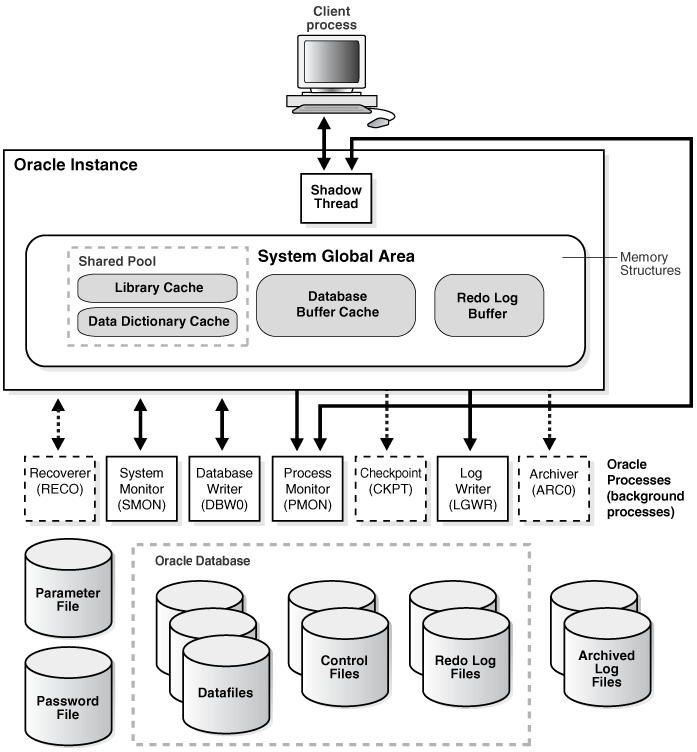
PLSQL - FAQs

Latest version of Oracle Database is **Oracle Database 23ai**

**NIQ interview questions:**

1. **Oracle PL/SQL Temporary Tables**
2. **Indexes**
3. **Partitioning**
4. **Analytical Queries (RANK, DENSE\_RANK, and CUBE)**
5. **Performance Tuning**
6. **Triggers – how to create insert, update, delete triggers fors specific table**
7. **Cursors – Implicit and Explicit**

**Architecture**



**1. Oracle Database vs. Oracle Instance**

* **Database**: A collection of logically related data, including tables, schemas, and other objects. It represents the actual information you store.
* **Instance**: A set of background processes, memory structures, and files that manage a single database. You can have multiple databases on a single server, each with its own instance

**2. Physical Components of Oracle Database**

* **Data Files**: Store the actual data (tables, indexes, etc.).
* **Redo Log Files**: Track changes made to the database for recovery purposes.
* **Control File**: Stores metadata about the database structure and location of data files.
* **Initialization Parameter File (SPFILE)**: Defines configuration settings for the instance.
* **Password File**: Stores user account passwords securely

**3. Memory Structures**

* **SGA (System Global Area)**: Shared memory region accessible by all user sessions. It holds crucial data like buffer cache (frequently accessed data), shared pool (parsed SQL statements), and redo log buffer.
* **PGA (Program Global Area)**: Private memory allocated for each user session. It holds temporary data specific to that session, like sort areas and variable storage

**4. Key Background Processes**

* **SMON (System Monitor)**: Cleans up orphaned sessions and performs instance recovery in case of crashes.
* **PMON (Process Monitor)**: Monitors background processes and restarts failed ones automatically

**5. Oracle Multitenant Architecture**

* **Container Database (CDB)**: Holds the Oracle instance and common objects.
* **Pluggable Databases (PDBs)**: Contain the actual application data and metadata. This architecture allows for easier database consolidation and management

**6. User Security and Privileges**

* **Roles**: Group related privileges that can be assigned to users.
* **Privileges**: Permissions to perform specific actions on database objects (e.g., SELECT, INSERT, DELETE).
* **Schemas**: Collections of database objects owned by a specific user

**7. Data Block, Extent, and Segment**

* **Data Block**: The smallest unit of logical storage for a database object.
* **Extent**: A group of contiguous data blocks allocated for a database object.
* **Segment**: All the extents allocated for a database object

**8. Checkpoint**

* **Purpose**: Synchronizes the database blocks in memory with the datafiles on disk. It establishes data consistency and enables faster database recovery

**9. SQL Execution Process**

* **Steps**: When a SQL statement is executed, Oracle checks the syntax and semantics, creates an execution plan, fetches data from datafiles if not already in buffer cache, and returns the result to the client

**SQL FAQs**

Sure! Here are some frequently asked questions (FAQs) about Oracle PL/SQL temporary tables, along with examples:

**FAQs on Oracle PL/SQL Temporary Tables**

**1. What are temporary tables in Oracle?** Temporary tables in Oracle are tables that hold data only for the duration of a session or transaction. They are useful for storing intermediate results temporarily.

**2. How do you create a temporary table in Oracle?** You can create a temporary table using the CREATE GLOBAL TEMPORARY TABLE statement. Here’s an example:

CREATE GLOBAL TEMPORARY TABLE temp\_table (

id INT,

description VARCHAR2(100)

) ON COMMIT DELETE ROWS;

This creates a transaction-specific temporary table that will be truncated after each commit.

**3. What is the difference between ON COMMIT DELETE ROWS and ON COMMIT PRESERVE ROWS?**

* ON COMMIT DELETE ROWS: The table is transaction-specific, meaning it is truncated after each commit.
* ON COMMIT PRESERVE ROWS: The table is session-specific, meaning it is truncated when the session ends, not after each commit.

**4. How do you insert data into a temporary table?** You can insert data into a temporary table just like a regular table. Here’s an example:

INSERT INTO temp\_table (id, description) VALUES (1, 'Temporary data');

**5. How do you query data from a temporary table?** You can query data from a temporary table using a standard SELECT statement:

SELECT id, description FROM temp\_table;

**6. What happens to the data in a temporary table after a transaction or session ends?**

* For transaction-specific tables (ON COMMIT DELETE ROWS), the data is deleted after each commit.
* For session-specific tables (ON COMMIT PRESERVE ROWS), the data is deleted when the session ends.

**Example Scenario**

Let's walk through an example of creating and using a transaction-specific temporary table:

1. **Create the temporary table:**

CREATE GLOBAL TEMPORARY TABLE temp\_table (

id INT,

description VARCHAR2(100)

) ON COMMIT DELETE ROWS;

1. **Insert data into the temporary table:**

INSERT INTO temp\_table (id, description) VALUES (1, 'Transaction specific data');

1. **Query the data:**

SELECT id, description FROM temp\_table;

1. **Commit the transaction:**

COMMIT;

1. **Query the data again (after commit):**

SELECT id, description FROM temp\_table;

After the commit, the table will be empty because it is transaction-specific.

**What is a Global Temporary Table in Oracle?**

A Global Temporary Table (GTT) in Oracle is a table that holds data temporarily for the duration of a session or transaction. Unlike regular tables, the data in a GTT is private to the session that inserts it, meaning other sessions cannot see this data[[1]](https://www.oracletutorial.com/oracle-basics/oracle-global-temporary-table/)[[2]](https://oracle-base.com/articles/misc/temporary-tables).

**How do you create a Global Temporary Table?**

You can create a GTT using the CREATE GLOBAL TEMPORARY TABLE statement. The syntax is similar to creating a regular table, but you include the GLOBAL TEMPORARY keywords and specify the ON COMMIT clause to define the table's behavior at commit time. Here is an example:

CREATE GLOBAL TEMPORARY TABLE temp\_table (

id NUMBER,

description VARCHAR2(100)

) ON COMMIT DELETE ROWS;

The ON COMMIT DELETE ROWS clause means that the data will be deleted at the end of each transaction

**What are the ON COMMIT DELETE ROWS and ON COMMIT PRESERVE ROWS options?**

* **ON COMMIT DELETE ROWS**: This option specifies that the data in the GTT will be deleted at the end of each transaction.
* **ON COMMIT PRESERVE ROWS**: This option specifies that the data will be preserved for the duration of the session and will only be deleted when the session ends[[1]](https://www.oracletutorial.com/oracle-basics/oracle-global-temporary-table/)[[2]](https://oracle-base.com/articles/misc/temporary-tables).

**Can you provide an example of creating a session-specific GTT?**

Certainly! Here is an example of creating a session-specific GTT:

CREATE GLOBAL TEMPORARY TABLE session\_temp\_table (

id NUMBER,

description VARCHAR2(100)

) ON COMMIT PRESERVE ROWS;

In this case, the data will remain in the table until the session ends[[1]](https://www.oracletutorial.com/oracle-basics/oracle-global-temporary-table/)[[2]](https://oracle-base.com/articles/misc/temporary-tables).

**What is the difference between Global Temporary Tables and Private Temporary Tables?**

* **Global Temporary Tables (GTTs)**: These are permanent database objects that store data temporarily for the duration of a session or transaction. The table structure is visible to all sessions, but the data is private to each session[[1]](https://www.oracletutorial.com/oracle-basics/oracle-global-temporary-table/)[[2]](https://oracle-base.com/articles/misc/temporary-tables).
* **Private Temporary Tables**: Introduced in Oracle 18c, these are memory-based tables that exist only for the duration of a session or transaction and are automatically dropped at the end of the session or transaction[[1]](https://www.oracletutorial.com/oracle-basics/oracle-global-temporary-table/)[[2]](https://oracle-base.com/articles/misc/temporary-tables).

**Are there any performance considerations when using GTTs?**

Yes, while GTTs can be very useful, they do have some performance considerations:

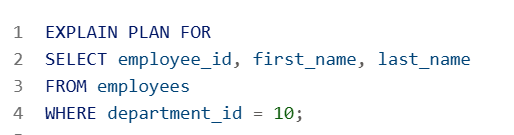
* The data in a GTT is stored in the temporary tablespace, but the associated undo data is stored in the regular undo tablespace, which can affect performance.
* Frequent creation and dropping of GTTs can lead to performance overhead. It is generally better to create GTTs once and reuse them[[3]](https://stackoverflow.com/questions/2671518/how-do-you-create-a-temporary-table-in-an-oracle-database)[[2]](https://oracle-base.com/articles/misc/temporary-tables).

1. **What is PL/SQL?**
   * **Answer**: PL/SQL stands for Procedural Language/SQL. It is an extension of SQL that allows for procedural constructs like loops and conditionals.
   * **Example**:
2. **What are the differences between SQL and PL/SQL?**
   * **Answer**: SQL is a standard language for querying databases, while PL/SQL is a procedural language that includes SQL and adds programming constructs.
   * **Example**:
3. **Explain the use of COMMIT, ROLLBACK, and SAVEPOINT statements.**
   * **Answer**:
     + **COMMIT**: Saves all changes made during the transaction.
     + **ROLLBACK**: Undoes all changes made during the transaction.
     + **SAVEPOINT**: Sets a point within a transaction to which you can later roll back.
   * **Example**:
4. **What are cursors in PL/SQL?**
   * **Answer**: Cursors are used to retrieve multiple rows from a query. There are two types: implicit cursors and explicit cursors.
   * **Example**:
5. **What is an exception in PL/SQL?**
   * **Answer**: Exceptions are used to handle errors and other exceptional conditions in PL/SQL. There are predefined exceptions and user-defined exceptions.
   * **Example**:
6. **What is a trigger in PL/SQL?**
   * **Answer**: A trigger is a stored procedure that is automatically invoked when a specified event occurs.
   * **Example**:

Oracle **SQL tuning** examples with explanations:

**1. Using Execution Plans**

**Example:**



**Explanation:**

* The EXPLAIN PLAN statement generates the execution plan for the query.
* The DBMS\_XPLAN.DISPLAY function displays the execution plan, showing how Oracle will execute the query, including the operations and their costs.

**2. Creating Indexes**

**Example:**

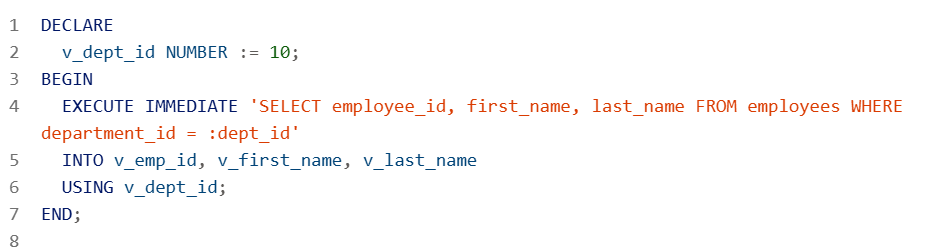
CREATE INDEX idx\_emp\_dept ON employees(department\_id);

**Explanation:**

* This creates an index on the department\_id column of the employees table.
* Indexes improve query performance by allowing faster data retrieval, especially for columns frequently used in WHERE clauses.

**3. Using Bind Variables**

**Example:**



**Explanation:**

* Bind variables (:dept\_id) are used to improve performance by allowing the database to reuse execution plans, reducing parsing overhead.

**4. Gathering Statistics**

**Example:**

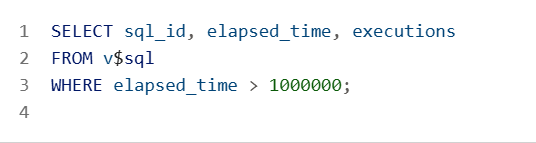


**Explanation:**

* This gathers statistics for the employees table in the HR schema.
* Up-to-date statistics help the optimizer choose the most efficient execution plans.

**5. Identifying Slow Queries**

**Example:**

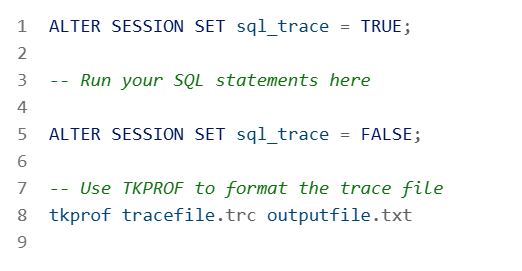


**Explanation:**

* This query retrieves SQL statements from the v$sql view that have an elapsed time greater than 1 second.
* Identifying slow queries helps in focusing tuning efforts on the most problematic SQL statements.

