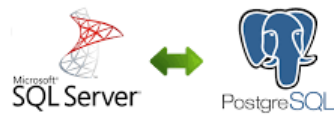




MIS 443: Business Data Management



The ETL Process Explained





Week 9 - Revision Create Insert and check database use PGAdmin



The ETL Process Explained



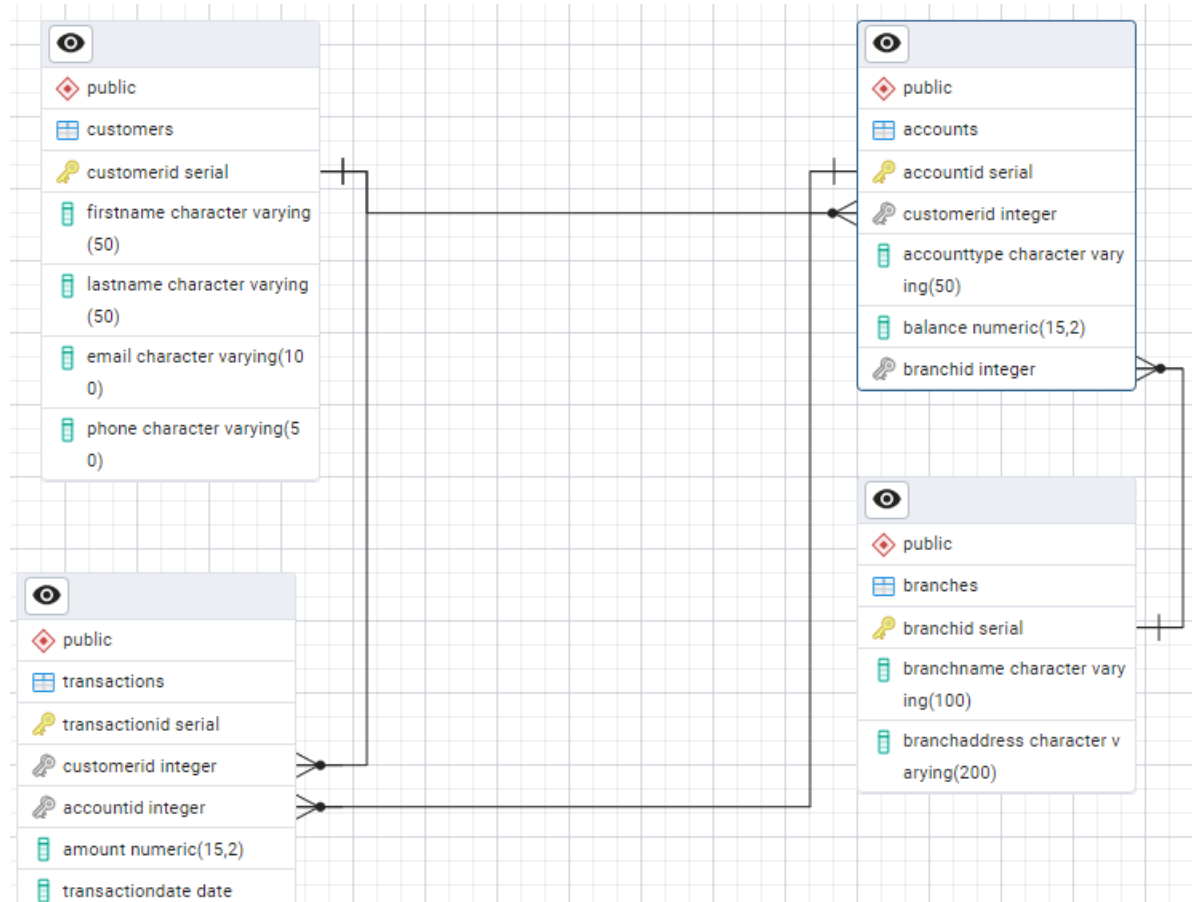
Objective

- ❖ Revision create database, tables
- ❖ Import data use SQL and PGAdmin
- ❖ Check data and ER
- ❖ Practice DML

Scenario

- ❖ Use SQL statements to create a database named “BankingDB” and the four tables described below with appropriate data types and constraints.
- ❖ Ensure each table includes at least one primary key and the Transactions table includes foreign keys that reference the Customers and Accounts tables.
 - Customers: Contains information about customers.
 - Accounts: Contains details of the bank accounts.
 - Transactions: Records each transaction made by the customers linking to the Customers and Accounts tables.
 - Branches: Lists the branches of the bank.

