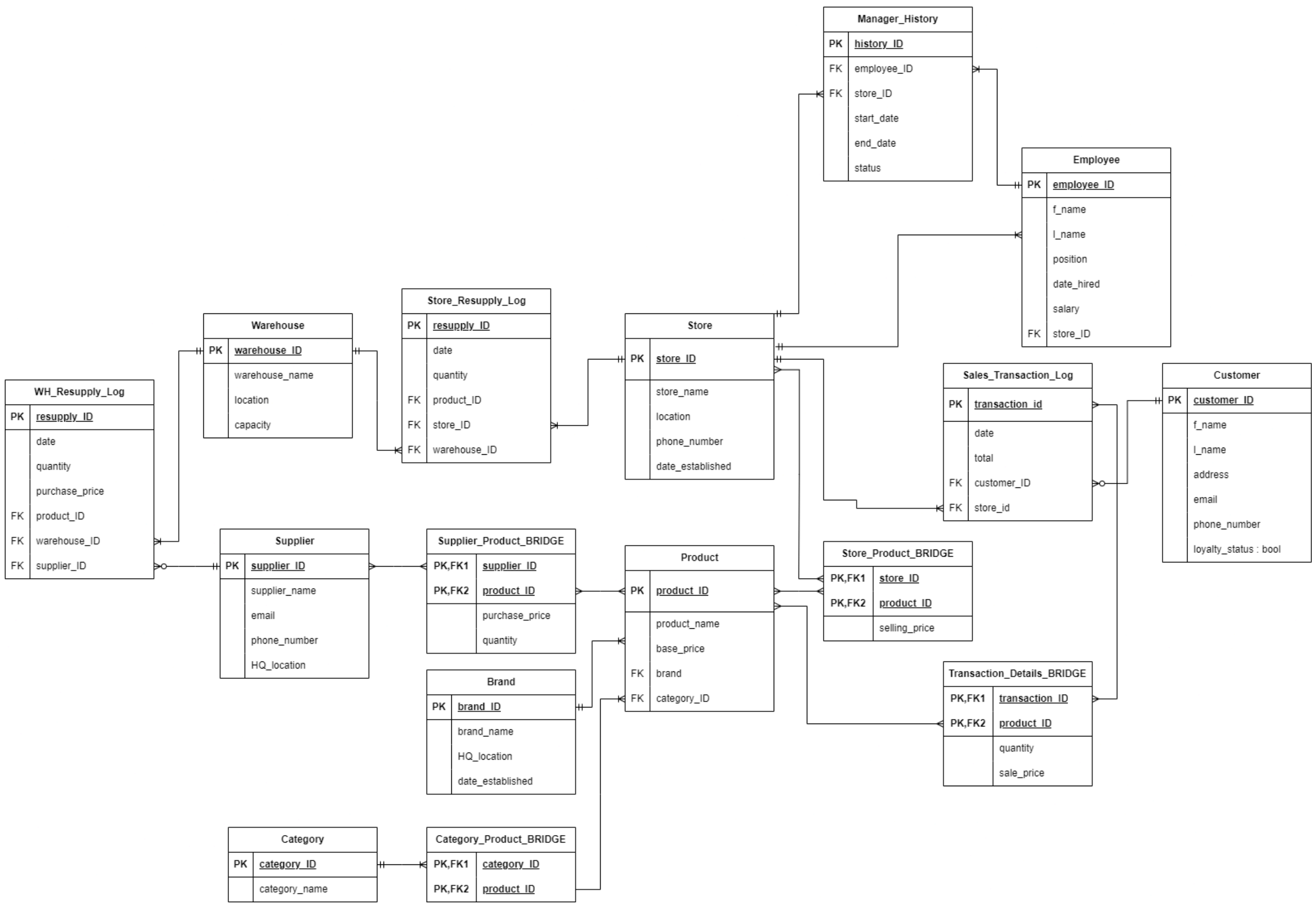
COSC 436

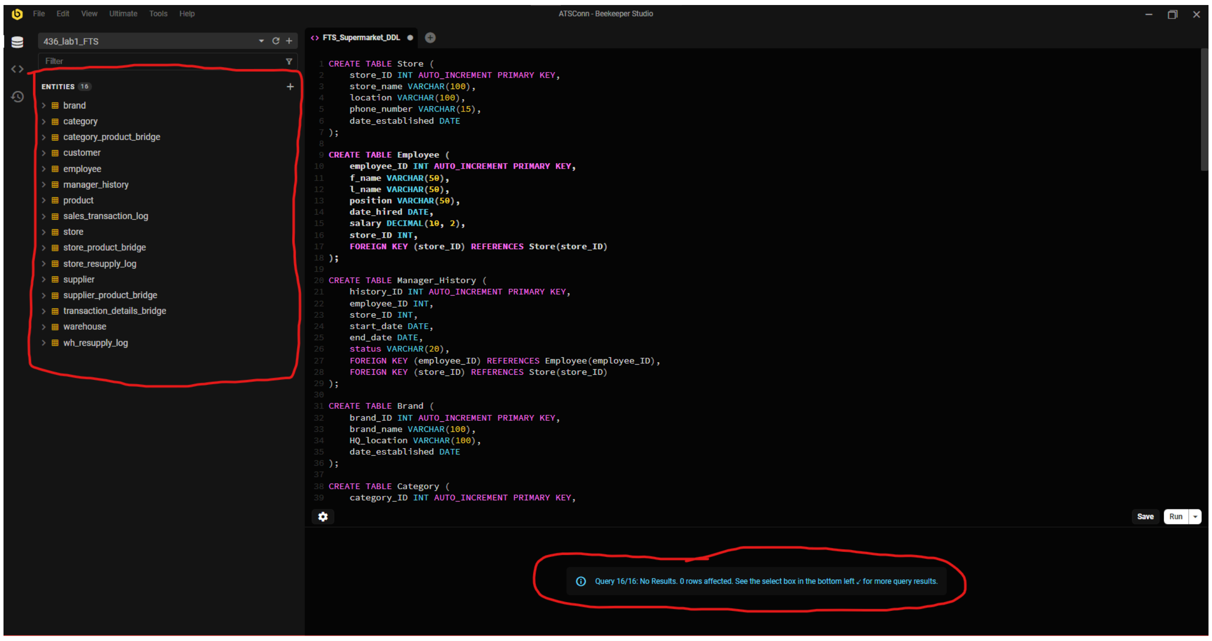
Lab 1 - FTS Supermarket

Jacob Rawlings - 300301869

ERD



Database Screenshots


File Edit View Ultimate Tools Help
ATSConn - Beekeeper Studio
436_lab1_FTS
Filter
FTS_Supermarket_DDL
Save
Run

