Sure! Below are some business scenario questions along with simple DDL, sample data, and PostgreSQL queries that demonstrate the use of **Sequences**, **Synonyms**, **Views**, **Operations on Views**, **Indexing**, and **Clusters**.

**1. Business Scenario for Sequence**

**Question:**  
A company needs a unique identifier for its employees. How can you implement a sequence to generate unique employee IDs automatically?

**DDL for Sequence**

sql

Copy code

CREATE SEQUENCE employee\_id\_seq

START WITH 1

INCREMENT BY 1

NO MINVALUE

NO MAXVALUE

CACHE 1;

**Inserting Data Using Sequence**

sql

Copy code

CREATE TABLE employees (

employee\_id INT DEFAULT nextval('employee\_id\_seq') PRIMARY KEY,

name VARCHAR(100),

department VARCHAR(50)

);

INSERT INTO employees (name, department) VALUES

('Alice Johnson', 'HR'),

('Bob Smith', 'Finance'),

('Charlie Brown', 'Engineering');

**Query to Display Employees**

sql

Copy code

SELECT \* FROM employees;

**2. Business Scenario for Synonym**

**Question:**  
You want to create an easier way to access a frequently used table. How can you create a synonym for that table?

**DDL for Synonym**

sql

Copy code

CREATE SYNONYM emp FOR employees;

**Query to Access Synonym**

sql

Copy code

SELECT \* FROM emp;

**3. Business Scenario for View**

**Question:**  
The management wants a summary view of employees showing only their names and departments. How can you create a view for this purpose?

**DDL for View**

sql

Copy code

CREATE VIEW employee\_summary AS

SELECT name, department

FROM employees;

**Query to Access View**

sql

Copy code

SELECT \* FROM employee\_summary;

**4. Operations on View**

**Question:**  
You want to update the department of an employee through the view. How can you accomplish that?

**Update Query on View**

sql

Copy code

UPDATE employee\_summary

SET department = 'Marketing'

WHERE name = 'Alice Johnson';

**Query to Confirm Update**

sql

Copy code

SELECT \* FROM employee\_summary;

**5. Business Scenario for Indexing**

**Question:**  
The company frequently queries employee data by department. How can you improve query performance by creating an index?

**DDL for Index**

sql

Copy code

CREATE INDEX idx\_department ON employees (department);

**Query Using Index**

sql

Copy code

SELECT \* FROM employees WHERE department = 'Engineering';

**6. Business Scenario for Clusters**

**Question:**  
To improve the performance of frequently accessed employee records, how can you cluster the employees table based on the department?

**DDL for Clustering**

sql

Copy code

CLUSTER employees USING idx\_department;

**Query to Display Employees After Clustering**

sql

Copy code

SELECT \* FROM employees;

**Summary of Operations:**

* **Sequence**: Automatically generates unique employee IDs.
* **Synonym**: Provides a shortcut to access the employees table.
* **View**: Creates a simplified view for employee summaries.
* **Update on View**: Allows updates to the original table through the view.
* **Indexing**: Enhances performance for queries based on departments.
* **Clustering**: Physically organizes the table to improve query performance.

These scenarios demonstrate practical uses of various PostgreSQL features that can be applied in a business context!

limitations of views (especially with DELETE operations), the differences between views and clustering, and the types of indexes, particularly **BRIN (Block Range INdex)** in PostgreSQL.

**1. Limitations on Views with DELETE Operations**

**Question:**  
What limitations exist when trying to perform DELETE operations on a view?

**Example of View Limitations**

Assuming we have a view called employee\_summary as defined earlier:

sql

Copy code

CREATE VIEW employee\_summary AS

SELECT name, department

FROM employees;

**Attempt to Delete Using the View**

sql

Copy code

DELETE FROM employee\_summary

WHERE name = 'Alice Johnson';

**Limitation:**  
This operation will throw an error if the view is not updatable. In PostgreSQL, for a view to be updatable, it must satisfy certain conditions:

* The view must be defined on a single base table.
* All columns must be directly updatable (i.e., not derived from calculations or aggregations).
* No DISTINCT, GROUP BY, or ORDER BY clauses should be present.