

Chapter 21:

Final Data Loading

Project

Public Safety: Bureau of Fire Prevention -
Active Violation Orders

NYC OpenData

Table of Contents

Loading the

Dataset.....
.....2

Loading the

Data.....
.....6

Normalizing the

Data.....
.....10

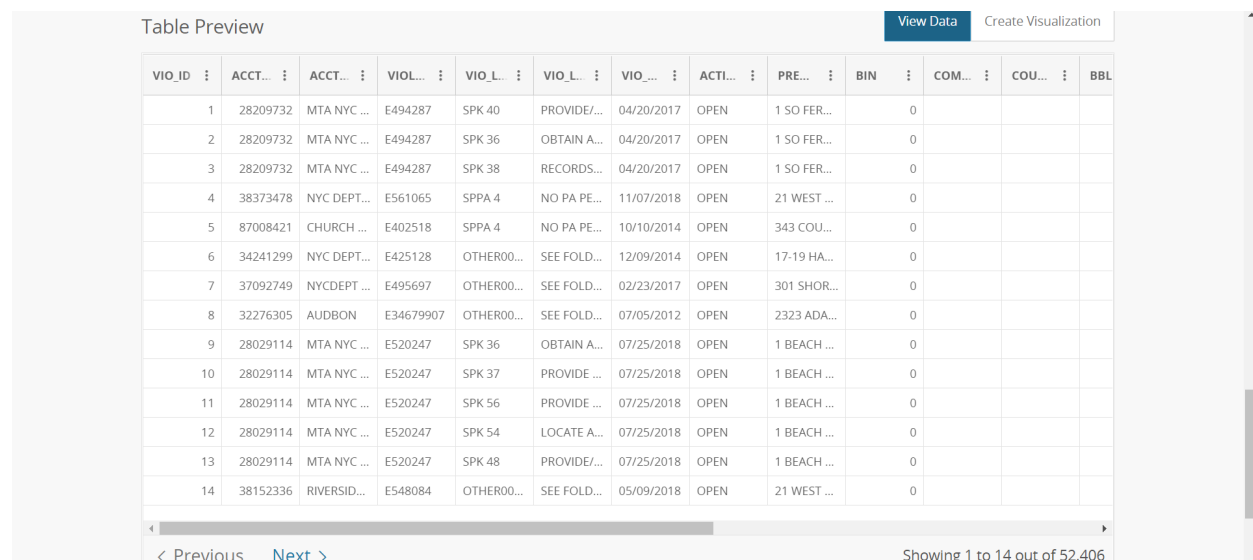
Cleaning the

Data.....
.....12

Part 1: Loading the Dataset

First a dataset chosen must be given as a single table, later on in the project the tables and data will be normalized so the following are crucial. In order for the data to be normalized properly the dataset must allow a many to one or many to many relationship. Also, to subdue any confusion, if the dataset chosen results in over 1 million rows you may delete rows.

First the dataset needs to be understood, without the understanding of the data difficulties will arise throughout the later portions of the project. As stated by the NYC OpenData website the Bureau of Fire Prevention dataset, when buildings go through NYC codes the FDNY provides a series of information given quarterly. If a violation takes place throughout the inspection process, the Violation ID, Building Identification Number, Premises Address, Violation Law Number, Violation Law Description, Violation Date, Account Number, and Account Owner are given. Other information such as the Status of Violation, Building Block Lot, Community Board, and Council District, etc. are provided throughout this dataset. The dataset can be found [here: NYC Bureau of Fire Prevention - Active Violation Orders.](#)



VIO_ID	ACCT...	ACCT...	VIOL...	VIO_L...	VIO_L...	VIO_...	ACTI...	PRE...	BIN	COM...	COU...	BBL
1	28209732	MTA NYC ...	E494287	SPK 40	PROVIDE/...	04/20/2017	OPEN	1 SO FER...	0			
2	28209732	MTA NYC ...	E494287	SPK 36	OBTAIN A...	04/20/2017	OPEN	1 SO FER...	0			
3	28209732	MTA NYC ...	E494287	SPK 38	RECORDS...	04/20/2017	OPEN	1 SO FER...	0			
4	38373478	NYC DEPT...	E561065	SPPA 4	NO PA PE...	11/07/2018	OPEN	21 WEST ...	0			
5	87008421	CHURCH ...	E402518	SPPA 4	NO PA PE...	10/10/2014	OPEN	343 COU...	0			
6	34241299	NYC DEPT...	E425128	OTHER00...	SEE FOLD...	12/09/2014	OPEN	17-19 HA...	0			
7	37092749	NYCDEPT ...	E495697	OTHER00...	SEE FOLD...	02/23/2017	OPEN	301 SHOR...	0			
8	32276305	AUDBON	E34679907	OTHER00...	SEE FOLD...	07/05/2012	OPEN	2323 ADA...	0			
9	28029114	MTA NYC ...	E520247	SPK 36	OBTAIN A...	07/25/2018	OPEN	1 BEACH ...	0			
10	28029114	MTA NYC ...	E520247	SPK 37	PROVIDE ...	07/25/2018	OPEN	1 BEACH ...	0			
11	28029114	MTA NYC ...	E520247	SPK 56	PROVIDE ...	07/25/2018	OPEN	1 BEACH ...	0			
12	28029114	MTA NYC ...	E520247	SPK 54	LOCATE A...	07/25/2018	OPEN	1 BEACH ...	0			
13	28029114	MTA NYC ...	E520247	SPK 48	PROVIDE/...	07/25/2018	OPEN	1 BEACH ...	0			
14	38152336	RIVERSID...	E548084	OTHER00...	SEE FOLD...	05/09/2018	OPEN	21 WEST ...	0			