**6. Using SQL Trace and TKPROF**

**Example:**



**Explanation:**

* SQL Trace is enabled for the session to capture detailed execution information.
* TKPROF formats the trace file, making it easier to analyze and identify performance issues.

**7. Using SQL Access Advisor**

**Example:**

A close-up of a white background

AI-generated content may be incorrect.

**Explanation:**

* The SQL Access Advisor provides recommendations for indexes, materialized views, and partitions to improve query performance.
* This example creates a tuning task for a specific query

**Type of Index**

1. **B-tree Index**
2. **Bitmap Index**
3. **Unique Index**
4. **Composite Index**
5. **Function-based Index**
6. **Domain Index**
7. **Clustered Index**

Oracle Database supports several types of indexes, each designed to optimize query performance in different scenarios. Here are the main types of indexes along with their purposes and examples:

**1. B-tree Index**

**Purpose:**

* B-tree indexes are the default type and are used for columns with high cardinality (many unique values).
* They are efficient for range queries and exact match queries.

**Example:**

CREATE INDEX idx\_emp\_lastname ON employees(last\_name);

**Explanation:**

* This creates a B-tree index on the last\_name column of the employees table.
* Useful for queries like SELECT \* FROM employees WHERE last\_name = 'Smith'.

**2. Bitmap Index**

**Purpose:**

* Bitmap indexes are used for columns with low cardinality (few unique values).
* They are efficient for queries involving multiple conditions on low-cardinality columns.

**Example:**

CREATE BITMAP INDEX idx\_emp\_gender ON employees(gender);

**Explanation:**

* This creates a bitmap index on the gender column of the employees table.
* Useful for queries like SELECT \* FROM employees WHERE gender = 'M' AND department\_id = 10.

**3. Unique Index**

**Purpose:**

* Unique indexes enforce the uniqueness of values in the indexed column(s).
* They prevent duplicate values in the column(s).

**Example:**

CREATE UNIQUE INDEX idx\_emp\_email ON employees(email);

**Explanation:**

* This creates a unique index on the email column of the employees table.
* Ensures that each email address in the table is unique.

**4. Composite Index**

**Purpose:**

* Composite indexes (also known as concatenated indexes) are created on multiple columns.
* They are useful for queries that filter on multiple columns.

**Example:**

CREATE INDEX idx\_emp\_name\_dept ON employees(last\_name, department\_id);

**Explanation:**

* This creates a composite index on the last\_name and department\_id columns.
* Useful for queries like SELECT \* FROM employees WHERE last\_name = 'Smith' AND department\_id = 10.

**5. Function-based Index**

**Purpose:**

* Function-based indexes are created on expressions involving one or more columns.
* They are useful for queries that use functions on columns.

**Example:**

CREATE INDEX idx\_emp\_upper\_lastname ON employees(UPPER(last\_name));

**Explanation:**

* This creates a function-based index on the expression UPPER(last\_name).
* Useful for queries like SELECT \* FROM employees WHERE UPPER(last\_name) = 'SMITH'.

**6. Domain Index**

**Purpose:**

* Domain indexes are used for complex data types and custom indexing schemes.
* They are useful for specialized applications like spatial data.

**Example:**

CREATE INDEX idx\_emp\_spatial ON employees(location) INDEXTYPE IS MDSYS.SPATIAL\_INDEX;

**Explanation:**

* This creates a domain index on the location column using the MDSYS.SPATIAL\_INDEX type.
* Useful for queries involving spatial data like SELECT \* FROM employees WHERE SDO\_WITHIN\_DISTANCE(location, 'POINT(10 20)', 'distance=5') = 'TRUE'.

**7. Clustered Index**

**Purpose:**

* Clustered indexes store table rows in the order of the index.
* They are useful for improving performance of range queries.

**Example:**

CREATE CLUSTER employees\_cluster (department\_id NUMBER, last\_name VARCHAR2(50));

CREATE INDEX idx\_emp\_cluster ON CLUSTER employees\_cluster;

**Explanation:**

* This creates a clustered index on the department\_id and last\_name columns.
* Useful for queries like SELECT \* FROM employees WHERE department\_id = 10 ORDER BY last\_name.

These indexes help optimize query performance by reducing the amount of data scanned and improving data retrieval efficiency. If you have any specific questions or need further details, feel free to ask!

**Oracle DML (Data Manipulation Language) statements using the WITH clause**

**1. What is the WITH clause in Oracle SQL?**

**Answer:**

* The WITH clause, also known as subquery factoring, allows you to define a subquery block that can be referenced multiple times within the main SQL query. It helps simplify complex queries and improve readability[[1]](https://oracle-base.com/articles/misc/with-clause).

**2. How does the WITH clause improve query performance?**

**Answer:**

* The WITH clause can improve performance by reducing the need to repeatedly execute the same subquery. Oracle may choose to materialize the subquery, storing its result temporarily, which can be more efficient than re-executing the subquery multiple times[[2]](https://stackoverflow.com/questions/57610712/when-do-we-use-with-clause-and-what-are-main-benefits-of-it).

**3. Can the WITH clause be used with DML statements like INSERT, UPDATE, and DELETE?**

**Answer:**

* Yes, the WITH clause can be used with DML statements to simplify complex operations and improve readability[[3]](https://www.complexsql.com/oracle-with-clause/).

**4. Example of using the WITH clause with an INSERT statement**

**Example:**

WITH dept\_avg AS (

SELECT department\_id, AVG(salary) AS avg\_salary

FROM employees

GROUP BY department\_id

)

INSERT INTO dept\_salary\_summary (department\_id, avg\_salary)

SELECT department\_id, avg\_salary

FROM dept\_avg;

**Explanation:**

* This example calculates the average salary for each department and inserts the results into the dept\_salary\_summary table.

**5. Example of using the WITH clause with an UPDATE statement**

**Example:**

WITH emp\_bonus AS (

SELECT employee\_id, salary \* 0.10 AS bonus

FROM employees

WHERE department\_id = 10

)

UPDATE employees e

SET e.salary = e.salary + (SELECT bonus FROM emp\_bonus WHERE emp\_bonus.employee\_id = e.employee\_id)

WHERE e.department\_id = 10;

**Explanation:**

* This example calculates a 10% bonus for employees in department 10 and updates their salaries accordingly.

**6. Example of using the WITH clause with a DELETE statement**

**Example:**

WITH old\_employees AS (

SELECT employee\_id

FROM employees

WHERE hire\_date < TO\_DATE('01-JAN-2000', 'DD-MON-YYYY')

)

DELETE FROM employees

WHERE employee\_id IN (SELECT employee\_id FROM old\_employees);

**Explanation:**

* This example identifies employees hired before January 1, 2000, and deletes them from the employees table.

**7. What are the benefits of using the WITH clause?**

**Answer:**

* The WITH clause helps reduce code repetition, improve query readability, and potentially enhance performance by allowing Oracle to optimize subquery execution[[1]](https://oracle-base.com/articles/misc/with-clause).

**8. Are there any risks associated with using the WITH clause?**

**Answer:**

* In some cases, Oracle may decide to materialize the subquery, which can lead to unexpected I/O operations and affect performance. It's important to test and analyze the performance implications on a case-by-case basis[[2]](https://stackoverflow.com/questions/57610712/when-do-we-use-with-clause-and-what-are-main-benefits-of-it).

These FAQs and examples should help you understand how to use the WITH clause effectively in Oracle SQL. If you have any specific questions or need further details, feel free to ask!

A complex query:

WITH category\_sales AS (

  SELECT p.category\_id, SUM(s.amount) AS total\_sales

  FROM products p

  JOIN sales s ON p.product\_id = s.product\_id

  GROUP BY p.category\_id

),

average\_sales AS (

  SELECT p.category\_id, AVG(s.amount) AS avg\_sales\_per\_product

  FROM products p

  JOIN sales s ON p.product\_id = s.product\_id

  GROUP BY p.category\_id

),

filtered\_sales AS (

  SELECT cs.category\_id, cs.total\_sales, as.avg\_sales\_per\_product

  FROM category\_sales cs

  JOIN average\_sales as ON cs.category\_id = as.category\_id

  WHERE cs.total\_sales > 10000

)

SELECT fs.category\_id, fs.total\_sales, fs.avg\_sales\_per\_product

FROM filtered\_sales fs

ORDER BY fs.total\_sales DESC;

**Database Partition**

1. **Range Partitioning**
2. **List Partitioning**
3. **Hash Partitioning**
4. **Composite Partitioning**
5. **Interval Partitioning**
6. **Reference Partitioning**

**1. Range Partitioning**

**Purpose:**

* Divides data into partitions based on a range of values.
* Useful for time-series data or any data that can be logically grouped by ranges.

**Example:**

CREATE TABLE sales (

sale\_id NUMBER,

sale\_date DATE,

amount NUMBER

)

PARTITION BY RANGE (sale\_date) (

PARTITION p1 VALUES LESS THAN (TO\_DATE('01-JAN-2022', 'DD-MON-YYYY')),

PARTITION p2 VALUES LESS THAN (TO\_DATE('01-JAN-2023', 'DD-MON-YYYY')),

PARTITION p3 VALUES LESS THAN (TO\_DATE('01-JAN-2024', 'DD-MON-YYYY'))

);

**sssExplanation:**

* This example partitions the sales table by the sale\_date column into yearly partitions.

**2. List Partitioning**

**Purpose:**

* Divides data into partitions based on a list of discrete values.
* Useful for categorical data like regions or departments.

**Example:**

CREATE TABLE employees (

employee\_id NUMBER,

department VARCHAR2(50)

)

PARTITION BY LIST (department) (

PARTITION p\_sales VALUES ('Sales'),

PARTITION p\_hr VALUES ('HR'),

PARTITION p\_it VALUES ('IT')

);

**Explanation:**

* This example partitions the employees table by the department column into partitions based on department names.

**3. Hash Partitioning**

**Purpose:**

* Distributes data evenly across partitions using a hash function.
* Useful for evenly distributing data and avoiding hotspots.

**Example:**

CREATE TABLE orders (

order\_id NUMBER,

customer\_id NUMBER

)

PARTITION BY HASH (customer\_id) PARTITIONS 4;

**Explanation:**

* This example partitions the orders table by the customer\_id column into four partitions using a hash function.

**4. Composite Partitioning**

**Purpose:**

* Combines two or more partitioning methods (e.g., range-hash, range-list).
* Useful for complex data distribution requirements.

**Example:**

CREATE TABLE transactions (

transaction\_id NUMBER,

transaction\_date DATE,

region VARCHAR2(50)

)

PARTITION BY RANGE (transaction\_date)

SUBPARTITION BY LIST (region) (

PARTITION p1 VALUES LESS THAN (TO\_DATE('01-JAN-2022', 'DD-MON-YYYY'))

(SUBPARTITION p1\_east VALUES ('East'),

SUBPARTITION p1\_west VALUES ('West')),

PARTITION p2 VALUES LESS THAN (TO\_DATE('01-JAN-2023', 'DD-MON-YYYY'))

(SUBPARTITION p2\_east VALUES ('East'),

SUBPARTITION p2\_west VALUES ('West'))

);

**Explanation:**

* This example partitions the transactions table by transaction\_date and further subpartitions by region.

**5. Interval Partitioning**

**Purpose:**

* Automatically creates partitions based on a specified interval.
* Useful for time-series data where new partitions are created as needed.

**Example:**

CREATE TABLE logs (

log\_id NUMBER,

log\_date DATE,

message VARCHAR2(255)

)

PARTITION BY RANGE (log\_date)

INTERVAL (NUMTOYMINTERVAL(1, 'MONTH')) (

PARTITION p\_initial VALUES LESS THAN (TO\_DATE('01-JAN-2022', 'DD-MON-YYYY'))

);

**Explanation:**

* This example partitions the logs table by log\_date and automatically creates monthly partitions.

**6. Reference Partitioning**

**Purpose:**

* Partitions a child table based on the partitioning of its parent table.
* Useful for maintaining partition alignment between related tables.

**Example:**

CREATE TABLE customers (

customer\_id NUMBER,

region VARCHAR2(50)

)

PARTITION BY LIST (region) (

PARTITION p\_east VALUES ('East'),

PARTITION p\_west VALUES ('West')

);

CREATE TABLE orders (

order\_id NUMBER,

customer\_id NUMBER,

order\_date DATE

)

PARTITION BY REFERENCE (customers);

**Explanation:**

* This example partitions the orders table based on the partitioning of the customers table.

**Triggers**

Certainly! Oracle triggers are powerful tools used to automatically execute a specified action when certain events occur in the database. Here are some key points about triggers along with examples that might be useful for interview purposes:

**Types of Triggers**

1. **Row-Level Triggers**: Execute once for each row affected by the triggering event.
2. **Statement-Level Triggers**: Execute once for the entire triggering event, regardless of the number of rows affected.
3. **Before Triggers**: Execute before the triggering event.
4. **After Triggers**: Execute after the triggering event.
5. **Instead of Triggers**: Execute instead of the triggering event, typically used for views.

**Example 1: Row-Level Trigger (Before Insert)**

**Purpose**: Automatically set the created\_date column to the current date before inserting a new row into the employees table.

CREATE OR REPLACE TRIGGER trg\_before\_insert\_employee

BEFORE INSERT ON employees

FOR EACH ROW

BEGIN

:NEW.created\_date := SYSDATE;

END;

**Explanation**:

* This trigger sets the created\_date column to the current date (SYSDATE) before a new row is inserted into the employees table.

**Example 2: Statement-Level Trigger (After Update)**

**Purpose**: Log changes to the salary column in the employees table after an update.

CREATE OR REPLACE TRIGGER trg\_after\_update\_salary

AFTER UPDATE ON employees

BEGIN

INSERT INTO salary\_changes (employee\_id, old\_salary, new\_salary, change\_date)

SELECT :OLD.employee\_id, :OLD.salary, :NEW.salary, SYSDATE

FROM dual

WHERE :OLD.salary <> :NEW.salary;

END;

**Explanation**:

* This trigger logs changes to the salary column by inserting a record into the salary\_changes table after an update occurs in the employees table.

**Example 3: Instead of Trigger (For Views)**

**Purpose**: Allow updates to a view that joins multiple tables.

CREATE OR REPLACE VIEW emp\_dept\_view AS

SELECT e.employee\_id, e.first\_name, e.last\_name, d.department\_name

FROM employees e

JOIN departments d ON e.department\_id = d.department\_id;

CREATE OR REPLACE TRIGGER trg\_instead\_of\_update\_emp\_dept

INSTEAD OF UPDATE ON emp\_dept\_view

FOR EACH ROW

BEGIN

UPDATE employees

SET first\_name = :NEW.first\_name, last\_name = :NEW.last\_name

WHERE employee\_id = :OLD.employee\_id;

UPDATE departments

SET department\_name = :NEW.department\_name

WHERE department\_id = (SELECT department\_id FROM employees WHERE employee\_id = :OLD.employee\_id);

END;

**Explanation**:

* This trigger allows updates to the emp\_dept\_view view by updating the underlying employees and departments tables.

**Example 4: Row-Level Trigger (Before Delete)**

**Purpose**: Prevent deletion of rows from the employees table if the employee is a manager.

CREATE OR REPLACE TRIGGER trg\_before\_delete\_employee

BEFORE DELETE ON employees

FOR EACH ROW

BEGIN

IF :OLD.job\_title = 'Manager' THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Managers cannot be deleted.');

END IF;

END;

**Explanation**:

* This trigger raises an error and prevents deletion of rows from the employees table if the employee's job title is 'Manager'.sss

**Example 5: Statement-Level Trigger (After Insert)**

**Purpose**: Automatically update a summary table after new rows are inserted into the sales table.

CREATE OR REPLACE TRIGGER trg\_after\_insert\_sales

AFTER INSERT ON sales

BEGIN

MERGE INTO sales\_summary ss

USING (SELECT product\_id, SUM(amount) AS total\_sales FROM sales GROUP BY product\_id) s

ON (ss.product\_id = s.product\_id)

WHEN MATCHED THEN

UPDATE SET ss.total\_sales = s.total\_sales

WHEN NOT MATCHED THEN

INSERT (product\_id, total\_sales) VALUES (s.product\_id, s.total\_sales);

END;

**Explanation**:

* This trigger updates the sales\_summary table to reflect the total sales for each product after new rows are inserted into the sales table.

**Exceptional**

**1. Predefined Exceptions**

These are exceptions that are automatically defined by Oracle.

* **NO*DATA*FOUND**: Raised when a SELECT INTO statement returns no rows.
* **TOO*MANY*ROWS**: Raised when a SELECT INTO statement returns more than one row.
* **ZERO\_DIVIDE**: Raised when an attempt is made to divide a number by zero.
* **INVALID\_CURSOR**: Raised when an illegal cursor operation is attempted.

**Example: Handling NO*DATA*FOUND**

DECLARE

v\_employee\_name VARCHAR2(50);

BEGIN

SELECT first\_name INTO v\_employee\_name

FROM employees

WHERE employee\_id = 9999; -- Assuming this ID does not exist

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

DBMS\_OUTPUT.PUT\_LINE('No employee found with the given ID.');

END;

**2. User-Defined Exceptions**

These are exceptions defined by the programmer.

**Example: Defining and Raising a User-Defined Exception**

DECLARE

e\_invalid\_salary EXCEPTION;

v\_salary NUMBER := -1000;

BEGIN

IF v\_salary < 0 THEN

RAISE e\_invalid\_salary;

END IF;

EXCEPTION

WHEN e\_invalid\_salary THEN

DBMS\_OUTPUT.PUT\_LINE('Salary cannot be negative.');

END;

**3. Named System Exceptions**

These are predefined exceptions that have names assigned to them.

**Example: Handling TOO*MANY*ROWS**

DECLARE

v\_employee\_name VARCHAR2(50);

BEGIN

SELECT first\_name INTO v\_employee\_name

FROM employees

WHERE department\_id = 10; -- Assuming this returns multiple rows

EXCEPTION

WHEN TOO\_MANY\_ROWS THEN

DBMS\_OUTPUT.PUT\_LINE('More than one employee found for the given department.');

END;

**4. Unhandled Exceptions (OTHERS)**

This handler catches all exceptions that are not explicitly handled.

**Example: Using OTHERS Handler**

DECLARE

v\_employee\_name VARCHAR2(50);

BEGIN

SELECT first\_name INTO v\_employee\_name

FROM employees

WHERE employee\_id = 9999; -- Assuming this ID does not exist

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

DBMS\_OUTPUT.PUT\_LINE('No employee found with the given ID.');

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('An unexpected error occurred: ' || SQLERRM);

END;

**5. Using PRAGMA EXCEPTION\_INIT**

This allows you to associate an Oracle error number with a user-defined exception.

**Example: Using PRAGMA EXCEPTION\_INIT**

DECLARE

e\_custom\_exception EXCEPTION;

PRAGMA EXCEPTION\_INIT(e\_custom\_exception, -20001);

BEGIN

RAISE\_APPLICATION\_ERROR(-20001, 'Custom error occurred.');

EXCEPTION

WHEN e\_custom\_exception THEN

DBMS\_OUTPUT.PUT\_LINE('Handled custom exception.');

END;

**6. Using SQLCODE and SQLERRM**

These functions provide the error code and message of the exception.

**Example: Using SQLCODE and SQLERRM**

DECLARE

v\_employee\_name VARCHAR2(50);

BEGIN

SELECT first\_name INTO v\_employee\_name

FROM employees

WHERE employee\_id = 9999; -- Assuming this ID does not exist

EXCEPTION

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('Error code: ' || SQLCODE);

DBMS\_OUTPUT.PUT\_LINE('Error message: ' || SQLERRM);

END;

**7. Raising Exceptions in Nested Blocks**

Exceptions can be raised and handled within nested PL/SQL blocks.

**Example: Nested Exception Handling**

DECLARE

v\_employee\_name VARCHAR2(50);

BEGIN

BEGIN

SELECT first\_name INTO v\_employee\_name

FROM employees

WHERE employee\_id = 9999; -- Assuming this ID does not exist

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

DBMS\_OUTPUT.PUT\_LINE('Inner block: No employee found.');

RAISE; -- Rethrow the exception

END;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

DBMS\_OUTPUT.PUT\_LINE('Outer block: No employee found.');

END;

**Most commonly encountered Oracle errors**

1. **ORA-12154: TNS:could not resolve the connect identifier specified**
   * **Description**: This error occurs when Oracle cannot resolve the database connection string.
   * **Example**:
2. sqlplus username/password@nonexistent\_db
   * **Solution**: Check the tnsnames.ora file for typos or missing entries.
3. **ORA-00942: Table or View Does Not Exist**
   * **Description**: This error happens when you reference a table or view that does not exist or you don't have access to.
   * **Example**:
4. SELECT \* FROM non\_existent\_table;
   * **Solution**: Verify the table/view name and check user permissions.
5. GRANT SELECT ON table\_name TO user;
6. **ORA-01017: Invalid Username/Password; Logon Denied**
   * **Description**: This error occurs when the provided username or password is incorrect.
   * **Example**:
7. sqlplus wrong\_user/wrong\_password@db
   * **Solution**: Double-check the credentials and reset the password if necessary.

ALTER USER user\_name IDENTIFIED BY new\_password;

1. **ORA-00001: Unique Constraint Violated**
   * **Description**: This error is triggered when you try to insert a duplicate value into a unique column.
   * **Example**:

INSERT INTO employees (employee\_id, name) VALUES (1, 'John');

INSERT INTO employees (employee\_id, name) VALUES (1, 'Jane');

* + **Solution**: Ensure the data being inserted is unique.

INSERT INTO employees (employee\_id, name) VALUES (2, 'Jane');

1. **ORA-01843: Not a Valid Month**
   * **Description**: This error occurs when a date is not in the correct format.
   * **Example**: sql SELECT TO\_DATE('2025-15-04', 'YYYY-MM-DD') FROM dual;
   * **Solution**: Use the correct date format. sql SELECT TO\_DATE('2025-04-15', 'YYYY-MM-DD') FROM dual;

**Data warehouse interview questions**

1. **What is a Data Warehouse?**
   * A data warehouse is a centralized repository that stores structured data from various sources. It is used primarily for reporting and data analysis, offering a unified, historical view of a company's data.
2. **What is the difference between OLAP and OLTP?**
   * **OLAP (Online Analytical Processing)**: Optimized for complex queries and historical data analysis. It is designed for read-heavy operations such as generating reports, visualizations, and trend analysis.
   * **OLTP (Online Transaction Processing)**: Focuses on real-time transaction management, such as processing orders or recording customer payments. It is optimized for fast, write-heavy operations.
3. **What is a Dimension Table?**
   * A dimension table contains attributes of measurements stored in fact tables. This table consists of hierarchies, categories, and logic that can be used to traverse in nodes.
4. **What is a Fact Table?**
   * A fact table contains the measurements of business processes and includes foreign keys for the dimension tables.
5. **What are the stages of Data Warehousing?**
   * The stages include:
     1. Data Warehouse Offline
     2. Operational Database Offline
     3. Real-Time Data Warehouse
     4. Integrated Data Warehouse
6. **What is Data Mining?**
   * Data mining is the process of analyzing data from different perspectives and summarizing it into useful information. It involves querying and retrieving data from databases in various formats.
7. **What is ETL?**
   * ETL stands for Extract, Transform, Load. It is a process used to extract data from various sources, transform it into a suitable format, and load it into a data warehouse.
8. **What is a Star Schema?**
   * A star schema is a type of database schema that consists of a central fact table surrounded by dimension tables. It is called a star schema because the diagram resembles a star.
9. **What is a Snowflake Schema?**
   * A snowflake schema is a more complex version of a star schema where dimension tables are normalized into multiple related tables, resembling a snowflake shape.
10. **What is a Data Mart?**
    * A data mart is a subset of a data warehouse, focused on a specific business area or department. It is designed to meet the needs of a particular group of users.

**How do you approach incremental loading in ETL processes?**

Incremental loading is a technique to update a data warehouse by loading only new or changed data, reducing processing time and resource usage. Common approaches include:

* **Timestamps:** Use a "last modified" timestamp column to identify new or updated records.
* **Change data capture (CDC):** Detect and extract changes directly from source systems, often through database logs or triggers.
* **Snapshot comparison:** Compare current data with previously loaded data to identify changes.

Incremental loading is especially important in large-scale data warehouses where full reloads would be impractical

**How would you optimize a query that takes too long to execute?**

This question combines coding and problem-solving. Steps to optimize:

* **Rewrite query:** Use efficient joins, avoid unnecessary subqueries, and apply proper indexing.
* **Use EXPLAIN plan:** Analyze the query execution plan to identify bottlenecks.
* **Partitioning:** If the table is large, use partitioning to reduce the data scanned.
* **Materialized views:** Pre-compute and cache expensive operations.

Example query improvement:

**Before:**

SELECT \*

FROM orders

WHERE YEAR(order\_date) = 2023;

[**Powered By**](https://www.datacamp.com/datalab)

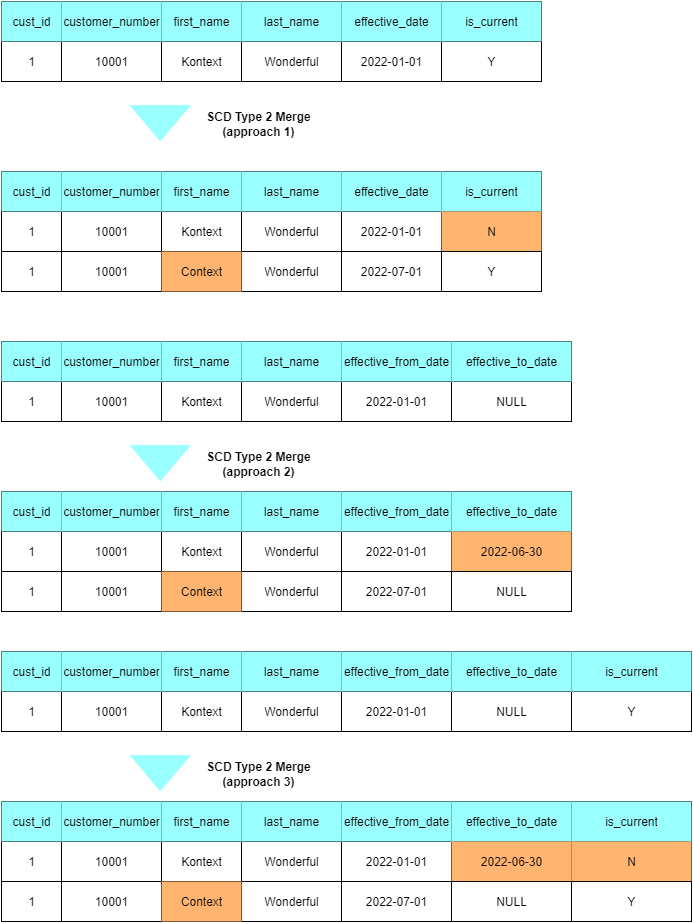
**Optimized:**

SELECT \*

FROM orders

WHERE order\_date >= '2023-01-01' AND order\_date < '2024-01-01';

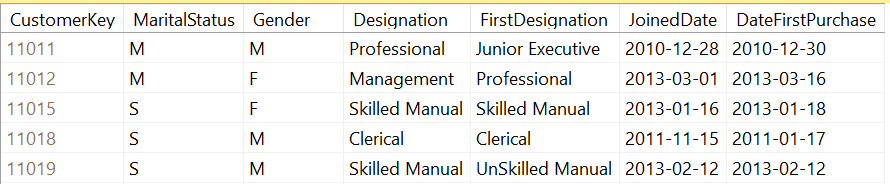
**Slowly changing Dimension**

****

**SCD Type 0**

There are situations where you ignore any changes. For example, when an employee joined an organization, there are joined related attributes such as joined Designation and JoinedDate, etc. that should not change over time.

The following is the example for Type 0 of Slowly Changing Dimensions in Data Warehouse.

