CHAPTER: 4 ADVANCE SQL: SQL PERFORMANCE TUNING (12 Marks)

4.1 VIEW

- View: Views are virtual relations mainly used for security purpose, and can be provided on request by a particular user.
- A view can contain all rows of a table or select rows from a table. A view can be created from one or many tables which depends on the written SQL query to create a view.
- Views are created for security reasons. Instead of coping same table multiple times for different requirements, views can be created.
- View is a logical copy of physical table.
- It doesn't exist physically. With the help of view, we can give restricted access to users. When view is used, underlying table is invisible, thus increasing security. Views can be used to see, insert, update and delete data from base table.
- **SQL CREATE VIEW Syntax**

  CREATE VIEW view_name
  AS SELECT column_name(s)
  FROM table_name
  WHERE condition;
- Where,

  view_name – is the name of view.

  SELECT statement- is used to define the columns & rows that you want to display in the view.
- **Note:** A view always shows up-to-date data! The database engineer recreates the data, using the view's SQL statement, every time a user queries a view.

- **Examples 1:-**
- To create a view on the product table the SQL query would be like.

  CREATE VIEW Product_view

  AS SELECT ProductID, ProductName

  FROM Product;
- We can query the view above as follows:
- **SQL**> SELECT * FROM Product_view;

- **Examples 2:-**

- **CREATE VIEW** emp_info **AS SELECT** Emp_no, Emp_name **FROM** Employee **WHERE** salary>12000;
- To describe content of view
  SQL> Select * from emp_info;

4.1.1 WITH CHECK OPTION

- The WITH CHECK OPTION is a CREATE VIEW statement option. The purpose of the WITH CHECK OPTION is to ensure that all UPDATE and INSERTs satisfy the condition(s) in the view definition.
- If they do not satisfy the condition(s), the UPDATE or INSERT returns an error.
- The **following is an example** of creating same view CUSTOMERS_VIEW with the WITH CHECK OPTION:
- **CREATE VIEW CUSTOMERS_VIEW AS**
  SELECT name, age
  FROM CUSTOMERS
  WHERE age IS NOT NULL
  WITH CHECK OPTION;
- The WITH CHECK OPTION in this case should deny the entry of any NULL values in the view's AGE column, because the view is defined by data that does not have a NULL value in the AGE column.

4.1.2 UPDATING A VIEWS

- A view can be updated under certain conditions:
  1. The SELECT clause may not contain the keyword DISTINCT.
  2. The SELECT clause may not contain summary functions.
  3. The SELECT clause may not contain set functions.
  4. The SELECT clause may not contain set operators.
  5. The SELECT clause may not contain an ORDER BY clause.
  6. The FROM clause may not contain multiple tables.
  7. The WHERE clause may not contain subqueries.
  8. The query may not contain GROUP BY or HAVING.
  9. Calculated columns may not be updated.

  So if a view satisfies all the above mentioned rules then you can update a view.

- **You can update a view by using the following syntax:**
- **SQL> UPDATE** CUSTOMERS_VIEW **SET** Age=35 **WHERE Name='ramesh';**
- **You can also update a view by using the following syntax:**
- **CREATE OR REPLACE VIEW** view_name **AS SELECT** column_name(s) **FROM** table_name **WHERE** condition;
- Ex**. CREATE OR REPLACE VIEW** [Current Product List] **AS SELECT** ProductID, ProductName, Category **FROM** Products **WHERE** Discontinued=No

4.1.3 Inserting Rows into a View:

- Rows of data can be inserted into a view. The same rules that apply to the UPDATE command also apply to the INSERT command.

4.1.4 Deleting Rows into a View:

- Rows of data can be deleted from a view. The same rules that apply to the UPDATE and INSERT commands apply to the DELETE command.
- Following is an example to delete a record having AGE= 22.
- SQL > **DELETE FROM** CUSTOMERS_VIEW **WHERE** age = 22;

4.1.5 SQL Dropping a View

- You can delete a view with the DROP VIEW command.
- SQL DROP VIEW Syntax

  SQL> DROP VIEW view_name;

4.1.6 SQL Joins

- SQL joins are used to combine rows from two or more tables.
- An SQL JOIN clause is used to combine rows from two or more tables, based on a common field between them.

