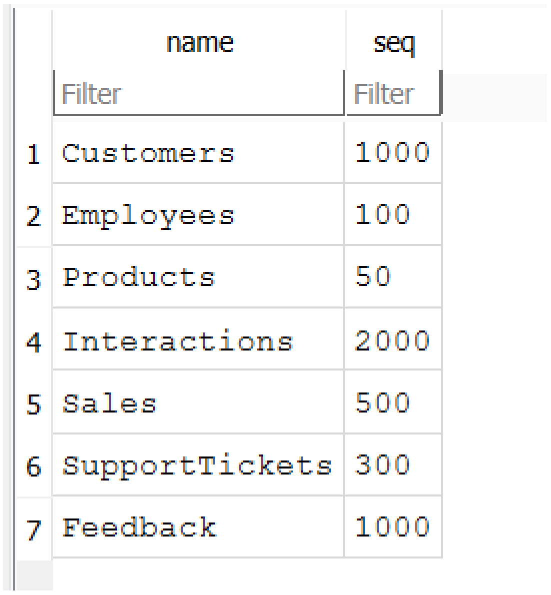
Name: Awaiz Hussain
Email: ah24adq@herts.ac.uk
UH-ID: 23100185

SQL ASSIGNMENT

This report details the design and justification of the database schema used in the customer interaction and sales management system. The schema consists of several tables representing different aspects of the system, such as customers, employees, products, interactions, sales, support tickets, and feedback.
Here is the code link for schema, python and sql code uploaded on github.

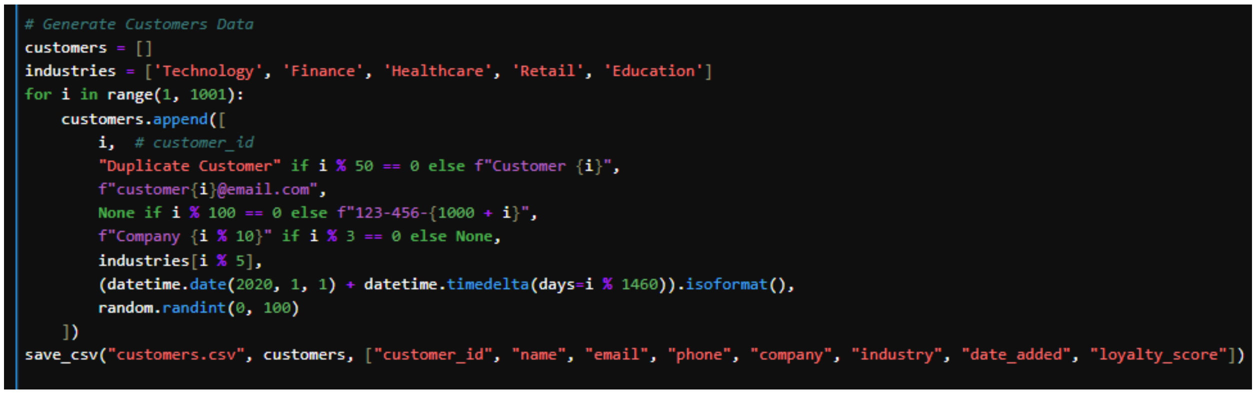


	name	seq
1	Customers	1000
2	Employees	100
3	Products	50
4	Interactions	2000
5	Sales	500
6	SupportTickets	300
7	Feedback	1000

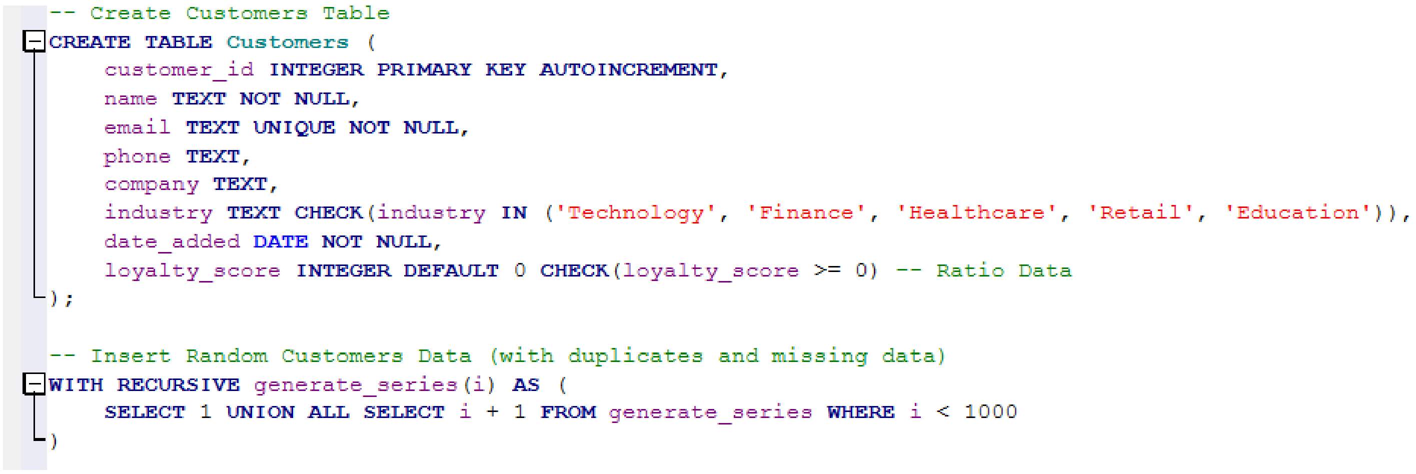
The data used in this report is generated programmatically using SQL **`WITH RECURSIVE`** queries to simulate a large set of records. Also the same data is generated and stored in CSV files using **`PYTHON`**. The use of randomization ensures that the dataset reflects a variety of real-world scenarios, including missing and duplicate data. Below is an overview of how data for each table was generated: 1st image with black background is python code to generate data and other one is sql query.

Customers Table:

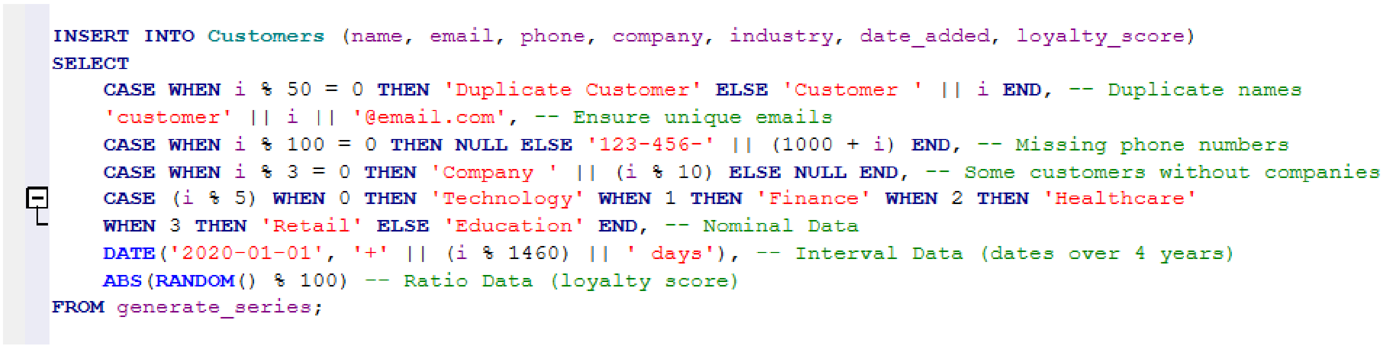
1000 records were generated, with random names, email addresses, phone numbers (some missing), company names (some missing), industry types, dates of addition, and loyalty scores.



```python
# Generate Customers Data
customers = []
industries = ['Technology', 'Finance', 'Healthcare', 'Retail', 'Education']
for i in range(1, 1001):
    customers.append([
        i,  # customer_id
        "Duplicate Customer" if i % 50 == 0 else f"Customer {i}",
        f"customer{i}@email.com",
        None if i % 100 == 0 else f"123-456-{1000 + i}",
        f"Company {i % 10}" if i % 3 == 0 else None,
        industries[i % 5],
        (datetime.date(2020, 1, 1) + datetime.timedelta(days=i % 1460)).isoformat(),
        random.randint(0, 100)
    ])
save_csv("customers.csv", customers, ["customer_id", "name", "email", "phone", "company", "industry", "date_added", "loyalty_score"])
```



