

JAY ARRE TALOSIG
COM-231
Mrs. Jensen Santillan

Objectives

By the end of this activity, students will be able to:

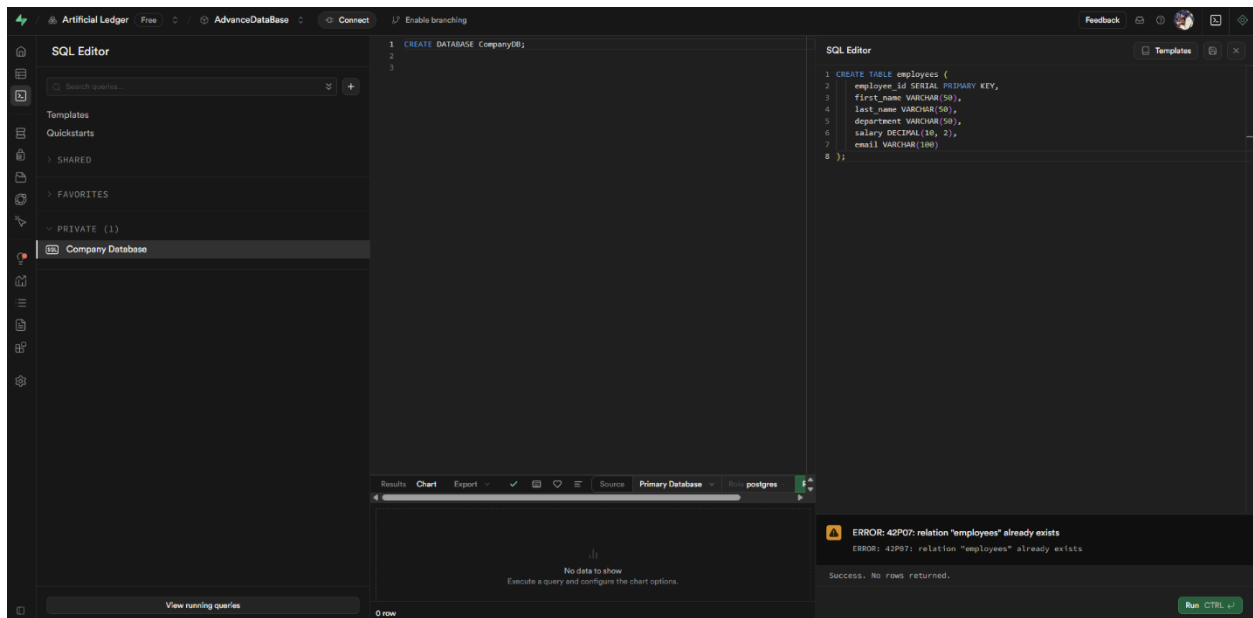
- Create and use Clustered and Non-clustered Indexes.
- Optimize queries using Composite and Covering Indexes.
- Work with Views, including Simple, Updatable, and Indexed Views.

Part 1: Setting Up the Database and Table

Step 1: Create a New Database

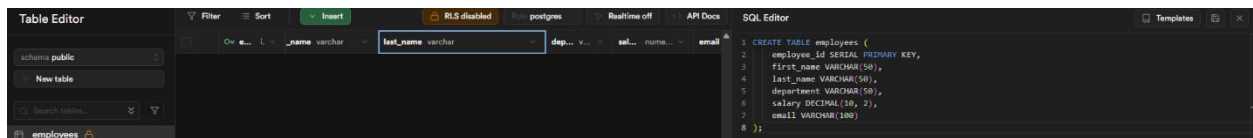
1. Open SQL Server Management Studio (SSMS).
2. Click New Query and enter:
CREATE DATABASE Company;
3. Execute the query by clicking F5 or the Execute button.
4. Select the new database to work with:

USE CompanyDB;



Part 2: Creating the Employees Table

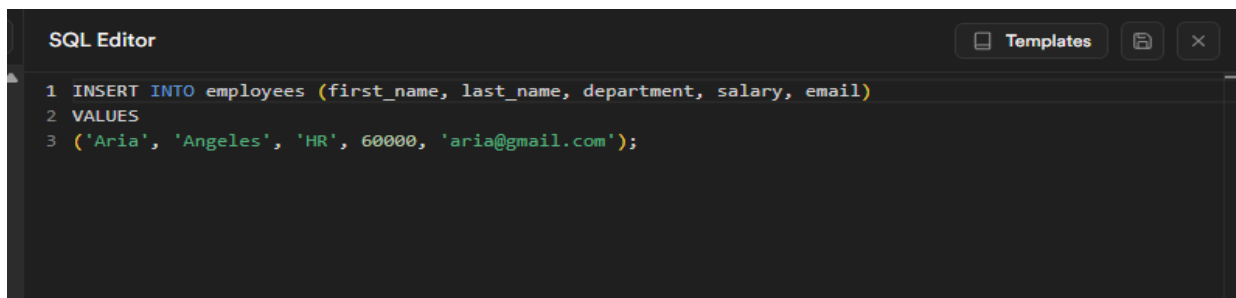
Step 2: Create an Employee Table



Part 3: Inserting Sample Data

Step 3: Insert Sample Employees

Run the following query to add employee records:



AdvanceDataBase Connect Enable branching Feedback 🔍 📄 ⚙️

Filter Sort Insert RLS disabled Role postgres Realtime off API Docs SQL Editor Templates ×

		first_name varchar	last_name varchar	dep... v...	sal... nume...	email varchar	+
<input type="checkbox"/>	2	Aria	Angeles	HR	60000.00	aria@email.com	
<input type="checkbox"/>	3	Klay	Santos	IT	70000.00	klay@email.com	
<input type="checkbox"/>	4	Noel	Santos	Finance	55000.00	noel@email.com	
<input type="checkbox"/>	5	Yuri	Hanamitchi	IT	80000.00	yuri@email.com	
<input type="checkbox"/>	6	Jasper Jean	Mariano	HR	62000.00	jayjay@email.com	
<input type="checkbox"/>	7	Kiefer	Watsons	Finance	58000.00	kiefer@email.com	

```

1 INSERT INTO employees (first_name, last_name, department, salary, email)
2 VALUES
3 ('Aria', 'Angeles', 'HR', 60000, 'aria@email.com'),
4 ('Klay', 'Santos', 'IT', 70000, 'klay@email.com'),
5 ('Noel', 'Santos', 'Finance', 55000, 'noel@email.com'),
6 ('Yuri', 'Hanamitchi', 'IT', 80000, 'yuri@email.com'),
7 ('Jasper Jean', 'Mariano', 'HR', 62000, 'jayjay@email.com'),
8 ('Kiefer', 'Watsons', 'Finance', 58000, 'kiefer@email.com');

```

Part 4: Indexing

Step 4: Create a Clustered Index

Run the following SQL:

```
CREATE CLUSTERED INDEX IDX_EmployeeID
ON Employees(EmployeeID);
```

TASK: Run the following **before and after** adding the index to compare execution time:

```
SELECT * FROM Employees
WHERE EmployeeID = 3;
```

SQL Editor

1 `SELECT * FROM Employees`
2 `WHERE employees.employee_id = 3;`

employee_id	first_name	last_name	department	salary	email
3	Klay	Santos	IT	70000.00	klay@email.com

1 rows

Hide Results

Run CTRL ↵

Step 5: Create a Non-Clustered Index

Run this SQL to speed up searches on **LastName**

```
CREATE NONCLUSTERED INDEX IDX_LastName  
ON Employees(LastName);
```

TASK: Run the following query before and after indexing

```
SELECT * FROM Employees  
WHERE LastName = 'Watsons';
```

SQL Editor

1 `SELECT * FROM Employees`
2 `WHERE employees.employee_id = 3;`

employee_id	first_name	last_name	department	salary	email
3	Klay	Santos	IT	70000.00	klay@email.com

1 rows

Hide Results

Run CTRL ↵

Step 6: Create a Composite Index

```
CREATE NONCLUSTERED INDEX IDX_DeptSalary  
ON Employees(Department, Salary);
```

