

#	Question	Answer / Key Points
1	What is PL/SQL? How is it different from SQL?	PL/SQL = Procedural Language/Structured Query Language. It's Oracle's extension of SQL with procedural features: variables, loops, conditionals, exception handling, modularization (procedures, functions, packages). SQL is declarative, used for querying and manipulating data; PL/SQL allows you to embed SQL in procedural logic.
2	What is the basic structure of a PL/SQL block?	A block has typically three parts (though exception and declare are optional):

- DECLARE (optional): declare variables, constants, cursors, types
- BEGIN: executable code
- EXCEPTION (optional): error-handling
- END;

Nested blocks are allowed.

DataCamp

+2

Intellipaat

+2

3 What are PL/SQL data types? Several kinds:

- Scalar: NUMBER, VARCHAR2, CHAR, DATE, BOOLEAN, etc.
- Composite: RECORD, TABLE, VARRAY.
- Reference types: REF CURSOR, etc.
- LOBs (CLOB, BLOB) if needed.

Also %TYPE and %ROWTYPE to refer to database column types.

Simplilearn.com

+2

mindmajix

+2

4 What are cursors? Difference between implicit & explicit cursor? Implicit cursor is used automatically by Oracle for single SQL statements (SELECT INTO, INSERT, UPDATE, DELETE etc.).

- Explicit cursor: you declare, open, fetch, close; used when you want to loop through multiple rows.

Cursor attributes: %FOUND, %NOTFOUND, %ROWCOUNT, %ISOPEN.

Simplilearn.com

+1

5 How do you handle exceptions in PL/SQL? What are predefined exceptions? In the EXCEPTION section of a block. Predefined ones include NO\_DATA\_FOUND, TOO\_MANY\_ROWS, ZERO\_DIVIDE, VALUE\_ERROR, etc. You can also define your own exceptions (using EXCEPTION clause and RAISE). Can use WHEN OTHERS to catch all. Also you can get error details via SQLCODE and SQLERRM.

Simplilearn.com

+2

Intellipaat

+2

6 What are triggers? Types of triggers? Trigger = PL/SQL block that automatically executes in response to certain DML events (INSERT, UPDATE, DELETE), or DDL, or database events. Types:

- Row level vs Statement level

- \* BEFORE \* vs \* AFTER \*

- Triggers can be on table, on view (INSTEAD OF), and database triggers (e.g., logon). Use cases: auditing, enforcing business rules, maintaining derived values. Beware of mutating table errors in row-level triggers.

mindmajix

+2

GeeksforGeeks

+2

7 What is a package in PL/SQL? What are benefits? A package groups related procedures, functions, variables, cursors, exceptions etc. Has two parts: Specification (public interface) and Body (implementation). Benefits: modularization, code reuse, encapsulation, hiding implementation, better performance (since package loads into memory once).

DataCamp

+2

Simplilearn.com

+2

8 Procedure vs Function – differences and when to use what. Function returns a value and can be used in SQL statements; Procedure does not return a value (but can via OUT parameters).

Functions should avoid DML (or commit) when used in SQL to prevent side effects. Procedure used when performing operations without needing return, or when many actions are needed. Use function when you need result, procedure otherwise. Also handling parameters: IN, OUT, IN OUT.

Simplilearn.com

+1

9 What are the parameter modes in PL/SQL (IN, OUT, IN OUT)? IN: parameter for input only (cannot change inside procedure and send back)

OUT: used for output; must be assigned before exiting; cannot read initial value

IN OUT: both read input and pass back possibly modified value.

Intellipaat

+1

10 What are %ROWTYPE and %TYPE? %TYPE is used to declare a variable with same data type as a column (so that if column's type changes, your variable auto-updates).

%ROWTYPE is used to declare a record type corresponding to an entire row of a table or cursor (all columns). Useful to fetch whole row without listing each column.

Simplilearn.com

+1

11 What is dynamic SQL in PL/SQL? How do you execute it? Dynamic SQL = SQL statements that are built at runtime (as string). Useful when table names, columns etc are dynamic. You can use EXECUTE IMMEDIATE for simple statements; for more complex or multiple rows, use DBMS\_SQL package. Need to handle binding variables to avoid SQL injection and for performance.

hirist.tech

+1

12 What are BULK COLLECT and FORALL? When do you use them? These are features to improve performance when dealing with large number of rows in PL/SQL:

- BULK COLLECT fetches many rows into collections in one go, reducing context switches.
- FORALL is used for bulk DML (INSERT / UPDATE / DELETE) using collections. Reduces overhead. Use them when processing many rows. Also need to handle memory usage, maybe use limits.

