PSQL Assignment

Database Setup:

* Create a fresh database titled **"university\_db"** or any other appropriate name.

I. Create the Database

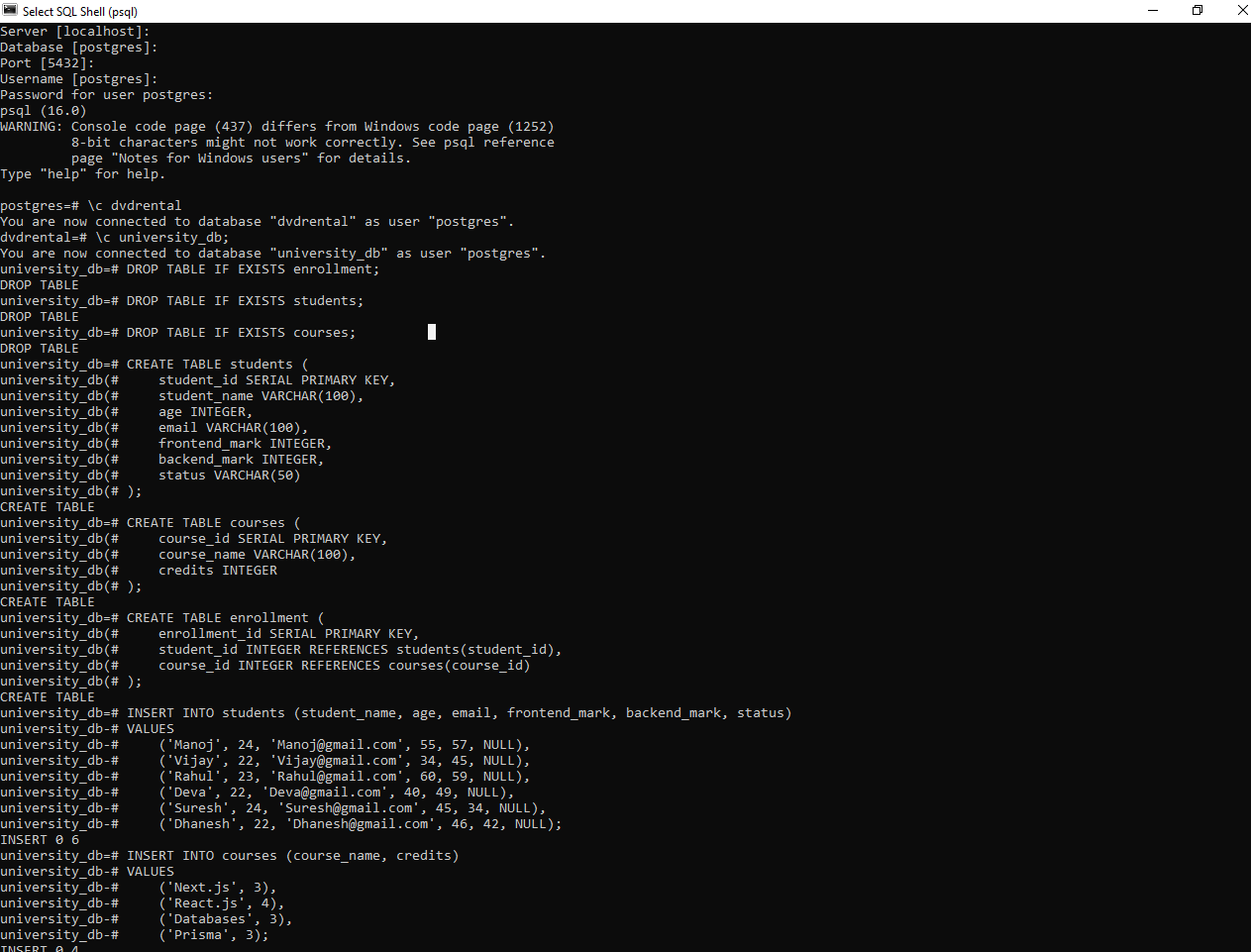
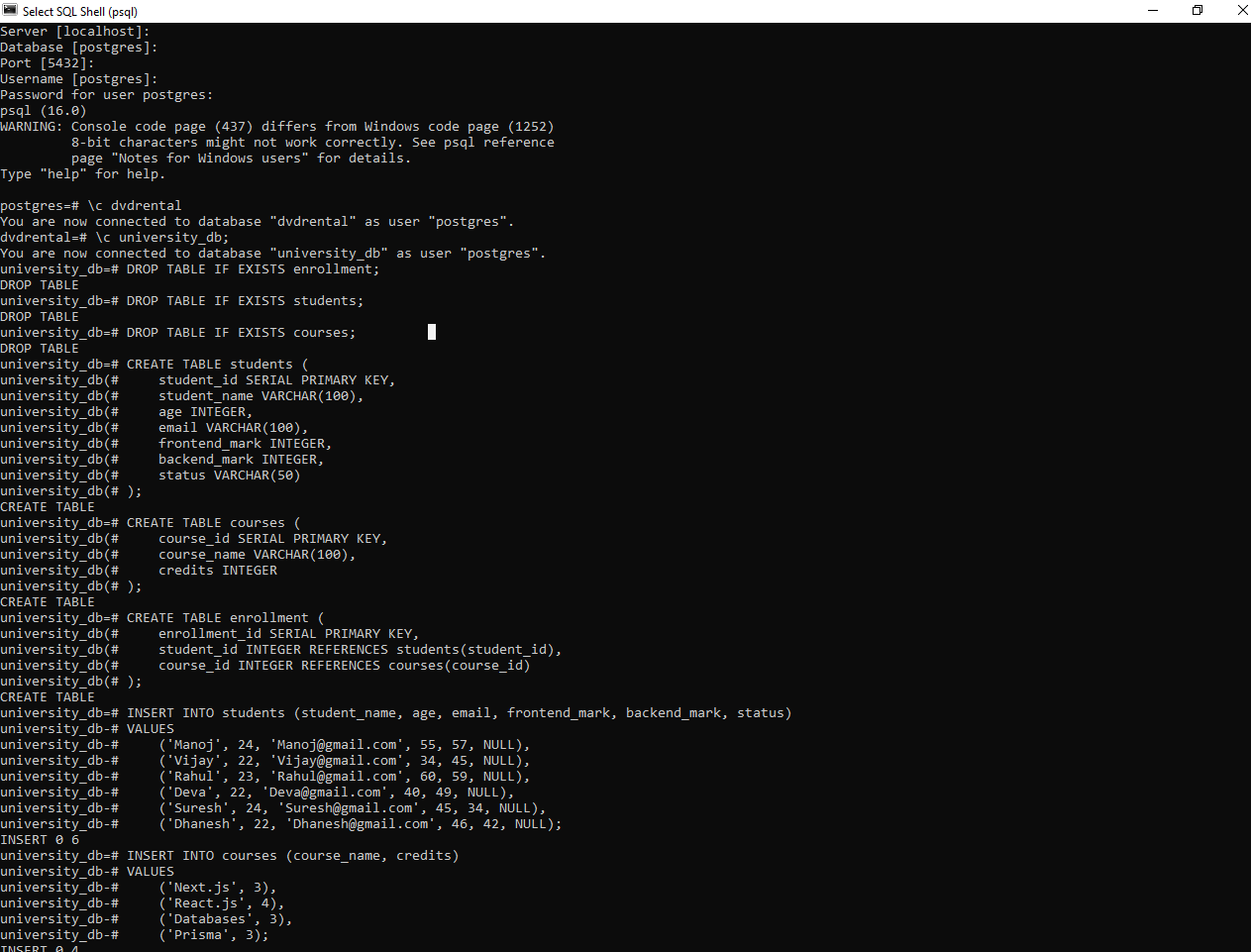
CREATE DATABASE university\_db;

