    (1, 1, "2023-04-04", "2023-04-04", 2, 3, 1),
    (1, 2, "2023-04-04", "2023-04-04", 4, 8, 1);

-- Query the order we just created with joins to get customer name, product name, and total price
SELECT order_ID,
    line,
    order_date,
    due_date,
    customer.name AS customer_name,
    product_metadata.name AS product_name,
    quantity,
    PRINTF("$%.2f", quantity*product_metadata.unit_price) AS total_cost
FROM product_order
    LEFT JOIN customer ON product_order.customer_ID = customer.customer_ID
    LEFT JOIN product_metadata ON product_order.product_ID = product_metadata.product_ID
WHERE order_ID = (SELECT MAX(order_ID) FROM product_order);
```

```
 * sqlite:///bakery.db
2 rows affected.
Done.
```

Out[12]:

order_ID	line	order_date	due_date	customer_name	product_name	quantity	total_cost
1	1	2023-04-04	2023-04-04	John Public	Rustic Sourdough	3	$23.85
1	2	2023-04-04	2023-04-04	John Public	Dark Rye Loaf	8	$38.00

Check Product Inventory After Order is Placed

In [13]:
```
%%sql
-- This query confirms that the product inventory trigger updates the inventory when an order is placed
SELECT product_inventory.product_ID,
    quantity,
    name
FROM product_inventory
    LEFT JOIN product_metadata ON product_inventory.product_ID = product_metadata.product_ID;
```

```
 * sqlite:///bakery.db
Done.
```

Out[13]:

product_ID	quantity	name
2	-3	Rustic Sourdough
4	-8	Dark Rye Loaf

Check Ingredients Inventory

Now that an order has been placed, we will check if we have sufficient ingredients on hand to complete the order. We can get the required ingredients quantities from the bill_of_materials table.

In [14]:
```
%%sql
-- This query grabs the order created above and compares the required ingredients to what is on hand
WITH required_ingredients AS (
    SELECT o.order_ID,
        b.ingredient_category,
        SUM(o.quantity*b.quantity) AS required_quantity,
        b.unit
    FROM product_order o, bill_of_materials b
    WHERE o.order_ID = (SELECT MAX(order_id) FROM product_order)
        AND o.product_ID = b.product_ID
    GROUP BY o.order_ID, b.ingredient_category, b.unit
)


SELECT r.order_ID,
    r.ingredient_category,
    r.required_quantity,
    COALESCE(i.quantity, 0) AS quantity_available,
    COALESCE(i.quantity, 0) - r.required_quantity AS quantity_delta
FROM required_ingredients r
    LEFT JOIN ingredient_inventory i ON r.ingredient_category = i.ingredient_category
;
```

```
 * sqlite:///bakery.db
Done.
```

Out[14]:

order_ID	ingredient_category	required_quantity	quantity_available	quantity_delta
1	Brown Sugar	240.0	0	-240.0
1	Medium Rye Flour	7500.0	0	-7500.0
1	Salt	170.0	0	-170.0
1	Wheat Flour	4500.0	0	-4500.0
1	Yeast	40.0	0	-40.0

Order Ingredients

Given the shortages we see from the query above, we need to order more ingredients.

In [15]:
```
%%sql
INSERT INTO supply_order (order_ID, line, supplier_ID, ingredient_ID, quantity, unit)
 VALUES
    (1, 1, 4, 17, 2500, "g"),
    (1, 2, 1, 3, 10000, "g"),
    (1, 3, 4, 15, 1000, "g"),
    (1, 4, 1, 1, 10000, "g"),
    (1, 5, 5, 20, 2500, "g");
```

```
 * sqlite:///bakery.db
5 rows affected.
```

Out[15]: []

Check Ingredient Inventory after Placing Supply Order

Now that we have order ingredients, let's check our ingredient inventory again to confirm the trigger that runs on supply order inserts is working as expected.

In [16]:
```
%%sql
-- This query grabs the order created above and compares the required ingredients to what is on hand
WITH required_ingredients AS (
    SELECT o.order_ID,
        b.ingredient_category,
        SUM(o.quantity*b.quantity) AS required_quantity,
        b.unit
    FROM product_order o, bill_of_materials b
    WHERE o.order_ID = (SELECT MAX(order_id) FROM product_order)
        AND o.product_ID = b.product_ID
    GROUP BY o.order_ID, b.ingredient_category, b.unit
)


SELECT r.order_ID,
    r.ingredient_category,
    r.required_quantity,
    COALESCE(i.quantity, 0) AS quantity_available,
    COALESCE(i.quantity, 0) - r.required_quantity AS quantity_delta
FROM required_ingredients r
    LEFT JOIN ingredient_inventory i ON r.ingredient_category = i.ingredient_category
;
```

```
 * sqlite:///bakery.db
Done.
```

Out[16]:

order_ID	ingredient_category	required_quantity	quantity_available	quantity_delta
1	Brown Sugar	240.0	2500.0	2260.0
1	Medium Rye Flour	7500.0	10000.0	2500.0
1	Salt	170.0	1000.0	830.0
1	Wheat Flour	4500.0	10000.0	5500.0
1	Yeast	40.0	2500.0	2460.0

Remove a Supplier

There may be a time when a supplier is removed for pricing, performance, or new contract. When a supplier is dropped, a trigger will remove all of their respective products from the ingredients_metadata table.

In [17]:
```
%%sql
-- First run a query to see current suppliers
SELECT * FROM supplier;
```

```
 * sqlite:///bakery.db
Done.
```

Out[17]:

supplier_ID	name	address	city	state	zip_code
1	The Flour Company	123 Main St	Anytown	CO	80000
2	Wheat House	456 Side St	Yonder	MO	99999
3	Powdered Plants	777 Warehouse Way	Boston	MA	88888
4	Everything Else, Inc	1115 1st Ave	Douglas	WY	66666
5	Other Ingredients Co	2356 Enterprise Dr	Chicken	AK	56789

In [18]:
```
%%sql
-- Query to see ingredients from The Flour Company
SELECT *
FROM ingredient_metadata
WHERE supplier_ID = 1;
```

```
 * sqlite:///bakery.db
Done.
```

Out[18]:

ingredient_ID	supplier_ID	name	category	description
1	1	Whole Wheat Bread Flour	Wheat Flour	100% whole wheat flour
2	1	Appalachian White Wheat Flour	White Flour	white bread flour
3	1	Medium Rye	Medium Rye Flour	medium dark rye flour
4	1	Light Rye	Light Rye Flour	light rye flour
5	1	Whole Spelt	Spelt Flour	whole grain spelt flour

In [19]:
```
%%sql
-- Drop The Flour Company as a supplier
DELETE FROM supplier WHERE supplier_ID = 1;
```

```
 * sqlite:///bakery.db
1 rows affected.
```

Out[19]: []

In [20]:
```
%%sql
-- Confirm removal of ingredients from The Flour Company
SELECT *
FROM ingredient_metadata
WHERE supplier_ID = 1;
```

```
 * sqlite:///bakery.db
Done.
```

Out[20]:

ingredient_ID	supplier_ID	name	category	description

The End

In [ ]: