**Задание 1: DML**

● Цель: Практика базовой вставки, выборки, простого обновления и удаления

данных.

● Действия:

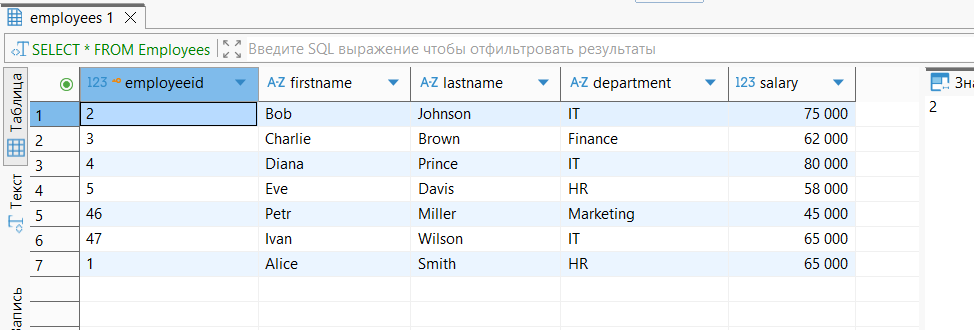
1. Вставить двух новых сотрудников в таблицу Employees.



INSERT INTO Employees (FirstName, LastName, Department, Salary): Указывает в какую таблицу будут вставлены данные и перечисляет столбцы в которые будут предоставлены значения.

VALUES ('Petr', 'Miller', 'Marketing', 45000.00), ('Ivan', 'Wilson', 'IT, 65000.00): Указывает значения, которые будут вставлены.

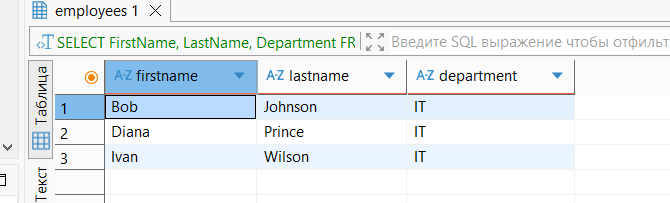
2. Выбрать всех сотрудников из таблицы Employees.



SELECT \*: Выбирает \*все столбцы\* из таблицы.

FROM Employees: Указывает таблицу Employees, которую необходимо отобразить.

3. Выбрать только FirstName и LastName сотрудников из отдела 'IT'.



SELECT FirstName, LastName, Department: Выбирает только столбцы FirstName, LastName, Department из таблицы.

FROM Employees: Указывает таблицу Employees из которые берутся данные.

WHERE Department = 'IT': Фильтрует результаты, чтобы включить только тех сотрудников, у которых значение столбца Department равно 'IT'.

4. Обновить Salary 'Alice Smith' до 65000.00.

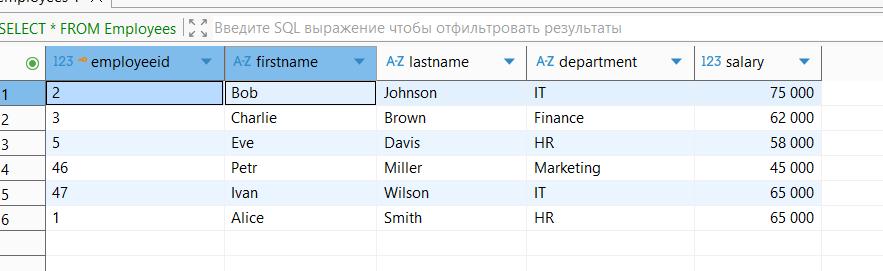


UPDATE Employees: Указывает, что мы хотим обновить данные в таблице Employees.

SET Salary = 65000.00: Устанавливает новое значение для столбца Salary.

WHERE FirstName = 'Alice' AND LastName = 'Smith' определяет, какие строки должны быть обновлены.

5. Удалить сотрудника, чья LastName — 'Prince'.



