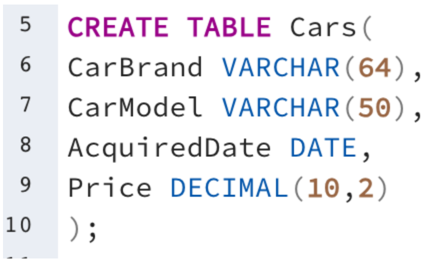
MET CS 669 Database Design and Implementation for Business
Lab 1 Katherine Rein: SQL From Scratch

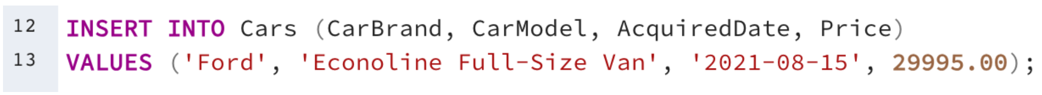
Section One – Absolute Fundamentals

1. *Creating a Table* – Create the Cars table. As a reminder, make sure to follow along in the Lab 1 Explanations document as it shows you how to create tables and complete the other steps.



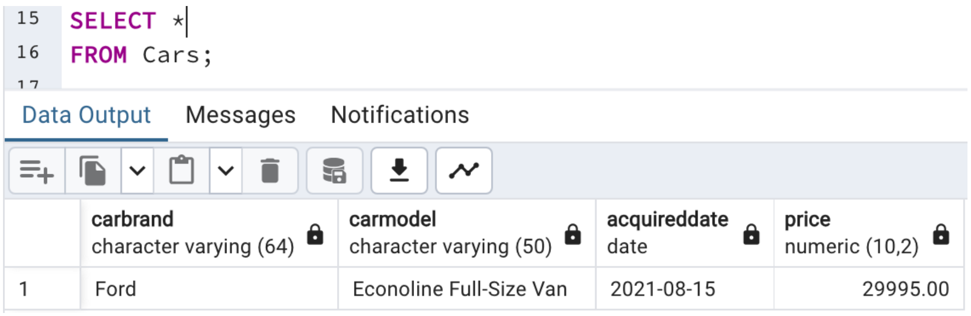
```sql
CREATE TABLE Cars(
CarBrand VARCHAR(64),
CarModel VARCHAR(50),
AcquiredDate DATE,
Price DECIMAL(10,2)
);
```

2. *Inserting a Row* – Insert the first row where the Car Brand name is “Ford”, the Car Model is “Econoline Full-Size Van”, the acquisition date for the car is August 15,2021, and the price is $29,995.00.



```sql
INSERT INTO Cars (CarBrand, CarModel, AcquiredDate, Price)
VALUES ('Ford', 'Econoline Full-Size Van', '2021-08-15', 29995.00);
```

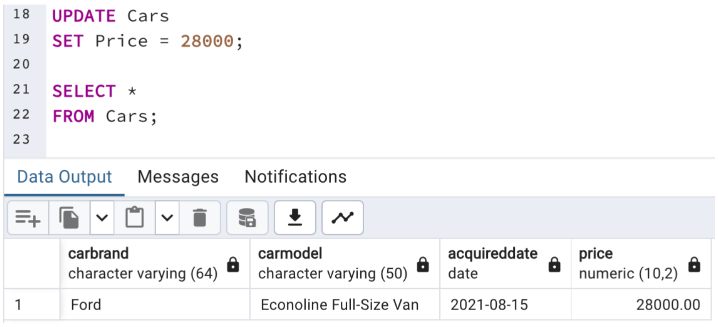
3. *Selecting All Rows* – Select all rows in the table to view the row you inserted.



```sql
SELECT *
FROM Cars;
```

	carbrand character varying (64)	carmodel character varying (50)	acquireddate date	price numeric (10,2)
1	Ford	Econoline Full-Size Van	2021-08-15	29995.00

4. *Updating All Rows* – Update the price of the row in the table to $28,000, then select all rows in the table to view the row you updated.