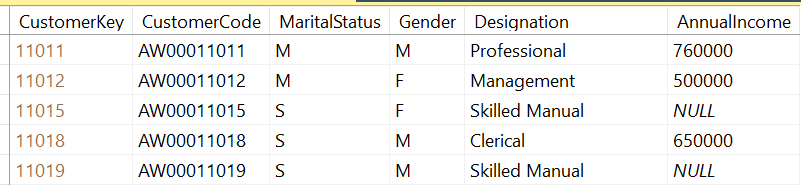


In the above Customer Dimension, **FirstDesignation**, **JoinedDate** and **DateFirstPurchase** are the attributes that will not be updated which is Type 0 SCD.

**SCD Type 1**

In the Type 1 SCD, you simply overwrite data in dimensions. There can be situations where you don’t have the entire data when the record is initiated in the dimension. For example, when the customer record is initiated, you may not get all attributes. Therefore, when the customer record is initiated at the operational database, there will be empty or null records in the customer records. Once the ETL is executed, those empty records will be created in the data warehouse. Once these attributes are filled in the operational databases, that has to be updated in the data warehouse.

Type 1 SCDs are identifying if the existing attributes are null and you are receiving a value from the operational table.

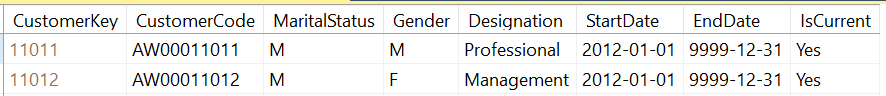


In the above Customer Dimension table, the **AnnualIncome** of customers **CustomerKey** 11015 and 11019 are NULL. When these records are updated in the operational database, those values should be updated in the data warehouse without considering those are historical values.

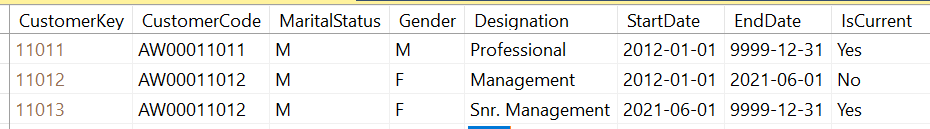
SCD Type 2

Type 2 Slowly Changing Dimensions in Data warehouse is the most popular dimension that is used in the data warehouse. As we discussed data warehouse is used for data analysis. If you need to analyze data, you need to accommodate historical aspects of data. Let us see how we can implement SCD Type 2.

For the SCD Type 2, we need to include three more attributes such as **StartDate**, **EndDate** and **IsCurrent** as shown below.



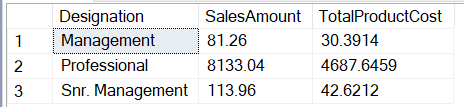
In the above customer dimension, there are two records and let us say that customer whose CustomerCode is AW00011012, has been promoted to Senior Management. However, if you simply update the record with the new value, you will not see the previous records. Therefore, a new record will be created with a new **CustomerKey** and a new Designation. However, other attributes will be remaining the same.



As you can see in the above figure, **CustomerCode** AW00011012 has a new record with 11013. All the new transactions will be related to **CustomerKey** 11013 while previous transactional are related to **CustomerKey** 11012. This mechanism helps to preserve the historic aspect of the customer as shown in the below query.

|  |  |
| --- | --- |
| 1  2  3  4 | SELECT  C.Designation,SUM(SalesAmount) SalesAmount,SUM(TotalProductCost)  TotalProductCost  FROM FactInternetSales F  INNER JOIN Dim\_Customer C ON F.CustomerKey = C.CustomerKey  GROUP BY C.Designation |
|  |  |

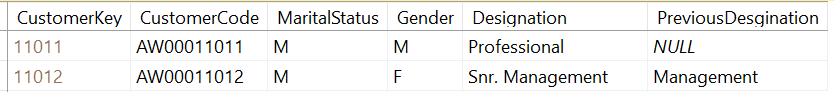
Once the query is executed, the following results will be observed.



As you can see Management designation can be seen in the above result which means that it has covered the historical aspects. Type 2 SCD is one of the implementations where you cannot avoid surrogate keys in dimensional tables in the data warehouse.

SCD Type 3

Type 3 Slowly Changing Dimension in Data warehouse is a simple implementation where history will be kept in the additional column. If we relate the same scenario that we discussed under Type 2 SCD to Type 3 SCD, the customer dimension would look like below.



As you can see, historical aspects of the data are preserved as a different column. However, this method will not be scalable if you want to preserve history. Further, this technique will allow only to keep the last version of the history, unlike Type 2 SCD.

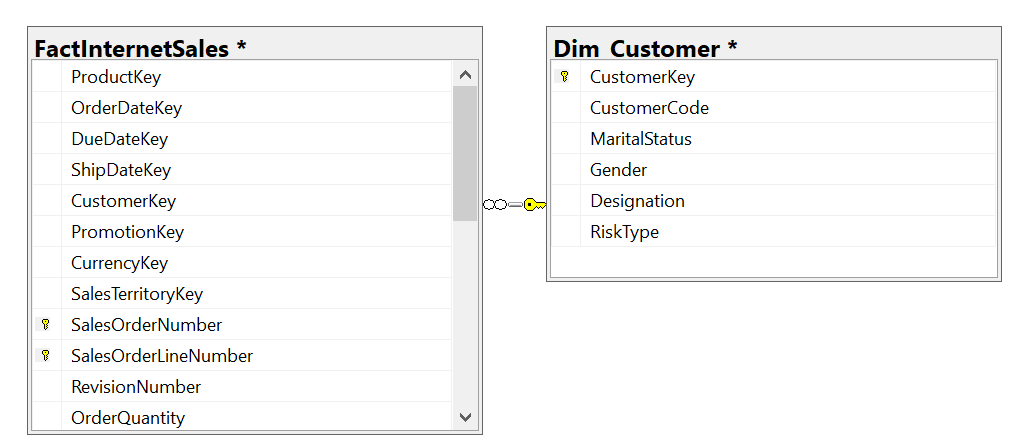
Typically, this would be better suited to implement name changes of an employee. In some cases, female employees will change their names after their marriage. In such situations, you can use Type 3 SCD since these types of changes will not occur rapidly.

**SCD Type 4**

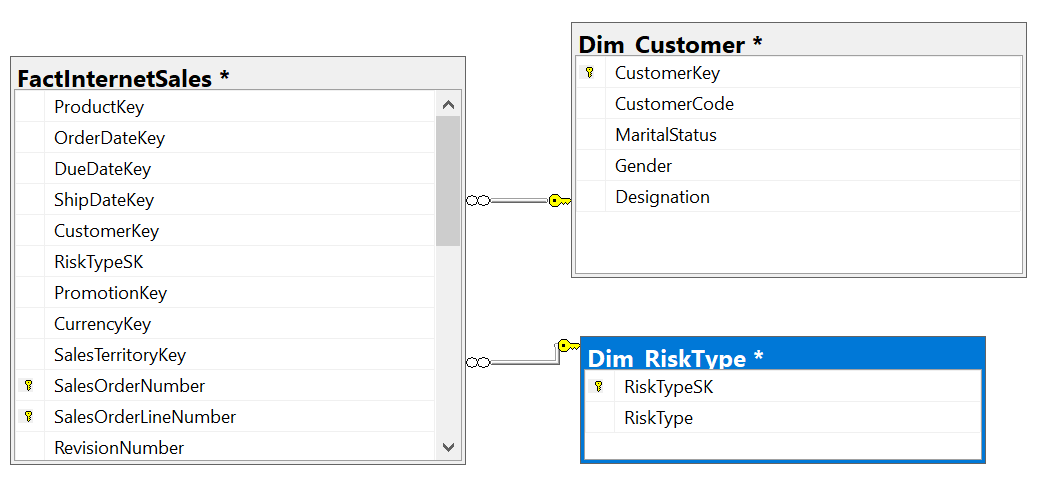
As we discussed in SCD type 2, we maintain the history by adding a different version of the row to the dimension. However, if the changes are rapid in nature Type 2 SCD will not be scalable.

For example, let us assume we want to keep the customer risk type depending on his previous payment. Since this is an attribute related to the customer, it should be stored in a customer dimension. This means every month there will be a new version of the customer record. If you have 1000 customers, you are looking at 12,000 records per month. As you can imagine this Slowly Changing Dimensions in Data Warehouse is not scalable.

Following is the relationship between the Fact and the Customer Dimension table.

[](https://www.sqlshack.com/wp-content/uploads/2021/09/schema-design-before-implementing-type-4-scd-.png)

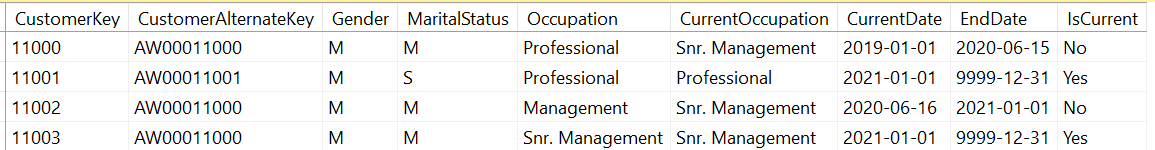
SCD Type 4 is introduced in order to fix this issue. In this technique, a rapidly changing column is moved out of the dimension and is moved to a new dimension table. This new dimension is linked to the fact table as shown in the below diagram.

[](https://www.sqlshack.com/wp-content/uploads/2021/09/schema-design-after-implementing-type-4-scd-.png)

With the above implementation of Type 4 Slowly Changing Dimensions in Data Warehouse, you are eliminating the unnecessary volume in the main dimension. However, still you have the capabilities of performing the required analysis.

**SCD Type 6**

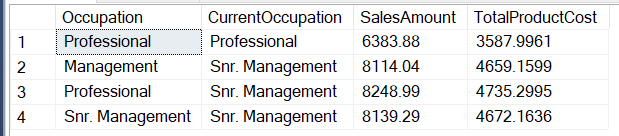
Type 6 Slowly Changing Dimensions in Data Warehouse is a combination of Type 2 and Type 3 SCDs. This means that Type 6 SCD has both columns are rows in its implementation.

[](https://www.sqlshack.com/wp-content/uploads/2021/09/sample-dataset-for-type-6-scd-.png)

With this implementation, you can further improve the analytical capabilities in the data warehouse. If you want to find out an analysis between current and historical occupation, you can use the following query.

|  |  |
| --- | --- |
|  | SELECT  C.Occupation,C.CurrentOccupation,  SUM(SalesAmount) SalesAmount,SUM(TotalProductCost)  TotalProductCost  FROM FactInternetSales F  INNER JOIN Dim\_Customer C ON F.CustomerKey = C.CustomerKey  GROUP BY C.Occupation,C.CurrentOccupation |

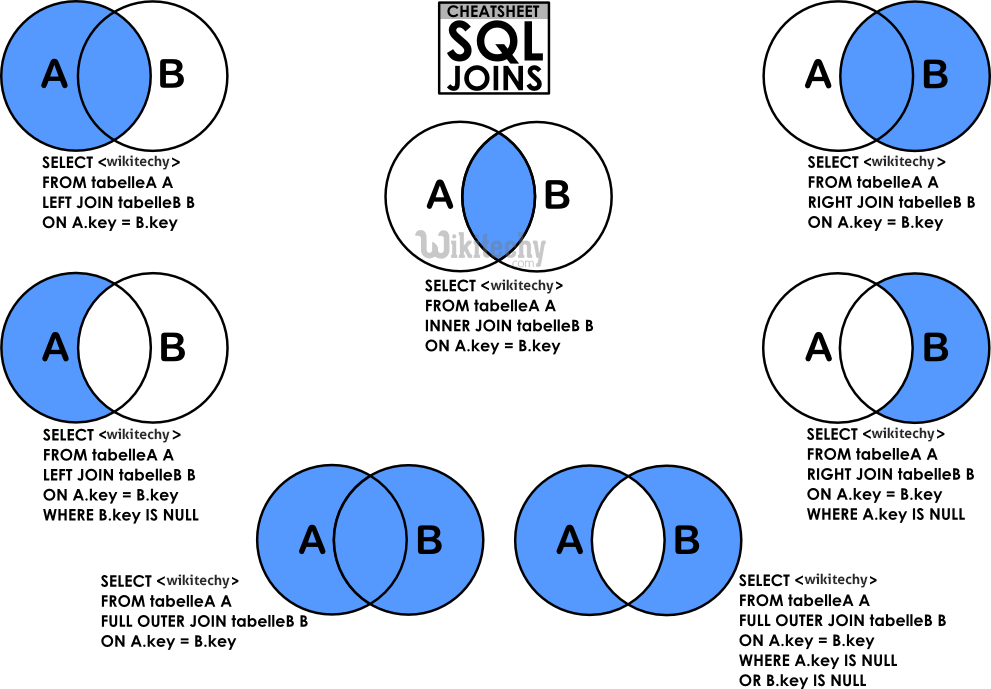
Above query will provide the following result:



Without Type 6, Slowly Changing Dimensions in Data Warehouse, complex queries have to be used.

In the Type 6 SCD, not only the current occupation, you can use the first occupation as well in order to provide more analysis.

**Oracle JOINs**



**Merge into fact table**

MERGE INTO fact\_table AS target

USING staging\_table AS source

ON target.id = source.id

WHEN MATCHED THEN

UPDATE SET target.value = source.value

WHEN NOT MATCHED THEN

INSERT (id, value) VALUES (source.id, source.value);

**Delete duplicate records**

DELETE

FROM some\_table

WHERE id IN (

SELECT id

FROM (

SELECT id, ROW\_NUMBER() OVER (PARTITION BY id ORDER BY created\_at) AS row\_num

FROM some\_table

) AS duplicates

WHERE row\_num > 1

);

**Oracle Cursor**

Certainly! Here are some frequently asked questions (FAQs) about Oracle Cursors, along with examples:

**1. What is a Cursor in Oracle?**

**Answer**: A cursor is a pointer that points to the result set of a query. It allows you to process individual rows returned by the query.

**2. What are the types of Cursors in Oracle?**

**Answer**: There are two types of cursors:

* **Implicit Cursors**: Automatically created by Oracle for single-row queries.
* **Explicit Cursors**: Defined by the programmer for queries that return multiple rows.