Необходимо было использовать **ON DELETE CASCADE** при создании внешнего ключа, поэтому сначала удаляем существующий внешний ключ **ALTER TABLE EmployeeProjects DROP CONSTRAINT employeeprojects\_employeeid\_fkey** и создаем новый **ALTER TABLE EmployeeProjects**

**ADD CONSTRAINT employeeprojects\_employeeid\_fkey**

**FOREIGN KEY (EmployeeID) REFERENCES Employees(EmployeeID) ON DELETE CASCADE;**

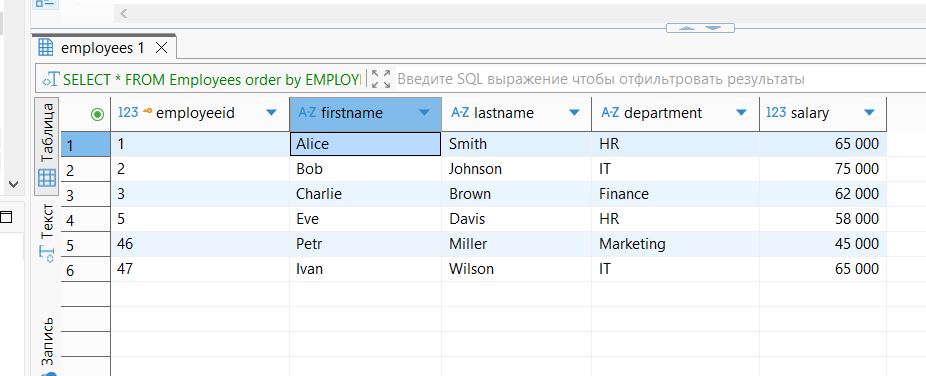
А затем удаляем сотрудника

**DELETE FROM Employees WHERE LastName = 'Prince';**

DELETE FROM Employees: Указывает, что нужно удалить строку из таблицы Employees.

WHERE LastName = 'Prince': WHERE указывает, какую строку необходимо удалить.

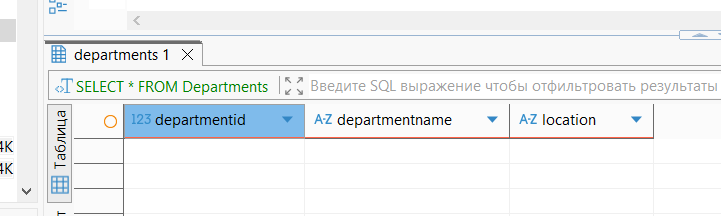
6. Проверить все изменения, используя SELECT \* FROM Employees;.



**Задание 2: DDL**

● Цель: Практика создания и изменения структуры таблиц.

● Действия:

1. Создать новую таблицу с именем Departments со столбцами: DepartmentID (SERIAL PRIMARY KEY), DepartmentName (VARCHAR(50), UNIQUE, NOT NULL), Location (VARCHAR(50)). 

2. Изменить таблицу Employees, добавив новый столбец с именем Email (VARCHAR(100)).

ALTER TABLE Employees: Указывает, что мы хотим изменить структуру таблицы Employees.

ADD COLUMN Email VARCHAR(100): Добавляет новый столбец с именем Email в таблицу Employees. Тип данных столбца - VARCHAR(100), что означает строку переменной длины с максимальной длиной 100 символов.

3. Добавить ограничение UNIQUE к столбцу Email в таблице Employees, предварительно заполнив любыми значениями

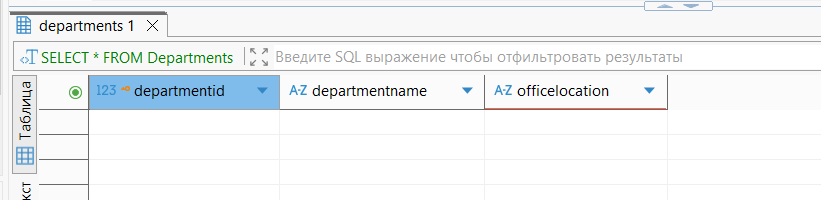
ALTER TABLE Employees: Указывает, что мы изменяем таблицу Employees.

ADD CONSTRAINT unique\_email UNIQUE (Email): Добавляет ограничение UNIQUE к столбцу Email.

CONSTRAINT unique\_email: Дает имя ограничению.

UNIQUE (Email): Указывает, что столбец Email должен содержать уникальные значения.

4. Переименовать столбец Location в таблице Departments в OfficeLocation.



ALTER TABLE Departments: Указывает какую таблицу мы хотим изменить.

