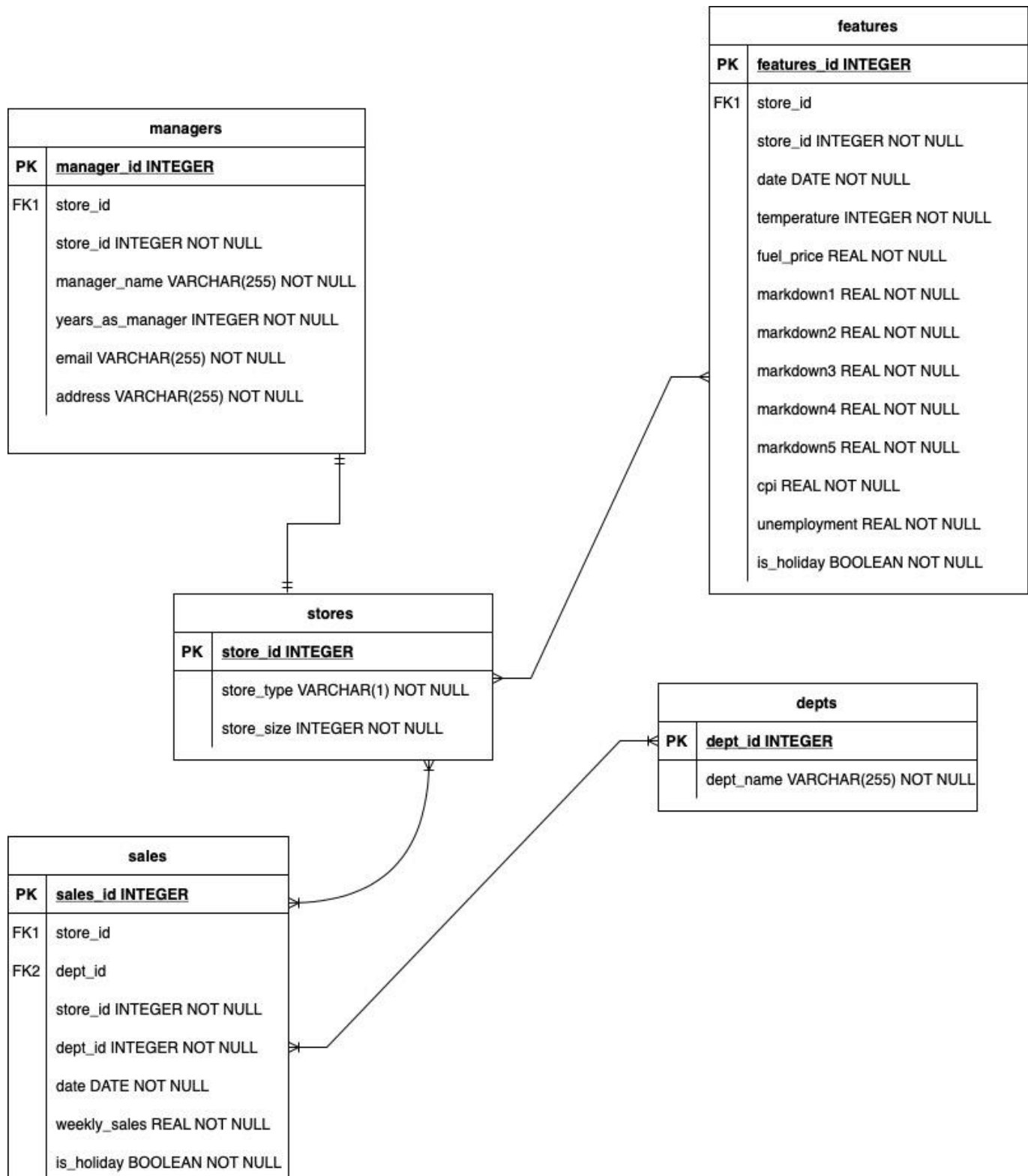


i) The entity relationship diagram



ii) Discussion of the design choices for the database.