Scenario

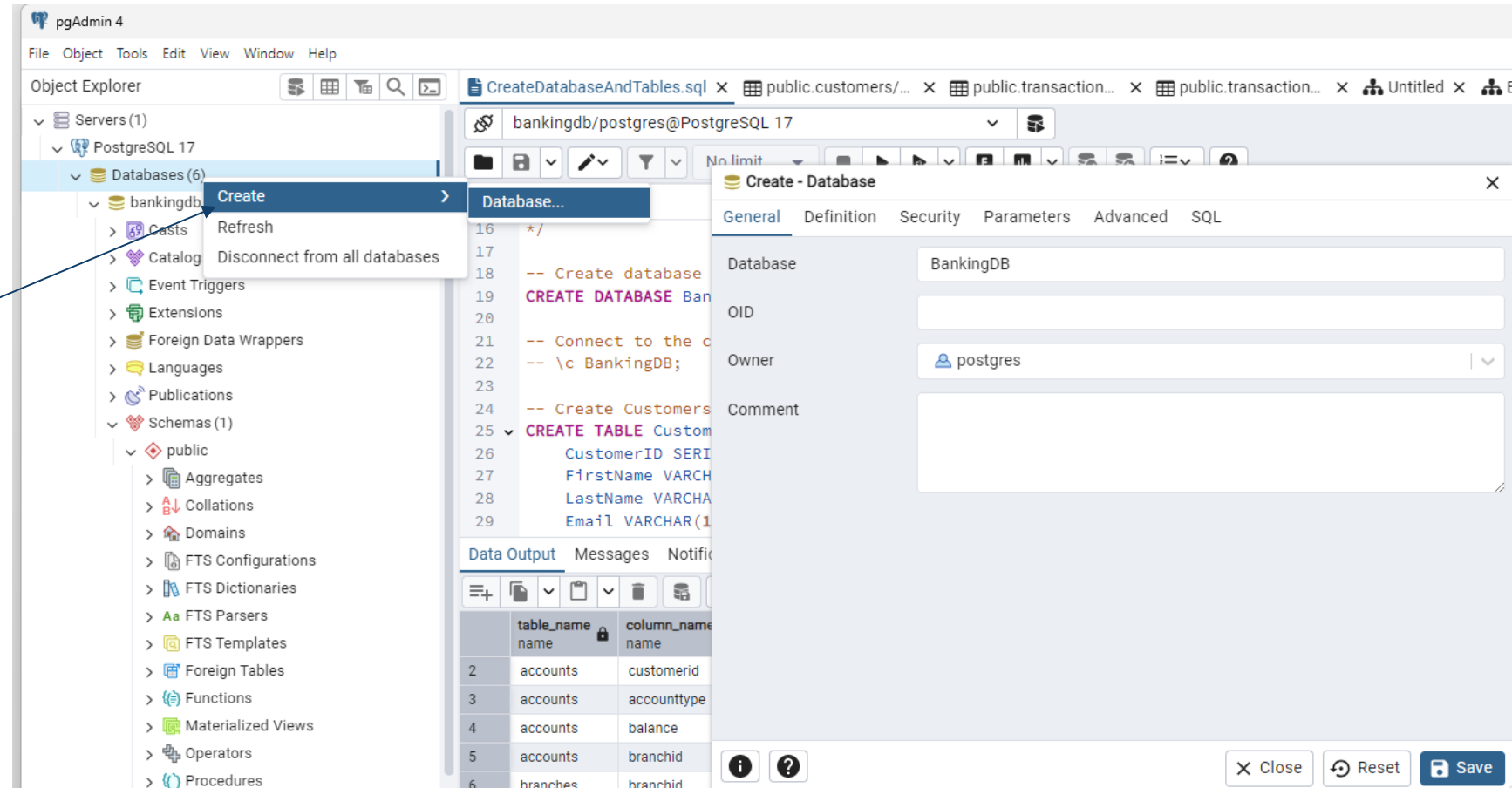


Create Database

❖ Use SQL

```
-- Create database
CREATE DATABASE BankingDB;
```

❖ Use pgAdmin GUI (Right Click on Database → Create → ...