**3. How do you declare an Explicit Cursor?**

**Example**:

DECLARE

CURSOR emp\_cursor IS

SELECT employee\_id, first\_name, last\_name FROM employees;

**Explanation**: This declares a cursor named emp\_cursor that selects employee\_id, first\_name, and last\_name from the employees table.

**4. How do you open a Cursor?**

**Example**:

OPEN emp\_cursor;

**Explanation**: This opens the emp\_cursor so that rows can be fetched from it.

**5. How do you fetch data from a Cursor?**

**Example**:

FETCH emp\_cursor INTO v\_employee\_id, v\_first\_name, v\_last\_name;

**Explanation**: This fetches the current row from emp\_cursor into the variables v\_employee\_id, v\_first\_name, and v\_last\_name.

**6. How do you close a Cursor?**

**Example**:

CLOSE emp\_cursor;

**Explanation**: This closes the emp\_cursor and releases any resources associated with it.

**7. What is a Cursor FOR Loop?**

**Example**:

FOR emp\_rec IN emp\_cursor LOOP

DBMS\_OUTPUT.PUT\_LINE(emp\_rec.first\_name || ' ' || emp\_rec.last\_name);

END LOOP;

**Explanation**: This loop automatically opens the cursor, fetches each row, and closes the cursor when done. It prints the first and last names of employees.

**8. What are Cursor Attributes?**

**Answer**: Cursor attributes provide information about the state of the cursor. Common attributes include:

* %FOUND: Returns TRUE if the last fetch returned a row.
* %NOTFOUND: Returns TRUE if the last fetch did not return a row.
* %ROWCOUNT: Returns the number of rows fetched so far.
* %ISOPEN: Returns TRUE if the cursor is open.

**9. How do you handle exceptions with Cursors?**

**Example**:

BEGIN

OPEN emp\_cursor;

FETCH emp\_cursor INTO v\_employee\_id, v\_first\_name, v\_last\_name;

IF emp\_cursor%NOTFOUND THEN

DBMS\_OUTPUT.PUT\_LINE('No data found.');

END IF;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

DBMS\_OUTPUT.PUT\_LINE('No data found exception.');

WHEN TOO\_MANY\_ROWS THEN

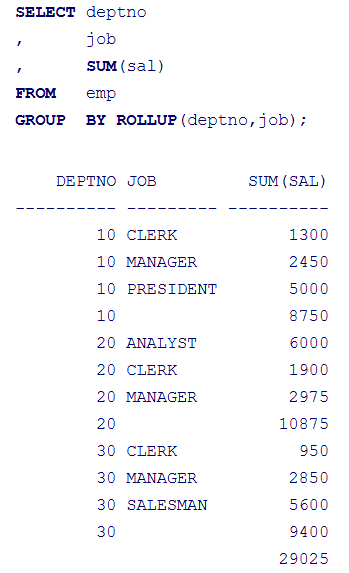
DBMS\_OUTPUT.PUT\_LINE('Too many rows exception.');

END;

**Explanation**: This handles exceptions such as NO\_DATA\_FOUND and TOO\_MANY\_ROWS when fetching data from the cursor.

**Analytical Functions**

**Rollup**

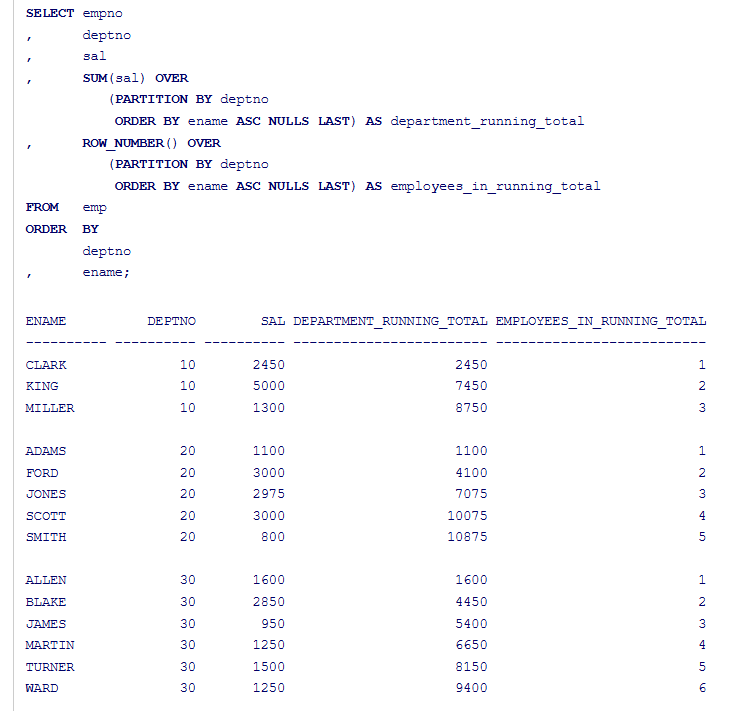
****

**Cube**

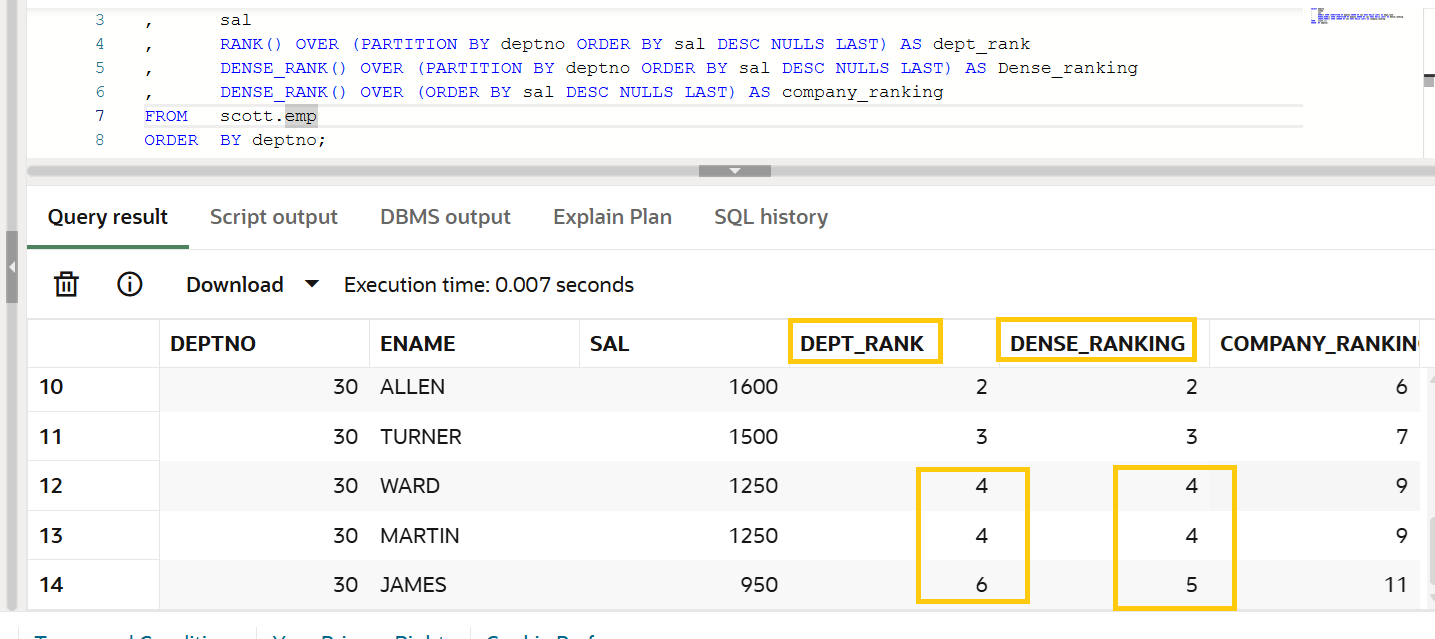
**A screenshot of a paper

AI-generated content may be incorrect.**

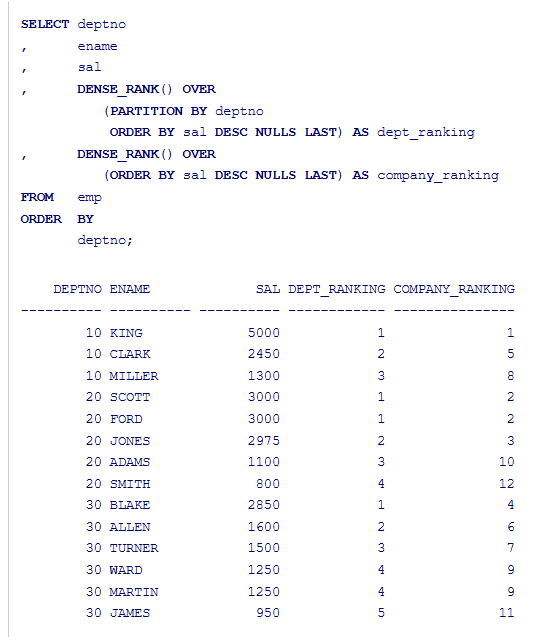
**Row Number**

****

**Rank**

****

**Dense Rank**

****

**Procedure / function and packages**

**Creating a Functions**

**Description**: A function is a subprogram that returns a single value. It can be used in SQL statements.

**Example**:

CREATE OR REPLACE FUNCTION get\_complete\_address(in\_person\_id IN NUMBER) RETURN VARCHAR2 IS

person\_details VARCHAR2(130);

BEGIN

SELECT 'Name-' || person.first\_name || ' ' || person.last\_name || ', City-' || address.city || ', State-' || address.state || ', Country-' || address.country || ', ZIP Code-' || address.zip\_code

INTO person\_details

FROM person\_info person, person\_address\_details address

WHERE person.person\_id = in\_person\_id AND address.person\_id = person.person\_id;

RETURN person\_details;

END get\_complete\_address;

**Explanation**: This function get\_complete\_address takes a person\_id as input and returns a formatted address string. It uses a SELECT INTO statement to fetch and concatenate the address details[[1]](https://mkyong.com/oracle/oracle-plsql-create-function-example/).

**Creating a Procedure**

**Description**: A procedure is a subprogram that performs a specific action but does not return a value directly.

**Example**:

CREATE OR REPLACE PROCEDURE print\_contact(in\_customer\_id IN NUMBER) IS

r\_contact contacts%ROWTYPE;

BEGIN

-- Get contact based on customer id

SELECT \* INTO r\_contact FROM contacts WHERE customer\_id = in\_customer\_id;

-- Print out contact's information

DBMS\_OUTPUT.PUT\_LINE(r\_contact.first\_name || ' ' || r\_contact.last\_name || '<' || r\_contact.email || '>');

EXCEPTION

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE(SQLERRM);

END print\_contact;

**Explanation**: This procedure print\_contact takes a customer\_id as input and prints the customer's contact information. It uses DBMS\_OUTPUT.PUT\_LINE to display the information and includes exception handling[[2]](https://www.oracletutorial.com/plsql-tutorial/plsql-procedure/).

**Interview Tipssssss**

1. **Understand the Syntax**: Be familiar with the syntax for creating functions and procedures, including the CREATE OR REPLACE statement.
2. **Know the Differences**: Be able to explain the differences between functions and procedures. Functions return values and can be used in SQL statements, while procedures perform actions and do not return values directly.
3. **Use Parameters**: Understand how to use IN, OUT, and INOUT parameters in procedures and functions.
4. **Handle Exceptions**: Be prepared to discuss exception handling in PL/SQL, including common exceptions like NO\_DATA\_FOUND and TOO\_MANY\_ROWS.
5. **Practice Examples**: Practice writing and explaining examples similar to the ones provided above. Be ready to discuss how they work and any potential issues.

These examples and tips should help you prepare effectively for your interview[[3]](https://docs.oracle.com/en/database/oracle/oracle-database/21/lnpls/CREATE-FUNCTION-statement.html)[[1]](https://mkyong.com/oracle/oracle-plsql-create-function-example/)[[2]](https://www.oracletutorial.com/plsql-tutorial/plsql-procedure/). If you need more detailed explanations or additional examples, feel free to ask!

**Chandani interview**

1. How Triggers worksss

2. Explain package

3. Which database objects used the most

4. How to transfer data from one database to another database

5. Can we access one schema table from another schema

6. Partition types and which are the partition i have used and explain how to do that

7. If one of the backed dated data deleted from the partition can we add it again , how ?

8. Normal table and global temporary table difference

9. oracle hints

10. Gather stats

11. What Explain plan shows (Plan Table)

12. How to optimize/tune a query

13. Truncate and drop command differences

14. If temporary table is created and 5 tabs open can we use same table in another tab  (Can we access temporary tables in another session ?)

What is Joining Methods (Hash Join, ..)

|  |  |  |  |
| --- | --- | --- | --- |
| **Country** | **Reg\_date** | **Reg\_users** | **Running Total** |
| **England** | **7/1/2024** | **25** |  |
| **England** | **7/2/2024** | **12** |  |
| **England** | **7/3/2024** | **10** |  |
| **India** | **7/1/2024** | **32** |  |
| **India** | **7/2/2024** | **15** |  |
| **India** | **7/3/2024** | **6** |  |

Mamata Interview Questions

|  |  |  |
| --- | --- | --- |
| Team A | Team B | Win |
| India | Pakistan | India |
| India | Srilanka | India |
| Srilanka | Pakistan | Pakistan |
| Srilanka | India | Srilanka |
| Australia | England | Tie |
| India | Australia | Tie |
| India | England | India |

Win = 2, Tie = 1, Loss =0

|  |  |
| --- | --- |
| Team | Points |
| India | 7 |
| Srilanka | 2 |
| Australia | 2 |
| Pakistan | 2 |
| England | 1 |

|  |  |  |
| --- | --- | --- |
| Employee Name | Employee ID | Manager ID |
| Sudha | 2 |  |
| Giri | 3 | 2 |
| Siva | 4 | 2 |
| Ram | 5 | 3 |
| Hari | 6 | 4 |

|  |  |  |  |
| --- | --- | --- | --- |
| Employee Name | Employee ID | Manager ID | Manager Name |
| Sudha | 2 |  |  |
| Giri | 3 | 2 | Sudha |
| Siva | 4 | 2 | Sudha |
| Ram | 5 | 3 | Giri |
| Hari | 6 | 4 | Siva |

**Anjali**

1. What is Hints and which hints you have Used

2. Global Temporary Tables, Can we create primary key on GT, can we do gather stats on GT and in which scenario

3. What is Dynamic Sampling why its used

4. If the EMP\_Id column is already duplicated can we create primary/unique constraints on it

5. How to drop a column from a table which has 500 millions of data and its shouldn't cause any performance issue

6. Mutating Trigger Error and how to avoid it

7. Can we subdivide a Partition if Yes then how ?

8. can we create partitions on a table which has already billions of data if yes then what's the approach ?