To normalize the database, I have used the third normal form (3NF). This form was efficient because it helped clean up the data and remove possible repetition from the tables obtained. In the Sales table, `sales_id` is the Primary Key, and it contains attributes such as `store_id` and `dept_id`. The same attributes have been used as Primary Keys to create other tables; the Store Table and Dept Table. The Store Table, which uses `store_id` as the Primary Key, has `store_type` and `store_size` as attributes. The Dept Table, on the other hand, uses `dept_id` as Primary Key with `dept_name` as the only attribute. Among the three tables, namely; Sales Table, Store Table, Dept Table there are no transitive functional dependencies. The Sales Table has been divided to create two more tables; Store Table and Dept Table.

iii) The explanation of the column names.

a. Store Table

- i. `store_id`: This column is the primary key for the table, it is unique for each store and is used to identify each store in the table.
- ii. `store_type`: This column stores the type of store, such as A, B, or C. It's not unique and it's also specified as NOT NULL and CHECK constraint of possible values being 'A', 'B' and 'C' to ensure the data quality.
- iii. `store_size`: This column stores the size of the store, in square footage. It's also specified as NOT NULL, because the size of the store is an important attribute of the store that shouldn't be null.
- iv. `UNIQUE (store_type, store_size)`: It's a unique constraint on the combination of 'store_type' and 'store_size' columns, it ensures that the `store_type` and `store_size` are unique for each store. This allows for efficient queries by store type and size, since the data is guaranteed to be unique across these columns.

b. Features table

- i. `features_id`: This column is the primary key for the table, it is unique for each feature and is used to identify each feature in the table.
- ii. `store_id`: This column stores the store identifier, it is a foreign key referencing the '`store_id`' column in the `stores` table, this allows the data in the '`features`' table to be linked back to the specific store.
- iii. `date`: This column stores the date on which the features were recorded.
- iv. `temperature`: This column stores the temperature on the date the features were recorded.
- v. `fuel_price`: This column stores the fuel price on the date the features were recorded.
- vi. `markdown1`, `markdown2`, `markdown3`, `markdown4`, `markdown5`: These columns store the price reductions or markdowns in the store, it is a common practice in retail to reduce prices in order to stimulate sales.
- vii. `cpi`: This column stores the Consumer Price Index on the date the features were recorded.
- viii. `unemployment`: This column stores the unemployment rate on the date the features were recorded.
- ix. `is_holiday`: This column stores a flag indicating whether the date the features were recorded is a holiday or not.

c. Dept table

- i. `dept_id`: This column is the primary key for the table, it is unique for each department and is used to identify each department in the table.
- ii. `dept_name`: This column stores the name of the department, it is not unique and can contain duplicate values.

d. Sales table

- i. `sales_id`: This column is the primary key for the table, it is unique for each sale and is used to identify each sale in the table.
- ii. `store_id`: This column stores the store identifier, it is a foreign key referencing the '`store_id`' column in the `stores` table, this allows the data in the '`sales`' table to be linked back to the specific store.
- iii. `dept_id`: This column stores the department identifier, it is a foreign key referencing the '`dept_id`' column in the `depts` table, this allows the data in the '`sales`' table to be linked back to the specific department.
- iv. `date`: This column stores the date on which the sales were recorded.
- v. `weekly_sales`: This column stores the weekly sales for the store and department on the date the sales were recorded.
- vi. `is_holiday`: This column stores a flag indicating whether the date the sales were recorded is a holiday or not.

e. Managers table

- i. `manager_id`: This column is the primary key for the table, it is unique for each manager and is used to identify each manager in the table.
- ii. `store_id`: This column stores the store identifier, it is a foreign key referencing the 'store_id' column in the `stores` table, this allows the data in the 'managers' table to be linked back to the specific store.
- iii. `manager_name`: This column stores the name of the manager for the store.
- iv. `years_as_manager`: This column stores the number of years the manager has been in their current position.
- v. `email`: This column stores the manager's email address.
- vi. `address`: This column stores the manager's contact address.