The screenshot shows the pgAdmin 4 interface. On the left, the 'Object Explorer' shows the 'Servers' tree with 'PostgreSQL 17' expanded, and 'Databases (6)' containing 'bankingdb'. A right-click context menu is open over 'bankingdb', with 'Create' selected. The 'Create - Database' dialog box is open, showing the 'General' tab. The 'Database' field is set to 'BankingDB'. The 'Owner' is set to 'postgres'. The 'SQL' tab is active, showing the SQL script for creating the database and tables. The 'Data Output' tab shows a table with columns 'table_name' and 'column_name'.

	table_name	column_name
2	accounts	customerid
3	accounts	accounttype
4	accounts	balance
5	accounts	branchid
6	branches	branchid

Create Tables

- Remember SQL command will difference when use difference platform



```
-- Create Customers table
CREATE TABLE Customers (
    CustomerID SERIAL PRIMARY KEY,
    FirstName VARCHAR(50),
    LastName VARCHAR(50),
    Email VARCHAR(100),
    Phone VARCHAR(50)
);
```

```
-- Create Branches table
CREATE TABLE Branches (
    BranchID SERIAL PRIMARY KEY,
    BranchName VARCHAR(100),
    BranchAddress VARCHAR(200)
);
```

**Create table
with primary key
first**

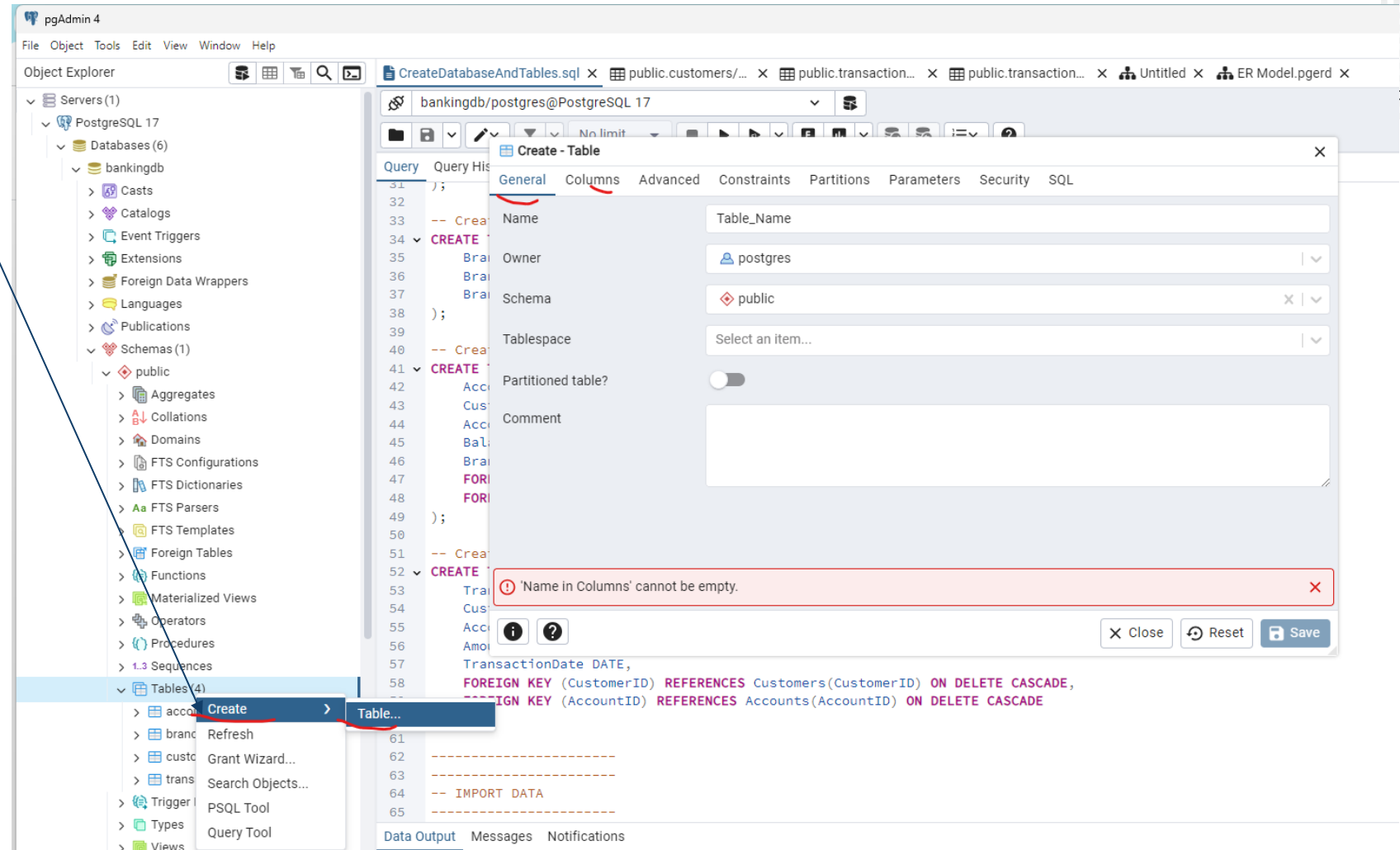
```
-- Create Accounts table
CREATE TABLE Accounts (
    AccountID SERIAL PRIMARY KEY,
    CustomerID INT,
    AccountType VARCHAR(50),
    Balance DECIMAL(15, 2),
    BranchID INT,
    FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID) ON DELETE CASCADE,
    FOREIGN KEY (BranchID) REFERENCES Branches(BranchID) ON DELETE SET NULL
);
```

```
-- Create Transactions table
CREATE TABLE Transactions (
    TransactionID SERIAL PRIMARY KEY,
    CustomerID INT,
    AccountID INT,
    Amount DECIMAL(15, 2),
    TransactionDate DATE,
    FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID) ON DELETE CASCADE,
    FOREIGN KEY (AccountID) REFERENCES Accounts(AccountID) ON DELETE CASCADE
);
```