13 What's a mutating table error? How can you solve or avoid it? A mutating table error occurs when in a row-level trigger you try to read or query the same table that is being modified. Because the table is in mid-change, Oracle disallows certain operations. To avoid: use statement-level triggers, use compound triggers (Oracle 11g+), use packages to hold data and process after triggering, or perform logic in AFTER statement trigger.

14 How do you debug or trace PL/SQL code? Use DBMS\_OUTPUT.PUT\_LINE for simple messages; use logging (tables) to capture runtime info; use Oracle's profiling tools; use EXCEPTION blocks to capture error messages (SQLERRM, SQLCODE); sometimes trace via SQL\*Trace, or Oracle tools like TKPROF; use DBMS\_TRACE / DBMS\_DEBUG etc.

15 What are the cursor attributes? %FOUND – true if last fetch found a row,

%NOTFOUND,

%ROWCOUNT – the number of rows fetched so far,

%ISOPEN – whether the cursor is open. For implicit cursors also partial of these (some only).

16 Explain transaction control statements in PL/SQL. The DML statements are grouped within transactions. Use COMMIT to make changes permanent, ROLLBACK to undo, SAVEPOINT to set intermediate points for partial rollback. Also SET TRANSACTION to define isolation etc. You need to ensure proper commit/rollback especially on exceptions.

17 What are views vs materialized views? View: virtual table defined by query; data not stored physically; every query on view runs underlying query.

Materialized view: stores actual data on disk; can be refreshed periodically. Useful for performance / reporting.

18 What is a pipelined function? Pipelined table functions can return rows as they are produced, rather than waiting for the full result set. They are used for performance when transforming or streaming large datasets. They allow interleaving of producer/consumer style.

19 Scenario: Write a procedure that takes department\_id and returns employees of that department (multiple rows) using cursor. Model code snippet:

```
CREATE OR REPLACE PROCEDURE get_emps_by_dept(p_dept_id IN NUMBER,
```

```
          p_emp_cursor OUT SYS_REFCURSOR) AS
```

```
BEGIN
```

```
    OPEN p_emp_cursor FOR SELECT * FROM employees WHERE department_id = p_dept_id;
```

```
END get_emps_by_dept;
```

- Using REF CURSOR so caller can fetch multiple rows. Discuss possible enhancements (e.g. only select required columns, performance etc). |

|20| What are the advantages and disadvantages of PL/SQL? | Advantages: Tight integration with Oracle DB; procedural features; can encapsulate business logic in DB; better performance because network round-trips reduced; security; reuse via packages. <br> Disadvantages: Vendor lock-in; can be misused (poorly maintained code, procedural spaghetti); debugging can be harder; performance issues if not optimized; memory usage; sometimes business logic should be in application rather than DB. |

|21| How is performance tuning done in PL/SQL? | Some strategies: <br>- Use BULK COLLECT / FORALL to reduce SQL-PL/PL-SQL context switches <br>- Avoid using SELECT \*; select only needed columns <br>- Proper indexing on tables; use explain plan <br>- Minimizing use of triggers where possible; avoid unnecessary nested loops <br>- Use bind variables; avoid dynamic SQL where static is possible <br>- Use caching or packages for frequently used data; optimize for commit frequency <br>- Monitor resource usage: memory, CPU, I/O; use Oracle tools e.g. SQL Trace / TKPROF. |

|22| What is an autonomous transaction? Where would you use it? | An autonomous transaction is a separate transaction that runs independently of the main one. It can commit or rollback independently;

used via the pragma AUTONOMOUS\_TRANSACTION. Use cases: logging or auditing where you want log writes to persist even if main transaction rolls back. Be careful with dependencies. |

|23| What is overloading (of procedures/functions)? | Defining multiple procedures or functions with the same name but different parameter signature (difference in number, type or order of parameters). Useful for similar operations with different inputs. |

|24| What are mutating and constraining tables (or the mutating table problem)? | Mutating table refers to the table being modified by DML in trigger; Oracle restricts reading from a mutating table in a row-level trigger; constraining table often refers to those involved in constraints or referential integrity. Solving mutating table problems via compound triggers, statement-level triggers, or delaying logic. |

|25| Tell me about a bug you faced / scenario/project you have worked where PL/SQL was used. What did you do? | (This is behavioural but technical) Talk about a project where you used PL/SQL: what you built (procedures/functions, triggers), what problem occurred (e.g. performance, wrong result, exceptions), how you debugged, optimized and the result. E.g. using BULK COLLECT reduced processing time or fixed exceptions with appropriate exception handling. |

If you like, I can also prepare Infosys-specific past PL/SQL interview questions (with what people have reported) so you can see exactly what they ask. Do you want that?