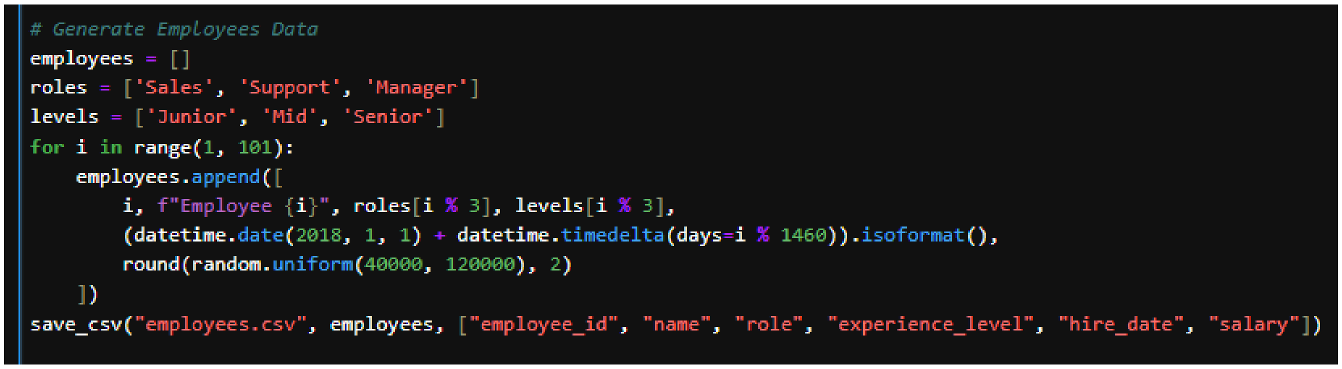
```sql
-- Create Customers Table
CREATE TABLE Customers (
    customer_id INTEGER PRIMARY KEY AUTOINCREMENT,
    name TEXT NOT NULL,
    email TEXT UNIQUE NOT NULL,
    phone TEXT,
    company TEXT,
    industry TEXT CHECK(industry IN ('Technology', 'Finance', 'Healthcare', 'Retail', 'Education')),
    date_added DATE NOT NULL,
    loyalty_score INTEGER DEFAULT 0 CHECK(loyalty_score >= 0) -- Ratio Data
);

-- Insert Random Customers Data (with duplicates and missing data)
WITH RECURSIVE generate_series(i) AS (
    SELECT 1 UNION ALL SELECT i + 1 FROM generate_series WHERE i < 1000
)
```



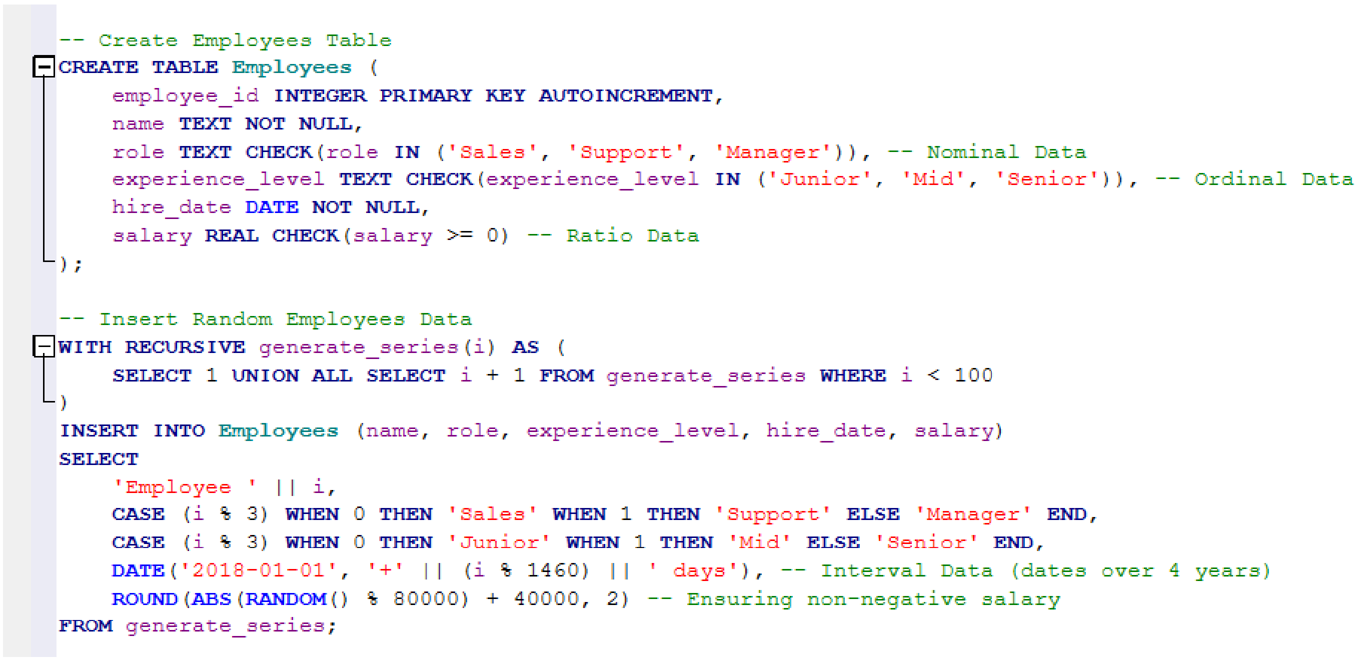
```sql
INSERT INTO Customers (name, email, phone, company, industry, date_added, loyalty_score)
SELECT
    CASE WHEN i % 50 = 0 THEN 'Duplicate Customer' ELSE 'Customer ' || i END, -- Duplicate names
    'customer' || i || '@email.com', -- Ensure unique emails
    CASE WHEN i % 100 = 0 THEN NULL ELSE '123-456-' || (1000 + i) END, -- Missing phone numbers
    CASE WHEN i % 3 = 0 THEN 'Company ' || (i % 10) ELSE NULL END, -- Some customers without companies
    CASE (i % 5) WHEN 0 THEN 'Technology' WHEN 1 THEN 'Finance' WHEN 2 THEN 'Healthcare'
    WHEN 3 THEN 'Retail' ELSE 'Education' END, -- Nominal Data
    DATE('2020-01-01', '+' || (i % 1460) || ' days'), -- Interval Data (dates over 4 years)
    ABS(RANDOM() % 100) -- Ratio Data (loyalty score)
FROM generate_series;
```

Employees Table:

100 records were created with random employee names, roles (Sales, Support, Manager), experience levels (Junior, Mid, Senior), hire dates, and salaries.



