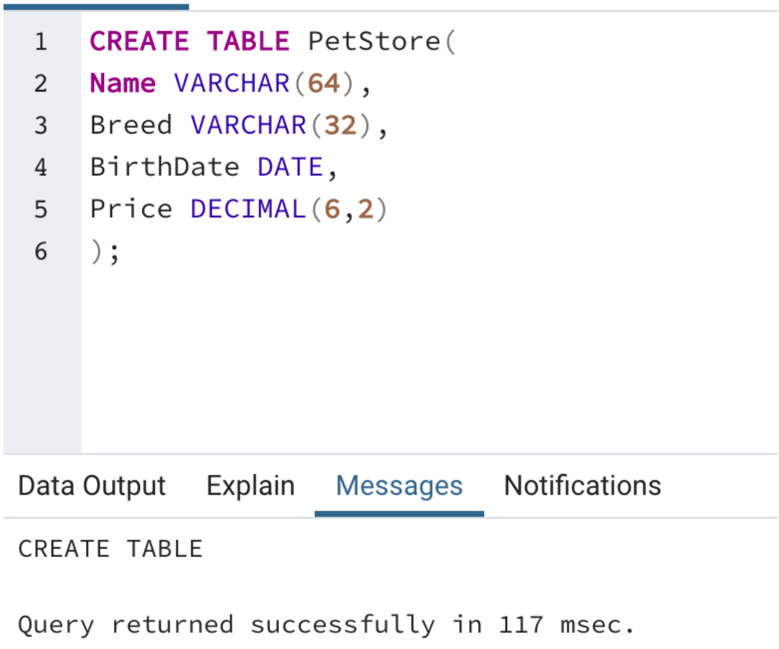
Session 1

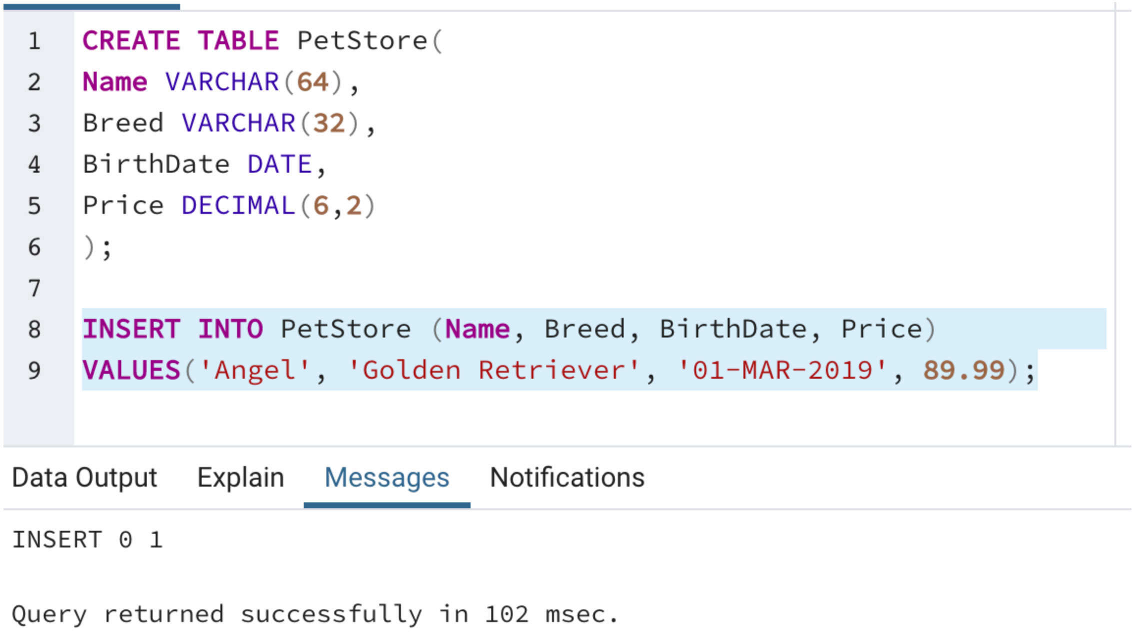
Step 1: Creating a Table



```
CREATE TABLE PetStore(
Name VARCHAR(64),
Breed VARCHAR(32),
BirthDate DATE,
Price DECIMAL(6,2)
);
```

```
CREATE TABLE

Query returned successfully in 117 msec.
```

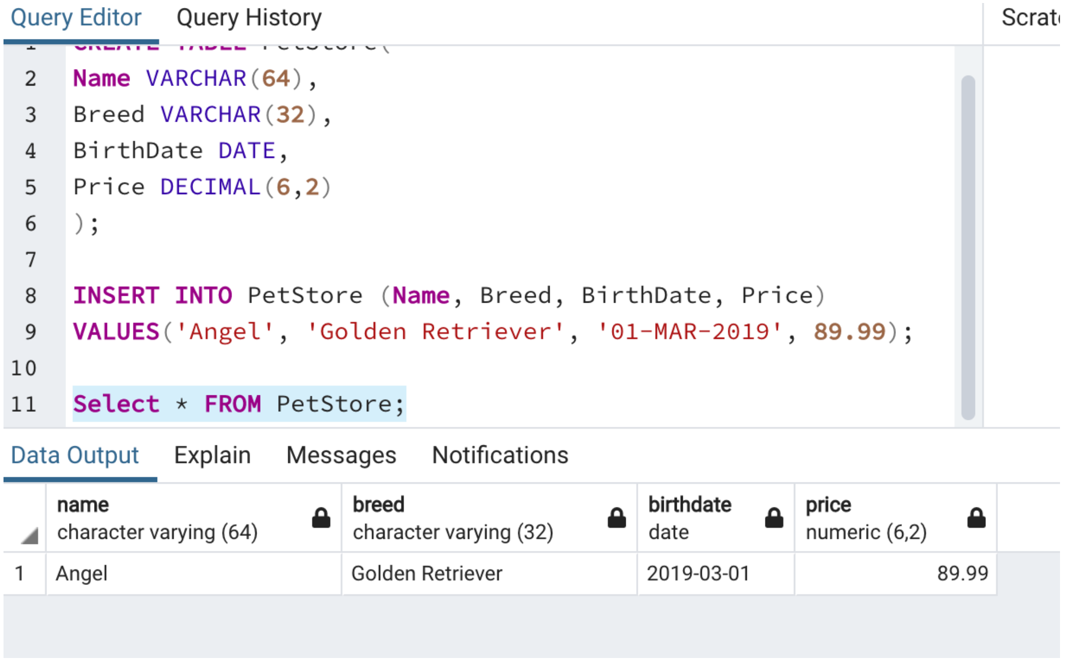
Step2: Inserting a Row



```
CREATE TABLE PetStore(
Name VARCHAR(64),
Breed VARCHAR(32),
BirthDate DATE,
Price DECIMAL(6,2)
);

INSERT INTO PetStore (Name, Breed, BirthDate, Price)
VALUES('Angel', 'Golden Retriever', '01-MAR-2019', 89.99);
```

```
INSERT 0 1

Query returned successfully in 102 msec.
```

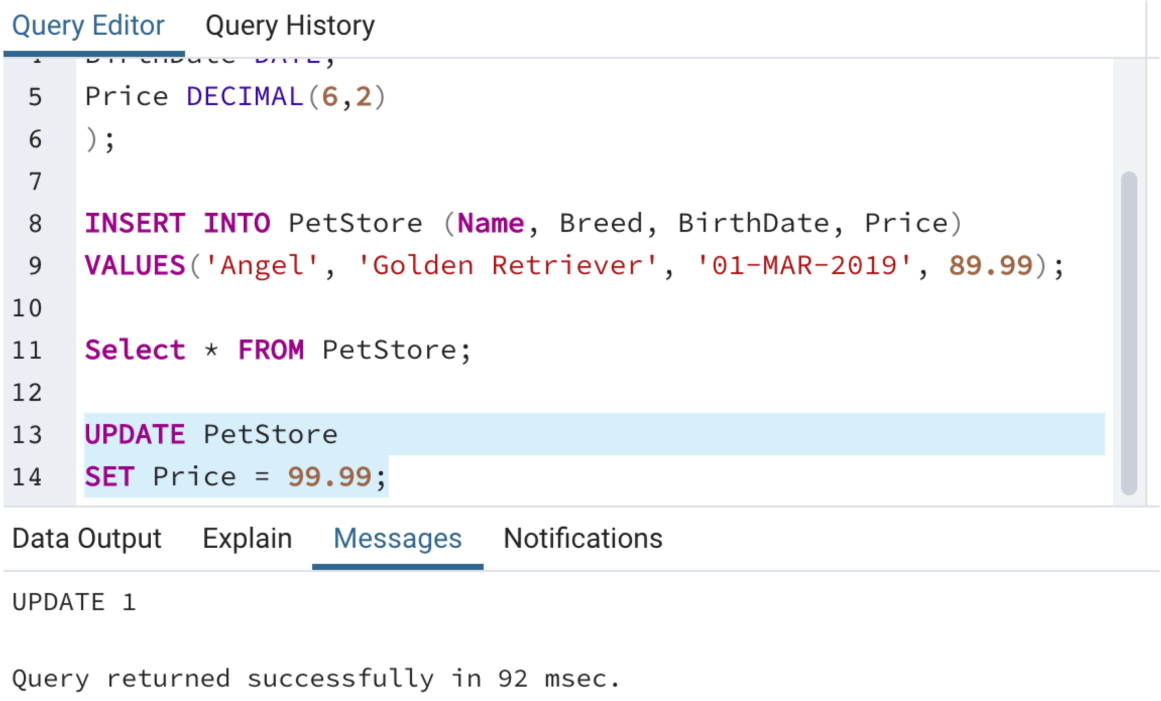
Step 3: Selecting All Rows



```
CREATE TABLE PetStore(
Name VARCHAR(64),
Breed VARCHAR(32),
BirthDate DATE,
Price DECIMAL(6,2)
);

INSERT INTO PetStore (Name, Breed, BirthDate, Price)
VALUES('Angel', 'Golden Retriever', '01-MAR-2019', 89.99);

Select * FROM PetStore;
```