"Order" Table		
OrderID	**CustomerID**	**OrderDate**
10308	2	1996-09-18
10309	37	1996-09-19
10310	77	1996-09-20

"Customers " Table			
CustomerID	**C_Name**	**ContactName**	**Country**
1	Alfred	Maria	Germany
2	Ana	Ana	Mexico
3	Antonio	Antonio	Mexico

- Example
- **SELECT** Orders.OrderID, Customers.C_Name, Orders.OrderDate **FROM** Orders **INNER JOIN** Customers **ON** Orders.C_ID=Customers.C_ID;
- The Resultant table is:-

"Order" Table		
OrderID	**C_Name**	**OrderDate**
10308	Ana	1996-09-18

4.2 SEQUENCES

- **Definition:-** A sequence refers to a database object that is capable of generating unique and sequential integer values.
- Syntax:

  CREATE SEQUENCE <seq_name>
  [Start with num]
  [Increment by num]
  [Maxvalue num] [Minvalue num]
  [Cycle/nocycle]
  [Cache num/nocache];

Where,

- **Increment by num:** Used to specify the interval between sequence numbers.
- **Start with num:** States the first sequence numbers that needs to be generated.
- **Minvalue num:** This is used to state the minimum value of the sequence.
- **Maxvalue num:** It states the maximum value generated by sequence.
- **Cycle:** Cycle indicates that the sequence will be continued for generating the values from the starting after reaching either its maximum value or minimum value.
- **Cache:** The cache option is used to pre-allocate a set of sequence numbers and keep these numbers in the memory so that they can be accessed.
- **No Cache**: This states that there is no pre-allocation for the values of sequence.
- **Example 1:-**
- SQL> **CREATE SEQUENCE** student_seq **START** with 1 **INCREMENT** by 1 **MAXVALUE** 60 nocycle;

- SQL> Sequence Created.
- On execution, oracle will create a sequence **student_seq.** its start with 1, incrementing the sequence number by 1. Maximum value that it can generate is 60.
- **Referencing a sequence:** it is referenced in SQL statement with the NEXTVAL & CURRVAL;
- Ex.2:- CREATE Sequence seq_1
  Start with 1
  Increment by 1
  Maxvalue 999
  Cycle;
- Suppose consider "info" table

"Info" Table	
ID	**Name**
1	**John**
2	**Smith**
3	**Hari**

- The SQL query will be,
- SQL> **INSERT** into info values(seq_1.nextval, 'Anu');
- Result table will look like,

"Info" Table	
ID	Name
1	John
2	Smith
3	Hari
1	Anu

- Once you use nextval the sequence will increment even if you don't insert any record into the table.
- SQL> **Select** seq_1.currval **from** info;

4.2.1 Altering Sequences

- Use the **ALTER SEQUENCE** statement to change the increment, minimum and maximum values, cached numbers, and behavior of an existing sequence.
- This statement affects only future sequence numbers.

- **Alter sequence** <seq_name>
  [Start with num]
  [Increment by num]
  [Maxvalue num] [Minvalue num]
  [Cycle/nocycle]
  [Cache num/no-cache];
- Ex.:-SQL> **Alter sequence** seq_1 maxvalue 1500;
- SQL> **Alter sequence** seq_1 cycle cache 10; (This statement turns on cycle & cache for the seq_1 sequence.)

4.2.2 Dropping Sequences

- Use the **DROP SEQUENCE** statement to remove a sequence from the database.
- The sequence must be in your own schema or you must have the **DROP ANY SEQUENCE** system privilege.
- **Syntax:**
- **SQL> Drop sequence** sequence_name;
- **Ex.:-**
- SQL> **Drop sequence** Info;

4.3 INDEXES

- An index can be created in a table to find data more quickly and efficiently.
- The users cannot see the indexes, they are just used to speed up searches/queries.
- An index provides direct and fast access to rows in a table. Indexes are created explicitly Or automatically.
- Syntax of **CREATE INDEX**
- **CREATE INDEX index_name ON table_name;**

4.3.1 Types of Index:

1. Simple index (Single column): An index created on single column of a table is called a Simple Index.

Syntax: - CREATE INDEX index_name **ON** table_name (column_name);

Ex. SQL> **CREATE INDEX** emp_ind **ON** EMP (empid);

2. Unique indexes are used not only for performance, but also for data integrity. A unique index does not allow any duplicate values to be inserted into the table.

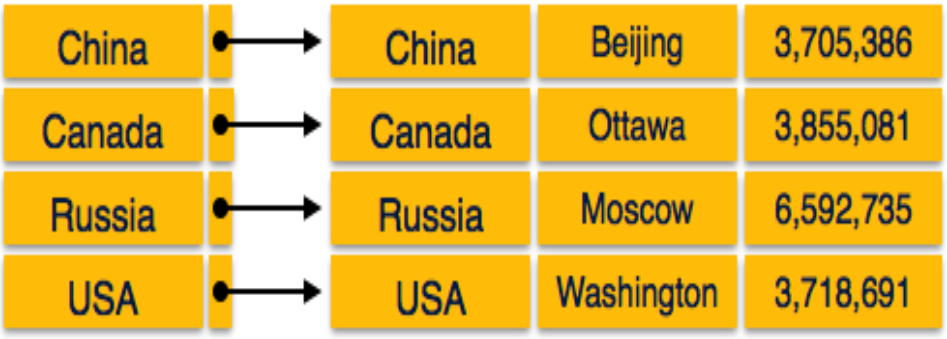
Syntax: - CREATE UNIQUE INDEX index_name **ON** table_name (column_name);

Ex: - CREATE UNIQUE INDEX emp_ind **ON** EMP (empid);

3. Composite (concatenated): Indexes that contain two or more columns from the same table which are useful for enforcing uniqueness in a table column where there's no single column that can uniquely identify a row.