You said:

sure go on

ChatGPT said:

Here are Infosys-specific / reported PL/SQL / Oracle interview questions + scenarios (from past candidates) and good answer sketches. These are actual kinds of things Infosys interviewers have asked people with ~2-3 yrs experience. Use them to guide your preparation.

### Questions Infosys Reportedly Asks

From Glassdoor, LinkedIn etc:

NodeFlair

+3

Glassdoor

+3

Glassdoor

+3

Here are some concrete ones:

What are triggers and types of triggers?

Glassdoor

+1

What is a cursor and what are REF CURSORs?

Glassdoor

What is a mutating trigger and how to resolve / avoid mutating table error?

Glassdoor

+1

Difference between DELETE and TRUNCATE.

Glassdoor

Data types, analytic functions, cursors/procedures/functions/collections/indexing/tuning.

Glassdoor

+1

Joins, views, triggers.

Glassdoor

+1

Queries to remove duplicates; queries based on ranking (top 2/3 salary) etc.

Glassdoor

Partitions, indexing.

Glassdoor

+1

Use of scheduler (for scheduling jobs) & dumping data using scheduler.

Glassdoor

Collections (PL/SQL collections), string/date functions.

Glassdoor

#### Sample Scenario / Coding Style Questions

From candidate reports, Infosys also gives scenario or small coding tasks. Some examples:

Write a trigger which audits changes (insert/update/delete) in a table (capture old/new values).

Write a procedure / function that returns multiple employees given some criteria (department, salary range, etc.).

Given that a table is being updated via a trigger and the table is mutating, how will you restructure to avoid errors.

Write a PL/SQL block using collections (nested table or VARRAY) to process a list of IDs and perform bulk updates.

Query that uses analytic functions (e.g. ROW\_NUMBER(), RANK()) to find the 2nd highest, or top N, per group.

#### Good Model Answers / Answer Sketches

Here are good ways to answer or code for the above kinds of questions.

##### Question      Good Answer / Sketch

What is a Trigger? Types?      Trigger is a stored PL/SQL block that fires when certain DML (INSERT/UPDATE/DELETE) or DDL or database events occur. Types: Row-level vs Statement-level; BEFORE vs AFTER triggers; INSTEAD OF triggers (on views); database event triggers (e.g. on logon). Use cases: enforcing business rules, auditing, derived columns, etc.

Mutating table error + resolution      Mutating table error arises when in a row-level trigger on a table you try to query (or refer to) the same table while it's being modified (because Oracle disallows that to maintain consistency). Resolutions: use statement-level triggers instead of row level; use compound triggers (available in newer Oracle versions) which let you accumulate row level data then process after statement; use package variables to store needed values; or redesign logic so that trigger does not need to query the same table.

Difference between DELETE & TRUNCATE      DELETE: deleting rows, can have WHERE clause, is DML, logs each row, can be rolled back. TRUNCATE: removes all rows, no WHERE clause, much faster, is DDL, cannot rollback (unless in some special Oracle setups), reclaims space, resets high water mark, etc.

Using collections and bulk operations      Suppose you have a list of employee IDs and you want to update their salary. Better to use a PL/SQL collection (e.g. nested table or varray) and then FORALL for bulk DML. Also if retrieving many rows, use BULK COLLECT to load into collections. This reduces context switches between SQL and PL/SQL, improving performance. Also care about memory: limit size, maybe process in batches. Example:

```
DECLARE
```

```
TYPE empid_tab IS TABLE OF employees.employee_id%TYPE;
l_empids empid_tab;
BEGIN
  SELECT employee_id BULK COLLECT INTO l_empids FROM employees WHERE department_id = 10;
  FORALL i IN l_empids.FIRST .. l_empids.LAST
    UPDATE employees SET salary = salary * 1.1 WHERE employee_id = l_empids(i);
  COMMIT;
END;
```|
```

| \*\*Write a procedure to fetch employees by parameters\*\* |