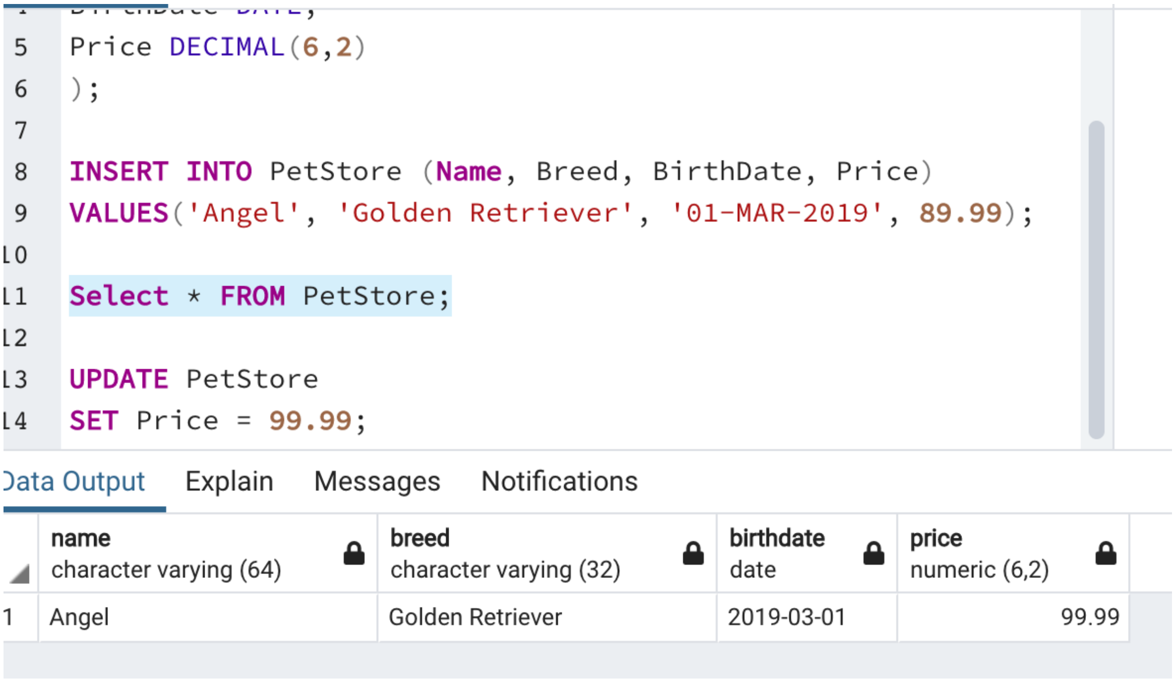
	name character varying (64)	breed character varying (32)	birthdate date	price numeric (6,2)
1	Angel	Golden Retriever	2019-03-01	89.99

Step 4: Updating All Rows



```
BirthDate DATE,
Price DECIMAL(6,2)
);

INSERT INTO PetStore (Name, Breed, BirthDate, Price)
VALUES('Angel', 'Golden Retriever', '01-MAR-2019', 89.99);

Select * FROM PetStore;

UPDATE PetStore
SET Price = 99.99;
```

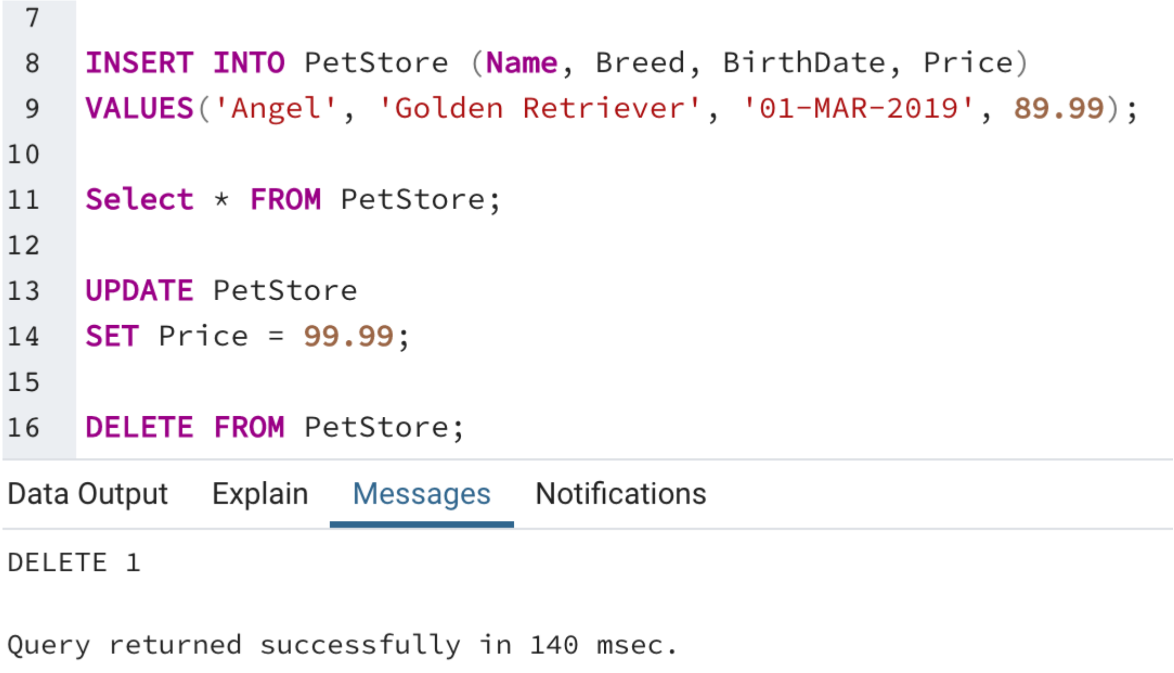
```
UPDATE 1

Query returned successfully in 92 msec.
```



```
BirthDate DATE,
Price DECIMAL(6,2)
);

INSERT INTO PetStore (Name, Breed, BirthDate, Price)
VALUES('Angel', 'Golden Retriever', '01-MAR-2019', 89.99);

Select * FROM PetStore;

UPDATE PetStore
SET Price = 99.99;
```

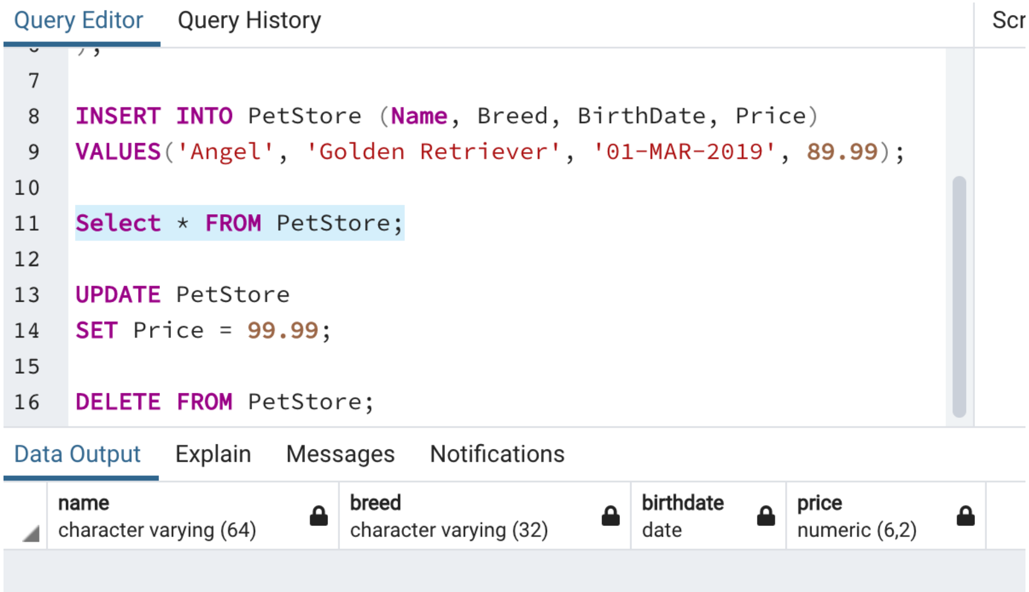
	name character varying (64)	breed character varying (32)	birthdate date	price numeric (6,2)
1	Angel	Golden Retriever	2019-03-01	99.99