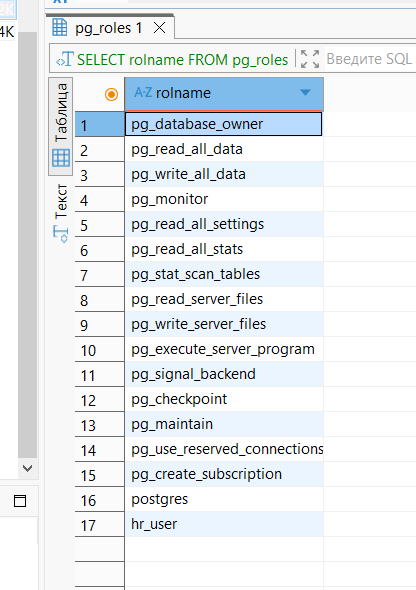
RENAME COLUMN Location TO OfficeLocation: Переименовывает столбец Location в OfficeLocation.

**Задание 3: DCL**

● Цель: Научиться создавать пользователей (роли в PostgreSQL) и предоставлять/отзывать базовые разрешения.

● Действия:

1. Создать нового пользователя PostgreSQL (роль) с именем hr\_user и простым паролем.



CREATE USER hr\_user: Создает нового пользователя с именем hr\_user.

WITH PASSWORD '163\_password': Устанавливает пароль для этого пользователя.

2. Предоставить hr\_user право SELECT на таблицу Employees.

GRANT SELECT: Предоставляет право на выполнение операции SELECT.

ON Employees: Указывает, что право предоставляется на таблицу Employees.

TO hr\_user: Указывает, что право предоставляется пользователю hr\_user.

3. Тест: В новой сессии подключиться как hr\_user и попытаться выполнить SELECT \* FROM Employees;. (Должно сработать).

4. Как hr\_user, попытаться выполнить INSERT нового сотрудника в Employees. (Должно завершиться неудачей).

5. Как пользователь-администратор, предоставить hr\_user права INSERT и UPDATE на таблицу Employees.

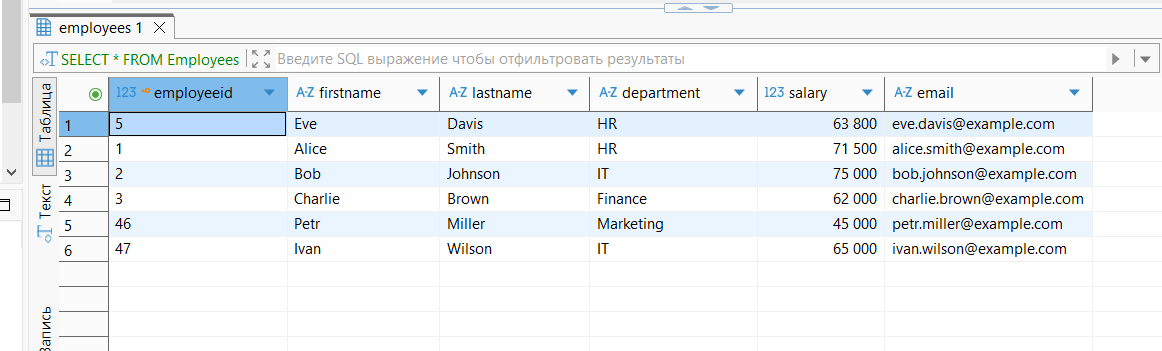
6. Тест: Как hr\_user, попробовать выполнить INSERT и UPDATE сотрудника. (Теперь должно сработать).

Задание 4: DML/DCL

● Цель: Практика более сложных DML-операций с использованием предложений WHERE, множественных обновлений

● Действия:

1. Увеличить Salary всех сотрудников в отделе 'HR' на 10%.