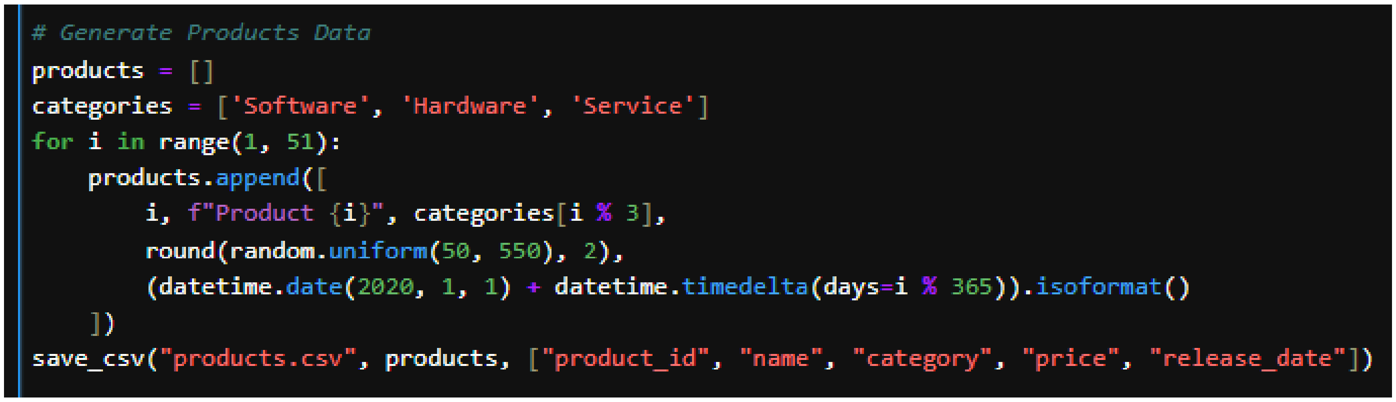
```python
# Generate Employees Data
employees = []
roles = ['Sales', 'Support', 'Manager']
levels = ['Junior', 'Mid', 'Senior']
for i in range(1, 101):
    employees.append([
        i, f"Employee {i}", roles[i % 3], levels[i % 3],
        (datetime.date(2018, 1, 1) + datetime.timedelta(days=i % 1460)).isoformat(),
        round(random.uniform(40000, 120000), 2)
    ])
save_csv("employees.csv", employees, ["employee_id", "name", "role", "experience_level", "hire_date", "salary"])
```



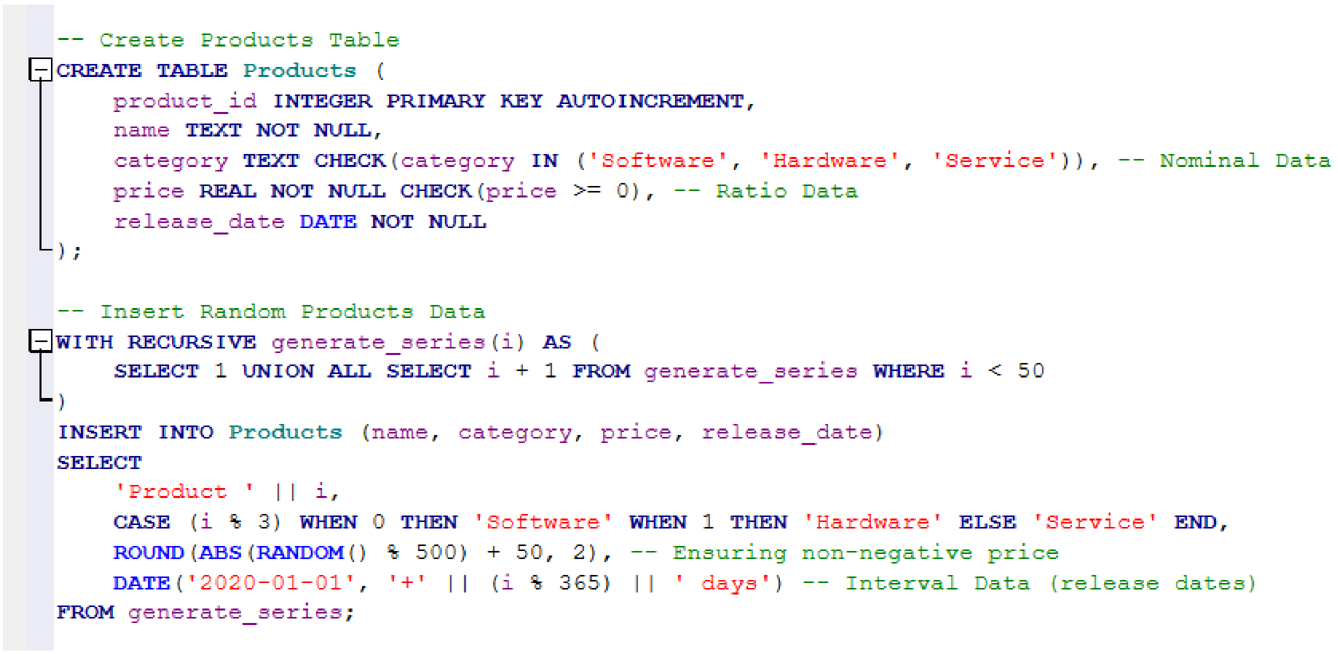
```sql
-- Create Employees Table
CREATE TABLE Employees (
    employee_id INTEGER PRIMARY KEY AUTOINCREMENT,
    name TEXT NOT NULL,
    role TEXT CHECK(role IN ('Sales', 'Support', 'Manager')), -- Nominal Data
    experience_level TEXT CHECK(experience_level IN ('Junior', 'Mid', 'Senior')), -- Ordinal Data
    hire_date DATE NOT NULL,
    salary REAL CHECK(salary >= 0) -- Ratio Data
);

-- Insert Random Employees Data
WITH RECURSIVE generate_series(i) AS (
    SELECT 1 UNION ALL SELECT i + 1 FROM generate_series WHERE i < 100
)
INSERT INTO Employees (name, role, experience_level, hire_date, salary)
SELECT
    'Employee ' || i,
    CASE (i % 3) WHEN 0 THEN 'Sales' WHEN 1 THEN 'Support' ELSE 'Manager' END,
    CASE (i % 3) WHEN 0 THEN 'Junior' WHEN 1 THEN 'Mid' ELSE 'Senior' END,
    DATE('2018-01-01', '+' || (i % 1460) || ' days'), -- Interval Data (dates over 4 years)
    ROUND(ABS(RANDOM() % 80000) + 40000, 2) -- Ensuring non-negative salary
FROM generate_series;
```

Products Table:

50 records were generated for different products with random names, categories (Software, Hardware, Service), prices, and release dates.