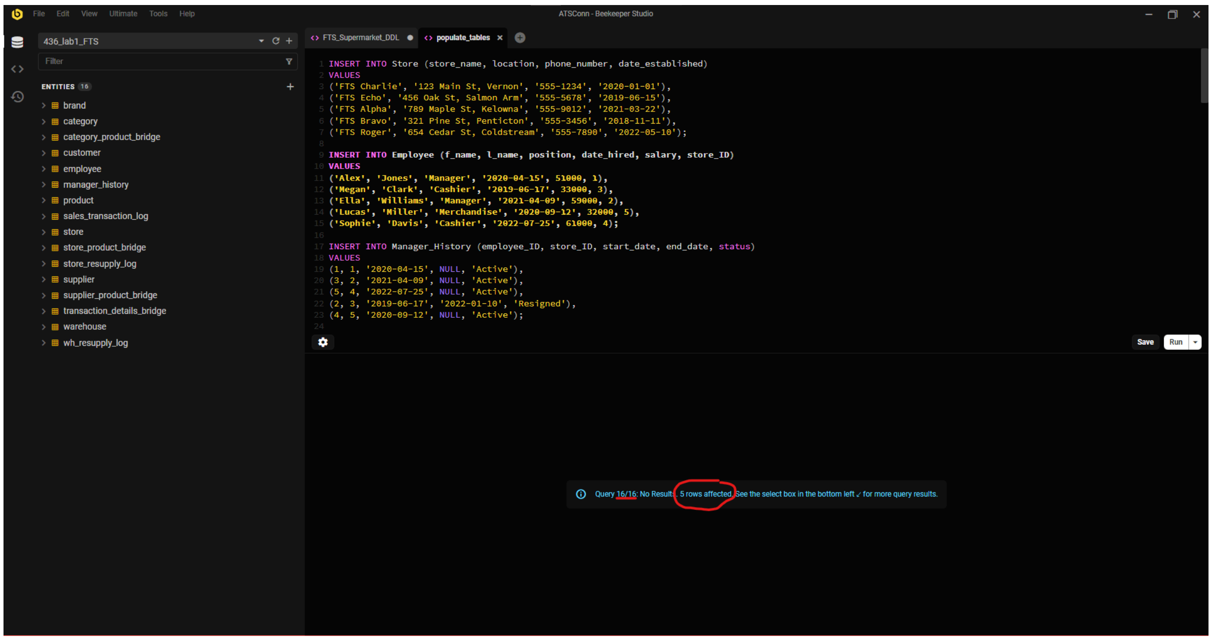
ENTITIES 16

- brand
- category
- category_product_bridge
- customer
- employee
- manager_history
- product
- sales_transaction_log
- store
- store_product_bridge
- store_resupply_log
- supplier
- supplier_product_bridge
- transaction_details_bridge
- warehouse
- wh_resupply_log

```sql
CREATE TABLE Store (
    store_ID INT AUTO_INCREMENT PRIMARY KEY,
    store_name VARCHAR(100),
    location VARCHAR(100),
    phone_number VARCHAR(15),
    date_established DATE
);

CREATE TABLE Employee (
    employee_ID INT AUTO_INCREMENT PRIMARY KEY,
    f_name VARCHAR(50),
    l_name VARCHAR(50),
    position VARCHAR(50),
    date_hired DATE,
    salary DECIMAL(10, 2),
    store_ID INT,
    FOREIGN KEY (store_ID) REFERENCES Store(store_ID)
);

CREATE TABLE Manager_History (
    history_ID INT AUTO_INCREMENT PRIMARY KEY,
    employee_ID INT,
    store_ID INT,
    start_date DATE,
    end_date DATE,
    status VARCHAR(20),
    FOREIGN KEY (employee_ID) REFERENCES Employee(employee_ID),
    FOREIGN KEY (store_ID) REFERENCES Store(store_ID)
);

CREATE TABLE Brand (
    brand_ID INT AUTO_INCREMENT PRIMARY KEY,
    brand_name VARCHAR(100),
    HQ_location VARCHAR(100),
    date_established DATE
);

CREATE TABLE Category (
    category_ID INT AUTO_INCREMENT PRIMARY KEY,
```

```
Query 16/16: No Results. 0 rows affected. See the select box in the bottom left ↙ for more query results.
```

Screenshot right after running DDL statements - showing the tables created on the left