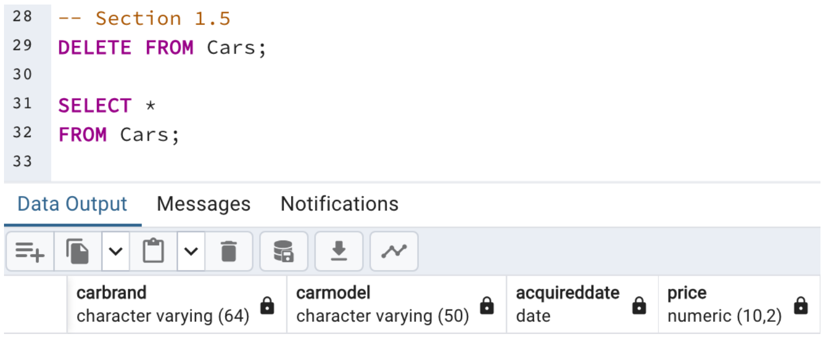


```sql
UPDATE Cars
SET Price = 28000;

SELECT *
FROM Cars;
```

	carbrand character varying (64)	carmodel character varying (50)	acquireddate date	price numeric (10,2)
1	Ford	Econoline Full-Size Van	2021-08-15	28000.00

Copyright 2019-2024 Boston University. All Rights Reserved.
Authored by Warren Mansur and Pamela Farr.

5. *Deleting All Rows* – Remove all rows from the table, then select all rows in the table to verify there are no rows.

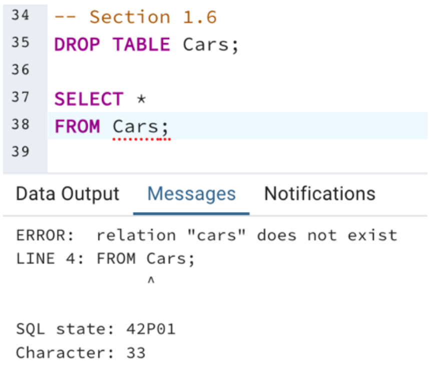


```
-- Section 1.5
DELETE FROM Cars;

SELECT *
FROM Cars;

```

Data Output Messages Notifications

	carbrand character varying (64)	carmodel character varying (50)	acquireddate date	price numeric (10,2)

6. *Dropping a Table* – Drop the Cars table, then select all rows in the table to verify the table doesn't exist. Explain how you would use the error message, in conjunction with the SELECT command, to diagnose the error.



```
-- Section 1.6
DROP TABLE Cars;

SELECT *
FROM Cars;

```

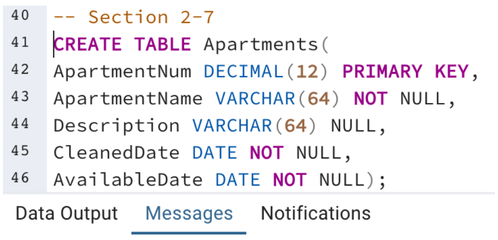
Data Output Messages Notifications

```
ERROR:  relation "cars" does not exist
LINE 4: FROM Cars;
             ^

SQL state: 42P01
Character: 33
```

The error message states that the table Cars does not exist. This indicates to us that we have effectively deleted/dropped the table.

Section Two – More Precise Data Handling

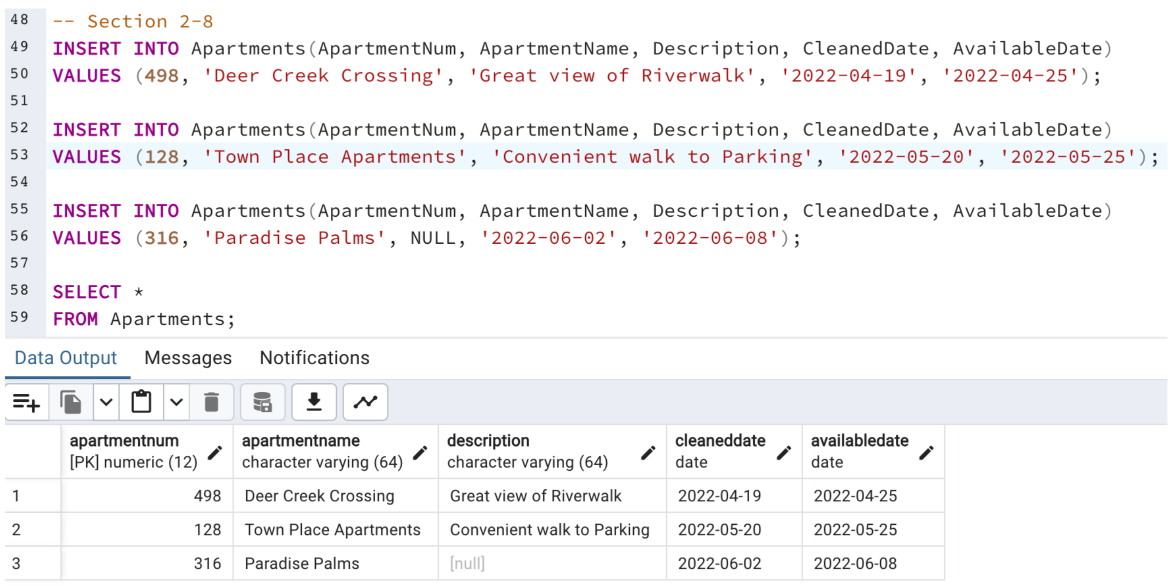
7. *Table Setup* – Create the Apartments table with its columns, datatypes, and constraints.



```
-- Section 2-7
CREATE TABLE Apartments(
ApartmentNum DECIMAL(12) PRIMARY KEY,
ApartmentName VARCHAR(64) NOT NULL,
Description VARCHAR(64) NULL,
CleanedDate DATE NOT NULL,
AvailableDate DATE NOT NULL);
```