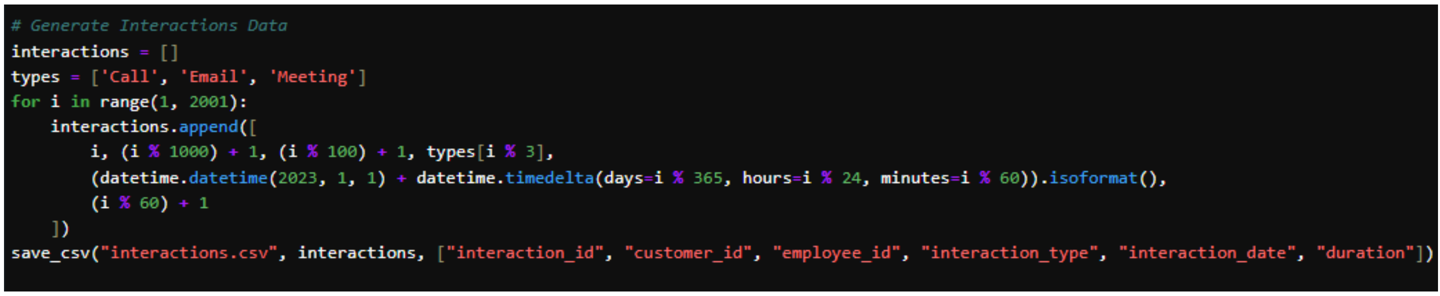
```python
# Generate Products Data
products = []
categories = ['Software', 'Hardware', 'Service']
for i in range(1, 51):
    products.append([
        i, f"Product {i}", categories[i % 3],
        round(random.uniform(50, 550), 2),
        (datetime.date(2020, 1, 1) + datetime.timedelta(days=i % 365)).isoformat()
    ])
save_csv("products.csv", products, ["product_id", "name", "category", "price", "release_date"])
```



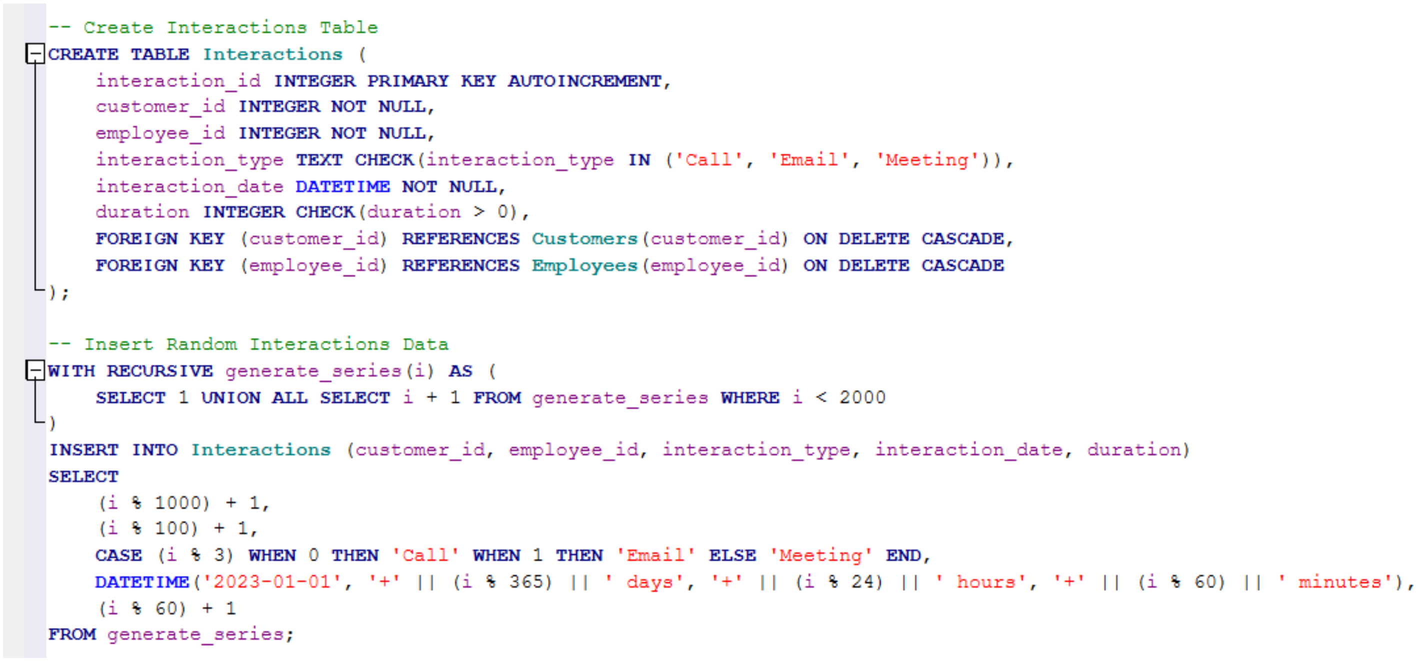
```sql
-- Create Products Table
CREATE TABLE Products (
    product_id INTEGER PRIMARY KEY AUTOINCREMENT,
    name TEXT NOT NULL,
    category TEXT CHECK(category IN ('Software', 'Hardware', 'Service')), -- Nominal Data
    price REAL NOT NULL CHECK(price >= 0), -- Ratio Data
    release_date DATE NOT NULL
);

-- Insert Random Products Data
WITH RECURSIVE generate_series(i) AS (
    SELECT 1 UNION ALL SELECT i + 1 FROM generate_series WHERE i < 50
)
INSERT INTO Products (name, category, price, release_date)
SELECT
    'Product ' || i,
    CASE (i % 3) WHEN 0 THEN 'Software' WHEN 1 THEN 'Hardware' ELSE 'Service' END,
    ROUND(ABS(RANDOM() % 500) + 50, 2), -- Ensuring non-negative price
    DATE('2020-01-01', '+' || (i % 365) || ' days') -- Interval Data (release dates)
FROM generate_series;
```

Interactions Table:

2000 interaction records were created, linking customers and employees with random interaction types (Call, Email, Meeting), interaction dates, and durations.