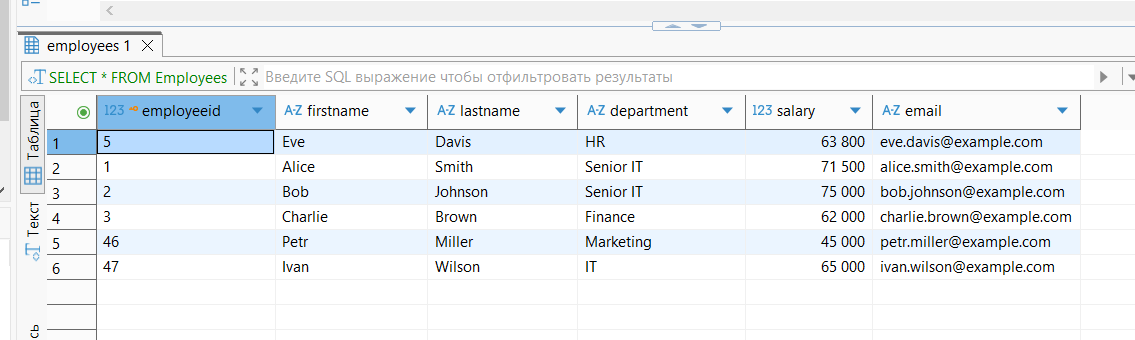


UPDATE Employees: Указываем, что мы собираемся обновить данные в таблице Employees.

SET Salary = Salary \* 1.10: 1.10 означает, что мы умножаем текущее значение Salary на 1.10, что эквивалентно увеличению на 10%.

WHERE Department = 'HR': Ограничивает обновление только сотрудниками, у которых значение в столбце Department равно 'HR'.

2. Обновить Department любого сотрудника с Salary выше 70000.00 на 'Senior IT'.



UPDATE Employees: Указываем, что мы собираемся обновить данные в таблице Employees.

SET Department = 'Senior IT': Устанавливаем новое значение для столбца Department равным 'Senior IT'.

WHERE Salary > 70000.00: Ограничивает обновление только сотрудниками, у которых значение в столбце Salary больше 70000.00.

3. Удалить всех сотрудников, которые не назначены ни на один проект в таблице EmployeeProjects. Подсказка: Используйте подзапрос NOT EXISTS или LEFT JOIN

DELETE FROM Employees: Указываем из какой таблицы будем удалять данные.

WHERE NOT EXISTS (...): Этот оператор проверяет, существует ли хотя бы одна запись, удовлетворяющая условию в подзапросе. Если ни одной записи не существует, то условие NOT EXISTS становится истинным, и строка из таблицы Employees будет удалена.

SELECT 1 FROM EmployeeProjects WHERE EmployeeProjects.EmployeeID = Employees.EmployeeID: Это подзапрос, который пытается найти запись в таблице EmployeeProjects, где EmployeeID соответствует EmployeeID из таблицы Employees.

4. Вставить новый проект и назначить на него двух существующих сотрудников с определенным количеством HoursWorked в EmployeeProjects, и все это в одном блоке BEGIN/COMMIT.



**Задание 5: Функции и представления**

● Цель: Понять и создать простые SQL-функции и представления в PostgreSQL.

● Действия:

1. Функция: Создать функцию PostgreSQL с именем CalculateAnnualBonus, которая принимает employee\_id и Salary в качестве входных данных и возвращает рассчитанную сумму бонуса (10 % от Salary) для этого сотрудника. Используйте PL/pgSQL для тела функции.

CREATE OR REPLACE FUNCTION CalculateAnnualBonus(employee\_id INT, Salary DECIMAL)

RETURNS DECIMAL AS $$

DECLARE

bonus DECIMAL;

BEGIN

-- Рассчитать бонус (10% от Salary)

bonus := Salary \* 0.10;

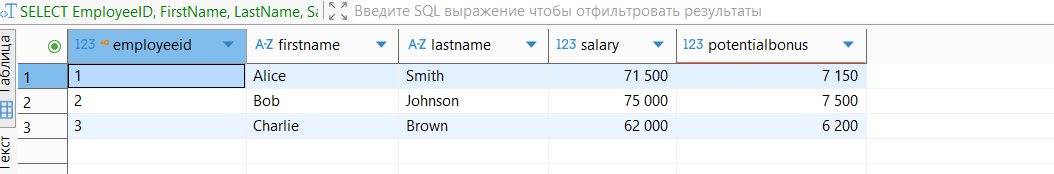
-- Вернуть рассчитанный бонус

RETURN bonus;

END;

$$ LANGUAGE plpgsql;

2. Использовать эту функцию в операторе SELECT, чтобы увидеть потенциальный бонус для каждого сотрудника.



**SELECT**

EmployeeID,

FirstName,

LastName,

Salary,

**CalculateAnnualBonus**(EmployeeID, Salary) **AS** *PotentialBonus*

**FROM**

Employees;

3. Представление (View): Создать представление с именем IT\_Department\_View, которое показывает EmployeeID, FirstName, LastName и Salary только для сотрудников из отдела 'IT'.