Data Output Messages Notifications

8. *Table Population* – Insert the rows illustrated in the figure above. Note that the description for Apartment 316 at Paradise Palms is null. Then select all rows from the Apartments table to show that the inserts were successful.



```sql
-- Section 2-8
INSERT INTO Apartments(ApartmentNum, ApartmentName, Description, CleanedDate, AvailableDate)
VALUES (498, 'Deer Creek Crossing', 'Great view of Riverwalk', '2022-04-19', '2022-04-25');

INSERT INTO Apartments(ApartmentNum, ApartmentName, Description, CleanedDate, AvailableDate)
VALUES (128, 'Town Place Apartments', 'Convenient walk to Parking', '2022-05-20', '2022-05-25');

INSERT INTO Apartments(ApartmentNum, ApartmentName, Description, CleanedDate, AvailableDate)
VALUES (316, 'Paradise Palms', NULL, '2022-06-02', '2022-06-08');

SELECT *
FROM Apartments;
```

Data Output Messages Notifications

	apartmentnum [PK] numeric (12)	apartmentname character varying (64)	description character varying (64)	cleaneddate date	availabledate date
1	498	Deer Creek Crossing	Great view of Riverwalk	2022-04-19	2022-04-25
2	128	Town Place Apartments	Convenient walk to Parking	2022-05-20	2022-05-25
3	316	Paradise Palms	[null]	2022-06-02	2022-06-08

9. *Invalid Insertion* – The following values leave the Apartment Name with no value.

ApartmentNum = 252
ApartmentName = NULL
Description = Close to Downtown shops
CleanedDate = 17-JUL-2020
AvailableDate = 13-JUL-2020

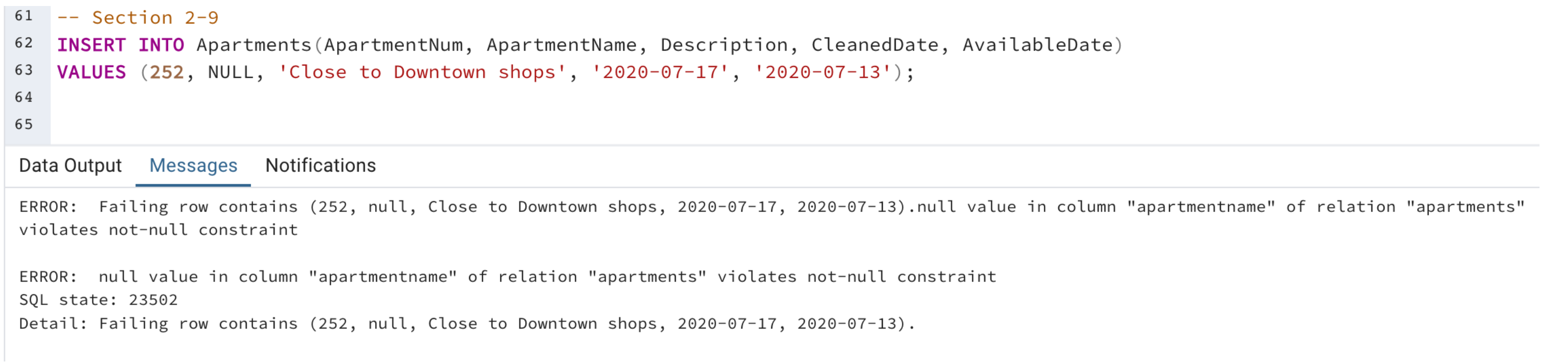
a. In your own words, explain what a null value is.

A null value is simply an empty field.

b. In your own words, explain what a NOT NULL constraint is.