Task: Run and Compare

```
SELECT * FROM Employees  
WHERE Department = 'IT' AND Salary > 60000;
```

SQL Editor

1 SELECT * FROM Employees
2 WHERE Department = 'IT' AND Salary > 60000;

employee_id	first_name	last_name	department	salary	email
3	Klay	Santos	IT	70000.00	klay@email.com
5	Yuri	Hanamitchi	IT	80000.00	yuri@email.com

2 rows

Hide Results

Run CTRL ↵

Step 7: Create a Covering Index

Create an index that covers **FirstName and LastName** when searching by Department:

```
CREATE NONCLUSTERED INDEX IDX_Covering  
ON Employees(Department) INCLUDE (FirstName, LastName);
```

TASK: Run the query before and after the indexing:

```
SELECT FirstName, LastName  
FROM Employees  
WHERE Department = 'HR'
```

The screenshot shows a database management interface. On the left, a table named 'employees' is displayed with columns: employee_id, first_name, last_name, and department. The table contains 6 records. On the right, the SQL Editor shows a query: `SELECT employees.first_name, employees.last_name FROM Employees WHERE Department = 'HR';`. The query results show 2 rows: (Aria, Angeles) and (Jasper Jean, Mariano). The interface also includes a Filter, Sort, and Insert menu, a status bar at the bottom, and a Run button.

employee_id	first_name	last_name	department
2	Aria	Angeles	HR
3	Klay	Santos	IT
4	Noel	Santos	Finance
5	Yuri	Hanamitchi	IT
6	Jasper Jean	Mariano	HR
7	Kiefer	Watsons	Finance

```
1 SELECT employees.first_name, employees.last_name
2 FROM Employees
3 WHERE Department = 'HR';
```

first_name	last_name
Aria	Angeles
Jasper Jean	Mariano

2 rows

Page 1 of 1 100 rows 6 records Refresh Data Definition Run CTRL + R

Step 8: Create a Unique Index

Ensure that **Email** is unique

```
ALTER TABLE Employees
ADD CONSTRAINT UQ_Email UNIQUE (Email);
```

TASK: Try inserting a duplicate email to see if it prevents it.

Part 5: Creating Views

Step 9: Create a Simple View

Create a view to show **high salary employees**:

```
CREATE VIEW vw_HighSalary AS
SELECT EmployeeID, FirstName, LastName, Salary
```

FROM Employees
WHERE Salary > 60000;

TASK: Run:

SELECT * FROM vw_HighSalary;

The screenshot shows a database management interface. On the left, a table view displays employee data. The table has columns: employee_id, first_name, last_name, and salary. The data rows are:

employee_id	first_name	last_name	salary
3	Klay	Santos	70000.00
5	Yuri	Hanamitchi	80000.00
6	Jasper Jean	Mariano	62000.00

On the right, the SQL Editor shows the query: `1 SELECT * FROM vw_HighSalary;`. Below the editor, a preview of the query results is shown, matching the table view on the left. At the bottom, there are controls for pagination (Page 1 of 1, 100 rows, 2.0 records estimated), a Refresh button, and a Run button (CTRL + R).

Insert a new employee with **Salary > 60,000** and check if they appear in the view.

Step 10: Create a View with Aggregation

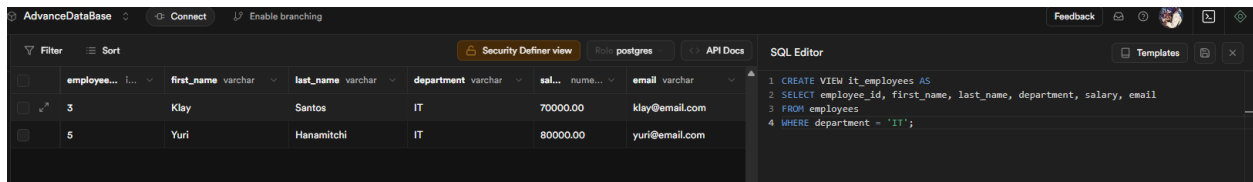
Create a view for **average salary per department**:

CREATE VIEW vw_AvgSalary AS
SELECT Department, AVG(Salary) AS AvgSalary
FROM Employees
GROUP BY Department;

Step 11: Create an Updatable View

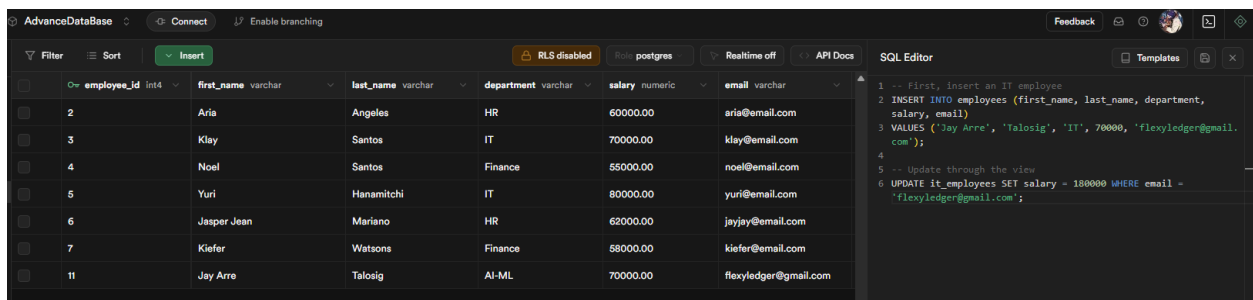
Create a **filtered view** for IT employees

```
CREATE VIEW vw_ITEmployees AS  
SELECT EmployeeID, FirstName, LastName, Salary  
FROM Employees  
WHERE Department = 'IT'  
WITH CHECK OPTION;
```



TASK: Try updating an employee's salary in this view:

```
UPDATE vw_ITEmployees  
SET Salary = 90000  
WHERE EmployeeID = 2;
```



Try inserting an employee from HR in the view. Does it work? Why not?

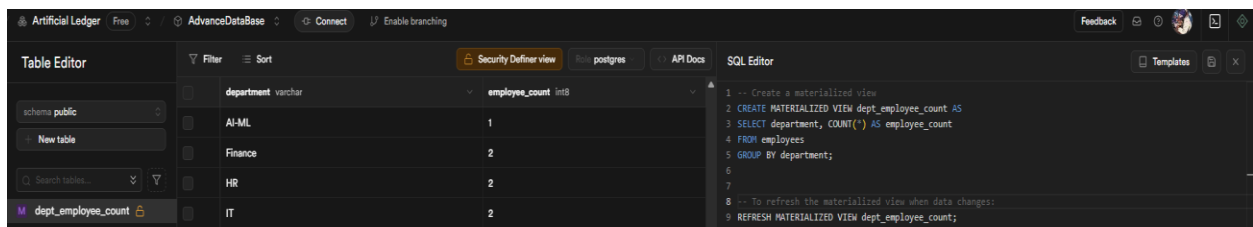
Step 12: Create an Indexed View

Create a view with indexing:

```

CREATE VIEW vw_IndexedSalary WITH SCHEMABINDING
AS
SELECT Department, COUNT_BIG(*)
AS EmpCount, AVG(Salary) AS AvgSalary
FROM dbo.Employees
GROUP BY Department;

```



Add a Clustered Index:

```

CREATE UNIQUE CLUSTERED INDEX IDX_vw_IndexedSalary
ON vw_IndexedSalary(Department);

```

TASK: Run and Compare query speed before and after.

```

SELECT * FROM vw_IndexedSalary
WHERE Department = 'HR';