Figure 1: NYC Bureau of Fire Prevention - Active Violation Order Dataset Preview

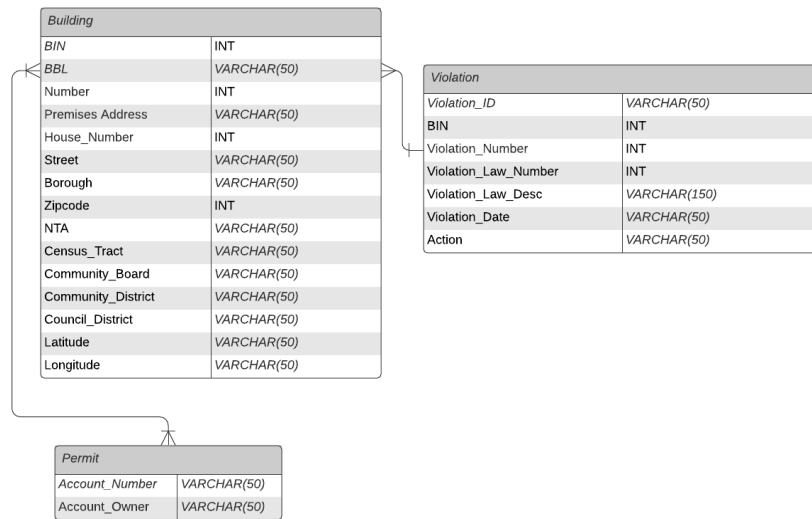
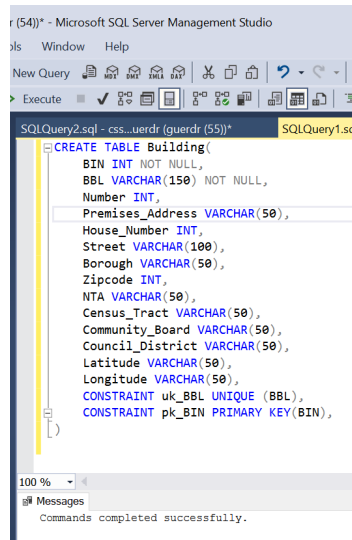


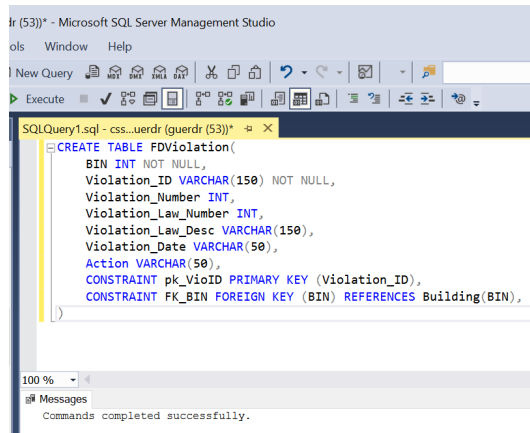
Figure 2: NYC Bureau of Fire Prevention - Active Violation Order ERD

Once the dataset chosen is understood and a 3NF or BCNF Entity Relationship Diagram (ERD) is created, the create table statements from the ERD are inputted into SQL Server as shown below.

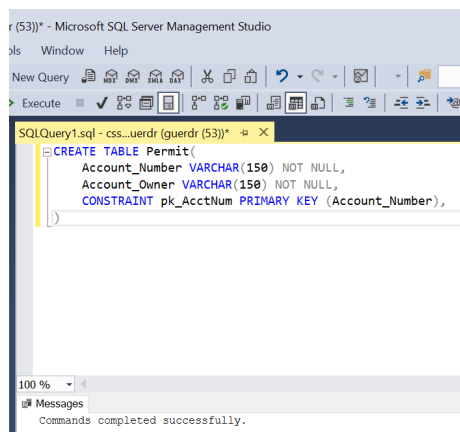
SQL Text:



```
CREATE TABLE Building(  
    BIN INT NOT NULL,  
    BBL VARCHAR(150) NOT NULL,  
    Number INT,  
    Premises_Address VARCHAR(50),  
    House_Number INT,  
    Street VARCHAR(100),  
    Borough VARCHAR(50),  
    Zipcode INT,  
    NTA VARCHAR(50),  
    Census_Tract VARCHAR(50),  
    Community_Board VARCHAR(50),  
    Council_District VARCHAR(50),  
    Latitude VARCHAR(50),  
    Longitude VARCHAR(50),  
    CONSTRAINT uk_BBL UNIQUE (BBL),  
    CONSTRAINT pk_BIN PRIMARY KEY(BIN),  
)
```



```
CREATE TABLE FdViolation(
    BIN INT NOT NULL,
    Violation_ID VARCHAR(150) NOT NULL,
    Violation_Number INT,
    Violation_Law_Number INT,
    Violation_Law_Desc VARCHAR(150),
    Violation_Date VARCHAR(50),
    Action VARCHAR(50),
    CONSTRAINT pk_VioID PRIMARY KEY (Violation_ID),
    CONSTRAINT FK_BIN FOREIGN KEY (BIN) REFERENCES Building(BIN),
)
```