Create Tables

❖ Use pgAdmin

Not recommend



Check tables and attributes

-- Check tables and attributes

```
SELECT table_name, column_name, data_type
FROM information_schema.columns
WHERE table_schema = 'public'
ORDER BY table_name, ordinal_position;
```

```
97 -- Check tables and attributes
98 SELECT table_name, column_name, data_ty
99 FROM information_schema.columns
100 WHERE table_schema = 'public'
101 ORDER BY table_name, ordinal_position;
102
```

Data Output Messages Notifications

	table_name	column_name	data_type
	name	name	character varying
2	accounts	customerid	integer
3	accounts	accounttype	character varying
4	accounts	balance	numeric
5	accounts	branchid	integer
6	branches	branchid	integer
7	branches	branchname	character varying
8	branches	branchaddress	character varying
9	customers	customerid	integer
10	customers	firstname	character varying
11	customers	lastname	character varying
12	customers	email	character varying
13	customers	phone	character varying
14	transactions	transactionid	integer
15	transactions	customerid	integer
16	transactions	accountid	integer
17	transactions	amount	numeric
18	transactions	transactiondate	date

Import data into Tables

❖ Use SQL

```
-- Import data into Customers table
COPY Customers(CustomerID, FirstName, LastName, Email, Phone)
FROM 'D:/MIS 443 - Business Data Management/Slide Teaching Q4 2023-2024/Final Exam/csv/Customers_final.csv'
DELIMITER ','
CSV HEADER;

-- Import data into Branches table
COPY Branches(BranchID, BranchName, BranchAddress)
FROM 'D:/MIS 443 - Business Data Management/Slide Teaching Q4 2023-2024/Final Exam/csv/Branches_final.csv'
DELIMITER ','
CSV HEADER;

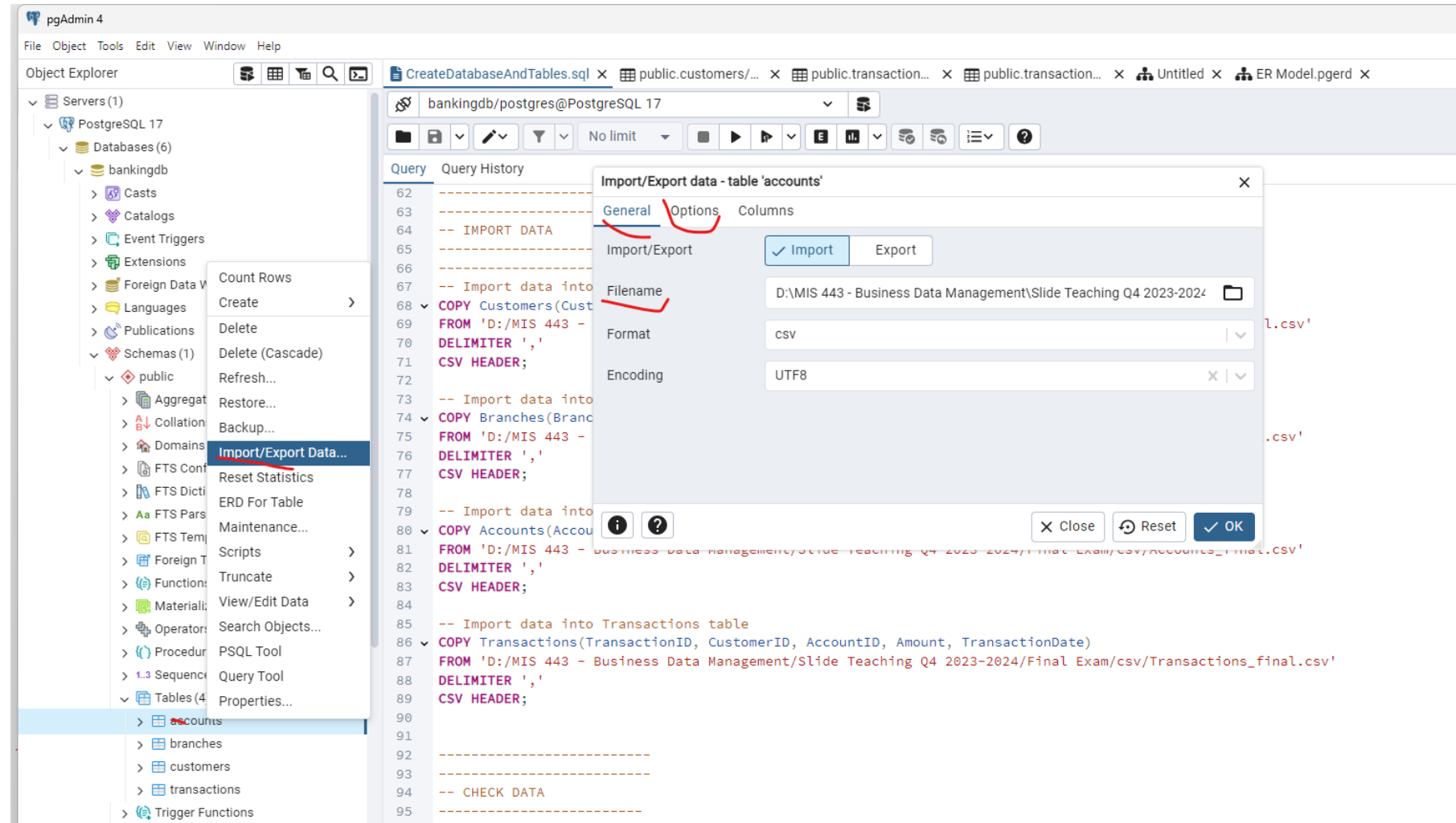
-- Import data into Accounts table
COPY Accounts(AccountID, CustomerID, AccountType, Balance, BranchID)
FROM 'D:/MIS 443 - Business Data Management/Slide Teaching Q4 2023-2024/Final Exam/csv/Accounts_final.csv'
DELIMITER ','
CSV HEADER;

-- Import data into Transactions table
COPY Transactions(TransactionID, CustomerID, AccountID, Amount, TransactionDate)
FROM 'D:/MIS 443 - Business Data Management/Slide Teaching Q4 2023-2024/Final Exam/csv/Transactions_final.csv'
DELIMITER ','
CSV HEADER;
```