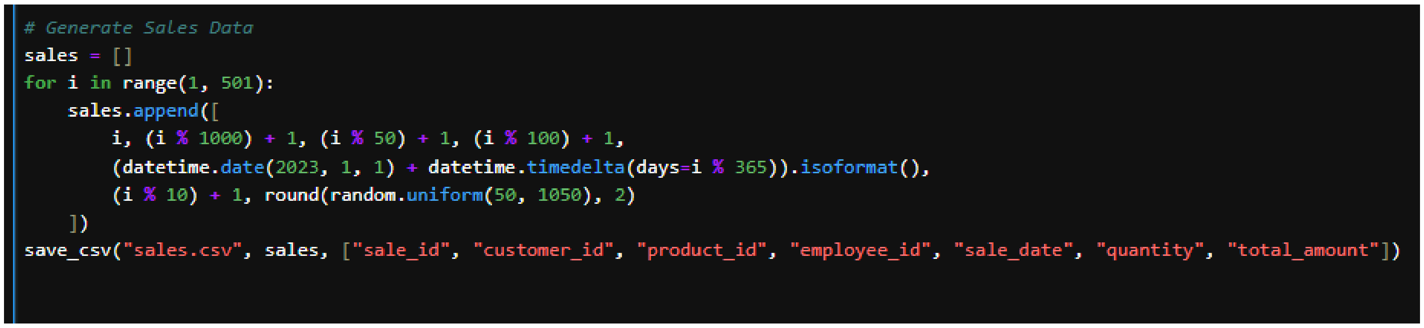
```python
# Generate Interactions Data
interactions = []
types = ['Call', 'Email', 'Meeting']
for i in range(1, 2001):
    interactions.append([
        i, (i % 1000) + 1, (i % 100) + 1, types[i % 3],
        (datetime.datetime(2023, 1, 1) + datetime.timedelta(days=i % 365, hours=i % 24, minutes=i % 60)).isoformat(),
        (i % 60) + 1
    ])
save_csv("interactions.csv", interactions, ["interaction_id", "customer_id", "employee_id", "interaction_type", "interaction_date", "duration"])
```



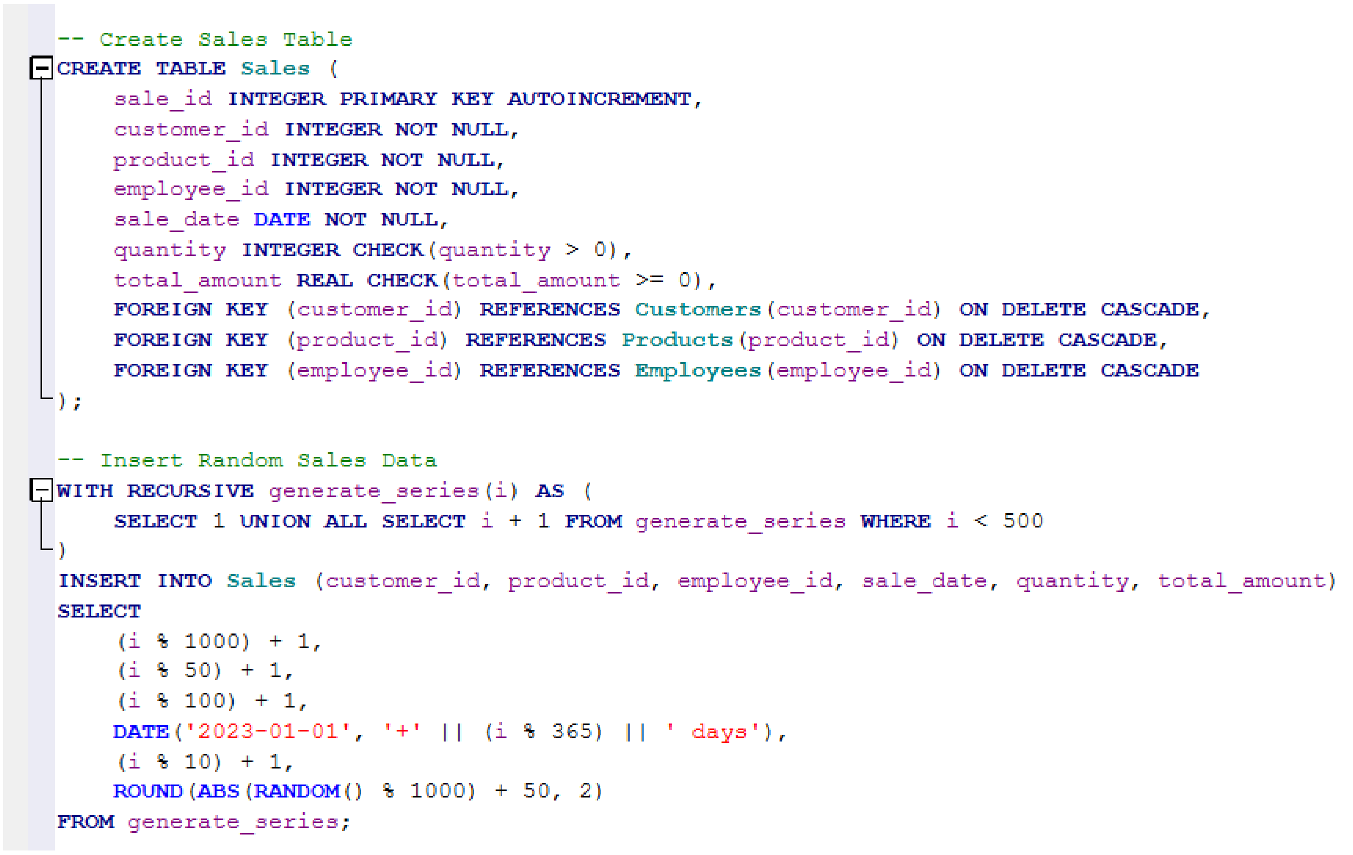
```sql
-- Create Interactions Table
CREATE TABLE Interactions (
    interaction_id INTEGER PRIMARY KEY AUTOINCREMENT,
    customer_id INTEGER NOT NULL,
    employee_id INTEGER NOT NULL,
    interaction_type TEXT CHECK(interaction_type IN ('Call', 'Email', 'Meeting')),
    interaction_date DATETIME NOT NULL,
    duration INTEGER CHECK(duration > 0),
    FOREIGN KEY (customer_id) REFERENCES Customers(customer_id) ON DELETE CASCADE,
    FOREIGN KEY (employee_id) REFERENCES Employees(employee_id) ON DELETE CASCADE
);

-- Insert Random Interactions Data
WITH RECURSIVE generate_series(i) AS (
    SELECT 1 UNION ALL SELECT i + 1 FROM generate_series WHERE i < 2000
)
INSERT INTO Interactions (customer_id, employee_id, interaction_type, interaction_date, duration)
SELECT
    (i % 1000) + 1,
    (i % 100) + 1,
    CASE (i % 3) WHEN 0 THEN 'Call' WHEN 1 THEN 'Email' ELSE 'Meeting' END,
    DATETIME('2023-01-01', '+' || (i % 365) || ' days', '+' || (i % 24) || ' hours', '+' || (i % 60) || ' minutes'),
    (i % 60) + 1
FROM generate_series;
```

Sales Table:

500 records were generated for sales transactions, including customer, product, employee references, sale dates, quantities, and total amounts.