Table Creation:

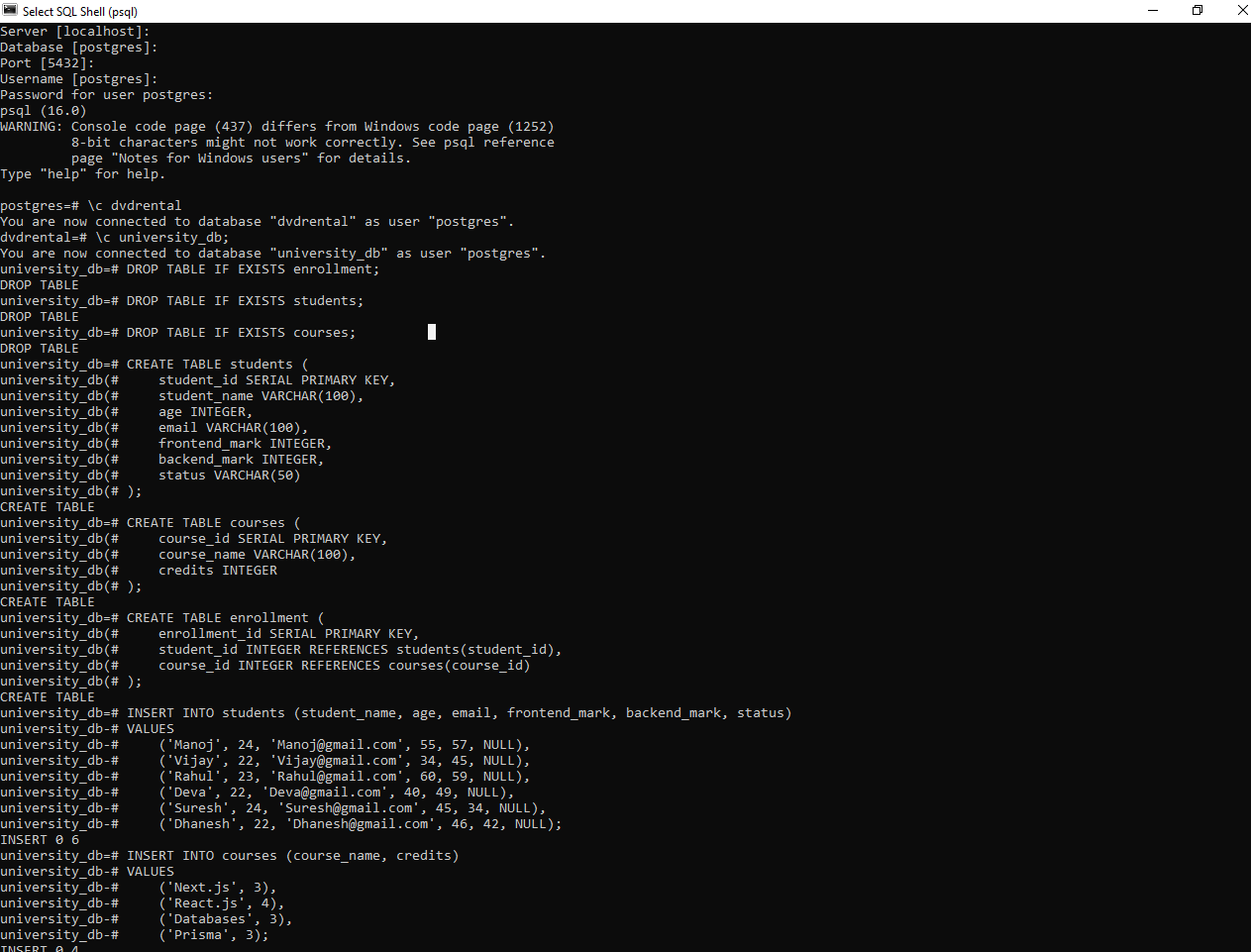
Create a **"students"** table with the following fields:

* student\_id (Primary Key): Integer, unique identifier for students.
* student\_name: String, representing the student's name.
* age: Integer, indicating the student's age.
* email: String, storing the student's email address.
* frontend\_mark: Integer, indicating the student's frontend assignment marks.
* backend\_mark: Integer, indicating the student's backend assignment marks.
* status: String, storing the student's result status.