Import data into Tables

❖ Use pgAdmin

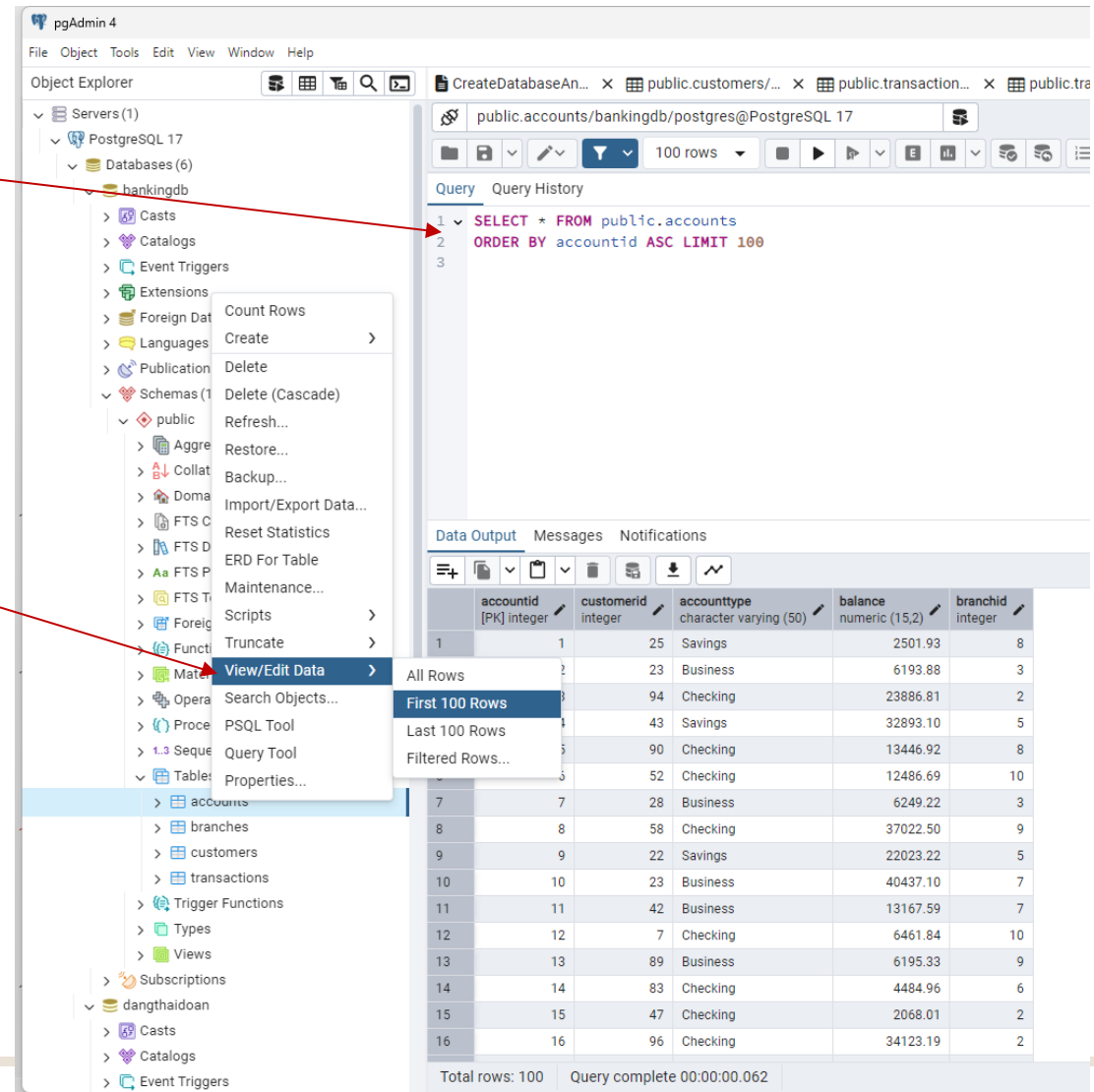
**Not
recommend**



Always check data after import

❖ Use SQL

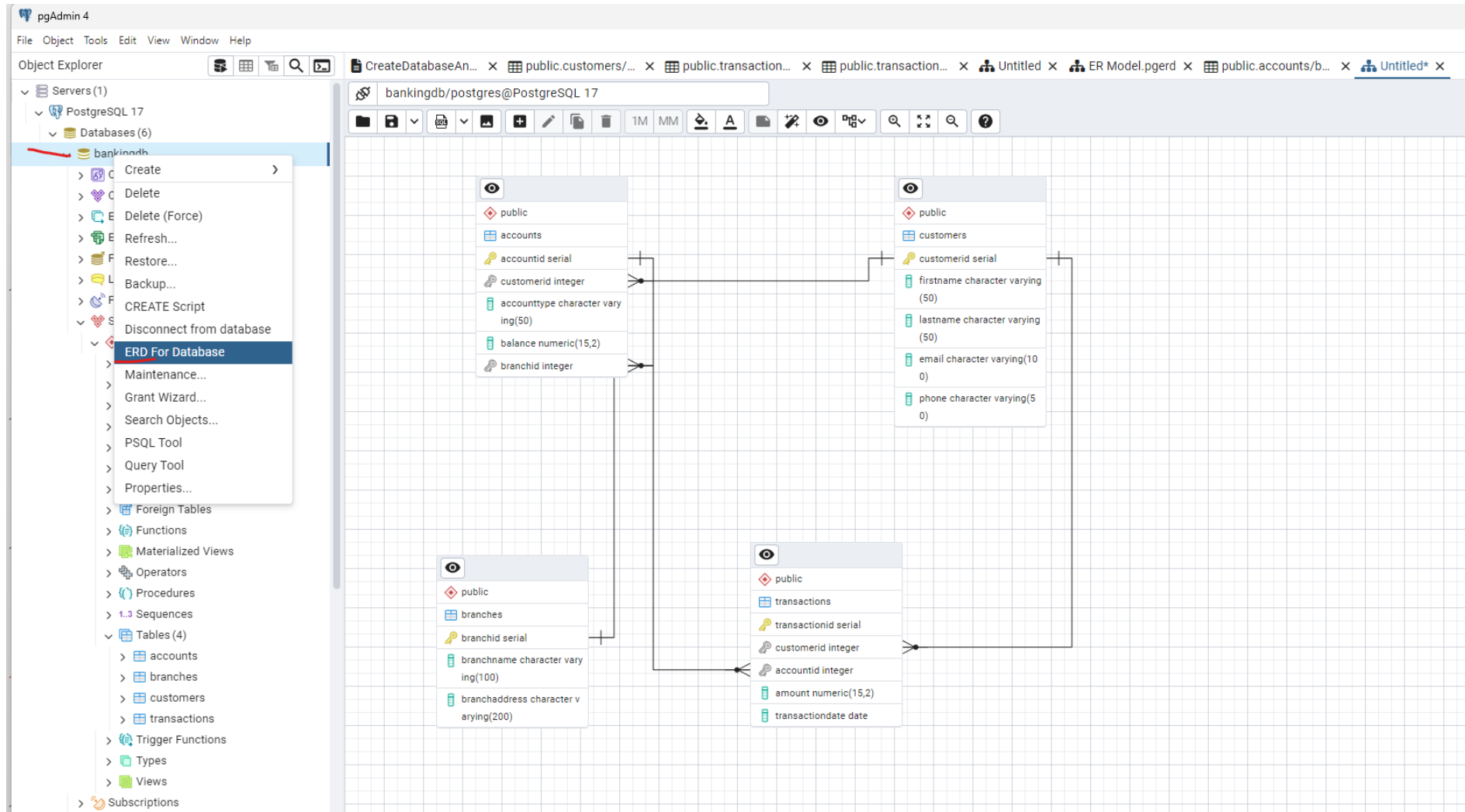
❖ Use GUI



The screenshot shows the pgAdmin 4 interface. On the left, the Object Explorer shows the database structure. A right-click context menu is open over the 'accounts' table in the 'public' schema, with the 'View/Edit Data' option selected. The main pane displays a SQL query: `SELECT * FROM public.accounts ORDER BY accountid ASC LIMIT 100`. Below the query, the 'Data Output' tab shows a table with 100 rows of data. The table has columns: accountid [PK] integer, customerid integer, accounttype character varying (50), balance numeric (15,2), and branchid integer. The status bar at the bottom indicates 'Total rows: 100' and 'Query complete 00:00:00.062'.