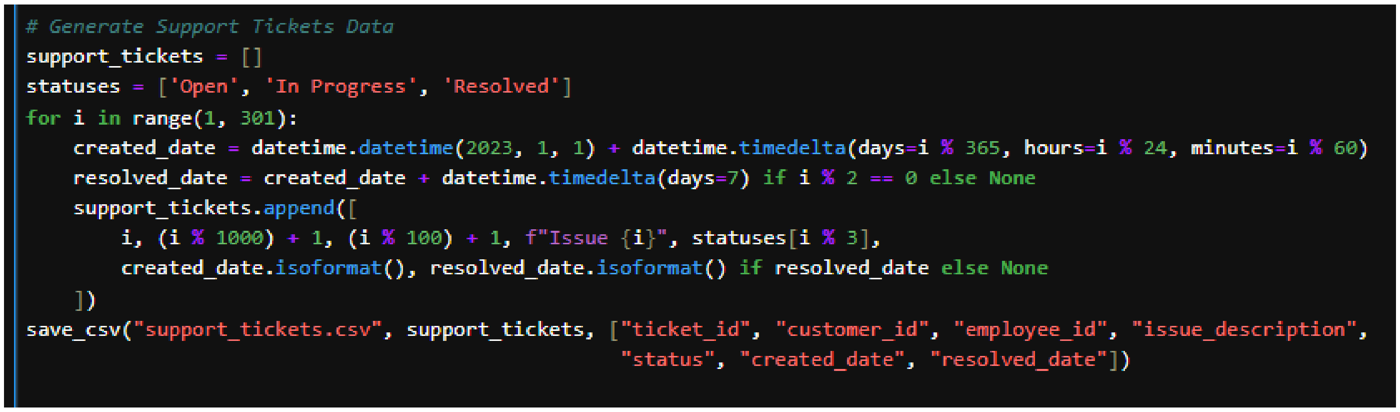
```python
# Generate Sales Data
sales = []
for i in range(1, 501):
    sales.append([
        i, (i % 1000) + 1, (i % 50) + 1, (i % 100) + 1,
        (datetime.date(2023, 1, 1) + datetime.timedelta(days=i % 365)).isoformat(),
        (i % 10) + 1, round(random.uniform(50, 1050), 2)
    ])
save_csv("sales.csv", sales, ["sale_id", "customer_id", "product_id", "employee_id", "sale_date", "quantity", "total_amount"])
```



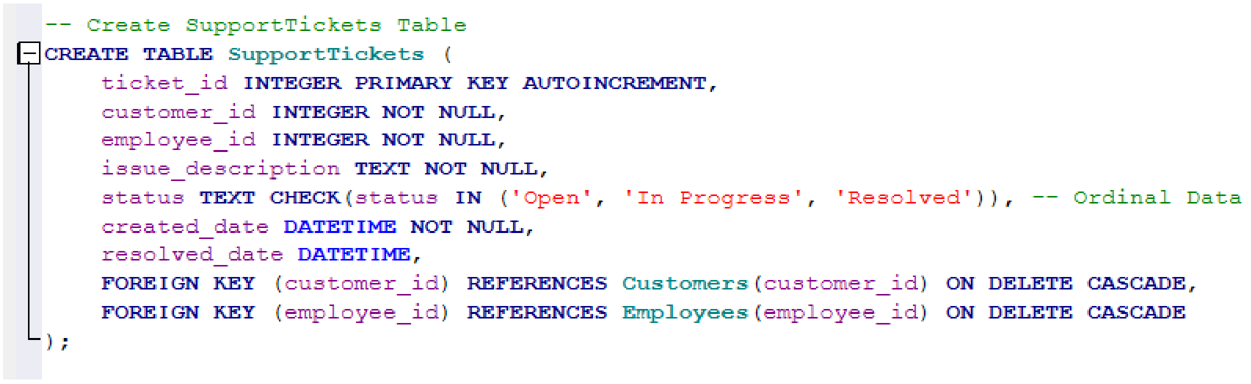
```sql
-- Create Sales Table
CREATE TABLE Sales (
    sale_id INTEGER PRIMARY KEY AUTOINCREMENT,
    customer_id INTEGER NOT NULL,
    product_id INTEGER NOT NULL,
    employee_id INTEGER NOT NULL,
    sale_date DATE NOT NULL,
    quantity INTEGER CHECK(quantity > 0),
    total_amount REAL CHECK(total_amount >= 0),
    FOREIGN KEY (customer_id) REFERENCES Customers(customer_id) ON DELETE CASCADE,
    FOREIGN KEY (product_id) REFERENCES Products(product_id) ON DELETE CASCADE,
    FOREIGN KEY (employee_id) REFERENCES Employees(employee_id) ON DELETE CASCADE
);

-- Insert Random Sales Data
WITH RECURSIVE generate_series(i) AS (
    SELECT 1 UNION ALL SELECT i + 1 FROM generate_series WHERE i < 500
)
INSERT INTO Sales (customer_id, product_id, employee_id, sale_date, quantity, total_amount)
SELECT
    (i % 1000) + 1,
    (i % 50) + 1,
    (i % 100) + 1,
    DATE('2023-01-01', '+' || (i % 365) || ' days'),
    (i % 10) + 1,
    ROUND(ABS(RANDOM() % 1000) + 50, 2)
FROM generate_series;
```

SupportTickets Table:

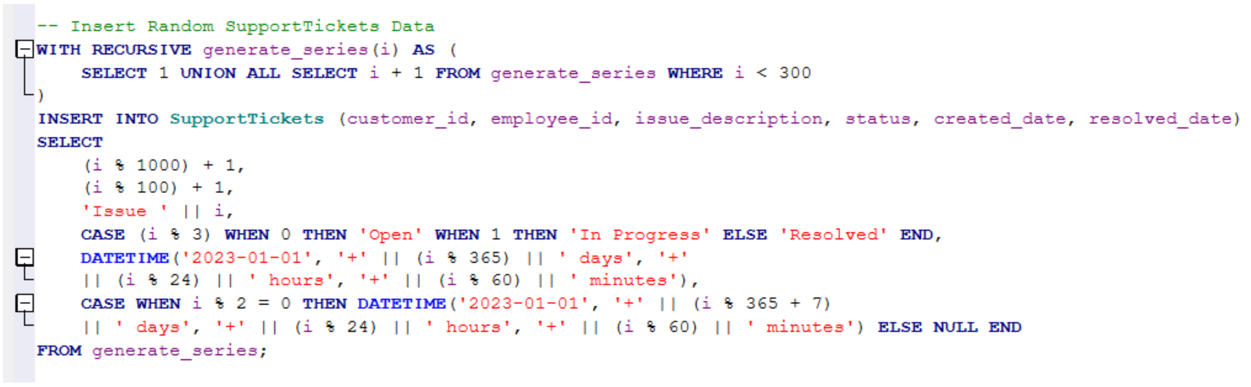
300 support tickets were created, linking customers and employees, including issue descriptions, statuses (Open, In Progress, Resolved), created dates, and resolved dates



```python
# Generate Support Tickets Data
support_tickets = []
statuses = ['Open', 'In Progress', 'Resolved']
for i in range(1, 301):
    created_date = datetime.datetime(2023, 1, 1) + datetime.timedelta(days=i % 365, hours=i % 24, minutes=i % 60)
    resolved_date = created_date + datetime.timedelta(days=7) if i % 2 == 0 else None
    support_tickets.append([
        i, (i % 1000) + 1, (i % 100) + 1, f"Issue {i}", statuses[i % 3],
        created_date.isoformat(), resolved_date.isoformat() if resolved_date else None
    ])
save_csv("support_tickets.csv", support_tickets, ["ticket_id", "customer_id", "employee_id", "issue_description",
                                                  "status", "created_date", "resolved_date"])
```