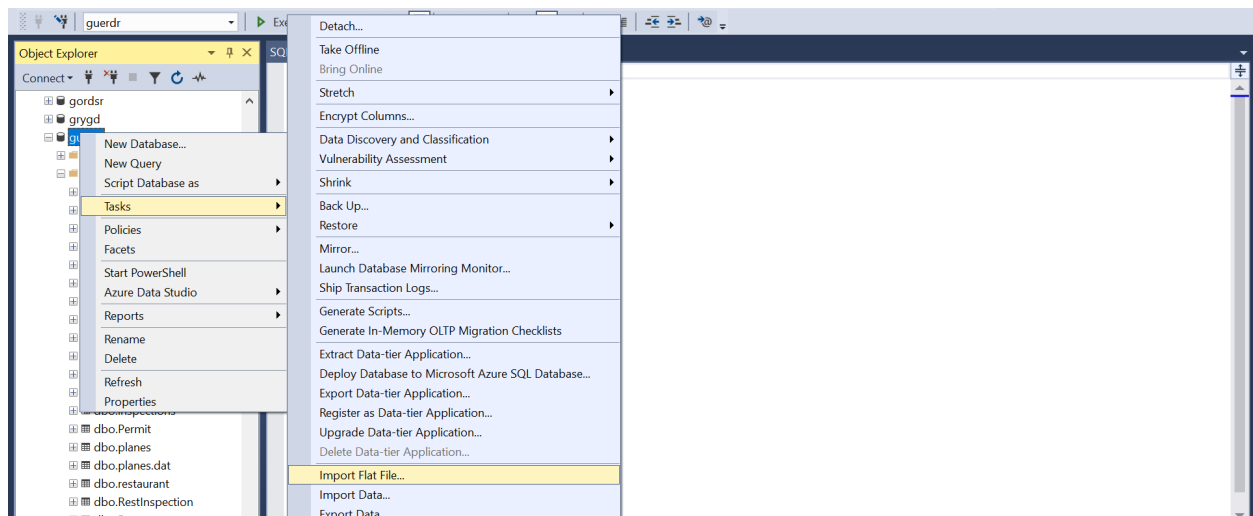
```
CREATE TABLE Permit(
    Account_Number VARCHAR(150) NOT NULL,
    Account_Owner VARCHAR(150) NOT NULL,
    CONSTRAINT pk_AcctNum PRIMARY KEY (Account_Number),
)
```

Part 2: Loading the Data

Once the tables are created and have attributes along with their proper constraints we can now load the data into our database so that when the queries are created and run you will get results that align with the information given in the dataset. Here is how to load a CSV file or flat file containing all that information to be stored into the database.

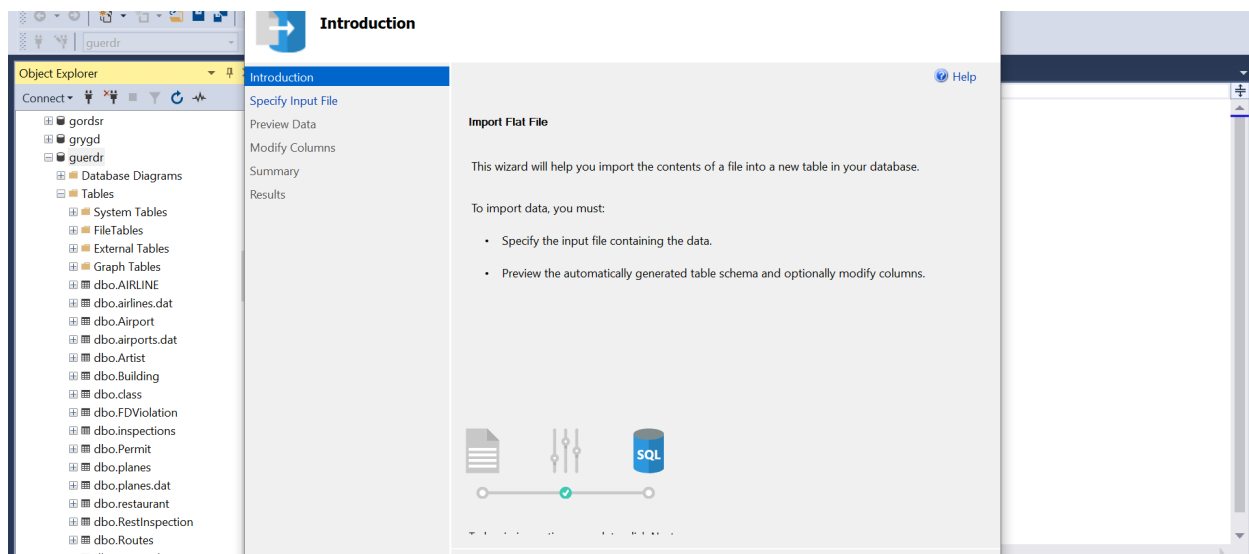
Step 1:

Right-Click onto your database and hover over tasks, then you will click on Import Flat File



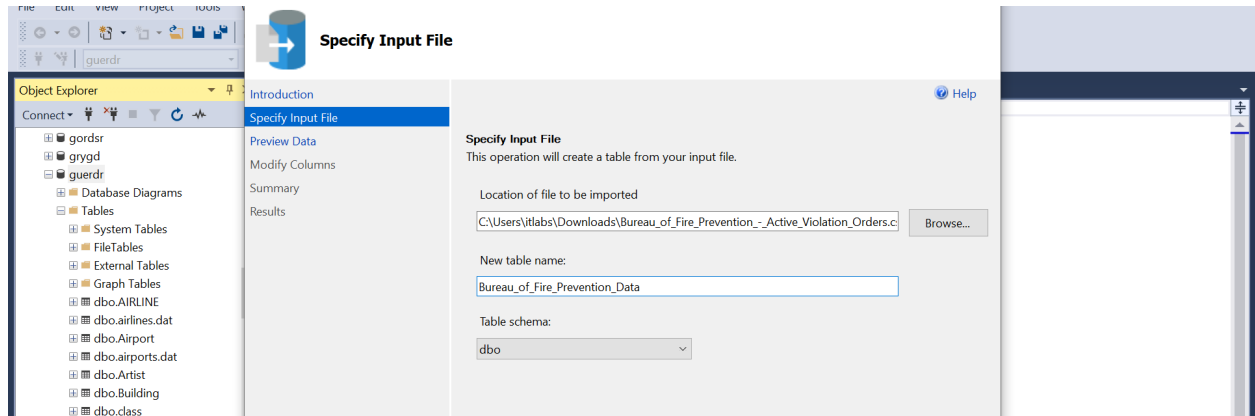
Step 2:

This screen will pop up, click the Next button to continue



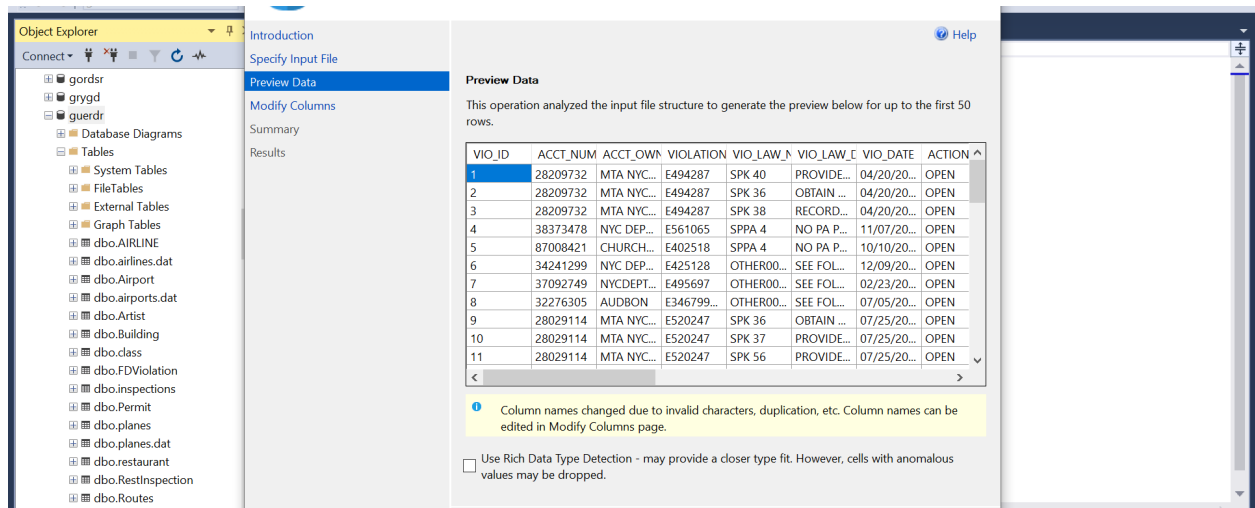
Step 3:

Then click on the Browse button to click on the CSV Flat File. You can change the new table name that best fits your Data Loading Project, once completed click Next.



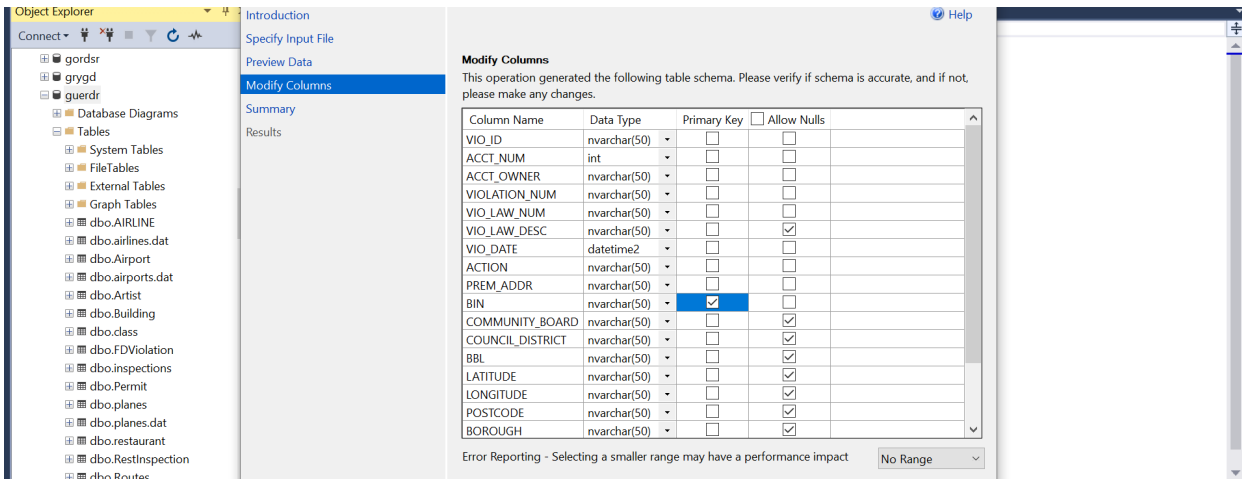
Step 4:

Click preview to review your work to ensure all columns are present, when done click Next.