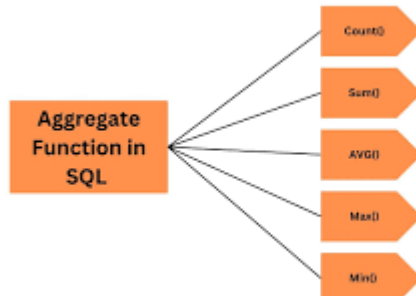
accountid [PK] integer	customerid integer	accounttype character varying (50)	balance numeric (15,2)	branchid integer
1	1	25 Savings	2501.93	8
2	2	23 Business	6193.88	3
3	3	94 Checking	23886.81	2
4	4	43 Savings	32893.10	5
5	5	90 Checking	13446.92	8
6	6	52 Checking	12486.69	10
7	7	28 Business	6249.22	3
8	8	58 Checking	37022.50	9
9	9	22 Savings	22023.22	5
10	10	23 Business	40437.10	7
11	11	42 Business	13167.59	7
12	12	7 Checking	6461.84	10
13	13	89 Business	6195.33	9
14	14	83 Checking	4484.96	6
15	15	47 Checking	2068.01	2
16	16	96 Checking	34123.19	2

Check Diagram and Relationship



Order	Element
5	SELECT
1	FROM
2	WHERE
3	GROUP BY
4	HAVING
6	ORDER BY

Now We Are Ready For Retrieve Data



LAG()	current row	LEAD()
0	a	b
a	b	c
b	c	d
c	d	0

LEFT JOIN



Everything on the left + anything on the right that matches

```
SELECT *
FROM TABLE_1
LEFT JOIN TABLE_2
ON TABLE_1.KEY = TABLE_2.KEY
```

ANTI LEFT JOIN



Everything on the left that is NOT on the right

```
SELECT *
FROM TABLE_1
LEFT JOIN TABLE_2
ON TABLE_1.KEY = TABLE_2.KEY
WHERE TABLE_2.KEY IS NULL
```

RIGHT JOIN



Everything on the right + anything on the left that matches

```
SELECT *
FROM TABLE_1
RIGHT JOIN TABLE_2
ON TABLE_1.KEY = TABLE_2.KEY
```

ANTI RIGHT JOIN



Everything on the right that is NOT on the left

```
SELECT *
FROM TABLE_1
RIGHT JOIN TABLE_2
ON TABLE_1.KEY = TABLE_2.KEY
WHERE TABLE_1.KEY IS NULL
```

OUTER JOIN



Everything on the right + Everything on the left

```
SELECT *
FROM TABLE_1
OUTER JOIN TABLE_2
ON TABLE_1.KEY = TABLE_2.KEY
```