**CREATE** **OR** **REPLACE** VIEW IT\_Department\_View **AS**

**SELECT**

EmployeeID,

FirstName,

LastName,

Salary

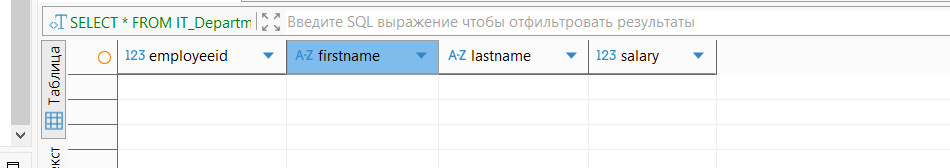
**FROM**

Employees

**WHERE**

Department = 'IT';

4. Выбрать данные из вашего представления IT\_Department\_View.



Вывод всех IT(их нет, т.к все IT перешли Senior IT)

**Задание 6: DML (Optional)**

● Цель: Объединение DML-операций с JOIN, подзапросами и условной логикой.

● Действия:

1. Найти ProjectName всех проектов, в которых 'Bob Johnson' работал более 150 часов.

**SELECT** P.ProjectName

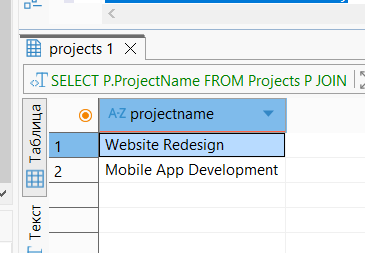
**FROM** Projects P

**JOIN** EmployeeProjects EP **ON** P.ProjectID = EP.ProjectID

**JOIN** Employees E **ON** EP.EmployeeID = E.EmployeeID

**WHERE** E.FirstName = 'Bob' **AND** E.LastName = 'Johnson'

**AND** EP.HoursWorked > 150;



2. Увеличить Budget всех проектов на 10%, если к ним назначен хотя бы один сотрудник из отдела 'IT'.

--Увеличиваем бюджет на 10%

**UPDATE** Projects

**SET** Budget = Budget \* 1.10

**WHERE** ProjectID **IN** (

**SELECT** EP.ProjectID

**FROM** EmployeeProjects EP

**JOIN** Employees E **ON** EP.EmployeeID = E.EmployeeID

**WHERE** E.Department = 'IT'

);

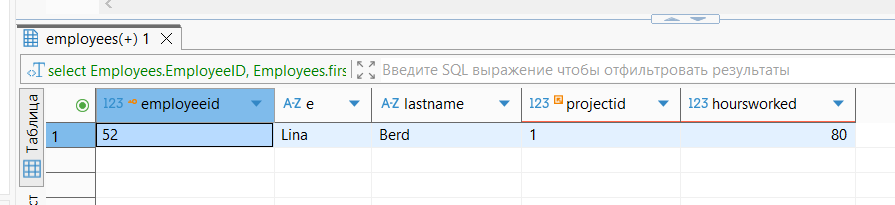
3. Для любого проекта, у которого еще нет EndDate (EndDate IS NULL), установить EndDate на один год позже его StartDate.

**UPDATE** Projects

**SET** EndDate = StartDate + **INTERVAL** '1 year'

**WHERE** EndDate **IS** **NULL**;

4. Вставить нового сотрудника и немедленно назначить его на проект 'Website Redesign' с 80 отработанными часами, все в рамках одной транзакции. Использовать предложение RETURNING, чтобы получить EmployeeID вновь вставленного сотрудника.



**WITH** NewEmployee **AS** (

**INSERT** **INTO** Employees (FirstName, LastName, Department, Salary)

**VALUES** ('Lina', 'Berd', 'IT', 45000.00)

**RETURNING** EmployeeID

),

TargetProject **AS** (

**SELECT** ProjectID

**FROM** Projects

**WHERE** ProjectName = 'Website Redesign'

**LIMIT** 1

)

**INSERT** **INTO** EmployeeProjects (EmployeeID, ProjectID, HoursWorked)

**SELECT** (**SELECT** EmployeeID **FROM** NewEmployee),

(**SELECT** ProjectID **FROM** TargetProject),

80;