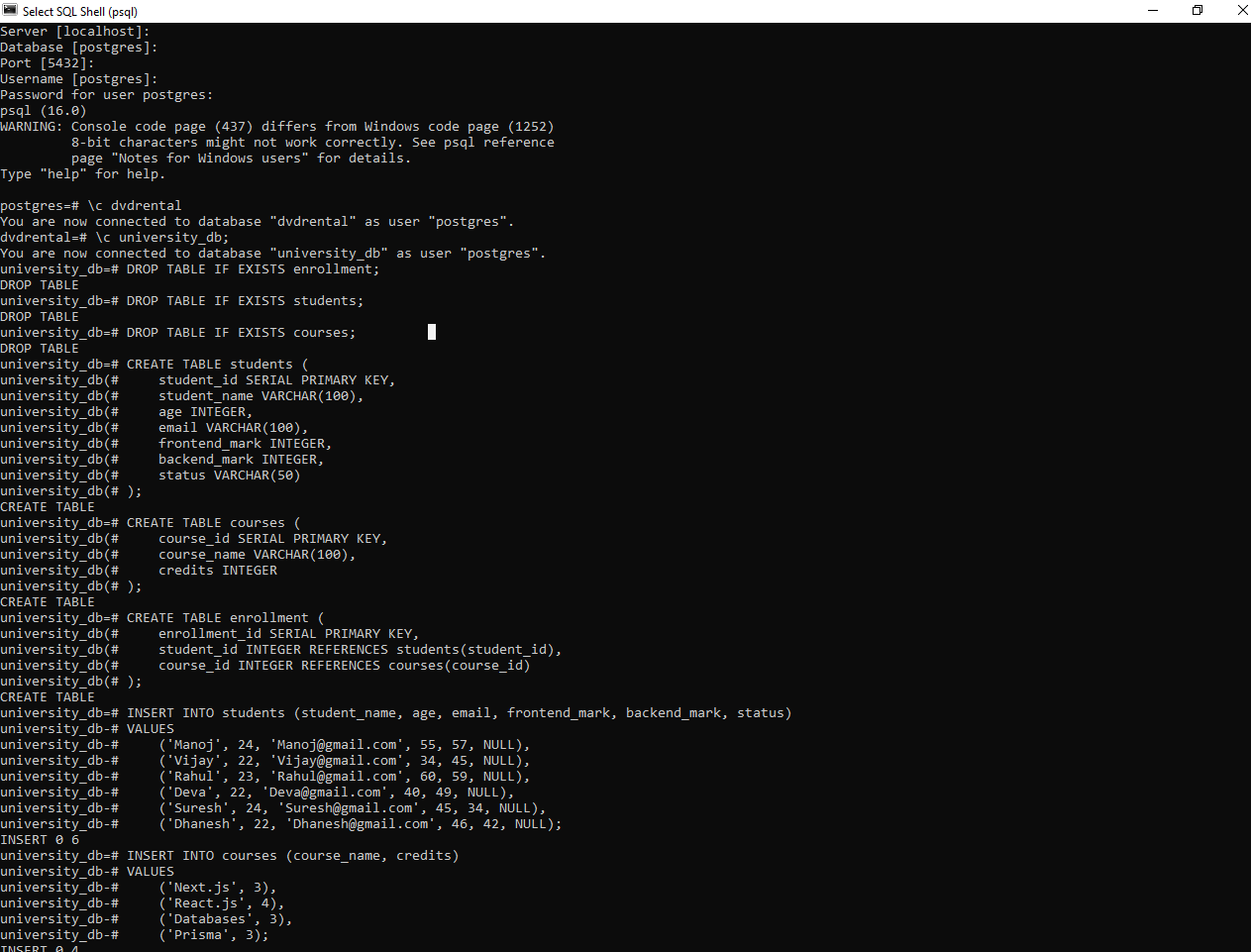


Create a **"courses"** table with the following fields:

* course\_id (Primary Key): Integer, unique identifier for courses.
* course\_name: String, indicating the course's name.
* credits: Integer, signifying the number of credits for the course.

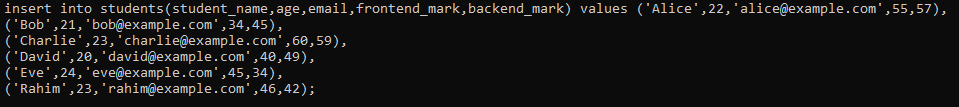


Create an **"enrollment"** table with the following fields:

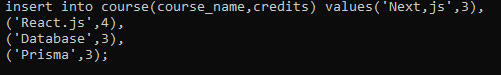
* enrollment\_id (Primary Key): Integer, unique identifier for enrollments.
* student\_id (Foreign Key): Integer, referencing student\_id in "Students" table.
* course\_id (Foreign Key): Integer, referencing course\_id in "Courses" table.