Syntax;- **CREATE INDEX** index_name **ON** table_name (column1, column2);

Ex.:- CREATE INDEX emp_ind **ON** EMP (empid, Dept_name);



Indexing with neat diagram

4.3.2 Indexes

- The DROP INDEX
- An index can be dropped using SQL DROP command.
- Syntax:-
- DROP INDEX index_name;
- Ex.:-
- DROP INDEX emp_ind;

4.4 SNAPSHOT

- A snapshot is read only, static view of a database. You cannot make a backup of a snapshot.
- It is also known as materialized view.
- It is a copy of either an entire single table or set of its rows or collection of tables, views or rows using join, grouping and selection criteria.
- Useful in distributed environment
- **It has two types:**
- Simple snapshot and complex snapshot.

1. **Simple** snapshot related to single table.
2. **Complex** snapshot related to joined tables, views or grouped.

4.4.1 Creating a snapshot

- Syntax:

  CREATE SNAPSHOT snapshot_name
  Pctfree n
  Pctused n
  Tablespace Tablespace_name
  Storage clause
  Refresh
  Start with data
  As select statement;
- **Will referred another syntax:**

  CREATE SNAPSHOT snapshot_name < List of parameters> **as** <Select query>
- **Example:-**
  SQL> CREATE SNAPSHOT snap_info **AS SELECT** * **FROM** customer **WHERE** age>25;

4.4.2 Altering a snapshot

- **Syntax:-**

  SQL> ALTER SNAPSHOT snapshot_name **<list of parameters>**;
- **Example**

  SQL> ALTER SNAPSHOT snap_info pctfree 10 pctused 80;

4.4.3 Dropping a snapshot

- **Syntax:-**

  SQL> DROP SNAPSHOT snapshot_name;
- **Example**

  SQL> ALTER SNAPSHOT snap_info;

4.5 SYNONYMS

- Use the create synonyms statement to create a synonyms, which is an alternative name for a table.
- It can provide a level of security by masking the name & owner of an object & by providing location transparency for remote objects of a distributed database.
- You can crate both public & private synonyms.

- A public synonym is owned by special user group named PUBLIC & is accessible to every user in a database.
- A Private synonym is contained of specific user only.

4.5.1 Creating Synonyms:-

Syntax:-

CREATE SYNONYM Synonym_Name **FOR** table_name;

Example:-

SQL> CREATE SYNONYM Emp FOR Employee;

SQL> Synonym created.

SQL> **SELECT * FROM** Emp;

4.5.2 Dropping Synonyms :-

- If a synonym is no longer required, you can drop the sequence using the DROP SYNONYM statement.

Syntax:-

DROP SYNONYM Synonym_Name;

Example:-

SQL> CREATE SYNONYM Emp;

SQL> SYNONYM created.

SUMMER-16

1. What are sequence? Why it is used? Create sequence for STUDENT table. *(Definition - 1 mark; Use - 1 mark; creating valid sequence example/pattern - 2 marks)*
2. What are views? Give its syntax and explain its advantages. *(Views - 2 marks; Syntax - 1 mark; Advantages (Any two) - 1 mark)*
3. What is index? Explain types of index. *(Index Definition - 2 marks; Any Two Types - 1 mark each)*
4. What are snapshots? Give its uses. How to create a snapshot?*(Definition - 1 mark; any one Use - 1 mark, Syntax/Example - 2 marks)*

WINTER– 16

1. What is view?
2. Explain the following terms with syntax and example.
   i) Creating snapshot
   ii) Altering snapshot
   iii) Dropping a snapshot.

WINTER – 15

1. Explain sequences with example. (Definition of sequence - 1 Mark, syntax – 2 Marks, Example - 1 Mark)
2. Explain views with example.(Explanation of view with syntax – 3 Marks, Example - 1 Mark)
3. Explain snapshot with example. (Explanation of snapshot – 3 Marks, example – 1 Mark)
4. Explain the concept of indexing with neat diagram. (Explanation - 3 Marks, Any relevant Diagram - 1 Mark)