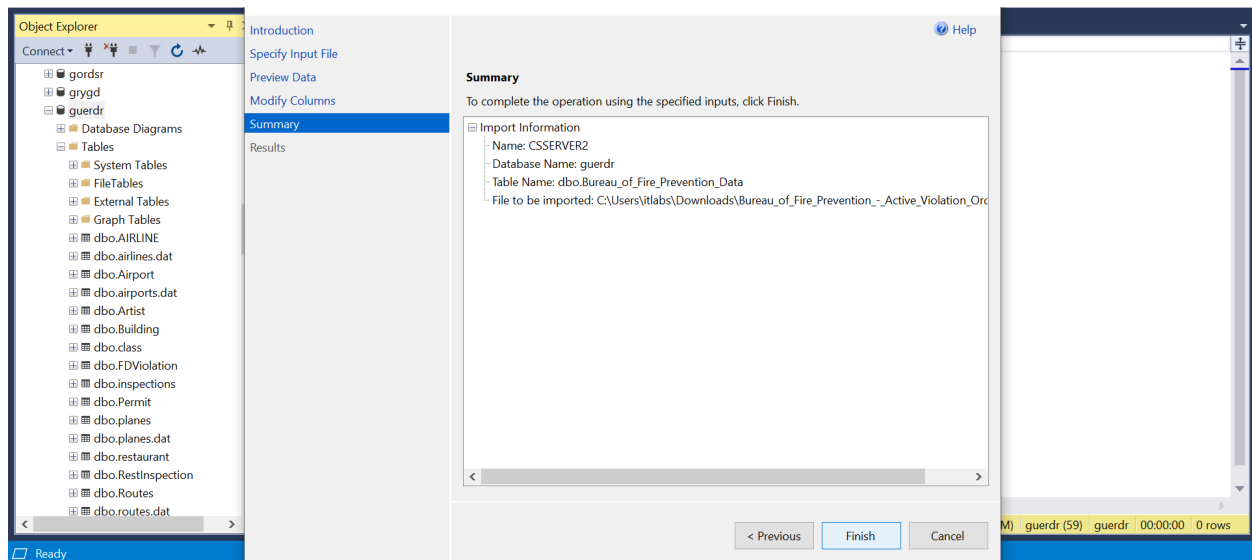
Step 5:

Then select the Primary Key of your new table and check off Allow Nulls to appropriate columns, when done click Next.



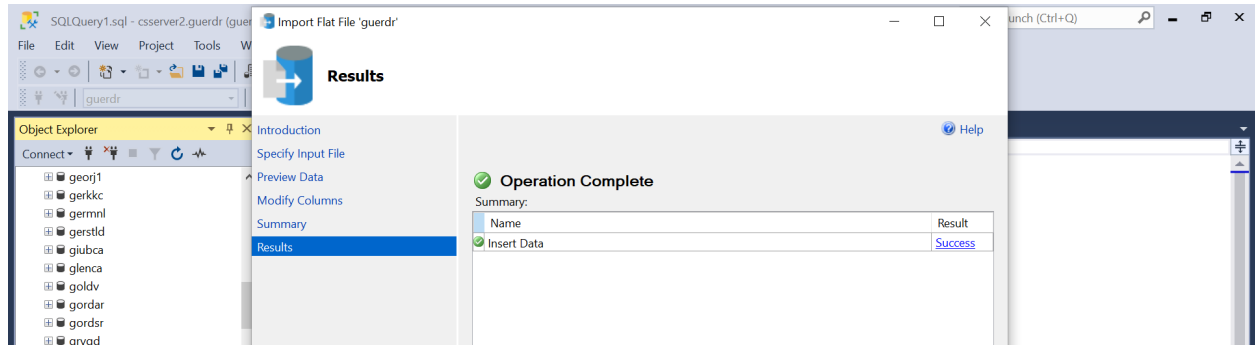
Step 6:

This screen should pop up, click on the Finish button.



Step 7:

There should be no errors when completed. If you receive an error message you must go back to step 5 to edit the data types, constraints, or nulls of the attribute you're having difficulty with. Here is the completed upload of the flat file.



It's important to keep, upkeep, and maintain updated information within your database to prevent inconsistencies such as duplication within the information, misspelling information, or even missing information within the table that you want to access. For example, to create and access a query that allows you to figure out which building had a violation code "ABC123" on July 23rd of 2017. A way to access that is to have a create statement to acquire such information. The query runs successfully, the date comes up fine yet the violation code doesn't match the specific tuple of information accessed. Without uploading the information properly it can alter the results and data in a table that can give faulty information. This can be changed by alter table statements and queries that allow the tables to reset their unique, primary, and foreign keys, delete duplicates, and alter/edit any misspellings in the table itself. This creates a better outcome and an easier way to the information given without having to continue adjusting the tables.

Part 3: Normalizing the Data

Data normalization is necessary for a database to reduce data inconsistencies. Three main reasons why data normalization is needed is to reduce duplicate data, to simplify within the queries, and to minimize data modification problems that can be prevented. Rules to normalize data include the following:

1. First Normal Form (1NF)	Entity Type is 1NF when there's no duplicates within the data.
2. Second Normal Form (2NF)	Entity Type is 2NF when it's in 1NF and all columns/attributes are dependent on the table's primary key attribute.
3. Third Normal Form (3NF)	Entity Type is 3NF when it's in 2NF and there is no transitive functional dependencies

For the chosen dataset, [Bureau of Fire Prevention Active Violation Orders](#) we're given the attributes and determine the best possible key attributes. Functional dependencies within the database are BIN which determines the Building Identification Number, Premises Address, Permit Account Number, etc. The Violation ID would return the Violation Number, Violation Law Number, Violation Law Description, Violation Date, and Action which gives us the status of the violation itself. Another functional dependency is Account Number which would give us the Account Owner attribute. Here we see that a functional dependency is when a chosen tuple can determine another tuple in the dataset.