A NOT NULL constraint is a limit that a coder puts on the table in SQL so that a null value is not allowed in that column.

c. Attempt to insert the values as listed and explain how the database handles this attempt.
Explain how you would interpret the error message conclude that the location column is missing a required value.



```sql
61 -- Section 2-9
62 INSERT INTO Apartments(ApartmentNum, ApartmentName, Description, CleanedDate, AvailableDate)
63 VALUES (252, NULL, 'Close to Downtown shops', '2020-07-17', '2020-07-13');
64
65
```

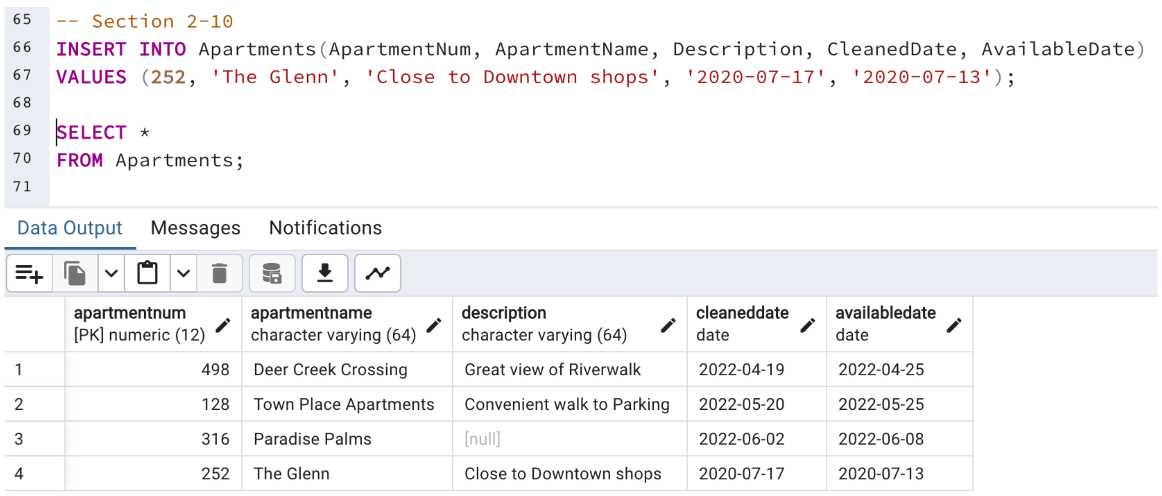
Data Output Messages Notifications

```
ERROR:  Failing row contains (252, null, Close to Downtown shops, 2020-07-17, 2020-07-13).null value in column "apartmentname" of relation "apartments"
violates not-null constraint

ERROR:  null value in column "apartmentname" of relation "apartments" violates not-null constraint
SQL state: 23502
Detail: Failing row contains (252, null, Close to Downtown shops, 2020-07-17, 2020-07-13).
```

When the database gets to line 63, it comes to the null value for the ApartmentName column. This is not allowed and so the database throws an error. The error message tells us this by saying that the null value in the ApartmentName column violates the not-null constraint. This tells us what row and what part of that row is an issue.

10. *Valid Insertion* – Now insert the row with the Apartment Name intact, with the following values.

ApartmentNum = 252
ApartmentName = The Glenn
Description = Close to Downtown shops
CleanedDate = 17-JUL-2020
AvailableDate = 13-JUL-2020

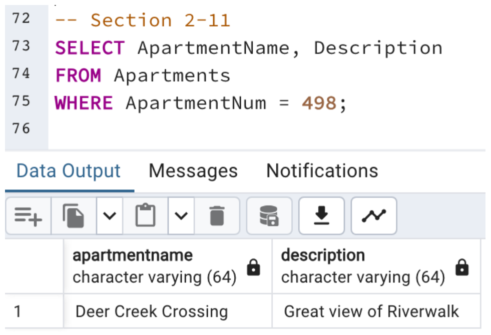


```
-- Section 2-10
INSERT INTO Apartments(ApartmentNum, ApartmentName, Description, CleanedDate, AvailableDate)
VALUES (252, 'The Glenn', 'Close to Downtown shops', '2020-07-17', '2020-07-13');

SELECT *
FROM Apartments;
```

Data Output | Messages | Notifications

	apartmentnum [PK] numeric (12)	apartmentname character varying (64)	description character varying (64)	cleaneddate date	availabledate date
1	498	Deer Creek Crossing	Great view of Riverwalk	2022-04-19	2022-04-25
2	128	Town Place Apartments	Convenient walk to Parking	2022-05-20	2022-05-25
3	316	Paradise Palms	[null]	2022-06-02	2022-06-08
4	252	The Glenn	Close to Downtown shops	2020-07-17	2020-07-13