Step 5: Deleting All Rows



```

INSERT INTO PetStore (Name, Breed, BirthDate, Price)
VALUES('Angel', 'Golden Retriever', '01-MAR-2019', 89.99);

Select * FROM PetStore;

UPDATE PetStore
SET Price = 99.99;

DELETE FROM PetStore;
```

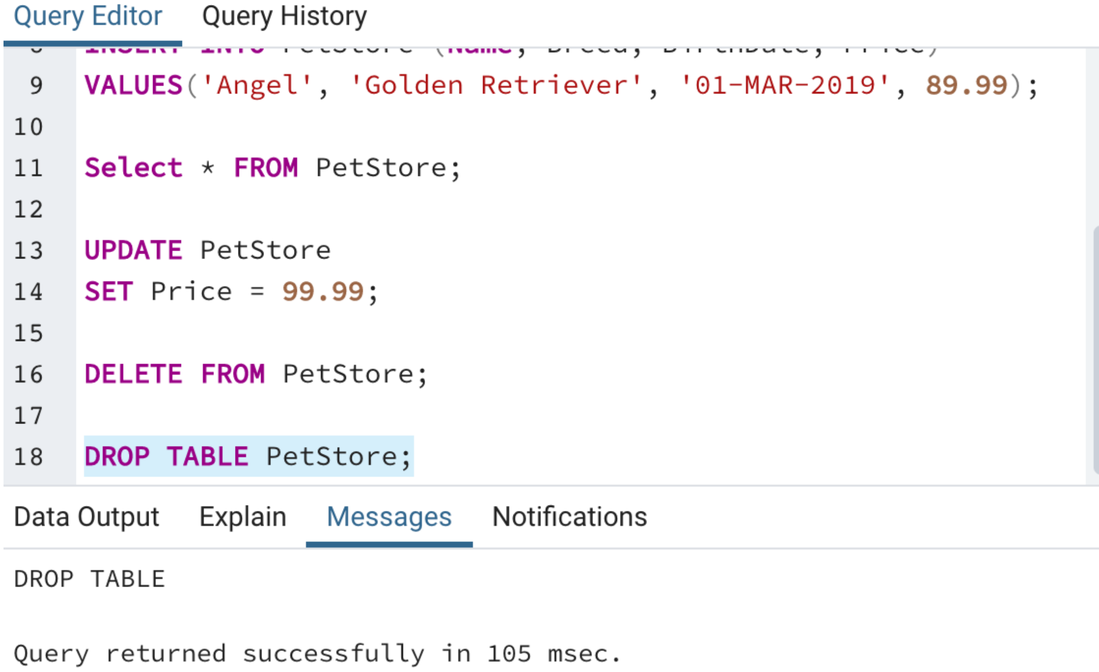
```
DELETE 1

Query returned successfully in 140 msec.
```



```
);

INSERT INTO PetStore (Name, Breed, BirthDate, Price)
VALUES('Angel', 'Golden Retriever', '01-MAR-2019', 89.99);

Select * FROM PetStore;

UPDATE PetStore
SET Price = 99.99;

DELETE FROM PetStore;
```

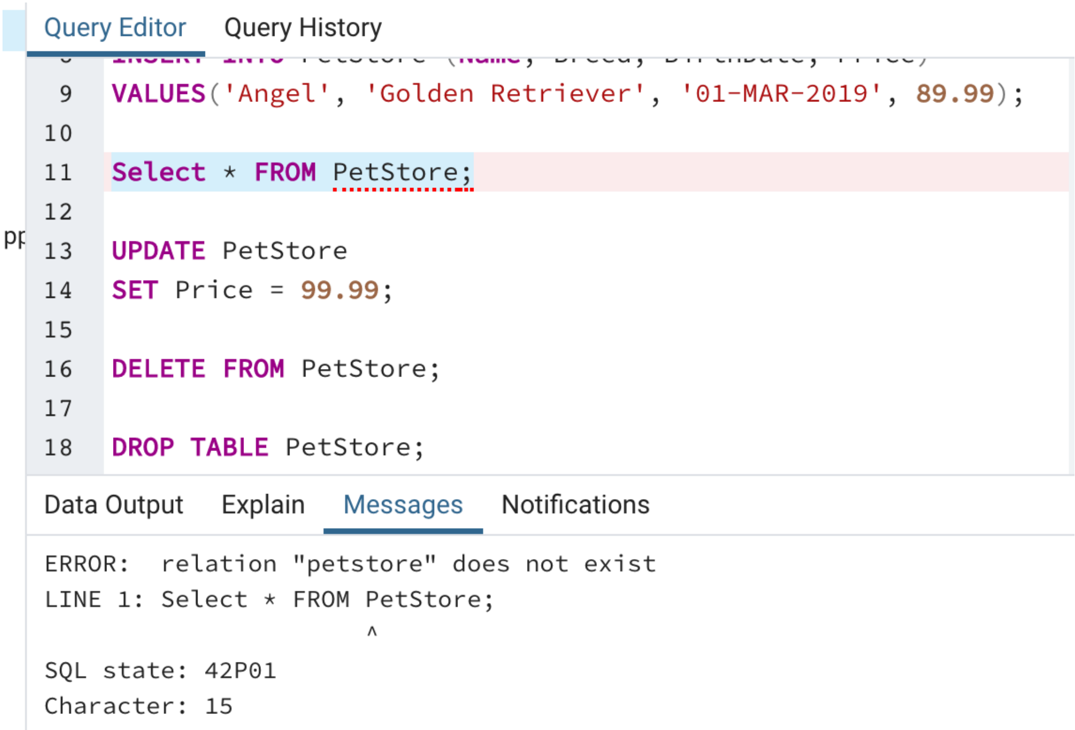
name character varying (64)	breed character varying (32)	birthdate date	price numeric (6,2)

Step 6: Dropping a Table



```
INSERT INTO PetStore (Name, Breed, BirthDate, Price)
VALUES('Angel', 'Golden Retriever', '01-MAR-2019', 89.99);

Select * FROM PetStore;

UPDATE PetStore
SET Price = 99.99;

DELETE FROM PetStore;

DROP TABLE PetStore;
```

```
DROP TABLE

Query returned successfully in 105 msec.
```



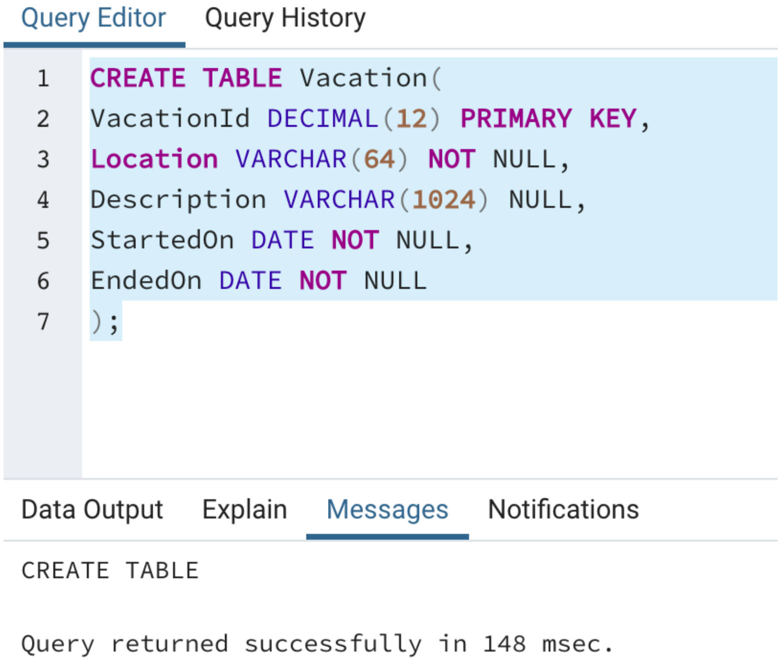
```sql
INSERT INTO PetStore (Name, Breed, BirthDate, Price)
VALUES('Angel', 'Golden Retriever', '01-MAR-2019', 89.99);

Select * FROM PetStore;

UPDATE PetStore
SET Price = 99.99;

DELETE FROM PetStore;

DROP TABLE PetStore;
```

```
ERROR:  relation "petstore" does not exist
LINE 1: Select * FROM PetStore;
                      ^
SQL state: 42P01
Character: 15
```

Dropping the table would get rid of the whole table. After executing DROP TABLE PetStore; then Select * FROM PetStore; is to read the table, but the error message tells that now PetStore no longer exists.