List of Functional Dependencies:

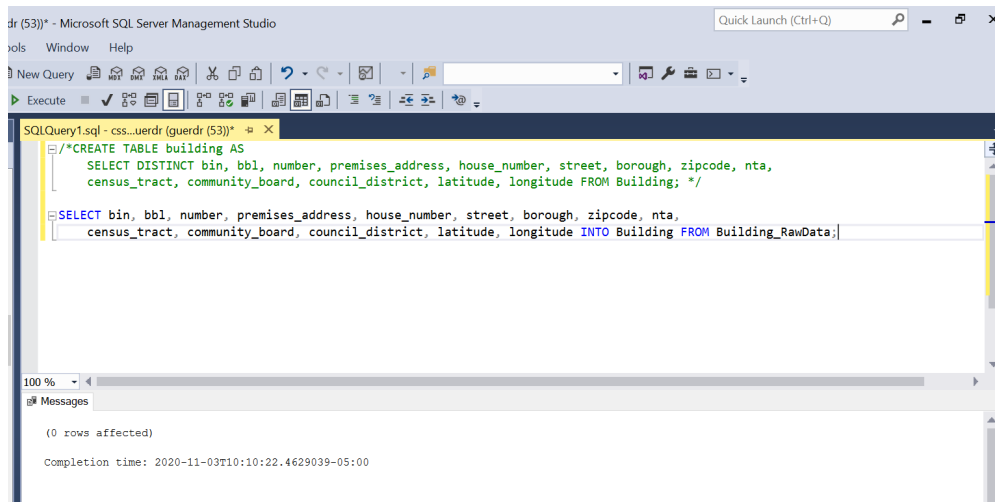
BIN → Building Identification Number, Premises Address, Permit Account Number

Violation ID → Violation Number, Violation Law Number, Violation Law Description, Violation Date, Action

Account Number → Account Number, Account Owner

Step 1: Creation of New Tables

The creation of new tables will be done through “SELECT...INTO...FROM” statements so that the new tables contain data from our raw data tables for Building, FDViolation, and Permit. Below are the “SELECT...INTO” statements for each new table. Afterwards, the constraints for each table will be given.

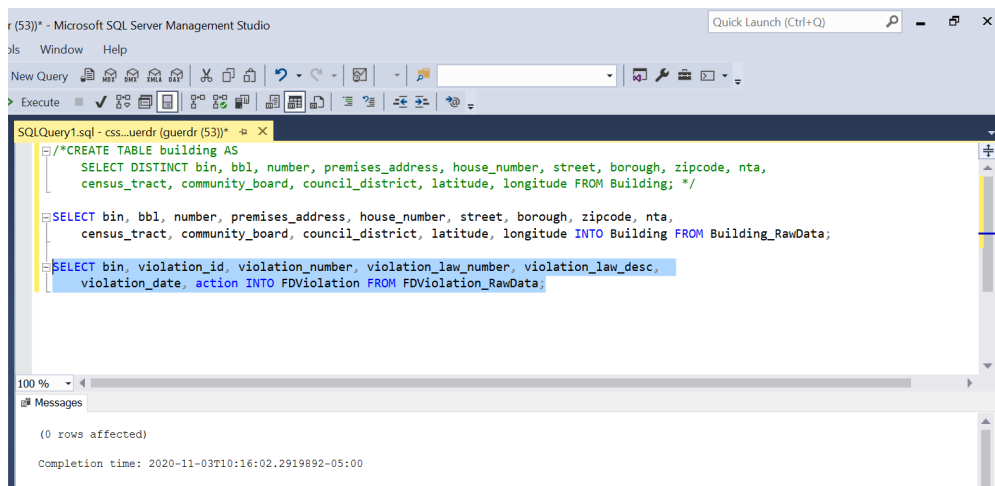


The screenshot shows the Microsoft SQL Server Management Studio interface. The main window displays a SQL query in the 'SQLQuery1.sql' file. The query consists of two parts: a CREATE TABLE statement followed by a SELECT DISTINCT statement, and a SELECT statement with an INTO clause. The messages pane at the bottom indicates that 0 rows were affected and provides the completion time.

```
SQLQuery1.sql - css...uerdr (guerdr (53)) *  
/*CREATE TABLE building AS  
SELECT DISTINCT bin, bbl, number, premises_address, house_number, street, borough, zipcode, nta,  
census_tract, community_board, council_district, latitude, longitude FROM Building; */  
  
SELECT bin, bbl, number, premises_address, house_number, street, borough, zipcode, nta,  
census_tract, community_board, council_district, latitude, longitude INTO Building FROM Building_RawData;
```

100 %
Messages
(0 rows affected)
Completion time: 2020-11-03T10:10:22.4629039-05:00

SELECT bin, bbl, number, premises_address, house_number, street, borough, zipcode, nta,
census_tract, community_board, council_district, latitude, longitude INTO Building
FROM Building_RawData;

