1. DCL (GRANT and REVOKE)

Grant Permissions: Write a query to grant `SELECT` permission on the `employees` table to a user named `user1`.
sql
Copy code
```
GRANT SELECT ON employees TO user1;
```

1.

Revoke Permissions: Write a query to revoke `SELECT` permission on the `employees` table from `user1`.
sql
Copy code
```
REVOKE SELECT ON employees FROM user1;
```

2.

2. View and Materialized View

Create a View: Create a view named `employee_view` that shows the `employee_id`, `first_name`, and `department_id` from the `employees` table.
sql
Copy code
```
CREATE VIEW employee_view AS
SELECT employee_id, first_name, department_id
FROM employees;
```

1.

Create a Materialized View: Create a materialized view named `employee_mv` that refreshes every day at midnight, containing the `employee_id` and `salary` from the `employees` table.
sql
Copy code
```
CREATE MATERIALIZED VIEW employee_mv
BUILD IMMEDIATE
REFRESH COMPLETE
START WITH SYSDATE
NEXT SYSDATE + 1
AS SELECT employee_id, salary FROM employees;
```

2.

3. Synonyms

Create a Synonym: Create a synonym named `emp_syn` for the `employees` table.
sql
Copy code
```
CREATE SYNONYM emp_syn FOR employees;
```

1.

4. Sequences

Create a Sequence: Create a sequence named `emp_seq` that starts at 1 and increments by 1.
sql
Copy code
```
CREATE SEQUENCE emp_seq
START WITH 1
INCREMENT BY 1;
```

1.

Use the Sequence: Insert a new employee into the `employees` table using the `emp_seq` sequence for the `employee_id`.
sql
Copy code
```
INSERT INTO employees (employee_id, first_name, last_name,
department_id)
VALUES (emp_seq.NEXTVAL, 'John', 'Doe', 10);
```

2.

5. Correlated Query

Correlated Subquery: Write a query to find all employees whose salary is above the average salary in their department.
sql
Copy code
```
SELECT employee_id, first_name, salary
FROM employees e
WHERE salary > (
    SELECT AVG(salary)
    FROM employees
    WHERE department_id = e.department_id
```

```
);
```

1.

6. Index

Create an Index: Create an index on the `last_name` column of the `employees` table.
sql
Copy code

```
CREATE INDEX idx_last_name ON employees(last_name);
```

1.

7. Row Number

Row Number: Write a query to assign a row number to each employee, ordered by `salary` in descending order.
sql
Copy code

```
SELECT employee_id, first_name, salary,
       ROW_NUMBER() OVER (ORDER BY salary DESC) AS row_num
FROM employees;
```

1.

8. Regex

Regular Expression: Write a query to find all employees whose first name starts with 'J' and ends with 'n'.
sql
Copy code

```
SELECT employee_id, first_name
FROM employees
WHERE REGEXP_LIKE(first_name, '^J.*n$');
```

1. ***What is materialized view***

A materialized view is a database object that contains the results of a query. It is different from a standard view in that it stores the query result data physically, rather than just storing the query definition. This means that materialized views can significantly improve query performance by allowing users to access precomputed results instead of re-executing complex queries.

Key Characteristics of Materialized Views

1. **Physical Storage**: Unlike regular views, which are virtual tables and do not store data themselves, materialized views store the actual data returned by the query.
2. **Refresh Mechanism**: Materialized views need to be refreshed to keep their data up to date with the underlying base tables. This can be done automatically (scheduled) or manually.
3. **Performance**: Because the data is precomputed and stored, queries that use materialized views can be much faster, especially for complex aggregations and joins.
4. **Use Cases**: Materialized views are particularly useful in data warehousing and reporting applications where query performance is critical.

Creating a Materialized View

Here is an example of creating a materialized view:

sql
Copy code

```
CREATE MATERIALIZED VIEW sales_summary_mv
BUILD IMMEDIATE
REFRESH COMPLETE
START WITH SYSDATE
NEXT SYSDATE + 1
AS
SELECT product_id, SUM(quantity_sold) AS total_quantity_sold,
SUM(amount_sold) AS total_amount_sold
FROM sales
GROUP BY product_id;
```

Explanation of the Example

- **BUILD IMMEDIATE**: This clause tells the database to populate the materialized view immediately when it is created.
- **REFRESH COMPLETE**: This specifies that the entire materialized view will be refreshed (rebuilt) when it is updated. Other options include `REFRESH FAST` for incremental updates.
- **START WITH SYSDATE**: This specifies the start time for the first refresh.
- **NEXT SYSDATE + 1**: This specifies the interval between refreshes (e.g., every day).
- **AS SELECT...**: This is the query whose results will be stored in the materialized view.

Refreshing a Materialized View

Materialized views can be refreshed manually or automatically. Here's an example of manually refreshing a materialized view:

sql
Copy code

```
EXEC DBMS_MVIEW.REFRESH('sales_summary_mv');
```

Automatic refresh can be scheduled as shown in the creation example, using `START WITH` and `NEXT` clauses.

Advantages of Materialized Views

1. **Improved Performance**: By storing the results of complex queries, materialized views can speed up query performance.
2. **Precomputed Aggregations**: They are useful for storing precomputed aggregations, which can be expensive to compute on the fly.
3. **Offloading Query Load**: By using materialized views, you can offload the query load from the base tables, which can be particularly useful in data warehousing scenarios.

Disadvantages of Materialized Views

1. **Storage Overhead**: Since they store data, they require additional storage space.
2. **Maintenance Overhead**: Keeping the materialized view in sync with the base tables can require significant maintenance, especially if the data changes frequently.
3. **Complexity**: Managing refresh schedules and understanding when and how to use materialized views adds complexity to the database management.

In summary, materialized views are a powerful feature for improving query performance and are especially beneficial in environments where read performance is critical, and data changes are relatively infrequent.