9. which Oracle package is used to convert a non partitioned table into a partitioned table

10. if some data has been mistakenly deleted from a table and the same table had a backup table before with all the correct data how will you insert those missed data again from the backup table to main table without having any slowness or performance issue

11. Performance Tuning strategy

12. Access Paths, Joining Methods

13. what can you check in Explain Plan describe

14. how to check statistics stale percentage of data

15. where to check parallelism of the running query in Explain Plan

16. RAG (Retrieval Augmented Generation)

17. How we can create automatic range partitions

**Q: Based on below table create cumulative Sum**

Ename  Salary

SMITH   800

JAMES  950

ADAMS  1100

WARD   1250

MARTIN  1250

MILLER  1300

**sResults:**

Ename  Salary  Cummulative\_Sum

SMITH   800   800

JAMES  950   1750

ADAMS  1100   2850

WARD   1250   4100

MARTIN  1250  5350

MILLER  1300  6650

**Answers:**

select

ename,

salary,

sum(salary) over(

order by

salary rows between unbounded preceding

and current row

) as cummulative\_sum

from

emp;

Team A   Team B   Win

India   Pakistan   India

India   Srilanka   India

Srilanka   Pakistan   Pakistan

Srilanka   India   Srilanka

Australia   England   Tie

India   Australia   Tie

India   England   India

**Answers:**

Team   Points

India   7

Srilanka  2

Australia  2

Pakistan  2

England  1

Win - 2 Tie - Both Team 1 point each and Lossing - 0 points

**Query:**

select

team,

sum(points) as points

from

(

select

teamA as team,

case when win = teamA then 2 when win = '' Tie then 1 else 0 end as points

from

match

Union all

select

teamB as team,

case when win = teamB then 2 when win = '' Tie then 1 else 0 end as points

from

match

)

group by

team

order by

points desc;

Date  Stocks -- This is the Main table below two tables we have to achieve by writing sql query

1-Sep  34

2-Sep  32

3-Sep  32

4-Sep  20

5-Sep  50

6-Sep  50

7-Sep  46

Date  Old Stocks  Stocks

1-Sep  0  34

2-Sep  34  32

3-Sep  32  32

4-Sep  32  20

5-Sep  20  50

6-Sep  50  50

7-Sep  50  46

Date  Old Stocks  Stocks  Sales

1-Sep  0  34  0

2-Sep  34  32  2

3-Sep  32  32  0

4-Sep  32  20  2

5-Sep  20  50  0

6-Sep  50  50  0

7-Sep  50  46  4

-- with CTE as (select date,stocks,lag(stocks) over(order by date) as old\_stocks

from sales order by date) a select date,old\_stocks,stock,stocks - old\_stocks sales\_per from sales,a on [a.date](http://a.date)=[sales.date](http://sales.date" \t "_blank" \o "http://sales.date/);

**Pavitra Interview FAQs**

1)When we should go for option of "rebuilding indexes" .In which scenario?

2)In which scenario you would choose the option of "gathering statistics". And when we gatherstatistics for a table etc exactly what happens in backend ?

3)When we drop an table whether its indexes are dropped automatically

4)If there is table name "tablea" in schema1 ie:schema1.tablea and if we want to insert the data in same tablea from another schema ie schema2 . can we do that ?

Write a query to grant permission to schema2 for a table which is present in schema1 ie :schema1.tablea

5)How can we copy data from one db to another?

If using databaselinks where we will create the database link

6)Consider an table which has a column with numeric values .

If we create an range partition with values like 40 to 60

And if the values mentioned in the partition are not present in table column, then whether the partition will be created or not?

7)Can we drop a table in package.

8)Consider a package which has insert , update etc code inside it

And if we want to capture how many rows are inserted or updated in a table through that package ,how can we do that?

9)What is exchange partition

10)What is split partition

11)If we have "tablea" in schema1 and if we want to access the same "tablea" from schema2 but with different name . Then how can we do that

Ans : we can create synonym for "tablea" in schema1 and use it in schema2

12)while using parallel hints what is the max number we can mention

Eg: parallel(4). Instead of 4 what max we can use

13) When we use parallel(4). Each of the query will go to sid in backend. So what happens in backend sid

Q-1 : What is Hints and which hints you have Used

In Oracle, **hints** are directives that you can include in SQL statements to influence the execution plan chosen by the Oracle optimizer. They can be used to improve performance by guiding the optimizer to use specific indexes, join methods, or other execution strategies.

**Commonly Used Hintsss**

1. **INDEX**: Forces the optimizer to use a specific index.

SELECT /\*+ INDEX(e emp\_idx) \*/ employee\_id, first\_name FROM employees e WHERE employee\_id = 100;

1. **FULL**: Forces a full table scan.

SELECT /\*+ FULL(e) \*/ employee\_id, first\_name FROM employees e;

1. **USE\_NL**: Forces a nested loop join.

SELECT /\*+ USE\_NL(e d) \*/

 e.employee\_id,

d.department\_name

FROM hr.employees e

JOIN hr.departments d ON e.department\_id = d.department\_id;

1. **PARALLEL**: Enables parallel execution.

SELECT /\*+ PARALLEL(e, 4) \*/ employee\_id, first\_name FROM employees e;

1. **NO\_INDEX**: Prevents the use of a specific index.

SELECT /\*+ NO\_INDEX(e emp\_idx) \*/ employee\_id, first\_name

FROM employees e WHERE employee\_id = 100;

**Example with Hints**

Here's an example of a query using the INDEX hint:

SELECT /\*+ INDEX(e emp\_idx) \*/

employee\_id, first\_name, last\_name

FROM employees e WHERE employee\_id = 100;

**Explanation**

* **/\*+ INDEX(e emp\_idx) \*/**: This hint tells the optimizer to use the emp\_idx index on the employees table (e).

Hints can be very powerful, but they should be used judiciously as they can sometimes lead to suboptimal performance if not used correctly. If you have any specific scenario or query in mind, feel free to share, and I can help you with the appropriate hint!

Q: When we should go for option of "rebuilding indexes" , In which scenario?

Rebuilding indexes in Oracle can be beneficial in specific scenarios to improve performance and maintain database efficiency. Here are some common situations where rebuilding indexes might be necessary:

1. **High Percentage of Deleted Rows**:
   * If the percentage of deleted rows in an index exceeds 30% of the total rows, it may indicate fragmentation. This can be checked using the DEL\_LF\_ROWS and LF\_ROWS columns in the INDEX\_STATS view
2. **Height of the Index**:
   * If the height (or depth) of the index tree is greater than 4, it might be a sign that the index is becoming inefficient
3. **Significant Changes in Data**:
   * After performing bulk DELETE or INSERT operations, the index might become fragmented. Rebuilding the index can help in reorganizing the data and improving performance
4. **Partition Maintenance**:
   * After operations like dropping, truncating, splitting, or merging partitions, rebuilding the partitioned indexes can help maintain performance and integrity s
5. **Performance Degradation**:
   * If you notice a significant degradation in query performance and suspect that the index might be the cause, rebuilding the index can sometimes resolve the issue

**Example of Rebuilding an Index**

Here's how you can rebuild an index in Oracle:

ALTER INDEX employees\_idx REBUILD;

**Explanation**

* **ALTER INDEX employees\_idx REBUILD**: This command rebuilds the index named employees\_idx.ssss

Rebuilding indexes should be done judiciously, as it can be resource-intensive. It's always a good practice to monitor performance before and after the rebuild to ensure that it has the desired effect.

Q: In which scenario you would choose the option of "gathering statistics". And when we gatherstatistics for a table etc exactly what happens in backend ?

Gathering statistics in Oracle is crucial for optimizing query performance. Here are some scenarios where you should consider gathering statistics:

**Scenarios for Gathering Statistics**

1. **After Bulk Data Changes**:
   * When you perform large INSERT, UPDATE, or DELETE operations, the data distribution changes significantly. Gathering statistics helps the optimizer understand the new data distribution
2. **New Indexes or Partitions**:
   * After creating new indexes or partitions, gathering statistics ensures the optimizer has accurate information to make efficient query plans
3. **Stale Statistics**:
   * Regularly check for **stale statistics**. If the statistics are outdated, the optimizer might choose suboptimal execution plans
4. **Performance Issues**:
   * If you notice performance degradation in queries, gathering statistics can help the optimizer choose better execution plans
5. **Schema Changes**:
   * After significant schema changes, such as adding or modifying columns, gathering statistics helps the optimizer understand the new structure

**What Happens in the Backend?**

When you gather statistics for a table, Oracle collects various types of information about the table, its columns, and indexes. This information includes:

* **Table Statistics**: Number of rows, blocks, average row length.
* **Column Statistics**: Number of distinct values, histograms, null values.
* **Index Statistics**: B-tree levels, leaf blocks, clustering factor.
* **System Statistics**: CPU speed, I/O performance

**Example of Gathering Table Statistics**

Here's how you can gather statistics for a table:

EXEC DBMS\_STATS.GATHER\_TABLE\_STATS('HR', 'EMPLOYEES');

**Explanation**

* **DBMS*STATS.GATHER*TABLE\_STATS**: This procedure collects statistics for the specified table (EMPLOYEES) in the specified schema (HR).

Gathering statistics ensures that the Oracle optimizer has accurate and up-to-date information to make the best possible execution plans for queries, ultimately improving performance.

If you have any specific questions or need further details, feel free to ask!

Sure! Here are some code examples for gathering statistics in Oracle:

**Example 1: Gathering Table Statistics**

BEGIN

DBMS\_STATS.GATHER\_TABLE\_STATS(

ownname => 'HR',

tabname => 'EMPLOYEES',

estimate\_percent => DBMS\_STATS.AUTO\_SAMPLE\_SIZE,

method\_opt => 'FOR ALL COLUMNS SIZE AUTO'

);

END;

**Explanation**

* **ownname**: Specifies the schema name (HR).
* **tabname**: Specifies the table name (EMPLOYEES).
* **estimate\_percent**: Uses the automatic sample size for estimating statistics.
* **method\_opt**: Collects statistics for all columns with automatic histogram size.

**Example 2: Gathering Index Statistics**

BEGIN

DBMS\_STATS.GATHER\_INDEX\_STATS(

ownname => 'HR',

indname => 'EMPLOYEES\_IDX',

estimate\_percent => DBMS\_STATS.AUTO\_SAMPLE\_SIZE

);

END;

**Explanation**

* **ownname**: Specifies the schema name (HR).
* **indname**: Specifies the index name (EMPLOYEES\_IDX).
* **estimate\_percent**: Uses the automatic sample size for estimating statistics.

**Example 3: Gathering Schema Statistics**

BEGIN

DBMS\_STATS.GATHER\_SCHEMA\_STATS(

ownname => 'HR',

estimate\_percent => DBMS\_STATS.AUTO\_SAMPLE\_SIZE,

method\_opt => 'FOR ALL COLUMNS SIZE AUTO'

);

END;

**Explanation**

* **ownname**: Specifies the schema name (HR).
* **estimate\_percent**: Uses the automatic sample size for estimating statistics.
* **method\_opt**: Collects statistics for all columns with automatic histogram size.

**Example 4: Gathering Database Statistics**

BEGIN

DBMS\_STATS.GATHER\_DATABASE\_STATS(

estimate\_percent => DBMS\_STATS.AUTO\_SAMPLE\_SIZE,

method\_opt => 'FOR ALL COLUMNS SIZE AUTO'

);

END;

**Explanation**

* **estimate\_percent**: Uses the automatic sample size for estimating statistics.
* **method\_opt**: Collects statistics for all columns with automatic histogram size.

These examples should help you understand how to gather statistics in different contexts within Oracle. If you have any specific requirements or further questions, feel free to ask!

**Q: What is exchange partition / What is split partition**

**Exchange Partition**

The **EXCHANGE PARTITION** operation in Oracle allows you to swap the data between a partition of a partitioned table and a non-partitioned table. This operation is useful for moving data efficiently without physically copying rows, which can be beneficial for data maintenance and performance ss

**Example**

Let's say we have a partitioned table sales and a non-partitioned table sales\_archive. We want to move the data from the sales partition for the year 2024 to the sales\_archive table.

-- Create partitioned table

CREATE TABLE sales (

sale\_id NUMBER,

sale\_date DATE,

amount NUMBER

) PARTITION BY RANGE (sale\_date) (

PARTITION sales\_2023 VALUES LESS THAN (TO\_DATE('01-JAN-2024', 'DD-MON-YYYY')),

PARTITION sales\_2024 VALUES LESS THAN (TO\_DATE('01-JAN-2025', 'DD-MON-YYYY'))

);

-- Create non-partitioned table

CREATE TABLE sales\_archive (

sale\_id NUMBER,

sale\_date DATE,

amount NUMBER

);

-- Exchange partition

ALTER TABLE sales EXCHANGE PARTITION sales\_2024 WITH TABLE sales\_archive WITHOUT VALIDATION;

**Split Partition**

The **SPLIT PARTITION** operation in Oracle allows you to divide an existing partition into two or more partitions. This is useful when a partition becomes too large and needs to be split for better performance and manageability [[2]](https://oracle-base.com/articles/12c/online-split-partition-and-subpartition-12cr2).

**Example**

Let's say we have a partitioned table sales with a partition sales\_2024 that we want to split into two partitions: sales\_2024\_H1 for the first half of the year and sales\_2024\_H2 for the second half.