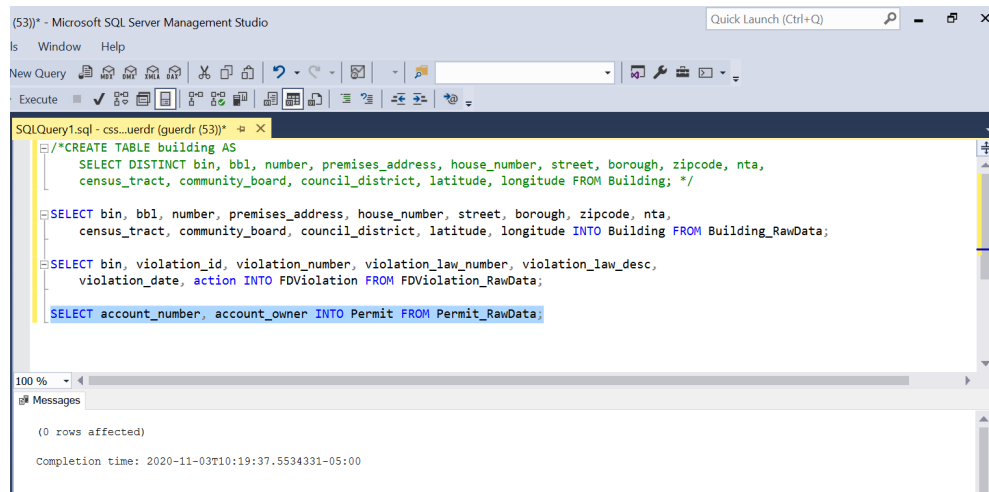


The screenshot shows the Microsoft SQL Server Management Studio interface. The main window displays a SQL query in the 'SQLQuery1.sql' file. The query consists of two parts: a CREATE TABLE statement followed by a SELECT DISTINCT statement, and a SELECT statement with an INTO clause. The messages pane at the bottom indicates that 0 rows were affected and provides the completion time.

```
SQLQuery1.sql - css...uerdr (guerdr (53)) *  
/*CREATE TABLE building AS  
SELECT DISTINCT bin, bbl, number, premises_address, house_number, street, borough, zipcode, nta,  
census_tract, community_board, council_district, latitude, longitude FROM Building; */  
  
SELECT bin, bbl, number, premises_address, house_number, street, borough, zipcode, nta,  
census_tract, community_board, council_district, latitude, longitude INTO Building FROM Building_RawData;  
  
SELECT bin, violation_id, violation_number, violation_law_number, violation_law_desc,  
violation_date, action INTO FDViolation FROM FDViolation_RawData;
```

100 %
Messages
(0 rows affected)
Completion time: 2020-11-03T10:16:02.2919892-05:00

SELECT bin, violation_id, violation_number, violation_law_number, violation_law_desc,
violation_date, action INTO FDViolation FROM FDViolation_RawData;



SELECT account_number, account_owner INTO Permit FROM Permit_RawData;

Step 2: Table Constraints

To match the tables we created above these constraints would be added to alter table statements to include the following key attributes:

ALTER TABLE Building ADD CONSTRAINT pk_binbuilding PRIMARY KEY(BIN);

ALTER TABLE FDViolation ADD CONSTRAINT pk_fdviolationid PRIMARY KEY(Violation_ID);

ALTER TABLE Permit ADD CONSTRAINT pk_permitactnum PRIMARY KEY(Account_Number);

Part 4: Cleaning the Data

Cleaning your data in your database is important because the information you obtain from your results must be accurate to be considered efficient and effective. Without the transformation of your data, the tables you've created will continue to show results with duplicates, null values will be skewed, unnecessary indexes will appear within your created tables, and you will not be able to add proper primary and foreign keys to your tables in your dataset. Other possibilities regarding inefficient cleaning results in invalid records within your

tables that will not be able to be used for data analysis and data that won't fit within the schema and will affect your importing of the information.

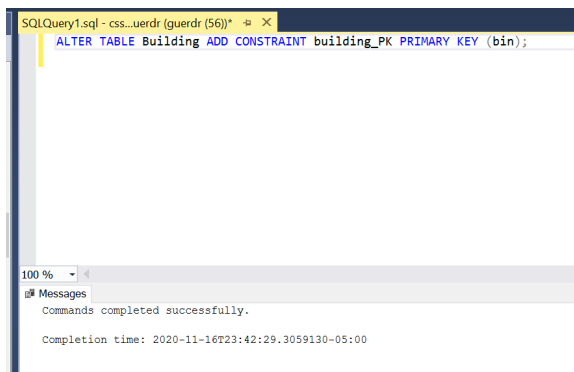
Let's say we have results that end up with inconsistencies and invalid results within our dataset. How are we able to properly clean and appropriately adjust our tables to give an accurate as possible outcome? Well, an approach would be to convert data values into their appropriate data type, delete and get rid of any duplicates shown within the tables, attempt to create statements within your SQL Server to run as fast as possible with results that give you the answer you're looking for.

For example, we're looking for a score within the building violation database that gives us tuples that are within a failing grade. One of the tuples gives us a failing grade however, the other does not and only shows NULL. We can create a SELECT COUNT statement that shows us how many tuples have a duplicate failing grade and the same BIN (Building Identification Number). However, this SELECT COUNT statement is using a SELF JOIN so it takes a lot of time to run and give us the results we want. If an index is created for those fields and then we rerun the same SELECT COUNT query we end up getting the results in a faster and more efficient approach.

How to Clean the Bureau of Fire Prevention Dataset Building Table:

Step 1: Add constraints to your table

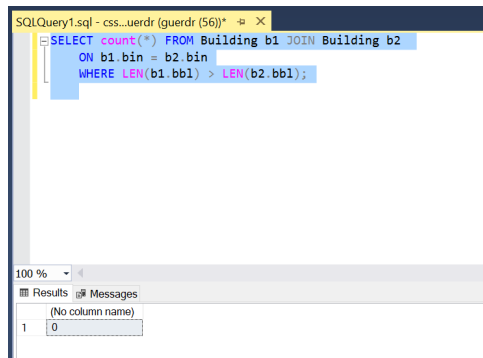
ALTER TABLE Building ADD CONSTRAINT building_PK PRIMARY KEY (bin);



Step 2: Check for duplicates

```
SELECT count(*) FROM Building b1 JOIN Building b2
      ON b1.bin = b2.bin
      WHERE LEN(b1.bbl) > LEN(b2.bbl);
```

In the Building table, every attribute was added besides the building identification number to find duplicates that show inconsistencies. However, no column had a duplicate within the Building table.