Session2

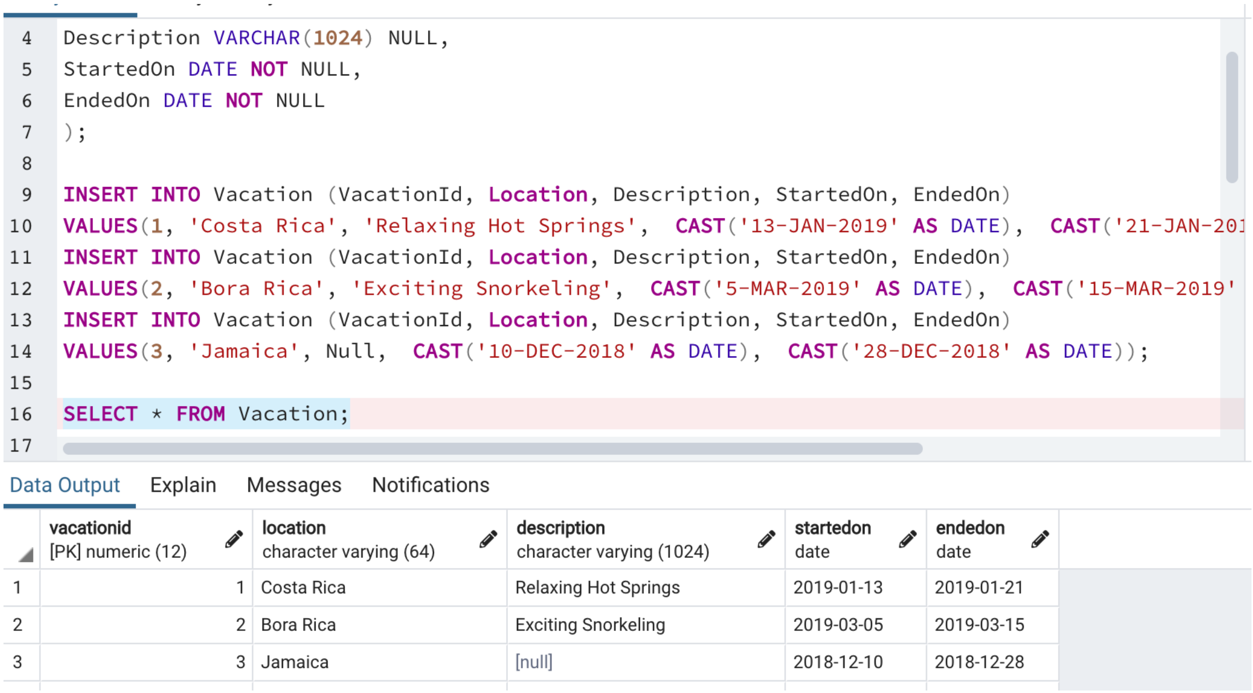
Step 7: Table Setup



```sql
CREATE TABLE Vacation(
VacationId DECIMAL(12) PRIMARY KEY,
Location VARCHAR(64) NOT NULL,
Description VARCHAR(1024) NULL,
StartedOn DATE NOT NULL,
EndedOn DATE NOT NULL
);
```

```
CREATE TABLE

Query returned successfully in 148 msec.
```

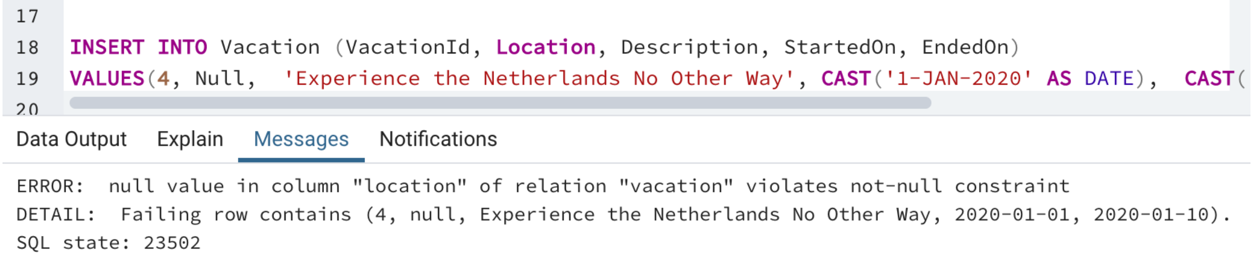
Step 8: Table Population



```
Description VARCHAR(1024) NULL,
StartedOn DATE NOT NULL,
EndedOn DATE NOT NULL
);

INSERT INTO Vacation (VacationId, Location, Description, StartedOn, EndedOn)
VALUES(1, 'Costa Rica', 'Relaxing Hot Springs',  CAST('13-JAN-2019' AS DATE),  CAST('21-JAN-201
INSERT INTO Vacation (VacationId, Location, Description, StartedOn, EndedOn)
VALUES(2, 'Bora Rica', 'Exciting Snorkeling',  CAST('5-MAR-2019' AS DATE),  CAST('15-MAR-2019'
INSERT INTO Vacation (VacationId, Location, Description, StartedOn, EndedOn)
VALUES(3, 'Jamaica', Null,  CAST('10-DEC-2018' AS DATE),  CAST('28-DEC-2018' AS DATE));

SELECT * FROM Vacation;
```

	vacationid [PK] numeric (12)	location character varying (64)	description character varying (1024)	startedon date	endedon date
1	1	Costa Rica	Relaxing Hot Springs	2019-01-13	2019-01-21
2	2	Bora Rica	Exciting Snorkeling	2019-03-05	2019-03-15
3	3	Jamaica	[null]	2018-12-10	2018-12-28

Step 9: Invalid Insertion



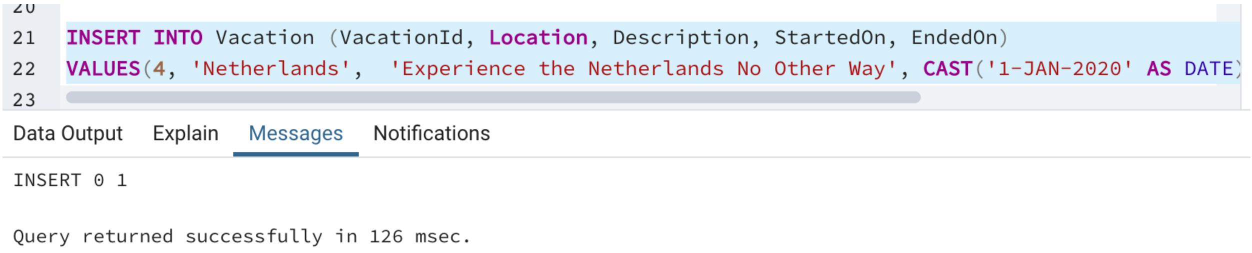
```

INSERT INTO Vacation (VacationId, Location, Description, StartedOn, EndedOn)
VALUES(4, Null,  'Experience the Netherlands No Other Way', CAST('1-JAN-2020' AS DATE),  CAST(
```

```
ERROR:  null value in column "location" of relation "vacation" violates not-null constraint
DETAIL:  Failing row contains (4, null, Experience the Netherlands No Other Way, 2020-01-01, 2020-01-10).
SQL state: 23502
```

Explain how you would interpret the error message to conclude that the location column is missing a required value.

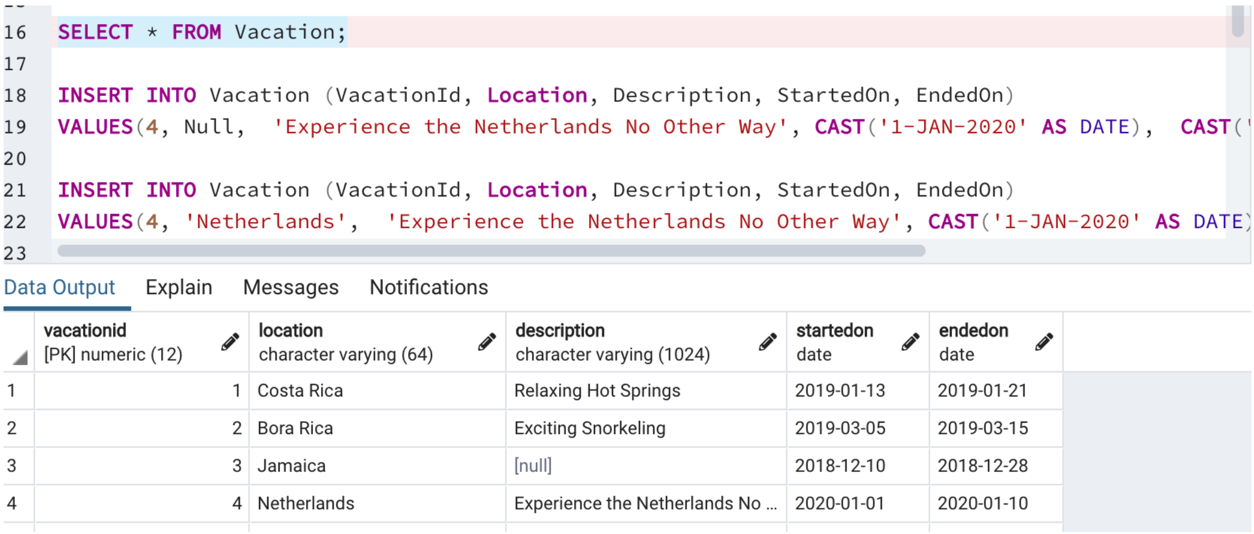
When we set up the table, we set the location column as NOT NULL, which means we have to have a value under this Location column, which can not be null.