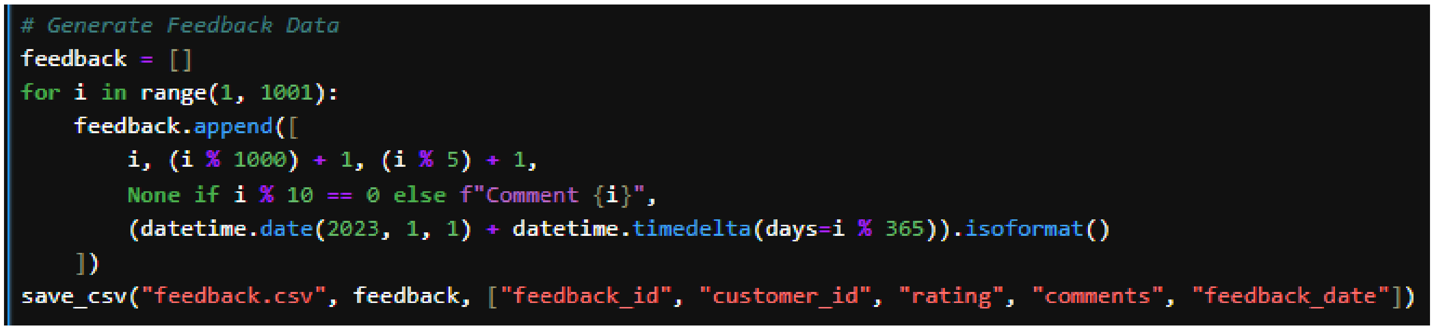
```sql
-- Create SupportTickets Table
CREATE TABLE SupportTickets (
    ticket_id INTEGER PRIMARY KEY AUTOINCREMENT,
    customer_id INTEGER NOT NULL,
    employee_id INTEGER NOT NULL,
    issue_description TEXT NOT NULL,
    status TEXT CHECK(status IN ('Open', 'In Progress', 'Resolved')), -- Ordinal Data
    created_date DATETIME NOT NULL,
    resolved_date DATETIME,
    FOREIGN KEY (customer_id) REFERENCES Customers(customer_id) ON DELETE CASCADE,
    FOREIGN KEY (employee_id) REFERENCES Employees(employee_id) ON DELETE CASCADE
);
```



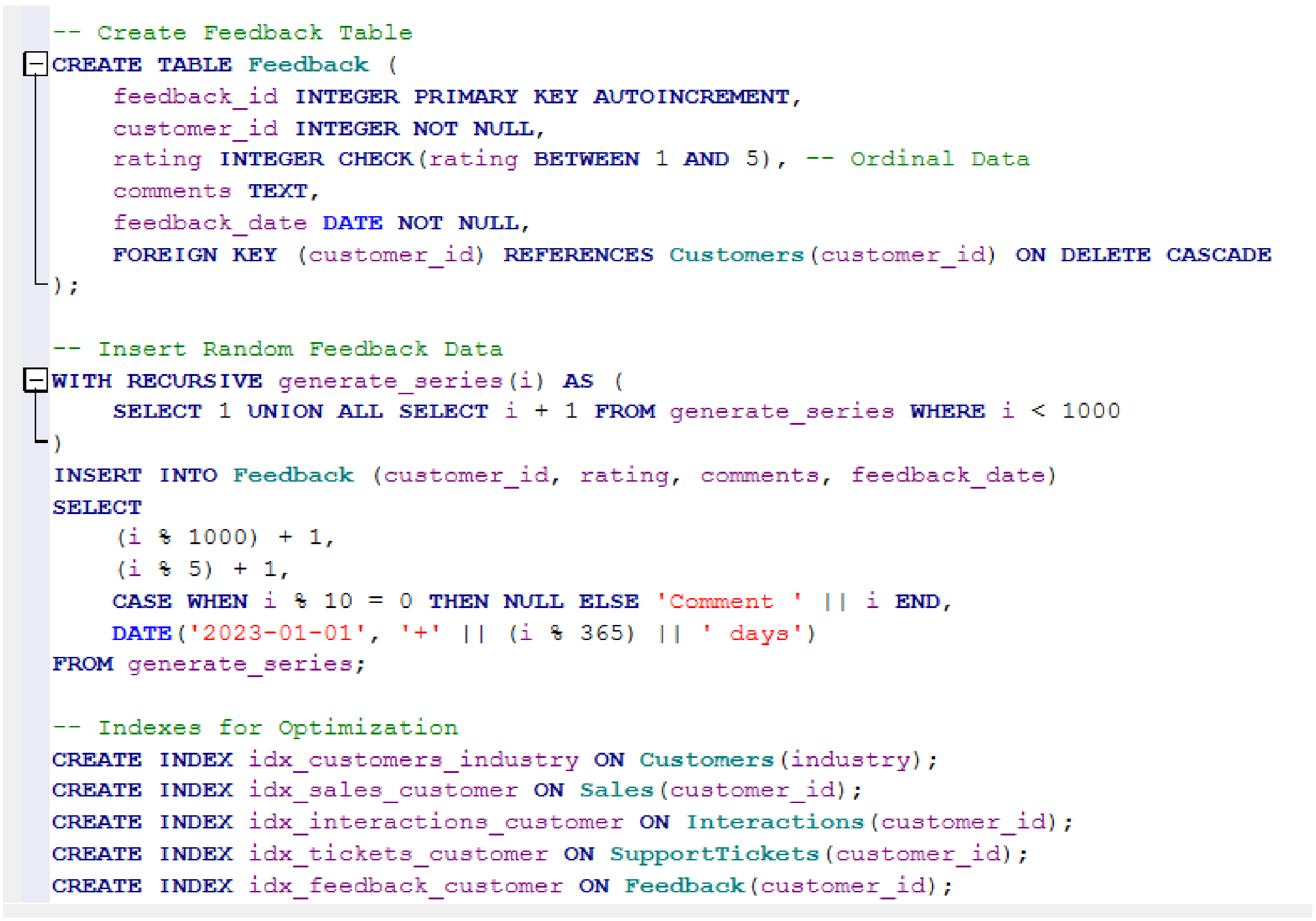
```sql
-- Insert Random SupportTickets Data
WITH RECURSIVE generate_series(i) AS (
    SELECT 1 UNION ALL SELECT i + 1 FROM generate_series WHERE i < 300
)
INSERT INTO SupportTickets (customer_id, employee_id, issue_description, status, created_date, resolved_date)
SELECT
    (i % 1000) + 1,
    (i % 100) + 1,
    'Issue ' || i,
    CASE (i % 3) WHEN 0 THEN 'Open' WHEN 1 THEN 'In Progress' ELSE 'Resolved' END,
    DATETIME('2023-01-01', '+' || (i % 365) || ' days', '+'
    || (i % 24) || ' hours', '+' || (i % 60) || ' minutes'),
    CASE WHEN i % 2 = 0 THEN DATETIME('2023-01-01', '+' || (i % 365 + 7)
    || ' days', '+' || (i % 24) || ' hours', '+' || (i % 60) || ' minutes') ELSE NULL END
FROM generate_series;
```

Feedback Table:

1000 feedback entries were generated, with customer feedback ratings (1-5), optional comments, and feedback dates.