```

Part 6: Joins and Indexes

Step 13: Optimized Joins

1. Create a Departments Table:

```

CREATE TABLE Departments (

```

```
DeptID INT PRIMARY KEY,  
    DeptName NVARCHAR(50)  
);
```

2. Insert sample data.

```
INSERT INTO Departments  
VALUES (1, 'HR'), (2, 'IT'), (3, 'Finance');
```

3. Add DeptID to Employees.

```
ALTER TABLE Employees ADD DeptID INT;
```

4. Assign DeptID values.

```
UPDATE Employees SET DeptID = 1 WHERE Department = 'HR';  
UPDATE Employees SET DeptID = 2 WHERE Department = 'IT';  
UPDATE Employees SET DeptID = 3 WHERE Department = 'Finance';
```

5. Optimize the join:

```
CREATE NONCLUSTERED INDEX IDX_DeptID  
ON Employees(DeptID);
```

The screenshot displays a database management interface. On the left, a table view for 'dept_name' is shown with columns 'dept_name' and 'location'. It contains four rows: (1, 'HR', 'Floor 1'), (2, 'IT', 'Floor 2'), (3, 'Engineering', 'Floor 3'), and (4, 'Marketing', 'Floor 4'). On the right, the SQL Editor shows a script with 31 lines of SQL code. The script includes comments and SQL statements for creating a 'departments' table, inserting sample data, adding a 'dept_id' column to the 'employees' table, creating a foreign key relationship, and creating a non-clustered index 'IDX_DeptID' on the 'employees' table.

```
1 -- Create a Departments Table  
2 CREATE TABLE departments (  
3     dept_id SERIAL PRIMARY KEY,  
4     dept_name VARCHAR(50),  
5     location VARCHAR(100)  
6 );  
7  
8 -- Insert sample data  
9 INSERT INTO departments (dept_name, location)  
10 VALUES  
11 ('HR', 'Floor 1'),  
12 ('IT', 'Floor 2'),  
13 ('Engineering', 'Floor 3'),  
14 ('Marketing', 'Floor 4');  
15  
16 -- Add dept_id to Employees  
17 ALTER TABLE employees ADD COLUMN dept_id INTEGER;  
18  
19 -- Create a foreign key relationship  
20 ALTER TABLE employees  
21 ADD CONSTRAINT fk_employees_departments  
22 FOREIGN KEY (dept_id) REFERENCES departments(dept_id);  
23  
24 -- Assign dept_id values based on department names  
25 UPDATE employees e  
26 SET dept_id = d.dept_id  
27 FROM departments d  
28 WHERE e.department = d.dept_name;  
29  
30 -- Create an index to optimize joins  
31 CREATE INDEX idx_employees_dept_id ON employees(dept_id);
```

TASK: Run and Compare:

```
SELECT e.EmployeeID, e.FirstName, e.LastName, d.DeptName  
FROM Employees e  
JOIN Departments d ON e.DeptID = d.DeptID;
```

SQL Editor

Templates

```
1 SELECT e.first_name, e.last_name, d.dept_name, d.location
2 FROM employees e
3 JOIN departments d ON e.dept_id = d.dept_id
4 WHERE d.location = 'Floor 2';
```

first_name	last_name	dept_name	location
Yuri	Hanamitchi	IT	Floor 2
Klay	Santos	IT	Floor 2

2 rows

Hide Results

Run CTRL ↵

Artificial LedgerFree

AdvanceDatabaseConnectEnable branching

Table Editor

FilterSortInsertRLS disabledpostgresRealtime offAPI Docs

schema public

New table

Search tables

departments

dept_employee_count

employees

inventory_logs

it_employees

medications

schema_change_logs

schema_objects

user_access_logs

vw_mgrsalary

vw_highsalary

1

2

3

4

dept_name

location

HR

IT

Engineering

Marketing

Floor 1

Floor 2

Floor 3

Floor 4

SQL Editor

Templates

1 SELECT e.first_name, e.last_name, d.dept_name, d.location
FROM employees e
3 JOIN departments d ON e.dept_id = d.dept_id
4 WHERE d.location = 'Floor 2';

first_name

last_name

dept_name

location

Vuri

Hananitchi

IT

Floor 2

Klay

Santos

IT

Floor 2

2 rows

Hide Results

Page 1 of 1100 rows4 records

RefreshDataDefinition

Run CTRL + R