iv) The SQL schema for all the tables.

```
CREATE TABLE stores (  
    store_id INTEGER PRIMARY KEY,  
    store_type VARCHAR(1) NOT NULL CHECK (store_type in ('A','B','C')),  
    store_size INTEGER NOT NULL,  
    UNIQUE (store_type,store_size)  
);  
  
CREATE TABLE features (  
    features_id INTEGER PRIMARY KEY,  
    store_id INTEGER NOT NULL,  
    date DATE NOT NULL,  
    temperature INTEGER NOT NULL,  
    fuel_price REAL NOT NULL,  
    markdown1 REAL NOT NULL,  
    markdown2 REAL NOT NULL,  
    markdown3 REAL NOT NULL,  
    markdown4 REAL NOT NULL,  
    markdown5 REAL NOT NULL,
```

```

        cpi REAL NOT NULL,
        unemployment REAL NOT NULL,
        is_holiday BOOLEAN NOT NULL,
        FOREIGN KEY (store_id) REFERENCES stores (store_id)
    );

CREATE TABLE depts (
    dept_id INTEGER PRIMARY KEY,
    dept_name VARCHAR(255) NOT NULL
);

CREATE TABLE sales (
    sales_id INTEGER PRIMARY KEY,
    store_id INTEGER NOT NULL,
    dept_id INTEGER NOT NULL,
    date DATE NOT NULL,
    weekly_sales REAL NOT NULL,
    is_holiday BOOLEAN NOT NULL,
    FOREIGN KEY (store_id) REFERENCES stores (store_id),
    FOREIGN KEY (dept_id) REFERENCES depts (dept_id)
);

CREATE TABLE managers (
    manager_id INTEGER PRIMARY KEY,
    store_id INTEGER NOT NULL,
    manager_name VARCHAR(255) NOT NULL,
    years_as_manager INTEGER NOT NULL,
    email VARCHAR(255) NOT NULL,
    address VARCHAR(255) NOT NULL,
    FOREIGN KEY (store_id) REFERENCES stores (store_id)
);

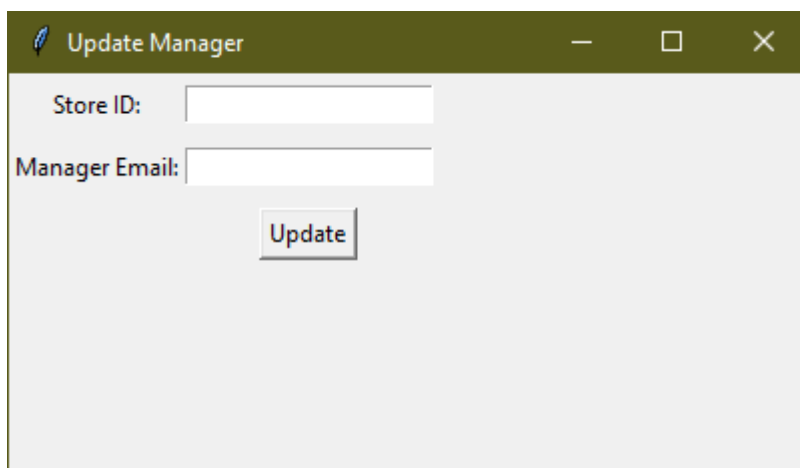
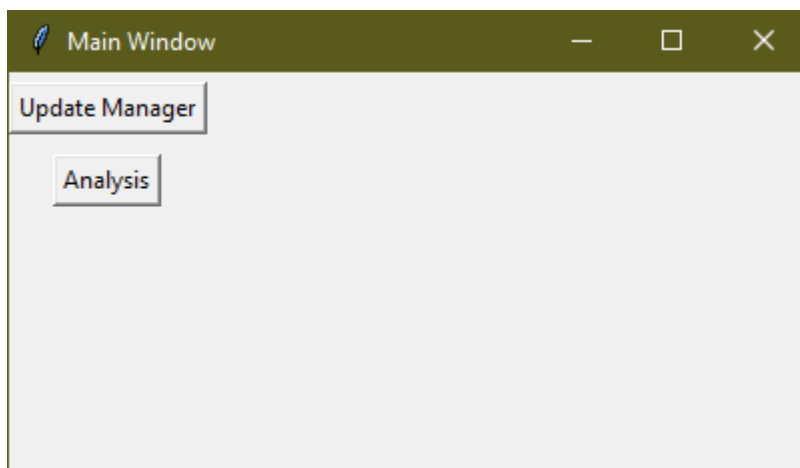
```