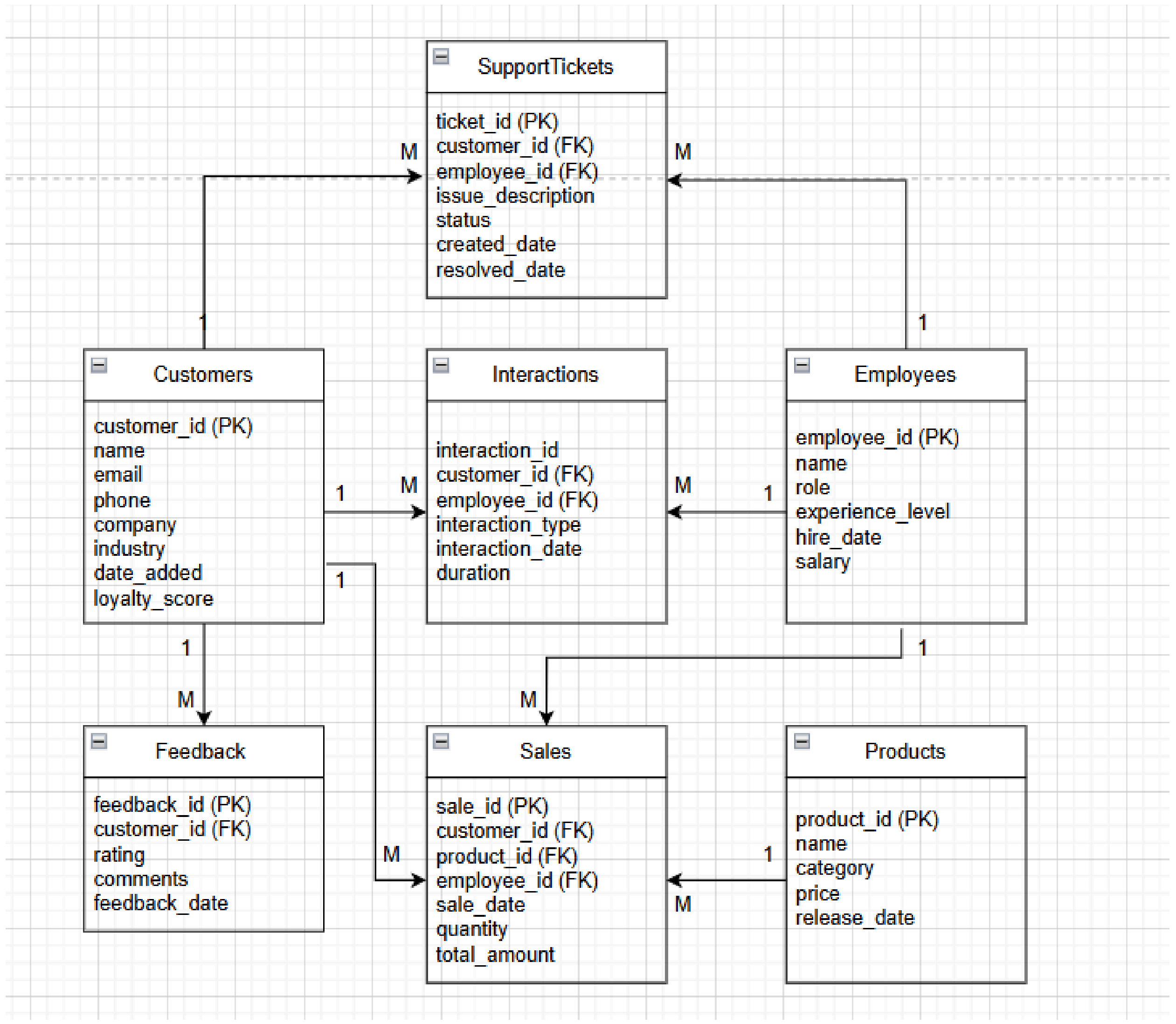


```python
# Generate Feedback Data
feedback = []
for i in range(1, 1001):
    feedback.append([
        i, (i % 1000) + 1, (i % 5) + 1,
        None if i % 10 == 0 else f"Comment {i}",
        (datetime.date(2023, 1, 1) + datetime.timedelta(days=i % 365)).isoformat()
    ])
save_csv("feedback.csv", feedback, ["feedback_id", "customer_id", "rating", "comments", "feedback_date"])
```



```sql
-- Create Feedback Table
CREATE TABLE Feedback (
    feedback_id INTEGER PRIMARY KEY AUTOINCREMENT,
    customer_id INTEGER NOT NULL,
    rating INTEGER CHECK(rating BETWEEN 1 AND 5), -- Ordinal Data
    comments TEXT,
    feedback_date DATE NOT NULL,
    FOREIGN KEY (customer_id) REFERENCES Customers(customer_id) ON DELETE CASCADE
);

-- Insert Random Feedback Data
WITH RECURSIVE generate_series(i) AS (
    SELECT 1 UNION ALL SELECT i + 1 FROM generate_series WHERE i < 1000
)
INSERT INTO Feedback (customer_id, rating, comments, feedback_date)
SELECT
    (i % 1000) + 1,
    (i % 5) + 1,
    CASE WHEN i % 10 = 0 THEN NULL ELSE 'Comment ' || i END,
    DATE('2023-01-01', '+' || (i % 365) || ' days')
FROM generate_series;

-- Indexes for Optimization
CREATE INDEX idx_customers_industry ON Customers(industry);
CREATE INDEX idx_sales_customer ON Sales(customer_id);
CREATE INDEX idx_interactions_customer ON Interactions(customer_id);
CREATE INDEX idx_tickets_customer ON SupportTickets(customer_id);
CREATE INDEX idx_feedback_customer ON Feedback(customer_id);
```

ERD DIAGRAM:



The decision to create separate tables for each entity is based on normalization principles, aimed at reducing data redundancy and improving data integrity:

- Customers Table: Contains essential customer information, ensuring that customer data is stored in a single location to avoid duplication and maintain consistency. A unique `customer_id` is used as a primary key for relationships with other tables.

- Employees Table: Employee-related information is stored separately to allow for flexible employee assignments to different roles, interactions, and sales.

- Products Table: Products are separated into their own table to allow easy updates and management of product information without affecting other entities like sales or interactions.

- Interactions Table: Captures the communication history between customers and employees. By keeping it separate, we can record interactions independently of sales or support tickets, providing a detailed view of customer engagement.

- Sales Table: Sales data needs to be distinct from customer interactions and support tickets. By linking sales to both customers and products, we ensure accurate tracking of transactions.

- SupportTickets Table: This table handles customer service issues separately to allow for detailed tracking of issues, statuses, and resolutions.

- Feedback Table: Feedback is stored separately, allowing for analysis of customer satisfaction and feedback without affecting the core customer or sales data.

Conclusion:

The database schema for the customer interaction and sales management system is designed with normalization principles in mind, ensuring that data is stored efficiently and consistently. The separation of concerns into different tables allows for easy maintenance and scalability. Ethical considerations related to data privacy and security are crucial, and the schema design takes these factors into account by implementing constraints and considering data retention and access control policies.