-- Split partition

ALTER TABLE sales SPLIT PARTITION sales\_2024 AT (TO\_DATE('01-JUL-2024', 'DD-MON-YYYY'))

INTO (

PARTITION sales\_2024\_H1 VALUES LESS THAN (TO\_DATE('01-JUL-2024', 'DD-MON-YYYY')),

PARTITION sales\_2024\_H2 VALUES LESS THAN (TO\_DATE('01-JAN-2025', 'DD-MON-YYYY'))

);

**Explanation**

* **EXCHANGE PARTITION**: Swaps data between a partition and a non-partitioned table without physically moving rows
* **SPLIT PARTITION**: Divides an existing partition into smaller partitions based on a specified boundary

These operations help in managing large datasets efficiently and maintaining optimal performance. If you have any specific scenarios or further questions, feel free to ask!

**Q: And if the values mentioned in the partition are not present in table column, then whether the partition will be created or not?**

Yes, the partition will still be created even if the values specified in the partition are not present in the table column. The partitioning scheme defines the boundaries for the partitions, but it doesn't require the actual data to match those boundaries at the time of creation.

**Example Scenario**

Let's say we have a table sales partitioned by range on the sale\_date column, and we create partitions for future dates that don't yet have corresponding data.

CREATE TABLE sales (

sale\_id NUMBER,

sale\_date DATE,

amount NUMBER

) PARTITION BY RANGE (sale\_date) (

PARTITION sales\_2023 VALUES LESS THAN (TO\_DATE('01-JAN-2024', 'DD-MON-YYYY')),

PARTITION sales\_2024 VALUES LESS THAN (TO\_DATE('01-JAN-2025', 'DD-MON-YYYY'))

);

**Explanation**

* **sales*2023*** *and* ***sales*2024** partitions are created with boundaries for dates in 2023 and 2024, respectively.
* Even if there are no rows with sale\_date values in 2024, the partition sales\_2024 will still be created.

**Practical Use**

Creating partitions for future dates or ranges where data might not yet exist is common in scenarios where you anticipate data growth or want to organize data efficiently for future operations.

**Q: EXPLAIN PLAN statement in Oracle is used**

EXPLAIN PLAN statement in Oracle is used to display the execution plan that the Oracle optimizer chooses for a SQL statement. This can help you understand how a query will be executed and identify potential performance issues.

**Example Scenario**

Let's say we have a table employees and we want to analyze the execution plan for a query that retrieves employee details based on their department.

**Table Structure**

CREATE TABLE employees (

employee\_id NUMBER PRIMARY KEY,

first\_name VARCHAR2(50),

last\_name VARCHAR2(50),

department\_id NUMBER,

salary NUMBER

);

CREATE INDEX emp\_dept\_idx ON employees(department\_id);

**Query to Analyze**

SELECT employee\_id, first\_name, last\_name, salary

FROM employees

WHERE department\_id = 10;

**Generating the Execution Plan**

1. **Create a Plan Table**: If not already created, you need a table to store the execution plan.

CREATE TABLE plan\_table AS

SELECT \* FROM TABLE(DBMS\_XPLAN.DISPLAY);

1. **Explain the Plan**:

EXPLAIN PLAN FOR

SELECT employee\_id, first\_name, last\_name, salary

FROM employees

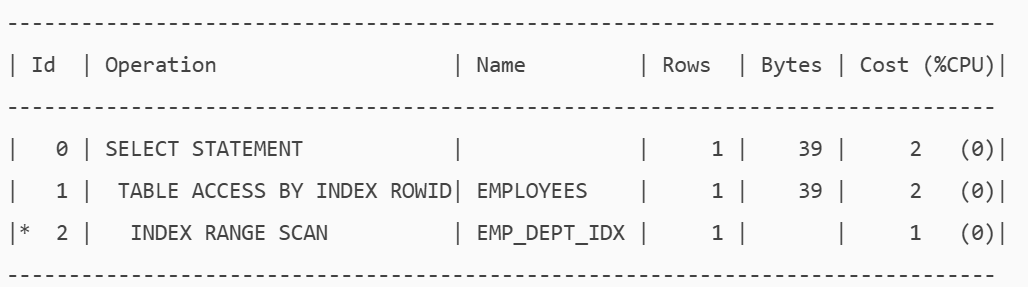
WHERE department\_id = 10;

1. **Display the Plan**:

SELECT \* FROM TABLE(DBMS\_XPLAN.DISPLAY);

**Example Output**

The output might look something like this:



**Explanation**

* **Id**: Step number in the execution plan.
* **Operation**: Type of operation performed (e.g., TABLE ACCESS, INDEX RANGE SCAN).
* **Name**: Name of the table or index involved.
* **Rows**: Estimated number of rows processed.
* **Bytes**: Estimated number of bytes processed.
* **Cost**: Estimated cost of the operation.

**Key Points for Interviews**

* **Understanding Execution Plans**: Be prepared to explain how to interpret the different operations and their impact on performance.
* **Indexes**: Discuss how indexes can improve query performance and how the optimizer uses them.
* **Cost-Based Optimization**: Explain how Oracle uses statistics to choose the most efficient execution plan.

**Triggers**

In Oracle, triggers are special types of stored procedures that automatically execute or "fire" when certain events occur. There are several types of triggers based on the timing and the event that causes them to fire.

**Types of Triggers**

1. **Before Triggers**: Execute before the triggering event.
2. **After Triggers**: Execute after the triggering event.
3. **Row-Level Triggers**: Execute once for each row affected by the triggering event.
4. **Statement-Level Triggers**: Execute once for the entire triggering event.
5. **Instead of Triggers**: Used on views to perform DML operations.

**Examples**

**1. Before Insert Trigger**

Executes before an insert operation on the employees table.

CREATE OR REPLACE TRIGGER before\_insert\_employee

BEFORE INSERT ON employees

FOR EACH ROW

BEGIN

:NEW.employee\_id := employees\_seq.NEXTVAL;

END;

**Explanation**: This trigger automatically assigns a new employee ID from a sequence before inserting a new row into the employees table.

**2. After Update Trigger**

Executes after an update operation on the employees table.

CREATE OR REPLACE TRIGGER after\_update\_employee

AFTER UPDATE ON employees

FOR EACH ROW

BEGIN

INSERT INTO employees\_audit (employee\_id, old\_salary, new\_salary, change\_date)

VALUES (:OLD.employee\_id, :OLD.salary, :NEW.salary, SYSDATE);

END;

**Explanation**: This trigger logs changes to the salary column in an employees\_audit table after an update operation.

**3. Row-Level Trigger**

Executes for each row affected by an insert operation on the employees table.

CREATE OR REPLACE TRIGGER row\_level\_insert\_employee

BEFORE INSERT ON employees

FOR EACH ROW

BEGIN

IF :NEW.salary < 0 THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Salary cannot be negative');

END IF;

END;

**Explanation**: This trigger checks if the salary is negative before inserting a new row and raises an error if it is.

**4. Statement-Level Trigger**

Executes once for the entire insert operation on the employees table.

CREATE OR REPLACE TRIGGER statement\_level\_insert\_employee

AFTER INSERT ON employees

BEGIN

DBMS\_OUTPUT.PUT\_LINE('Insert operation completed on employees table');

END;

**Explanation**: This trigger outputs a message after the insert operation is completed on the employees table.

**5. Instead of Trigger**

Used on a view to perform an insert operation.

CREATE OR REPLACE TRIGGER instead\_of\_insert\_employee\_view

INSTEAD OF INSERT ON employees\_view

FOR EACH ROW

BEGIN

INSERT INTO employees (employee\_id, first\_name, last\_name, department\_id, salary)

VALUES (:NEW.employee\_id, :NEW.first\_name, :NEW.last\_name, :NEW.department\_id, :NEW.salary);

END;

**Explanation**: This trigger allows inserting data into a view (employees\_view) by performing the insert operation on the underlying employees table.

These examples cover the basic types of triggers in Oracle. If you have any specific requirements or further questions, feel free to ask!

Example on INSERT, UPDATE and Delete

CREATE OR REPLACE TRIGGER log\_employee\_changes

AFTER INSERT OR UPDATE OR DELETE ON employees

FOR EACH ROW

BEGIN

    IF INSERTING THEN

        INSERT INTO employees\_log (

            log\_id, employee\_id, operation, new\_first\_name, new\_last\_name, new\_salary, change\_date

        ) VALUES (

            employees\_log\_seq.NEXTVAL, :NEW.employee\_id, 'INSERT',

:NEW.first\_name, :NEW.last\_name, :NEW.salary, SYSDATE

        );

    ELSIF UPDATING THEN

        INSERT INTO employees\_log (

            log\_id, employee\_id, operation, old\_first\_name, new\_first\_name,

old\_last\_name, new\_last\_name, old\_salary, new\_salary, change\_date

        ) VALUES (

            employees\_log\_seq.NEXTVAL, :OLD.employee\_id, 'UPDATE', :OLD.first\_name,

:NEW.first\_name, :OLD.last\_name, :NEW.last\_name, :OLD.salary, :NEW.salary, SYSDATE

        );

    ELSIF DELETING THEN

        INSERT INTO employees\_log (

            log\_id, employee\_id, operation, old\_first\_name, old\_last\_name, old\_salary, change\_date

        ) VALUES (

            employees\_log\_seq.NEXTVAL, :OLD.employee\_id, 'DELETE',

:OLD.first\_name, :OLD.last\_name, :OLD.salary, SYSDATE

        );

    END IF;

END;

**Cursor**

**sImplicit Cursor**

**Oracle automatically creates implicit cursors for SELECT statements and DML operations (INSERT, UPDATE, DELETE). You don't need to declare or open implicit cursors; they are managed by Oracle.**

**Example of Implicit Cursor**

**BEGIN**

**UPDATE employees**

**SET salary = salary \* 1.10**

**WHERE department\_id = 10;**

**IF SQL%ROWCOUNT > 0 THEN**

**DBMS\_OUTPUT.PUT\_LINE(SQL%ROWCOUNT || ' rows updated.');**

**ELSE**

**DBMS\_OUTPUT.PUT\_LINE('No rows updated.');**

**END IF;**

**END;**

**Explanation**

* **SQL%ROWCOUNT: An implicit cursor attribute that returns the number of rows affected by the last DML operation.**

**Explicit Cursor**

**Explicit cursors are defined by the programmer and provide more control over the context area. You need to declare, open, fetch, and close explicit cursors.**

**Example of Explicit Cursor**

DECLARE

CURSOR emp\_cursor IS

SELECT employee\_id, first\_name, last\_name, salary

FROM employees

WHERE department\_id = 10;

v\_employee\_id employees.employee\_id%TYPE;

v\_first\_name employees.first\_name%TYPE;

v\_last\_name employees.last\_name%TYPE;

v\_salary employees.salary%TYPE;

BEGIN

OPEN emp\_cursor;

LOOP

FETCH emp\_cursor INTO v\_employee\_id, v\_first\_name, v\_last\_name, v\_salary;

EXIT WHEN emp\_cursor%NOTFOUND;

DBMS\_OUTPUT.PUT\_LINE('Employee ID: ' || v\_employee\_id || ', Name: ' || v\_first\_name || ' ' || v\_last\_name || ', Salary: ' || v\_salary);

END LOOP;

CLOSE emp\_cursor;

END;

**Explanation**

* **DECLARE CURSOR emp\_cursor IS: Declares an explicit cursor named emp\_cursor.**
* **OPEN emp\_cursor: Opens the cursor.**
* **FETCH emp\_cursor INTO: Fetches rows from the cursor into variables.**
* **EXIT WHEN emp\_cursor%NOTFOUND: Exits the loop when there are no more rows to fetch.**
* **CLOSE emp\_cursor: Closes the cursor.**

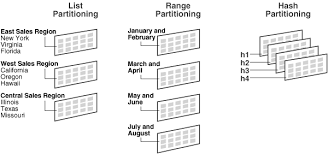
**Key Differences**

* **Implicit Cursor: Automatically managed by Oracle, used for single-row operations.**
* **Explicit Cursor: Defined and managed by the programmer, used for multi-row operations.**

**These examples should help you understand the use of implicit and explicit cursors in Oracle. If you have any specific scenarios or further questions, feel free to ask!**

**Krishan Interview (25-04-2025)**

1.What is difference b/w range partition / list partition and hash partition



Oracle Database offers several partitioning strategies to manage large tables and indexes efficiently. Here are the key differences between range partitioning, list partitioning, and hash partitioning:

**Range Partitioning**

* **Definition**: Data is divided into partitions based on a range of values for a specified column.
* **Use Case**: Ideal for date or numeric columns where data can be logically grouped into ranges (e.g., monthly sales data).
* **Example**: A table partitioned by a date column, with each partition representing a month.

CREATE TABLE sales (

order\_id NUMBER,

order\_date DATE,

amount NUMBER

)

PARTITION BY RANGE (order\_date) (

PARTITION p1 VALUES LESS THAN (TO\_DATE('2023-01-01', 'YYYY-MM-DD')),

PARTITION p2 VALUES LESS THAN (TO\_DATE('2023-02-01', 'YYYY-MM-DD')),

PARTITION p3 VALUES LESS THAN (TO\_DATE('2023-03-01', 'YYYY-MM-DD'))

);

**List Partitioning**

* **Definition**: Data is divided into partitions based on a list of discrete values for a specified column.
* **Use Case**: Suitable for categorical data where specific values can be grouped together (e.g., regions or product categories).
* **Example**: A table partitioned by a region column, with each partition representing a different region.

CREATE TABLE customers (

customer\_id NUMBER,

region VARCHAR2(50),

name VARCHAR2(100)

)

PARTITION BY LIST (region) (

PARTITION p1 VALUES ('North'),

PARTITION p2 VALUES ('South'),

PARTITION p3 VALUES ('East'),

PARTITION p4 VALUES ('West')

);

**Hash Partitioning**

* **Definition**: Data is divided into partitions based on a hash function applied to a specified column.
* **Use Case**: Useful when data does not fit neatly into ranges or lists, providing an even distribution of data across partitions.
* **Example**: A table partitioned by a customer ID column, with data distributed evenly across partitions.