Step 10: Valid Insertion



```
INSERT INTO Vacation (VacationId, Location, Description, StartedOn, EndedOn)
VALUES(4, 'Netherlands',  'Experience the Netherlands No Other Way', CAST('1-JAN-2020' AS DATE
```

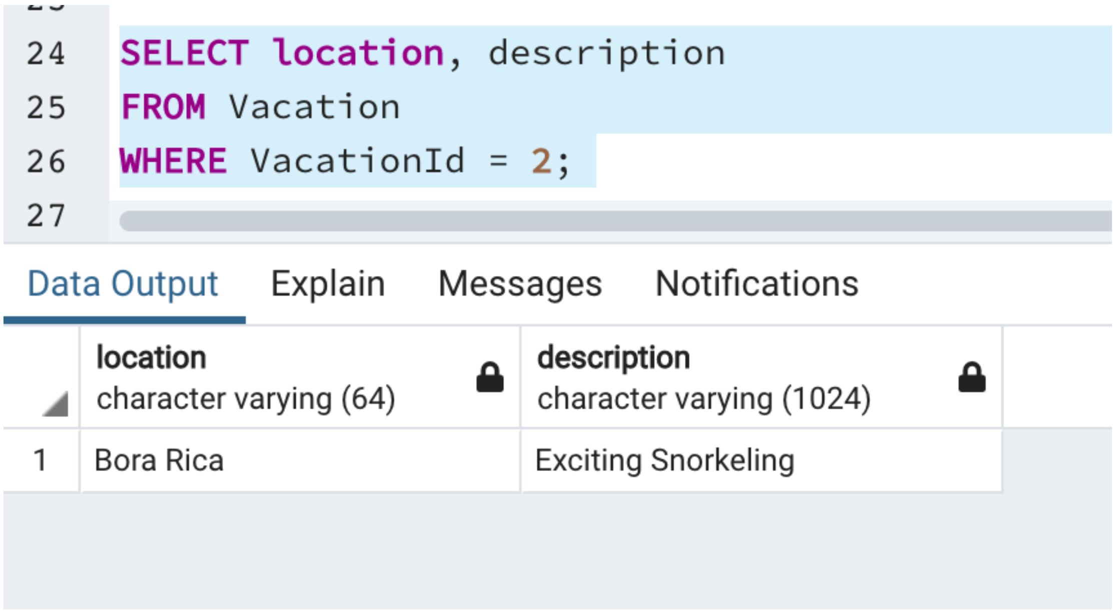
```
INSERT 0 1

Query returned successfully in 126 msec.
```



```
SELECT * FROM Vacation;

INSERT INTO Vacation (VacationId, Location, Description, StartedOn, EndedOn)
VALUES(4, Null,  'Experience the Netherlands No Other Way', CAST('1-JAN-2020' AS DATE),  CAST('

INSERT INTO Vacation (VacationId, Location, Description, StartedOn, EndedOn)
VALUES(4, 'Netherlands',  'Experience the Netherlands No Other Way', CAST('1-JAN-2020' AS DATE
```

	vacationid [PK] numeric (12)	location character varying (64)	description character varying (1024)	startedon date	endedon date
1	1	Costa Rica	Relaxing Hot Springs	2019-01-13	2019-01-21
2	2	Bora Rica	Exciting Snorkeling	2019-03-05	2019-03-15
3	3	Jamaica	[null]	2018-12-10	2018-12-28
4	4	Netherlands	Experience the Netherlands No ...	2020-01-01	2020-01-10

Step 11: Filtered Results



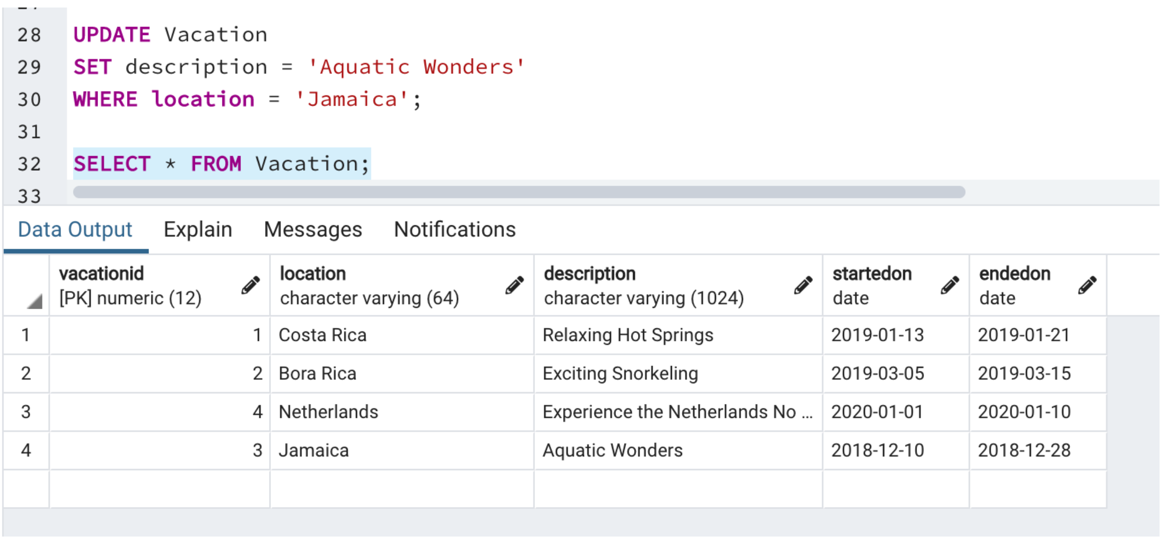
```
SELECT location, description
FROM Vacation
WHERE VacationId = 2;
```

	location character varying (64)	description character varying (1024)
1	Bora Rica	Exciting Snorkeling

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

For the sake of efficiency. If we have many roles and columns, it will be unnecessary for the database to obtain all the columns and rows while we just need a limited target. In this example, we have indicated that we would like to see the location and description columns, but no other columns.

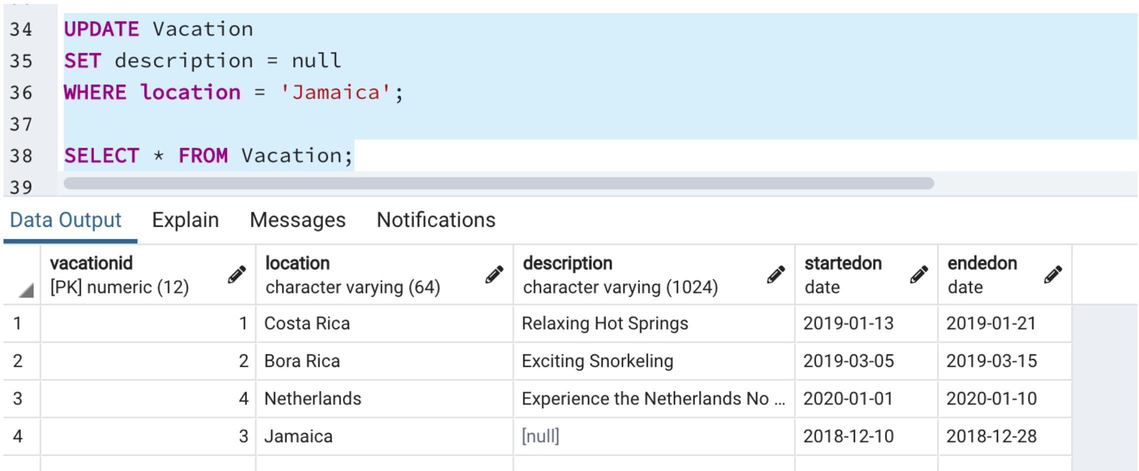
Step 12: Targeted Update



```sql
UPDATE Vacation
SET description = 'Aquatic Wonders'
WHERE location = 'Jamaica';

SELECT * FROM Vacation;
```

	vacationid [PK] numeric (12)	location character varying (64)	description character varying (1024)	startedon date	endedon date
1	1	Costa Rica	Relaxing Hot Springs	2019-01-13	2019-01-21
2	2	Bora Rica	Exciting Snorkeling	2019-03-05	2019-03-15
3	4	Netherlands	Experience the Netherlands No ...	2020-01-01	2020-01-10
4	3	Jamaica	Aquatic Wonders	2018-12-10	2018-12-28