Example, if there was an inconsistency and duplicates appeared within the premises address attribute this is how you would delete and get rid of duplicates using the following statement.

```
DELETE b1 FROM Building b1 JOIN Building b2
      ON b1.bin = b2.bin
      WHERE LEN(b1.premises_address) > LEN(b2.premises_address);
```

How to Clean the Bureau of Fire Prevention Dataset FDViolation Table:

★ Disclaimer: Repeat steps above to clean data. ★

Step 1: Add constraints to your table

```
ALTER TABLE FDViolation ADD CONSTRAINT violationid_PK PRIMARY KEY
(violation_id);
```

```
SQLQuery1.sql - css...uerdr (guerdr (56))* X
ALTER TABLE FDViolation ADD CONSTRAINT violationid_PK PRIMARY KEY (violation_id);

100 %
Messages
Commands completed successfully.
Completion time: 2020-11-17T00:12:26.0053468-05:00
```

ALTER TABLE FDViolation ADD CONSTRAINT bin_FK
FOREIGN KEY(bin) REFERENCES Building(bin);

```
SQLQuery1.sql - css...uerdr (guerdr (56))* X
ALTER TABLE FDViolation ADD CONSTRAINT bin_FK
FOREIGN KEY(bin) REFERENCES Building(bin);

100 %
Messages
Commands completed successfully.
Completion time: 2020-11-17T00:19:39.7944874-05:00
```

Step 2: Check for duplicates

```
SELECT count(*) FROM FDViolation fdv1 JOIN FDViolation fdv2
ON fdv1.bin = fdv2.bin
WHERE LEN(fdv1.action) > LEN(fdv2.action);
```

In the FDViolation table, every attribute was added to the statement above to check to see if there were any duplicates that show inconsistencies within the FDViolation table. However, no column had a duplicate within the FDViolation table.

```
SQLQuery1.sql - css...uerdr (guerdr (56))* X
SELECT count(*) FROM FDViolation fdv1 JOIN FDViolation fdv2
ON fdv1.bin = fdv2.bin
WHERE LEN(fdv1.action) > LEN(fdv2.action);

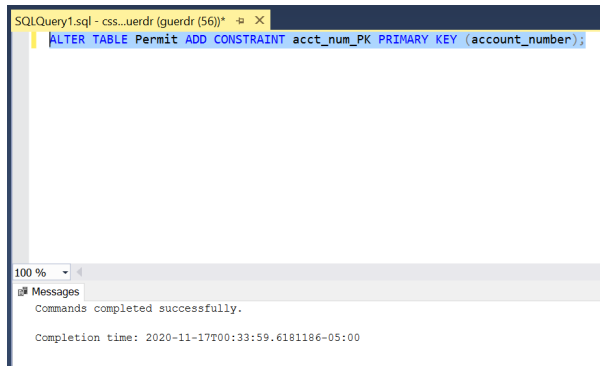
100 %
Results Messages
(No column name)
1 0
```


How to Clean the Bureau of Fire Prevention Dataset Permit Table:

★ Disclaimer: Repeat steps above to clean data. ★

Step 1: Add constraints to your table

```
ALTER TABLE Permit ADD CONSTRAINT acct_num_PK PRIMARY KEY (account_number);
```

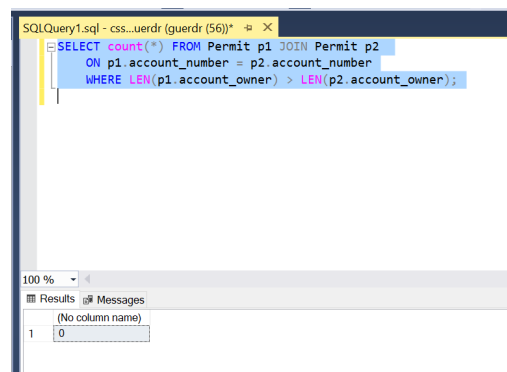


The screenshot shows a SQL query window titled 'SQLQuery1.sql - css...uerdr (56)' with the following text: `ALTER TABLE Permit ADD CONSTRAINT acct_num_PK PRIMARY KEY (account_number);`. Below the query, a 'Messages' pane displays 'Commands completed successfully.' and 'Completion time: 2020-11-17T00:33:59.6181186-05:00'.

Step 2: Check for duplicates

```
SELECT count(*) FROM Permit p1 JOIN Permit p2
      ON p1.account_number = p2.account_number
      WHERE LEN(p1.account_owner) > LEN(p2.account_owner);
```

In the Permit table, the account owner attribute was added to the statement above to check to see if there were any duplicates that show inconsistencies within the Permit table. However, there were no duplicates shown within the Permit table.



The screenshot shows a SQL query window titled 'SQLQuery1.sql - css...uerdr (56)' with the following text: `SELECT count(*) FROM Permit p1 JOIN Permit p2 ON p1.account_number = p2.account_number WHERE LEN(p1.account_owner) > LEN(p2.account_owner);`. Below the query, a 'Results' pane shows a single row with the value '0' under the column '(No column name)'.