Sample Data

* Insert the following sample data into the **"students"** table:



* Insert the following sample data into the **"courses"** table:



* Insert the following sample data into the **"enrollment"** table:



Execute SQL queries to fulfill the ensuing tasks:

Query 1:

Insert a new student record with the following details:

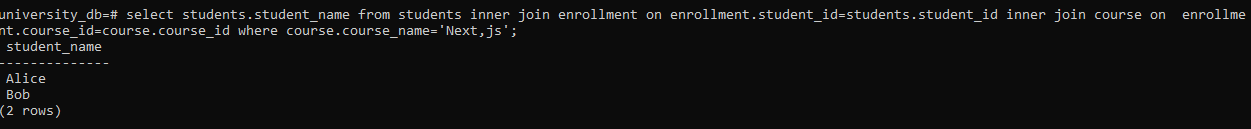
* Name: YourName
* Age: YourAge
* Email: YourEmail
* Frontend-Mark: YourMark
* Backend-Mark: YourMark
* Status: NULL



Query 2:

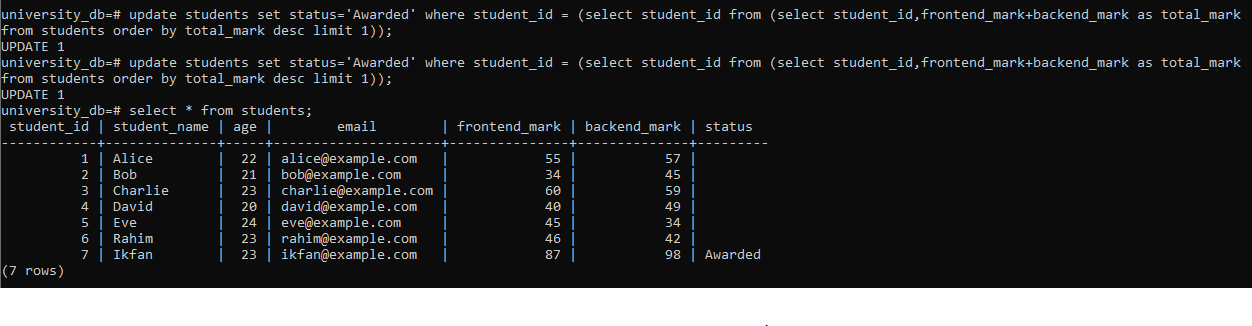
Retrieve the names of all students who are enrolled in the course titled 'Next.js'.

**Sample Output:**



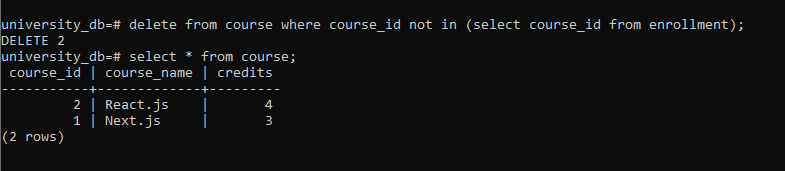
Query 3:

Update the status of the student with the highest total (frontend\_mark + backend\_mark) mark to 'Awarded'



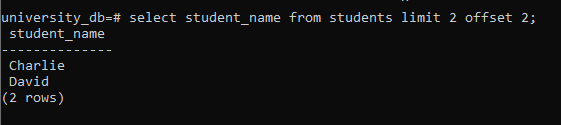
Query 4:

Delete all courses that have no students enrolled.



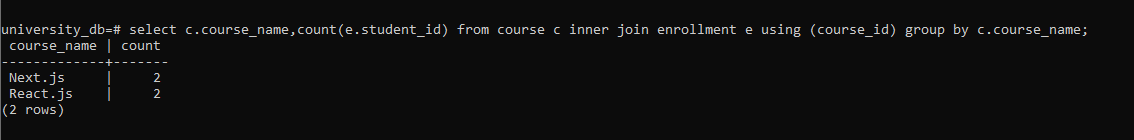
Query 5:

Retrieve the names of students using a limit of 2, starting from the 3rd student.