File Edit View Ultimate Tools Help
ATSConn - Beekeeper Studio
436_lab1_FTS
Filter
FTS_Supermarket_DDL
populate_tables
Save
Run

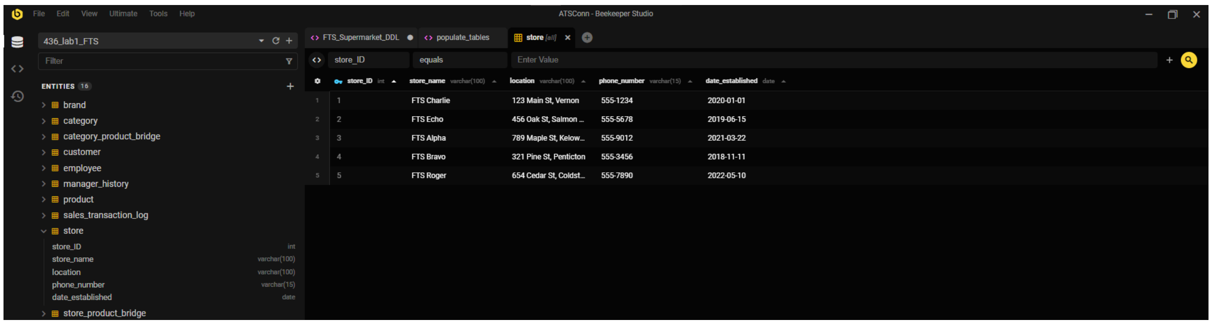
ENTITIES 16

- brand
- category
- category_product_bridge
- customer
- employee
- manager_history
- product
- sales_transaction_log
- store
- store_product_bridge
- store_resupply_log
- supplier
- supplier_product_bridge
- transaction_details_bridge
- warehouse
- wh_resupply_log

```sql
INSERT INTO Store (store_name, location, phone_number, date_established)
VALUES
('FTS Charlie', '123 Main St, Vernon', '555-1234', '2020-01-01'),
('FTS Echo', '456 Oak St, Salmon Arm', '555-5678', '2019-06-15'),
('FTS Alpha', '789 Maple St, Kelowna', '555-9012', '2021-03-22'),
('FTS Bravo', '321 Pine St, Penticton', '555-3456', '2018-11-11'),
('FTS Roger', '654 Cedar St, Coldstream', '555-7890', '2022-05-10');

INSERT INTO Employee (f_name, l_name, position, date_hired, salary, store_ID)
VALUES
('Alex', 'Jones', 'Manager', '2020-04-15', 51000, 1),
('Megan', 'Clark', 'Cashier', '2019-06-17', 33000, 3),
('Ella', 'Williams', 'Manager', '2021-04-09', 59000, 2),
('Lucas', 'Miller', 'Merchandise', '2020-09-12', 32000, 5),
('Sophie', 'Davis', 'Cashier', '2022-07-25', 61000, 4);

INSERT INTO Manager_History (employee_ID, store_ID, start_date, end_date, status)
VALUES
(1, 1, '2020-04-15', NULL, 'Active'),
(3, 2, '2021-04-09', NULL, 'Active'),
(5, 4, '2022-07-25', NULL, 'Active'),
(2, 3, '2019-06-17', '2022-01-10', 'Resigned'),
(4, 5, '2020-09-12', NULL, 'Active');
```