ANTI OUTER JOIN



Everything on the left and right that is unique to each side

```
SELECT *
FROM TABLE_1
OUTER JOIN TABLE_2
ON TABLE_1.KEY = TABLE_2.KEY
WHERE TABLE_1.KEY IS NULL
OR TABLE_2.KEY IS NULL
```

INNER JOIN



Only the things that match on the left AND the right

```
SELECT *
FROM TABLE_1
INNER JOIN TABLE_2
ON TABLE_1.KEY = TABLE_2.KEY
```

CROSS JOIN



All combination of rows from the right and the left (cartesian product)

```
SELECT *
FROM TABLE_1
CROSS JOIN TABLE_2
```

Question 1

Question 1 (10 marks): Create a database named BankingDB and the four tables: Customers, Accounts, Transactions, and Branches using SQL statements. Ensure each table includes appropriate primary and foreign keys, and data types. Submit the SQL script as part of your answer.

```
SELECT table_name, column_name, data_type  
FROM information_schema.columns  
WHERE table_schema = 'public'  
ORDER BY table_name, ordinal_position;
```

Question 2

- ❖ Question 2 (10 marks): List all transactions that occurred in the year 2024 displaying the TransactionID, CustomerID, AccountID, and TransactionDate. Arrange the results by TransactionDate in ascending order.

```
SELECT TransactionID, CustomerID, AccountID, TransactionDate
FROM Transactions
WHERE EXTRACT(YEAR FROM TransactionDate) = 2024
ORDER BY TransactionDate ASC;
```


Question 3

- ❖ Question 3 (20 marks): Calculate the total number of transactions made by each customer. Show the customer's ID, name, and their total number of transactions. Display the top 5 customers with the highest number of transactions. Order the results by the total number of transactions in descending order.

```
SELECT Customers.CustomerID, Customers.FirstName, Customers.LastName, COUNT(Transactions.TransactionID) AS TotalTransactions
FROM Customers
JOIN Transactions ON Customers.CustomerID = Transactions.CustomerID
GROUP BY Customers.CustomerID, Customers.FirstName, Customers.LastName
ORDER BY TotalTransactions DESC
LIMIT 5;
```

Question 4

- ❖ Question 4 (20 marks): Display of the top 5 customers who made the most recent transactions. Include the customer's ID, name, and the date of their most recent transaction. For customers who haven't made any transactions, their last transaction date should be shown as NULL. Order the list by the date of the last transaction in descending order.
- ❖ Notes: *Using a Common Table Expression (CTE)*

```
-- Get top 5 customers with the latest transactions
SELECT Customers.CustomerID, Customers.FirstName, Customers.LastName, MAX(Transactions.TransactionDate) AS LastTransactionDate
FROM Customers
LEFT JOIN Transactions ON Customers.CustomerID = Transactions.CustomerID
GROUP BY Customers.CustomerID, Customers.FirstName, Customers.LastName
ORDER BY LastTransactionDate DESC
LIMIT 5;

-- Using a Common Table Expression (CTE) to get the latest transaction date per customer
WITH LatestTransaction AS (
    SELECT CustomerID, MAX(TransactionDate) AS LastTransactionDate
    FROM Transactions
    GROUP BY CustomerID
)
SELECT c.CustomerID, c.FirstName, c.LastName, lt.LastTransactionDate
FROM Customers c
LEFT JOIN LatestTransaction lt ON c.CustomerID = lt.CustomerID
ORDER BY lt.LastTransactionDate DESC
LIMIT 5;
```

Question 5

- ❖ Question 5 (20 marks): List each transaction, including the customer's ID, name, account type, amount, and the transaction date. Order the results by transaction date in descending order.

```
SELECT Customers.FirstName, Customers.LastName, Accounts.AccountType, Transactions.Amount, Transactions.TransactionDate
FROM Transactions
JOIN Customers ON Transactions.CustomerID = Customers.CustomerID
JOIN Accounts ON Transactions.AccountID = Accounts.AccountID
ORDER BY Transactions.TransactionDate DESC;
```

Question 6

- ❖ Question 6 (20 marks): Rank the branches based on the total amount of transactions handled. Display the branch name, total transaction amount, and its rank. Branches with the same transaction amount should share the same rank.

```
SELECT Branches.BranchName, SUM(Transactions.Amount) AS TotalTransactionAmount,  
       RANK() OVER (ORDER BY SUM(Transactions.Amount) DESC) AS Ranking  
FROM Transactions  
JOIN Accounts ON Transactions.AccountID = Accounts.AccountID  
JOIN Branches ON Accounts.BranchID = Branches.BranchID  
GROUP BY Branches.BranchID, Branches.BranchName  
ORDER BY TotalTransactionAmount DESC;
```