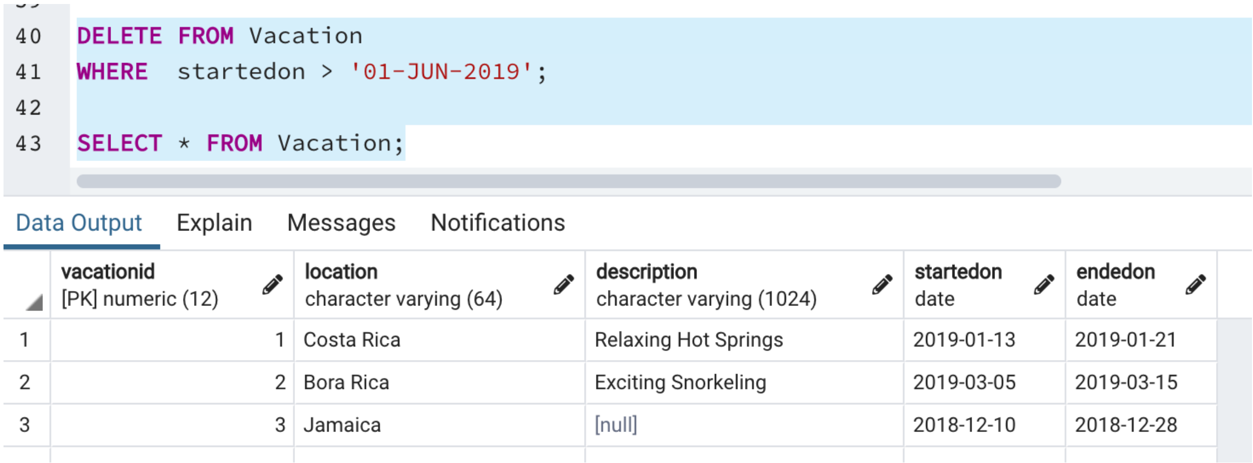
Step 13: Updating to Null



```sql
UPDATE Vacation
SET description = null
WHERE location = 'Jamaica';

SELECT * FROM Vacation;
```

	vacationid [PK] numeric (12)	location character varying (64)	description character varying (1024)	startedon date	endedon date
1	1	Costa Rica	Relaxing Hot Springs	2019-01-13	2019-01-21
2	2	Bora Rica	Exciting Snorkeling	2019-03-05	2019-03-15
3	4	Netherlands	Experience the Netherlands No ...	2020-01-01	2020-01-10
4	3	Jamaica	[null]	2018-12-10	2018-12-28

Step 14: Targeted Deletion

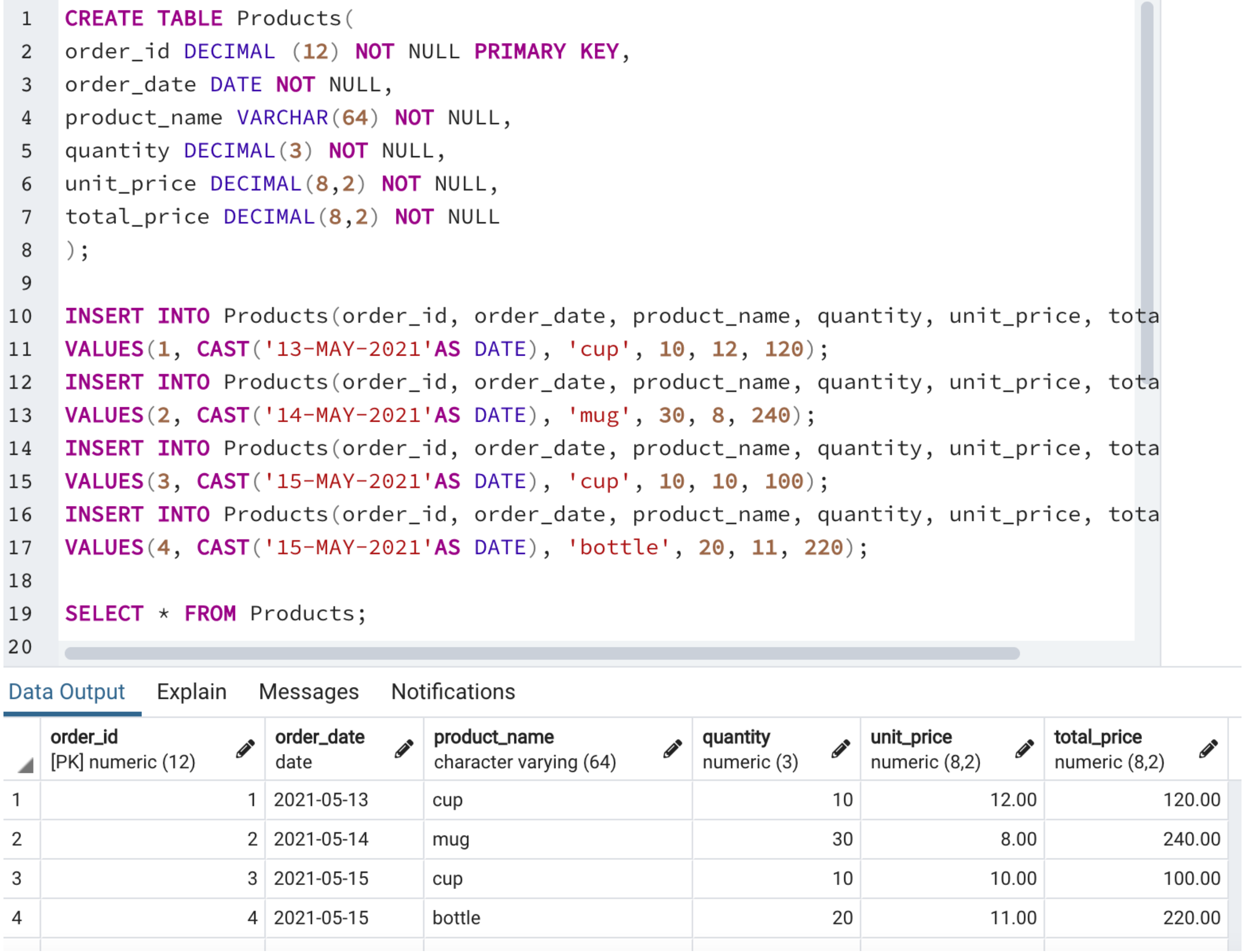


```sql
DELETE FROM Vacation
WHERE  startedon > '01-JUN-2019';

SELECT * FROM Vacation;
```

	vacationid [PK] numeric (12)	location character varying (64)	description character varying (1024)	startedon date	endedon date
1	1	Costa Rica	Relaxing Hot Springs	2019-01-13	2019-01-21
2	2	Bora Rica	Exciting Snorkeling	2019-03-05	2019-03-15
3	3	Jamaica	[null]	2018-12-10	2018-12-28

Session3

Step 15 – Data Anomalies



```
CREATE TABLE Products(
order_id DECIMAL (12) NOT NULL PRIMARY KEY,
order_date DATE NOT NULL,
product_name VARCHAR(64) NOT NULL,
quantity DECIMAL(3) NOT NULL,
unit_price DECIMAL(8,2) NOT NULL,
total_price DECIMAL(8,2) NOT NULL
);

INSERT INTO Products(order_id, order_date, product_name, quantity, unit_price, tota
VALUES(1, CAST('13-MAY-2021'AS DATE), 'cup', 10, 12, 120);
INSERT INTO Products(order_id, order_date, product_name, quantity, unit_price, tota
VALUES(2, CAST('14-MAY-2021'AS DATE), 'mug', 30, 8, 240);
INSERT INTO Products(order_id, order_date, product_name, quantity, unit_price, tota
VALUES(3, CAST('15-MAY-2021'AS DATE), 'cup', 10, 10, 100);
INSERT INTO Products(order_id, order_date, product_name, quantity, unit_price, tota
VALUES(4, CAST('15-MAY-2021'AS DATE), 'bottle', 20, 11, 220);

SELECT * FROM Products;