11. *Filtered Results* – Retrieve only the Apartment Name and the Description for Deer Creek Crossing, using the primary key as the column that determines which row is retrieved.



```
-- Section 2-11
SELECT ApartmentName, Description
FROM Apartments
WHERE ApartmentNum = 498;
```

Data Output | Messages | Notifications

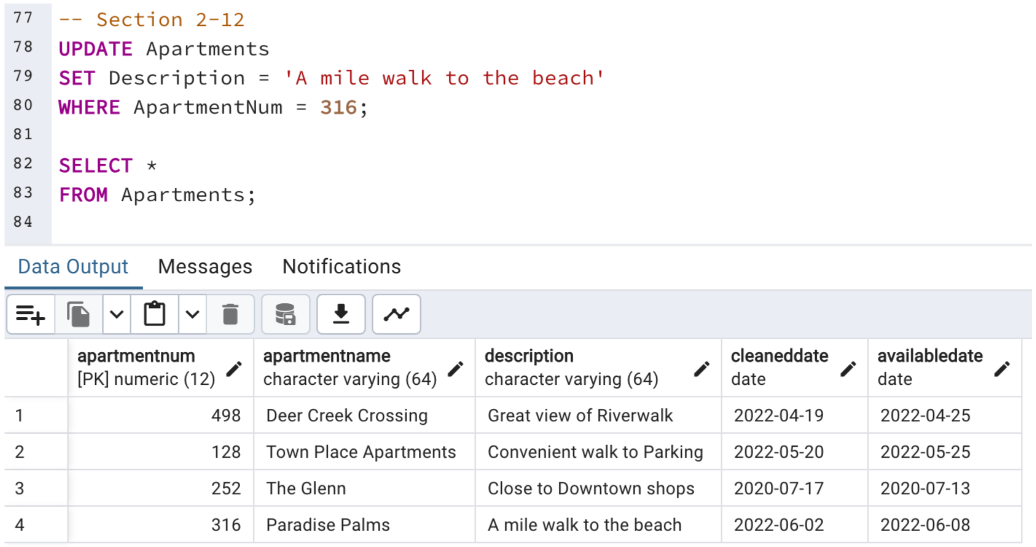
	apartmentname character varying (64)	description character varying (64)
1	Deer Creek Crossing	Great view of Riverwalk

Explain why it is useful to limit the number of rows and columns returned from a SELECT statement.

Selecting rows from a database is useful in terms of efficiency. If you have a lot of rows and need to do some calculations, it can take a lot of time and energy to try and look through that much data.

12. *Targeted Update* – The Paradise Palms apartment has no description. Update the row so that its description says "A mile walk to the beach".

Select all rows in the table to show that the update was successful.

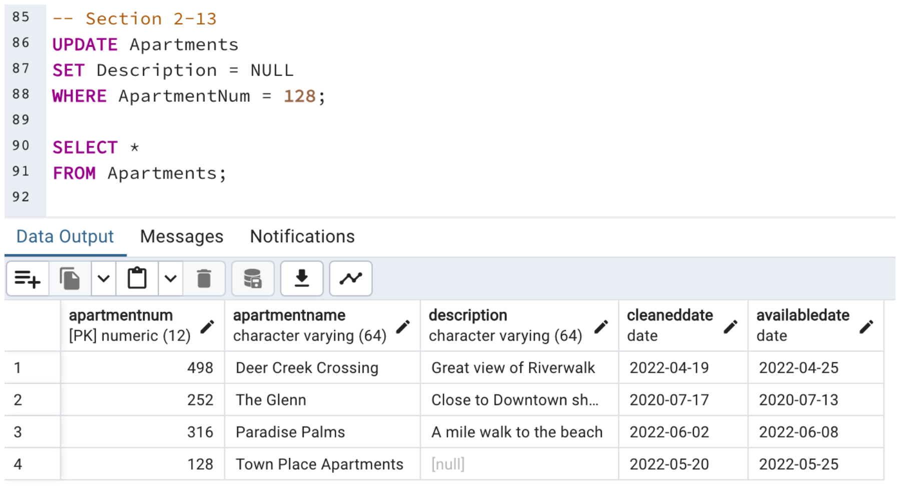


```sql
-- Section 2-12
UPDATE Apartments
SET Description = 'A mile walk to the beach'
WHERE ApartmentNum = 316;

SELECT *
FROM Apartments;
```

	apartmentnum [PK] numeric (12)	apartmentname character varying (64)	description character varying (64)	cleaneddate date	availabledate date
1	498	Deer Creek Crossing	Great view of Riverwalk	2022-04-19	2022-04-25
2	128	Town Place Apartments	Convenient walk to Parking	2022-05-20	2022-05-25
3	252	The Glenn	Close to Downtown shops	2020-07-17	2020-07-13
4	316	Paradise Palms	A mile walk to the beach	2022-06-02	2022-06-08

13. *Updating to Null* – Update the Town Place Apartments so that it no longer has a description (i.e., its description is null).