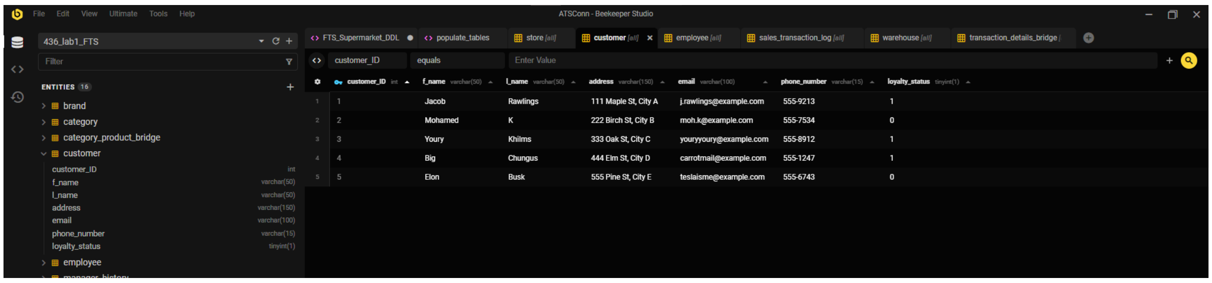
```
Query 16/16: No Result. 5 rows affected. See the select box in the bottom left ↙ for more query results.
```

Screenshot after running query to populate tables. Showing 16/16 run (1 per table) and 5 rows affected (per table).



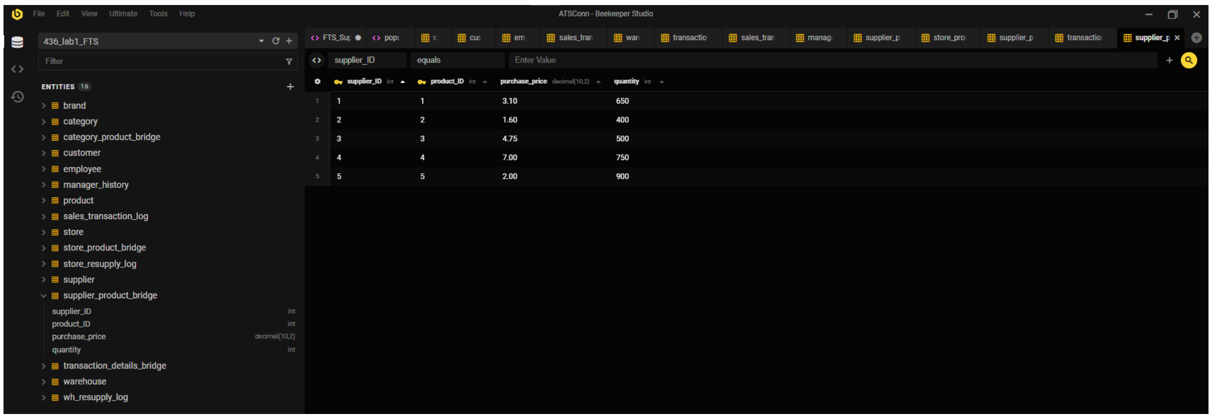
	store_ID int	store_name varchar(100)	location varchar(100)	phone_number varchar(15)	date_established date
1	1	FTS Charlie	123 Main St, Vernon	555-1234	2020-01-01
2	2	FTS Echo	456 Oak St, Salmon ...	555-5678	2019-06-15
3	3	FTS Alpha	789 Maple St, Kelow...	555-9012	2021-03-22
4	4	FTS Bravo	321 Pine St, Penticton	555-3456	2018-11-11
5	5	FTS Roger	654 Cedar St, Coldst...	555-7890	2022-05-10

Example data from the "store" table. As can be seen, the entries match the insert statement shown above.