```

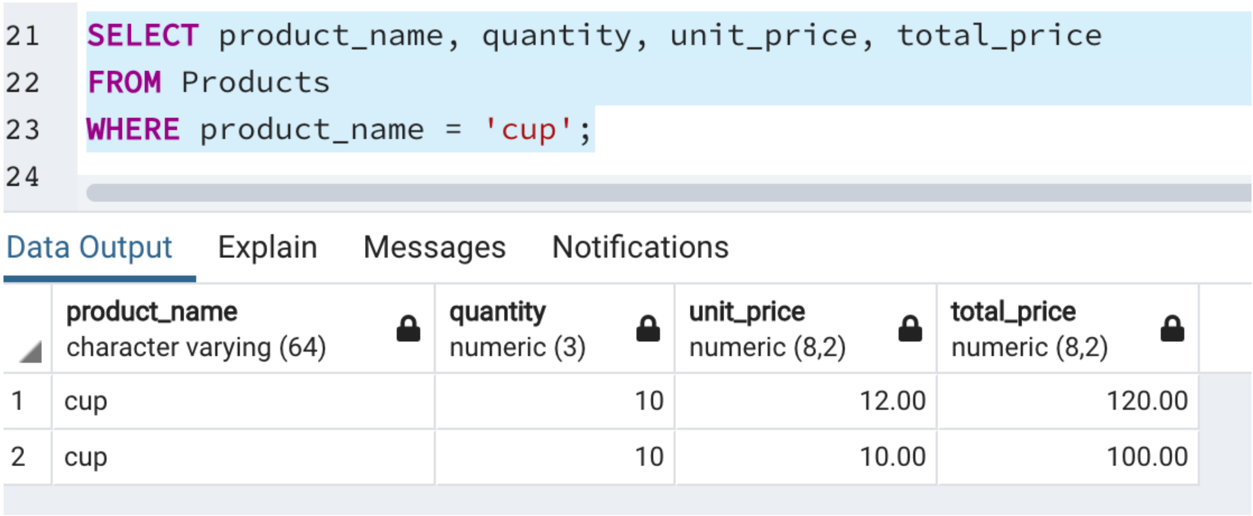
	order_id [PK] numeric (12)	order_date date	product_name character varying (64)	quantity numeric (3)	unit_price numeric (8,2)	total_price numeric (8,2)
1	1	2021-05-13	cup	10	12.00	120.00
2	2	2021-05-14	mug	30	8.00	240.00
3	3	2021-05-15	cup	10	10.00	100.00
4	4	2021-05-15	bottle	20	11.00	220.00

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.

Data anomaly occurs when data that is already present is added again with some different values which raises the question as to which data is accurate.

If I type:
SELECT product_name, quantity, total_price
FROM Products
WHERE product_name = 'cup';

It would be unclear how much is the unit price for the cup, because I have two entries for the cup with different unit prices (total_price/quantity).

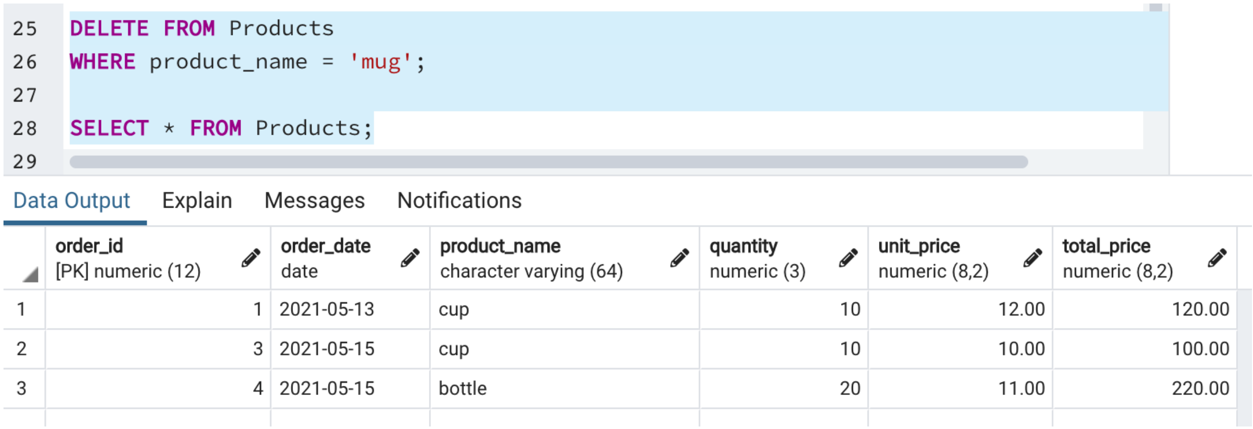


```sql
SELECT product_name, quantity, unit_price, total_price
FROM Products
WHERE product_name = 'cup';
```

	product_name character varying (64)	quantity numeric (3)	unit_price numeric (8,2)	total_price numeric (8,2)
1	cup	10	12.00	120.00
2	cup	10	10.00	100.00

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

If I type:
DELETE FROM Products
WHERE product_name = 'mug';



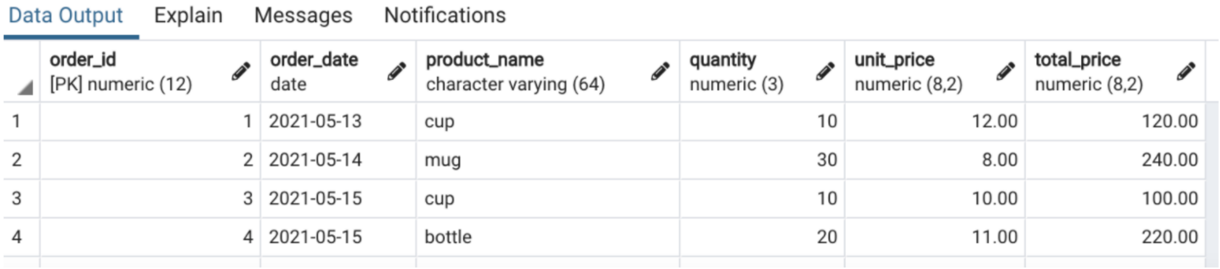
```sql
DELETE FROM Products
WHERE product_name = 'mug';

SELECT * FROM Products;
```

	order_id [PK] numeric (12)	order_date date	product_name character varying (64)	quantity numeric (3)	unit_price numeric (8,2)	total_price numeric (8,2)
1	1	2021-05-13	cup	10	12.00	120.00
2	3	2021-05-15	cup	10	10.00	100.00
3	4	2021-05-15	bottle	20	11.00	220.00

By deleting the mug product in our database, we have no more data related to ‘mug’, it no longer exists in our database, better to have different tables for each product.

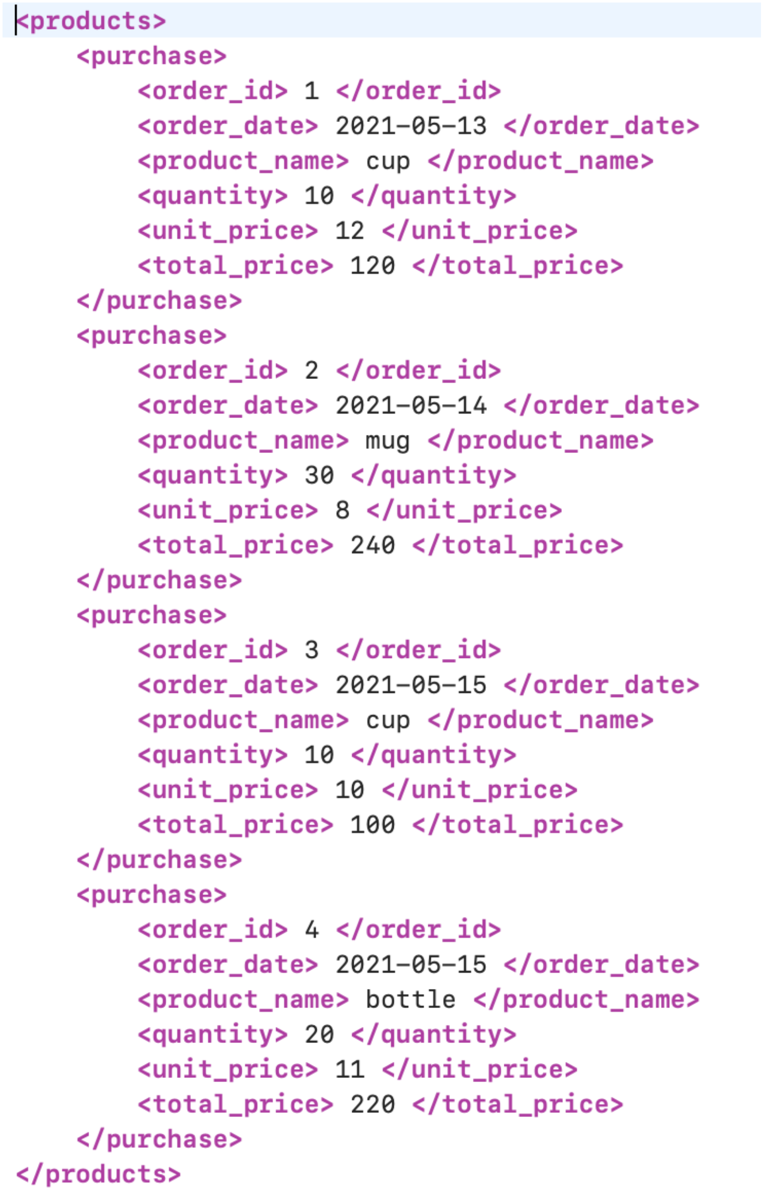
Step 16 – File and Database Table Comparison

Original Table from #15, and will convert the data to xml, json and txt to compare its Efficiency, Security, and Structural Independence.