Select all rows in the table to show that the update was successful.

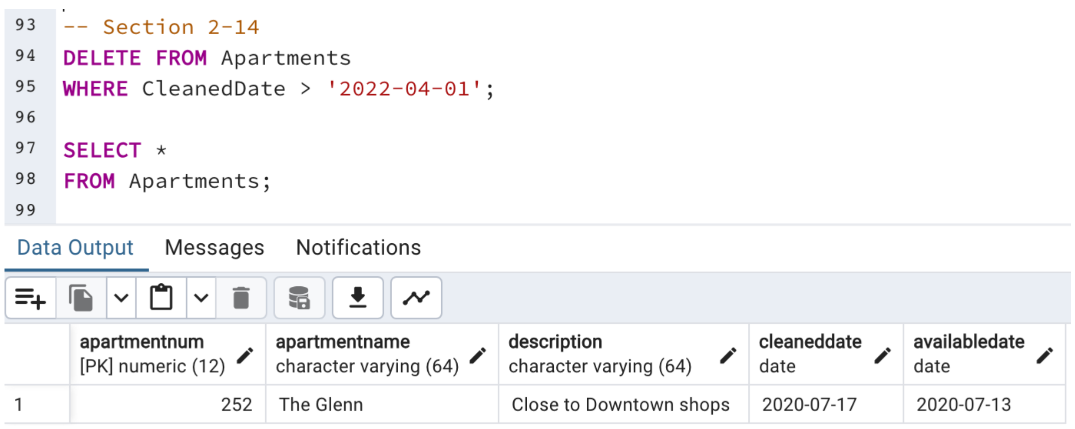


```sql
-- Section 2-13
UPDATE Apartments
SET Description = NULL
WHERE ApartmentNum = 128;

SELECT *
FROM Apartments;
```

	apartmentnum [PK] numeric (12)	apartmentname character varying (64)	description character varying (64)	cleaneddate date	availabledate date
1	498	Deer Creek Crossing	Great view of Riverwalk	2022-04-19	2022-04-25
2	252	The Glenn	Close to Downtown sh…	2020-07-17	2020-07-13
3	316	Paradise Palms	A mile walk to the beach	2022-06-02	2022-06-08
4	128	Town Place Apartments	[null]	2022-05-20	2022-05-25

14. *Targeted Deletion* – Delete all rows where the Cleaned date is greater than April 1, 2022, by using the Cleaned Date column as the determinant of which rows are deleted.

Select all rows in the table to show the delete was successful.

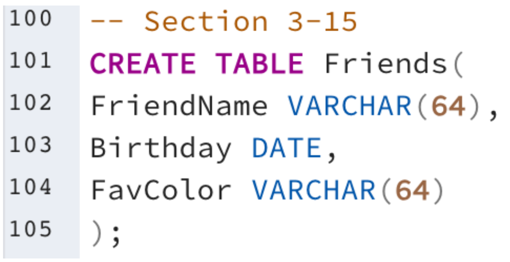


```sql
-- Section 2-14
DELETE FROM Apartments
WHERE CleanedDate > '2022-04-01';

SELECT *
FROM Apartments;
```

	apartmentnum [PK] numeric (12)	apartmentname character varying (64)	description character varying (64)	cleaneddate date	availabledate date
1	252	The Glenn	Close to Downtown shops	2020-07-17	2020-07-13