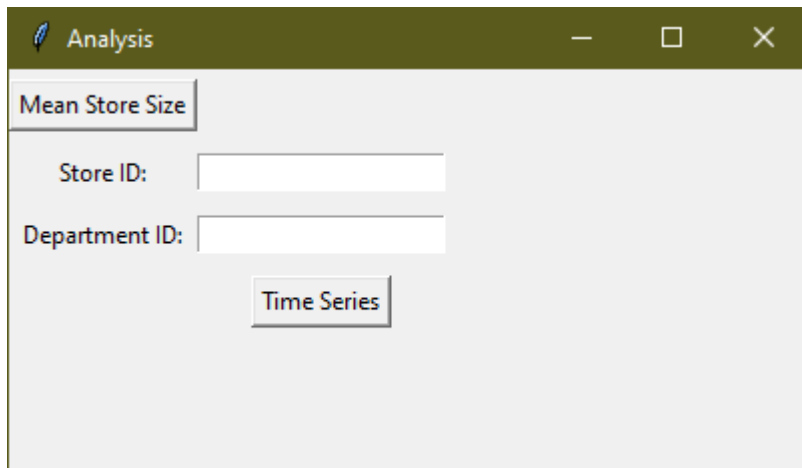
v) A discussion of any data cleaning required, if there are for example missing values.

`df.dropna()` method Was used to clean the data as it was bringing errors when trying to insert the data into the database.

The `df.dropna()` method removes all the rows that contain missing values, effectively removing any data where values are missing. It's important to consider that this method can result in a loss of data and should be used with caution. The use of this method may not be ideal if the dataset is large and the number of missing values is relatively small or if the missing values are informative. It might not be suitable if the data is collected from a survey or self-reported data with a large number of missing values.

vi) The screen-shots showing the working of the GUIs.





vii) A discussion about how AWS can be used in the project.

You can leverage a variety of services from AWS (Amazon Web Services) in a Tkinter project to increase the scalability, dependability, and security of your application.

Hosting the application on an EC2 (Elastic Compute Cloud) instance, a virtual server in the AWS cloud that may execute your application, is one option to use AWS in a Tkinter project. This gives you a way to operate the application in a safe, isolated environment and makes it simple to extend the program to handle more traffic.

S3 (Simple Storage Service), which may be used to store and retrieve massive amounts of data, such as user-uploaded files, logs, and backups, is another service that might be helpful for a Tkinter project. Because of this, you can stop worrying about running out of storage or dealing with data loss by offloading data storage to a highly scalable and reliable service.

If you wish to store data in a relational database, AWS also offers alternative services like RDS (Relational Database Service), which enables you to run a fully managed relational database in the cloud. AWS Lambda is a different service that enables you to run code without installing or managing servers, which might be handy for handling events or performing background chores.

Elasticache, DynamoDB, and other similar services are also available for caching and NoSQL databases, respectively.

Finally, in addition to these services, AWS also offers a wide range of others, such as AWS AppSync for creating GraphQL APIs and AWS Cognito for user authentication.

It's important to note that not all of these services must be used in a single project; instead, you can pick and select the ones that make the most sense for your use case. Remember that some services might come at an additional cost.

AWS Products and Monthly Cost

- RDS: A relational database service that is fully managed and supports a variety of database engines, including PostgreSQL, MySQL, Oracle, and more. The price will vary based on the read replica count, storage size, and engine of choice.
- Large volumes of data can be stored and retrieved using the S3 simple storage service. The price will change based on how much data is stored and how many queries are made.

Conclusion

All things considered, using AWS for a Tkinter project may offer scalability, dependability, and security.

The cost of using AWS RDS and S3 will vary depending on how these services are used specifically.

However, with careful planning, the Tkinter application may be moved to AWS with little disturbance.