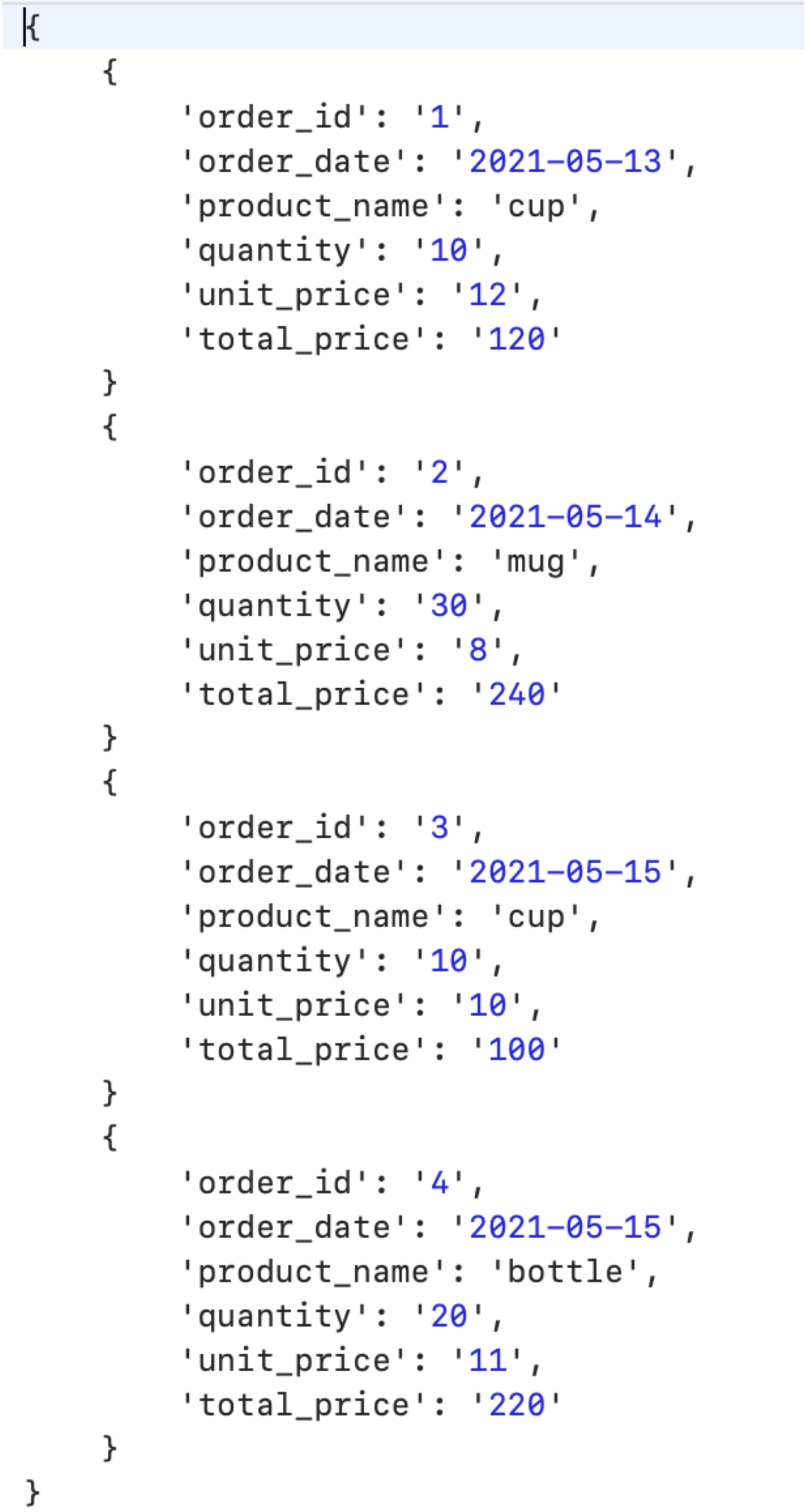
	order_id [PK] numeric (12)	order_date date	product_name character varying (64)	quantity numeric (3)	unit_price numeric (8,2)	total_price numeric (8,2)
1	1	2021-05-13	cup	10	12.00	120.00
2	2	2021-05-14	mug	30	8.00	240.00
3	3	2021-05-15	cup	10	10.00	100.00
4	4	2021-05-15	bottle	20	11.00	220.00

products-sql.xml



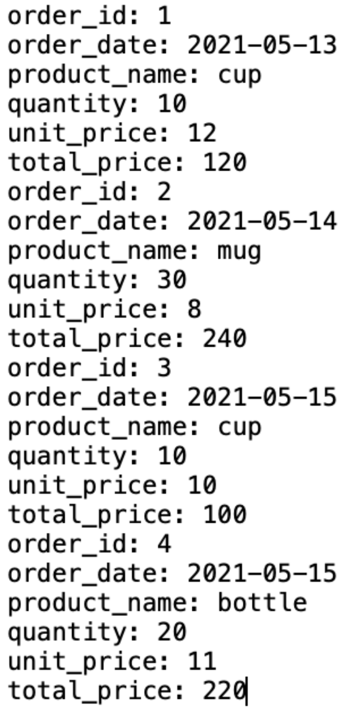
```
<products>
    <purchase>
        <order_id> 1 </order_id>
        <order_date> 2021-05-13 </order_date>
        <product_name> cup </product_name>
        <quantity> 10 </quantity>
        <unit_price> 12 </unit_price>
        <total_price> 120 </total_price>
    </purchase>
    <purchase>
        <order_id> 2 </order_id>
        <order_date> 2021-05-14 </order_date>
        <product_name> mug </product_name>
        <quantity> 30 </quantity>
        <unit_price> 8 </unit_price>
        <total_price> 240 </total_price>
    </purchase>
    <purchase>
        <order_id> 3 </order_id>
        <order_date> 2021-05-15 </order_date>
        <product_name> cup </product_name>
        <quantity> 10 </quantity>
        <unit_price> 10 </unit_price>
        <total_price> 100 </total_price>
    </purchase>
    <purchase>
        <order_id> 4 </order_id>
        <order_date> 2021-05-15 </order_date>
        <product_name> bottle </product_name>
        <quantity> 20 </quantity>
        <unit_price> 11 </unit_price>
        <total_price> 220 </total_price>
    </purchase>
</products>
```

products-sql.json



```
{
    {
        'order_id': '1',
        'order_date': '2021-05-13',
        'product_name': 'cup',
        'quantity': '10',
        'unit_price': '12',
        'total_price': '120'
    }
    {
        'order_id': '2',
        'order_date': '2021-05-14',
        'product_name': 'mug',
        'quantity': '30',
        'unit_price': '8',
        'total_price': '240'
    }
    {
        'order_id': '3',
        'order_date': '2021-05-15',
        'product_name': 'cup',
        'quantity': '10',
        'unit_price': '10',
        'total_price': '100'
    }
    {
        'order_id': '4',
        'order_date': '2021-05-15',
        'product_name': 'bottle',
        'quantity': '20',
        'unit_price': '11',
        'total_price': '220'
    }
}
```

products-sql.txt



```
order_id: 1
order_date: 2021-05-13
product_name: cup
quantity: 10
unit_price: 12
total_price: 120
order_id: 2
order_date: 2021-05-14
product_name: mug
quantity: 30
unit_price: 8
total_price: 240
order_id: 3
order_date: 2021-05-15
product_name: cup
quantity: 10
unit_price: 10
total_price: 100
order_id: 4
order_date: 2021-05-15
product_name: bottle
quantity: 20
unit_price: 11
total_price: 220
```

Efficiency – If there were millions of rows of data, it would be more efficient to access a single record in the relational table than the file, because the relational database is built for speed which allows us to pull back our desired row in seconds while the file takes much longer when there are a lot of data. If we have a big database with millions of records, the file may not even support such a big data file.
Also, the relational table can filter the result of the data. For example, you can check the order that happened on or after this particular date or you only see purchases that are for a specific item, a specific price, etc; which will be much more efficient to manage data in the relation table than file.

Security – It would be easier to securely restrict access to one specific row/record row in the relational table compared to the file, because file systems do not support security on the data within the file. We can set the permission to have certain people access to specific rows in the relational table while everyone could have the chance to access the file with the entire data present there and make changes to it.

Structural Independence – If the table structure was modified by adding or taking away columns, and equivalent changes were made to the file; these changes could affect an app using the table differently than an app using the file. Nothing breaks your SQL unless you actually change the table itself. It does not matter if I move my database from one machine to another, I just connect to the new instance and I can access all the records. However, If the file is moved to a different location, or changes its structure, or order of the element, the application may break. Relational database is structurally independent because applications do not rely on the location of the file, it depends only upon the structure of the table itself, not the file.