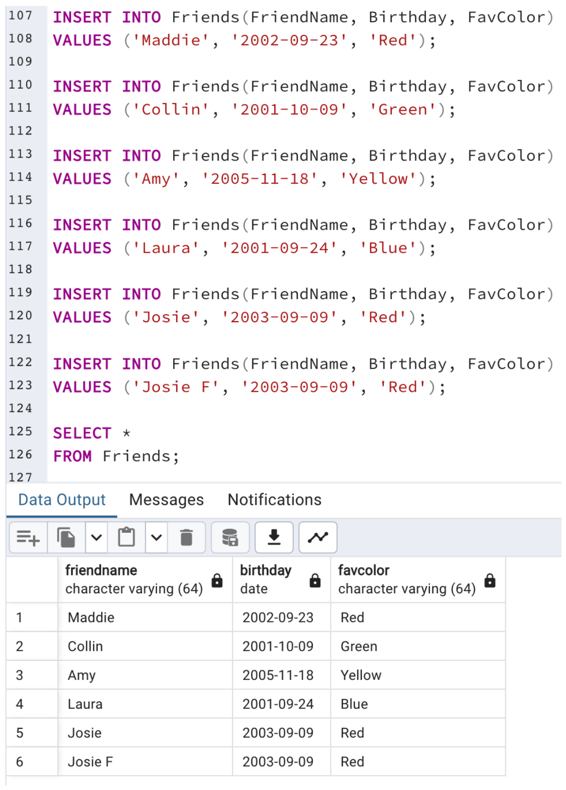
Section Three – Data Anomalies and Formats

15. *Data Anomalies* – In this step you demonstrate anomalies that can occur in improperly designed tables.
    a. Create a table of your choosing that has at least three columns. *Do not add constraints or primary keys to the table, as that may limit your ability to complete parts #b and #c.*



```
-- Section 3-15
CREATE TABLE Friends(
FriendName VARCHAR(64),
Birthday DATE,
FavColor VARCHAR(64)
);
```

    b. Using the table, demonstrate an anomaly that occurs when the same data is inserted multiple times with different values, and explain what the anomaly means for data integrity.

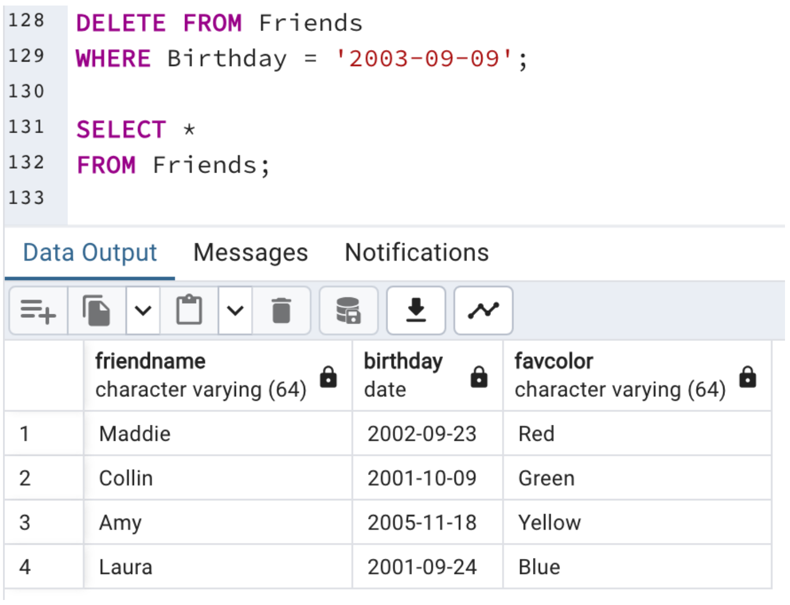


```
INSERT INTO Friends(FriendName, Birthday, FavColor)
VALUES ('Maddie', '2002-09-23', 'Red');

INSERT INTO Friends(FriendName, Birthday, FavColor)
VALUES ('Collin', '2001-10-09', 'Green');

INSERT INTO Friends(FriendName, Birthday, FavColor)
VALUES ('Amy', '2005-11-18', 'Yellow');

INSERT INTO Friends(FriendName, Birthday, FavColor)
VALUES ('Laura', '2001-09-24', 'Blue');

INSERT INTO Friends(FriendName, Birthday, FavColor)
VALUES ('Josie', '2003-09-09', 'Red');

INSERT INTO Friends(FriendName, Birthday, FavColor)
VALUES ('Josie F', '2003-09-09', 'Red');

SELECT *
FROM Friends;
```

	friendname character varying (64)	birthday date	favcolor character varying (64)
1	Maddie	2002-09-23	Red
2	Collin	2001-10-09	Green
3	Amy	2005-11-18	Yellow
4	Laura	2001-09-24	Blue
5	Josie	2003-09-09	Red
6	Josie F	2003-09-09	Red

The insertion anomaly in this scenario makes it seem like we have multiple friends with a September 9th birthday. This is not the case as they simply have different amounts of the name given.

    c. Using the table, demonstrate a deletion anomaly with SQL, and explain what the anomaly means for data integrity.