```
```plsql
CREATE OR REPLACE PROCEDURE get_emps_by_dept_salary(
  p_dept_id IN NUMBER,
  p_min_salary IN NUMBER,
  p_emp_cursor OUT SYS_REFCURSOR
) AS
BEGIN
  OPEN p_emp_cursor FOR
    SELECT employee_id, first_name, last_name, salary
    FROM employees
    WHERE department_id = p_dept_id
      AND salary >= p_min_salary;
END;
```

Then the caller can fetch rows from the REF CURSOR. Also mention checking for no rows (NO\_DATA\_FOUND), maybe handle exceptions. |

|Query: Find 2nd highest salary per department|

You could write:

```
SELECT department_id, MAX(salary) AS second_highest_salary
FROM employees e
WHERE salary < (
    SELECT MAX(salary) FROM employees e2 WHERE e2.department_id = e.department_id
)
GROUP BY department_id;
```

Or using analytic/window functions:

```
SELECT department_id, salary
FROM (
    SELECT department_id, salary,
        ROW_NUMBER() OVER (PARTITION BY department_id ORDER BY salary DESC) AS rn
    FROM employees
) WHERE rn = 2;
```

1) Trigger to audit INSERT / UPDATE / DELETE (capture old & new values)

Goal: maintain an audit table that records who changed a row, when, what operation, and old/new values.

## Setup (example DDL)

```
-- sample table (assumed to exist)
```

```
CREATE TABLE employees (
    employee_id NUMBER PRIMARY KEY,
    first_name VARCHAR2(100),
    last_name  VARCHAR2(100),
    department_id NUMBER,
    salary      NUMBER
);
```

```
-- audit table to store changes
```

```
CREATE TABLE emp_audit (
    audit_id      NUMBER PRIMARY KEY,
    employee_id   NUMBER,
    action_type   VARCHAR2(10),    -- 'INSERT'/'UPDATE'/'DELETE'
    old_first_name VARCHAR2(100),
    old_last_name VARCHAR2(100),
    old_salary    NUMBER,
    new_first_name VARCHAR2(100),
    new_last_name VARCHAR2(100),
    new_salary    NUMBER,
    changed_by    VARCHAR2(30),
    changed_on    TIMESTAMP
);
```

```
-- simple sequence for audit PK
```

```
CREATE SEQUENCE emp_audit_seq START WITH 1 INCREMENT BY 1;
```

Trigger (row-level AFTER trigger that handles all 3 DMLs):

```
CREATE OR REPLACE TRIGGER trg_emp_audit
AFTER INSERT OR UPDATE OR DELETE ON employees
FOR EACH ROW
BEGIN
  IF INSERTING THEN
    INSERT INTO emp_audit (
      audit_id, employee_id, action_type,
      old_first_name, old_last_name, old_salary,
      new_first_name, new_last_name, new_salary,
      changed_by, changed_on
    ) VALUES (
      emp_audit_seq.NEXTVAL,
      :NEW.employee_id, 'INSERT',
      NULL, NULL, NULL,
      :NEW.first_name, :NEW.last_name, :NEW.salary,
      USER, SYSTIMESTAMP
    );
  ELSIF UPDATING THEN
    INSERT INTO emp_audit (
      audit_id, employee_id, action_type,
      old_first_name, old_last_name, old_salary,
```

ELSIF UPDATING THEN

```
  INSERT INTO emp_audit (
    audit_id, employee_id, action_type,
    old_first_name, old_last_name, old_salary,
```

```
    new_first_name, new_last_name, new_salary,  
    changed_by, changed_on  
) VALUES (  
    emp_audit_seq.NEXTVAL,  
    :NEW.employee_id, 'UPDATE',  
    :OLD.first_name, :OLD.last_name, :OLD.salary,  
    :NEW.first_name, :NEW.last_name, :NEW.salary,  
    USER, SYSTIMESTAMP  
);
```

```
ELSIF DELETING THEN  
    INSERT INTO emp_audit (  
        audit_id, employee_id, action_type,  
        old_first_name, old_last_name, old_salary,  
        new_first_name, new_last_name, new_salary,  
        changed_by, changed_on  
    ) VALUES (  
        emp_audit_seq.NEXTVAL,  
        :OLD.employee_id, 'DELETE',  
        :OLD.first_name, :OLD.last_name, :OLD.salary,  
        NULL, NULL, NULL,  
        USER, SYSTIMESTAMP  
    );  
END IF;  
END trg_emp_audit;  
/
```

## Explanation & notes

:OLD and :NEW provide access to row values in row-level triggers.