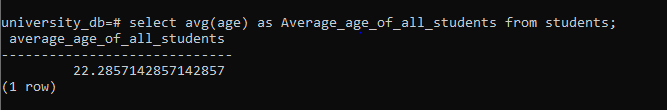
Query 6:

Retrieve the course names and the number of students enrolled in each course.



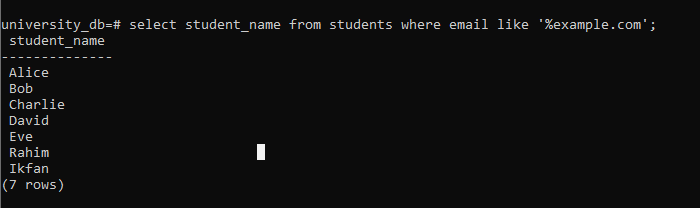
Query 7:

Calculate and display the average age of all students.



Query 8:

Retrieve the names of students whose email addresses contain 'example.com'.



**Answers for all the questions**

1. **Primary Key and Foreign Key concepts in PostgreSQL:**
   * **Primary Key:** A primary key uniquely identifies each record in a table. It ensures each row is unique and cannot contain NULL values. Example: ***student\_id*** in ***students*** table.
   * **Foreign Key:** A foreign key establishes a link between two tables, referencing the primary key of another table. It enforces referential integrity. Example: ***student\_id*** in ***enrollment*** table references ***student\_id*** in ***students*** table.
2. **Difference between VARCHAR and CHAR data types:**
   * **VARCHAR:** Variable-length character string. Stores characters of varying lengths up to a maximum specified length.
   * **CHAR:** Fixed-length character string. Pads shorter strings with spaces to match the defined length. Example: ***student\_name*** is VARCHAR because names vary in length.
3. **Purpose of the WHERE clause in a SELECT statement:**
   * The ***WHERE*** clause filters rows based on a specified condition. It allows retrieval of rows that satisfy the condition. Example: ***WHERE email LIKE*** ***'%example.com'*** filters students whose email addresses contain 'example.com'.
4. **LIMIT and OFFSET clauses:**
   * **LIMIT:** Specifies the maximum number of rows to return.
   * **OFFSET:** Specifies how many rows to skip before starting to return rows. Used for pagination. Example: ***LIMIT 2 OFFSET 2*** retrieves 2 rows starting from the 3rd row.
5. **Data modification using UPDATE statements:**
   * ***UPDATE*** statement modifies existing records in a table based on specified conditions. Example: ***UPDATE students SET status = 'Awarded' WHERE*** ... updates the status of a student based on certain criteria.
6. **Significance of the JOIN operation in PostgreSQL:**
   * **JOIN:** Combines rows from two or more tables based on related columns. Allows retrieval of related data from multiple tables in a single query. Example: ***LEFT JOIN enrollment ON c.course\_id = e.course\_id*** retrieves courses with the count of students enrolled.
7. **GROUP BY clause and its role in aggregation operations:**
   * **GROUP BY:** Groups rows that share a common value into summary rows. Used with aggregate functions like COUNT, SUM, AVG, etc., to perform operations on groups of data. Example: ***GROUP BY c.course\_name*** groups courses for counting enrolled students.
8. **Aggregate functions like COUNT, SUM, and AVG in PostgreSQL:**
   * **COUNT:** Counts the number of rows returned by a query.
   * **SUM:** Calculates the sum of values in a column.
   * **AVG:** Calculates the average of values in a column. Example: ***SELECT AVG(age) AS average\_age FROM students*** calculates the average age of all students.
9. **Purpose of an index in PostgreSQL and its optimization:**
   * **Index:** Improves query performance by reducing the number of data pages PostgreSQL needs to read. Speeds up data retrieval operations. Example: ***CREATE INDEX idx\_student\_email ON students(email);*** creates an index on the ***email*** column for faster email-based searches.
10. **Concept of a PostgreSQL view and its difference from a table:**
    * **View:** Virtual table based on the result set of a SELECT query. Stores a query definition but not data itself. Simplifies complex queries, provides data security by limiting access to columns. Example: ***CREATE VIEW view\_students AS SELECT student\_name, age FROM students WHERE age > 21***; creates a view of students older than 21.