CREATE TABLE transactions (

transaction\_id NUMBER,

customer\_id NUMBER,

amount NUMBER

)

PARTITION BY HASH (customer\_id) (

PARTITION p1,

PARTITION p2,

PARTITION p3,

PARTITION p4

);

**Summary**

* **Range Partitioning**: Best for sequential data with clear boundaries (e.g., dates).
* **List Partitioning**: Best for categorical data with specific values (e.g., regions).
* **Hash Partitioning**: Best for evenly distributing data without natural groupings (e.g., customer IDs).

Each partitioning strategy has its own advantages and is chosen based on the specific requirements of the data and the queries being executed [[1]](https://ittutorial.org/partitioning-types-range-list-hash-interval-in-oracle-database/) [[2]](https://docs.oracle.com/en/database/oracle/oracle-database/19/vldbg/recommendations-partition-strategy.html) [[3]](https://docs.oracle.com/cd/B10500_01/server.920/a96524/c12parti.htm).

2.what is spit partition and can we use in other partition like range/list/hash

3.What is Hints

4.What is Global temporary table and can we do gather stats on this.

5.Describe about Explain Plan

6.What is parallelism in explain plan

7.what is cardinality in explain plan

Write a query without use subquery, joins

|  |  |  |
| --- | --- | --- |
| Employee Name | Employee ID | Manager ID |
| Sudha | 2 |  |
| Giri | 3 | 2 |
| Siva | 4 | 2 |
| Ram | 5 | 3 |
| Hari | 6 | 4 |

|  |  |  |  |
| --- | --- | --- | --- |
| Employee Name | Employee ID | Manager ID | Manager Name |
| Sudha | 2 |  |  |
| Giri | 3 | 2 | Sudha |
| Siva | 4 | 2 | Sudha |
| Ram | 5 | 3 | Giri |
| Hari | 6 | 4 | Siva |

 ss

Q: Convert String to row:

INDIA

 ---------

I

N

D

I

A

Answer: SELECT REGEXP\_REPLACE('india', '(.)', '\1\n') AS converted\_text FROM dual;

Q:

|  |  |  |
| --- | --- | --- |
| Team A | Team B | Win |
| India | Pakistan | India |
| India | Srilanka | India |
| Srilanka | Pakistan | Pakistan |
| Srilanka | India | Srilanka |
| Australia | England | Tie |
| India | Australia | Tie |
| India | England | India |

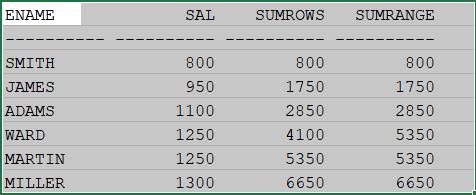
|  |  |
| --- | --- |
| Team | Points |
| India | 7 |
| Srilanka | 2 |
| Australia | 2 |
| Pakistan | 2 |
| England | 1 |

Win = 2\*3 = 6

6+1 = 7

|  |  |
| --- | --- |
| Ename | Salary |
| SMITH | 800 |
| JAMES | 950 |
| ADAMS | 1100 |
| WARD | 1250 |
| MARTIN | 1250 |
| MILLER | 1300 |

|  |
| --- |
| ENAME             SAL    SUMROWS   SUMRANGE |
| ---------- ---------- ---------- ---------- |
| SMITH             800        800        800 |
| JAMES             950       1750       1750 |
| ADAMS            1100       2850       2850 |
| WARD             1250       4100       5350 |
| MARTIN           1250       5350       5350 |
| MILLER           1300       6650       6650 |



Write a query without use subquery, joins

|  |  |  |
| --- | --- | --- |
| Employee Name | Employee ID | Manager ID |
| Sudha | 2 |  |
| Giri | 3 | 2 |
| Siva | 4 | 2 |
| Ram | 5 | 3 |
| Hari | 6 | 4 |

|  |  |  |  |
| --- | --- | --- | --- |
| Employee Name | Employee ID | Manager ID | Manager Name |
| Sudha | 2 |  |  |
| Giri | 3 | 2 | Sudha |
| Siva | 4 | 2 | Sudha |
| Ram | 5 | 3 | Giri |
| Hari | 6 | 4 | Siva |

SELECT

employee.Employee\_Name, employee.Employee\_Id,employee.Manager\_Id,

manager.Employee\_Name AS Manager\_Name

FROM Employees employee

JOIN Employees manager

ON employee.Manager\_Id = manager.Manager\_Id;

**Jadhav, Bhausaheb**

1.how to calculate cost in explain plan  
2.partition and types   
3.split partition  
4.Interval partition in brief   
5.exchange partition  
6.global and local index partition  
7.GTT and how it works  
8.when we can use split partition and can we use in hash partition   
9.what is partition pruning? how it works  
10.performance tunning techniques  
11.can we create partitions on a table which has already billions of data if yes then what's the approach ?  
12.Normal table and global temporary table difference  
13.where to check parallelism of the running query in Explain Plan  
14.When we use parallel statstics  
15.  
INDIA  
  
I  
N  
D  
I  
A  
  
SELECT REGEXP\_REPLACE('india', '(.)', '\1\n') AS converted\_text FROM dual;  
  
16.need to separate charactor and numbers

Table  
d  
y  
u  
3  
6  
7  
u  
i  
9  
IN  
NO  
24

:  
• What is Hints and which hints you have used?  
• Global Temporary Tables: Can we create a primary key on GT? Can we gather stats on GT and in which scenario?  
• What is Dynamic Sampling and why is it used?  
• If the EMP\_Id column is already duplicated, can we create primary/unique constraints on it?  
• How to drop a column from a table with 500 million rows without causing performance issues?  
• Mutating Trigger Error and how to avoid it?  
• Can we subdivide a Partition? If yes, how?  
• Can we create partitions on a table with billions of rows? If yes, what is the approach?  
• Which Oracle package is used to convert a non-partitioned table into a partitioned table?  
• If data is mistakenly deleted from a table and there is a backup table with correct data, how will you insert the missed data back without performance issues?  
• Performance Tuning strategy  
• Access Paths, Joining Methods  
• What can you check in Explain Plan? Describe.  
• How to check statistics stale percentage of data?  
• Where to check parallelism of the running query in Explain Plan?  
• RAG (Retrieval Augmented Generation)  
• How can we create automatic range partitions?  
• How to calculate cost in Explain Plan?  
• Partition and types  
• Split partition  
• Interval partition in brief  
• Exchange partition  
• Global and local index partition  
• GTT and how it works  
• When can we use split partition and can we use it in hash partition?  
• What is partition pruning? How does it work?  
• Normal table and global temporary table difference  
• When do we use parallel statistics?  
• What is the difference between external table and global temporary table?  
• Can we do indexing and gather statistics on GTT? In which scenario will we do that?  
• How does gathering statistics help improve performance?  
• There is a scenario where data coming at night 11 PM is wrong, and you need to delete it daily. How will you do that?  
• What is parallelism and how does it work?  
• When do we use use\_NL and use\_hash hints? What is the difference?  
• If there are 10 records in Table A and 5 records in Table B, how many rows will be displayed in the result of "Select \* From A, B"?  
• Select Id from A order by 2 desc; What will be the output of this query?  
• If a table has a column with numeric values and we create a range partition with values like 40 to 60, will the partition be created if the values are not present in the column?  
• Can we drop a table in a package?  
• If a package has insert, update, etc. code inside it, how can we capture how many rows are inserted or updated through that package?  
• If we have "tablea" in schema1 and want to access it from schema2 with a different name, how can we do that? (Answer: Create a synonym for "tablea" in schema1 and use it in schema2)  
• While using parallel hints, what is the max number we can mention? (e.g., parallel(4))  
• When using parallel(4), what happens in the backend SID?

⸻  
  
Fundamentals  
1. What are the key differences between SQL and PL/SQL?  
2. How do you identify performance bottlenecks in a PL/SQL procedure?  
3. What are bind variables, and why are they important for performance?  
4. Explain the difference between implicit and explicit cursors. Which is more performance-friendly?  
5. How do you avoid row-by-row processing in PL/SQL (the “row-by-row = slow-by-slow” issue)?  
  
⸻  
  
SQL Query Tuning  
6. How do you analyze an execution plan using EXPLAIN PLAN or DBMS\_XPLAN?  
7. What are some common reasons for a full table scan, and how do you fix it?  
8. When would you choose a hash join over a nested loop join?  
9. How do you tune a query that has multiple joins and subqueries?  
10. How do indexes affect SQL performance? When should you not use indexes?  
  
⸻  
  
PL/SQL Code Optimization  
11. What are the best practices for writing efficient PL/SQL code?  
12. What tools or techniques do you use to monitor and profile PL/SQL code performance?  
13. What is the use of BULK COLLECT and FORALL? When should they be used?  
14. What is context switching in PL/SQL and how do you reduce it?  
15. How do you handle large data loads efficiently in PL/SQL?  
  
⸻  
  
Oracle Internals & Troubleshooting  
16. How does Oracle’s Cost-Based Optimizer (CBO) work?  
17. What are statistics in Oracle, and how do they impact performance?  
18. How do you deal with SQLs stuck in the library cache or hard parsing?  
19. What is AWR, ASH, and how do you use them in performance troubleshooting?  
20. How do you detect and resolve lock contention issues?  
  
⸻  
  
Real-World Scenarios  
21. A query was fast yesterday but is slow today. How do you troubleshoot?  
22. You’ve been given a poorly performing stored procedure. What is your approach to optimize it?  
23. What steps would you take to tune a batch job that processes millions of rows overnight?  
24. Have you worked with partitioned tables? How does partitioning help performance?  
25. Describe a situation where you significantly improved the performance of a query or application.  
  
⸻

⸻  
  
Core PL/SQL Performance Optimization  
1. What are the common performance pitfalls in PL/SQL code, and how do you avoid them?  
2. How do you identify and eliminate context switching between SQL and PL/SQL?  
3. What is the difference between BULK COLLECT and FORALL? When should they be used?  
4. How do you handle large volume data processing efficiently in PL/SQL?  
5. What are the implications of using autonomous transactions?  
  
⸻  
  
Advanced SQL Query Tuning  
6. How do you read and interpret execution plans using EXPLAIN PLAN or DBMS\_XPLAN.DISPLAY?  
7. What causes a query to use a full table scan even when indexes exist?  
8. Explain the difference between nested loop, hash join, and merge join. How do you force a specific join method?  
9. How do you use hints in Oracle SQL? When is it appropriate to use them?  
10. How do you approach rewriting SQL for better performance?  
  
⸻  
  
Optimizer and Execution  
11. How does the Cost-Based Optimizer (CBO) work in Oracle?  
12. What are optimizer statistics and how do stale/missing stats affect performance?  
13. What is a cardinality misestimate and how do you troubleshoot it?  
14. Explain SQL Plan Baselines and SQL Profiles. How do they differ?  
15. How does adaptive query optimization (adaptive joins, adaptive plans) work in Oracle 12c and beyond?  
  
⸻  
  
Indexes and Partitioning  
16. What’s the difference between B-tree and Bitmap indexes? Use cases?  
17. What is Index Skip Scan? When does Oracle use it?  
18. How do function-based indexes affect query performance?  
19. How do you determine if a table should be partitioned? What are the performance benefits?  
20. How do you monitor and maintain index health?  
  
⸻  
  
PL/SQL Instrumentation and Tools  
21. What tools do you use to profile PL/SQL code? (e.g., DBMS\_PROFILER, DBMS\_HPROF)  
22. What is AWR? How do you read and interpret AWR reports?  
23. How do you use ASH and V$ views for performance diagnosis?  
24. What’s the difference between latch and lock waits? How do you troubleshoot them?  
25. What is the difference between wait events like db file sequential read and db file scattered read?  
  
⸻  
  
Real-World Tuning Scenarios  
26. Describe a production issue where you optimized a stored procedure or SQL and what steps you took.  
27. A query was fast in the morning and slow by afternoon. How would you troubleshoot it?  
28. How do you tune a batch job that processes millions of records?  
29. How do you handle cursor leaks or high open cursors count in production?  
30. How do you manage and tune long-running ETL or data migration scripts?  
  
⸻  
  
Memory and Resource Management  
31. What is PGA\_AGGREGATE\_TARGET and how does it impact performance?  
32. How do you monitor and tune memory usage in Oracle (SGA, PGA)?  
33. What are TEMP tablespaces used for and how do you manage temp space issues?  
34. How do you handle undo/redo usage during large DML operations?  
35. What is result cache in Oracle and when would you use it?  
  
⸻

⸻  
  
Basic Partitioning Concepts  
1. What is table partitioning in Oracle, and why is it used?  
2. What are the benefits of partitioning a table?  
3. What are the different types of partitioning available in Oracle?  
4. What is the difference between range and list partitioning?  
5. How does partitioning improve query performance?  
  
⸻  
  
Partitioning Types & Use Cases  
6. When would you use hash partitioning instead of range?  
7. What is composite partitioning? Give an example.  
8. What is interval partitioning and how does it differ from range partitioning?  
9. What are reference partitions, and when would you use them?  
10. Can you partition an index? What types of index partitioning are available?  
  
Partitioning Syntax & Management  
11. How do you create a range-partitioned table in Oracle?  
12. How do you add a new partition to an existing partitioned table?  
13. What is the purpose of the MAXVALUE keyword in range partitioning?  
14. Can you update the partition key column? Why or why not?  
15. How do you exchange a partition with a non-partitioned table?  
  
Performance & Query Optimization  
16. What is partition pruning, and how does it work?  
17. How do you verify if partition pruning is happening?  
18. What is partition-wise join and how does it help performance?  
19. How do statistics affect performance in partitioned tables?  
20. What are global vs local indexes on partitioned tables? What are the pros and cons of each?  
  
Advanced & Real-World Scenarios  
21. Have you faced performance issues with partitioned tables? How did you resolve them?  
22. How would you handle archiving or purging data in a partitioned table?  
23. Can you repartition a table that already contains data? What are the implications?  
24. How do you handle data loading into partitioned tables efficiently?  
25. Explain how partitioning fits into large data warehousing and ETL processes.