	customer_ID int	f_name varchar(50)	l_name varchar(50)	address varchar(150)	email varchar(100)	phone_number varchar(15)	loyalty_status tinyint(1)
1	1	Jacob	Rawlings	111 Maple St, City A	j.rawlings@example.com	555-9213	1
2	2	Mohamed	K	222 Birch St, City B	moh.k@example.com	555-7534	0
3	3	Youry	Khilms	333 Oak St, City C	youryyoury@example.com	555-8912	1
4	4	Big	Chungus	444 Elm St, City D	carrotmail@example.com	555-1247	1
5	5	Elon	Busk	555 Pine St, City E	teslaisme@example.com	555-6743	0

Another slice of data from the "customer" table.



	supplier_ID int	product_ID int	purchase_price decimal(10,2)	quantity int
1	1	1	3.10	650
2	2	2	1.60	400
3	3	3	4.75	500
4	4	4	7.00	750
5	5	5	2.00	900

Bridge table example from the supplier-product bridge table.

Questions

1. This database should be designed as a highly normalized database (if it's not the case, you may have done something wrong). Explain the following benefits of normalization using examples in this database:

- *reduced redundancy*
- *improved data integrity/consistency*
- *easy to maintain*

Normalization eliminates duplicate data in the DB. This directly counteracts <u>redundancy</u>, and as a separate benefit saves storage space. An example in the

FTS database is the “Brand” table. Products belong to a specific brand, and instead of storing the brand name over and over again in each entry of the “Product” table, I created the “Brand” entity so that we only need to store each brand once across the database. This also relates to data integrity and consistency, because having data stored in one place, and one place only, reduces the likelihood that data anomalies occur. Lastly, normalization makes a database easier to maintain because data is separated into organized and specifically focused tables. This is done through relationships and constraints. For example, if a new supplier were to be added to the system, we would simply add it to the “supplier_product_bridge” table. If the DB was not normalized, we would most likely have to update multiple tables and rows (anything relating to supplier).

2. Suppose a customer is using a self-service machine to check out. What database queries are likely to be invoked? Describe the activities and give a sample SQL statement related to the activity. (e.g., Customer inputs their phone number to the machine, the check-out software queries the customer info: SELECT cid, cfirstname, clastname, cpoints from customers WHERE cphone = '250123123';)

A query that may likely need to be invoked would be something that adds a product to the “transaction_details_log” after a customer makes a purchase (or scans an item, depending on how we would like to track the data). Example of the SQL:

```
INSERT INTO Transaction_Details_BRIDGE (transaction_ID, product_ID,
quantity, sale_price)
VALUES (101, 12345, 1, 3.50);
```

Of course, in a legitimate implementation the values would be placeholder variables whose values would be determined by the check-out till, but for this example hard coded example values are used.

3. Write down two other scenarios that the database may be used to support the business's daily operation. And what are the SQL statements that are likely being used?

Scanning a product at a till:

```
SELECT p.product_name, sp.selling_price
FROM Product p
JOIN Store_Product_BRIDGE sp ON p.product_ID = sp.product_ID
WHERE sp.store_ID = 1 AND p.product_ID = 12345;
```

The warehouse has been resupplied, and a new entry is added to the “WH_resupply_log”:
INSERT INTO WH_Resupply_Log (date, quantity, purchase_price, product_ID, warehouse_ID, supplier_ID)
VALUES ('2023-09-01', 500, 2.80, 101, 1, 3);

4. How does the database design avoid queries with large JOIN s during its daily operation?

This is done in the FTS database again made possible by normalization. For example, instead of having a single massive table with all the product information, transaction details, customer details etc. It is instead broken up to smaller tables (Customer, Product, Sales_Transaction_log, etc.) This makes it so that JOINS are needed less often, and when they ARE needed, they are only performed on smaller tables that have relation to eachother.

5. Is the database suitable for data analysis? Why or why not?

This could really be answered either way, but I am going to say it is <u>not</u> suitable for data analysis <u>*YET*</u>. However, it is set up so that potentially in the future it would be ready for analysis. The ways that it <u>IS</u> setup for analysis is that the structure is fairly well organized, and has clearly defined relationships between entities. It also contains lots of necessary information for analysis, like tracking suppliers, warehousing, management changes, and other business details that are relevant. It also contains foreign key restraints in the DDL, which is vital for data analysis.

With all this said, the only reason I say its <u>NOT</u> ready for analysis, is because it has a lack of historical data, and testing has not yet been performed on complex query performance. Once the database (in theory) has more historical information regarding the business, and once more queries have been ran and tested, with performance metrics being measured, then I believe the database would be ready for data analysis.