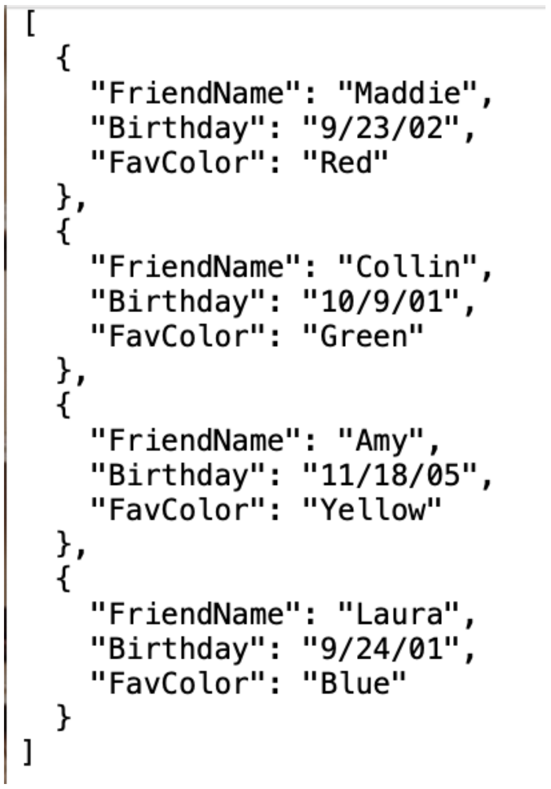
```sql
DELETE FROM Friends
WHERE Birthday = '2003-09-09';

SELECT *
FROM Friends;
```

Data Output | Messages | Notifications

	friendname character varying (64)	birthday date	favcolor character varying (64)
1	Maddie	2002-09-23	Red
2	Collin	2001-10-09	Green
3	Amy	2005-11-18	Yellow
4	Laura	2001-09-24	Blue

Since we deleted the duplicated birthdays, we now don't know Josie's birthday. A deletion anomaly results in lost data and not necessarily knowing that data even exists.

16. *File and Database Table Comparison* – In this step you compare the table created in #15 with a file that contains all the same information.
    a. Create a file in any format you'd like that contains all the same columns and at least 4 rows of information as the table you created in #15. There are many formats you can use. Some examples include XML, flat file, binary, text, and JSON; this list is not exhaustive. All columns and at least 4 rows should be present in the file in its new format. Make sure to provide the file or a screenshot of the file and to explain your choices.



```json
[
  {
    "FriendName": "Maddie",
    "Birthday": "9/23/02",
    "FavColor": "Red"
  },
  {
    "FriendName": "Collin",
    "Birthday": "10/9/01",
    "FavColor": "Green"
  },
  {
    "FriendName": "Amy",
    "Birthday": "11/18/05",
    "FavColor": "Yellow"
  },
  {
    "FriendName": "Laura",
    "Birthday": "9/24/01",
    "FavColor": "Blue"
  }
]
```

I inserted the data into an Excel sheet and then saved it as a CSV file. The CSV file was then converted into a JSON file. I chose the JSON file because it seemed most basic and organized.

b. With a few paragraphs, compare what it's like to access data in the table versus in the file. You may need to first research how applications typically access data in this type of file. Make sure to at least use these comparison points:
   i. Efficiency – If there were millions of rows of data, would it be more efficient to access a single record in the relational table, or the file, and why?

The relational table would be more efficient. This is because allows you to filter and select rows/columns. Without this it can be very time and energy consuming. Also, the databases have query optimization. This also helps with large numbers of rows. Without this any sort of efficiency is essentially gone.

   ii. Security – Imagine you needed to restrict access to one specific row/record, allowing only one person to access it, while the rest of the rows could be accessed by many people. Would it be easier or more difficult to secure this row in the relational table compared to the file, and why?

The relational table would be easier to secure. If you use SQL, you can utilize role-based access control (RBAC), row-level security (RLS), and column-level security. Additionally, you can use passwords, certificates, or external authentication systems. These tools could be possible with JSON but the tools are built in for SQL. Most of the security for the files is that only the files are protected but not the data within the file.

   iii. Structural Independence – Imagine the table structure was modified by adding or taking away columns, and equivalent changes were made to the file. Would these changes affect an app using the table differently than an app using the file, and why?

Yes, the SQL table wouldn't be affected but the JSON file would be. The JSON file needs the data to look the same as when you set up the app. The SQL table does not. The SQL table does not depend on the structure of the file but rather the structure of the table.