We use a separate audit table — this avoids mutating-table problems because the trigger writes to a different table.

USER and SYSTIMESTAMP capture who and when.

If you have many columns, you can add columns to emp\_audit per column, or use a single CLOB to store JSON/text representation of old/new row (Oracle JSON features or manual concatenation).

Be careful with heavy DML: triggers add overhead. For high-volume systems, consider asynchronous audit (e.g., write change records to a staging queue and process later).

## 2) Procedure / Function that returns multiple employees by criteria

Two common patterns: (A) SYS\_REFCURSOR (easy to call from Java/clients), (B) Pipelined table function (can be used in SQL FROM TABLE(...)).

### A) Using SYS\_REFCURSOR (procedure)

```
CREATE OR REPLACE PROCEDURE get_employees_refcur(
    p_dept_id IN NUMBER,
    p_min_salary IN NUMBER,
    p_max_salary IN NUMBER,
    p_rc      OUT SYS_REFCURSOR
) AS
```

```

BEGIN
  OPEN p_rc FOR
    SELECT employee_id, first_name, last_name, department_id, salary
    FROM employees
   WHERE (p_dept_id IS NULL OR department_id = p_dept_id)
     AND (p_min_salary IS NULL OR salary >= p_min_salary)
     AND (p_max_salary IS NULL OR salary <= p_max_salary)
  ORDER BY salary DESC;
END get_employees_refcur;
/

```

How to call (SQL\*Plus / SQL Developer):

```

VARIABLE rc REFCURSOR;
EXEC get_employees_refcur(10, 5000, 10000, :rc);
PRINT rc;

```

### Explanation

SYS\_REFCURSOR is flexible and easy to consume from Java (via CallableStatement and getObject/getResultSet) or from SQL clients.

NULL parameters act as wildcards via p\_param IS NULL OR ... patterns.

B) Pipelined table function (used directly in SQL)

Step 1 — SQL object & table types (schema-level types required for pipelined functions):

```
CREATE TYPE emp_obj AS OBJECT (
    employee_id NUMBER,
    first_name VARCHAR2(100),
    last_name  VARCHAR2(100),
    salary     NUMBER
);
/
CREATE TYPE emp_obj_tab AS TABLE OF emp_obj;
```

Step 2 — pipelined function:

```
CREATE OR REPLACE FUNCTION get_emps_pipe(
    p_dept_id  IN NUMBER,
    p_min_salary IN NUMBER,
    p_max_salary IN NUMBER
) RETURN emp_obj_tab PIPELINED AS
BEGIN
    FOR r IN (
        SELECT employee_id, first_name, last_name, salary
        FROM employees
        WHERE (p_dept_id IS NULL OR department_id = p_dept_id)
```

```

        AND (p_min_salary IS NULL OR salary >= p_min_salary)

        AND (p_max_salary IS NULL OR salary <= p_max_salary)

    ORDER BY salary DESC

) LOOP

PIPE ROW(emp_obj(r.employee_id, r.first_name, r.last_name, r.salary));

END LOOP;

RETURN;

END get_emps_pipe;
/

```

How to use:

```
SELECT * FROM TABLE(get_emps_pipe(10, 5000, 10000));
```

Explanation

Pipelined functions stream rows to the caller: useful when integrating with SQL or for large datasets (the caller can start consuming rows before the function completes).

Requires schema-level object/table type declarations.

Choose REF CURSOR if you want simpler interop with application code; choose PIPELINED if you need to use the result directly in SQL (join/filter/etc).

3) Mutating table error — cause and solutions (how to restructure)

Problem (brief):

ORA-04091 “table is mutating, trigger/function may not see it” occurs when a row-level trigger tries to query (or do certain operations on) the same table that is being modified — Oracle disallows querying that table during row-by-row operations to guarantee consistent state.

Common scenario: In a AFTER INSERT FOR EACH ROW trigger you attempt SELECT from the same table to compute aggregate values — that causes mutation.

Solutions (with examples)

A) Use a COMPOUND TRIGGER (Oracle 11g+)

Compound triggers let you collect row-level information and perform queries/updates after the statement completes (no mutating problem).

```
CREATE OR REPLACE TRIGGER trg_emp_compound  
FOR UPDATE OR INSERT OR DELETE ON employees  
COMPOUND TRIGGER
```

```
TYPE empid_tab IS TABLE OF employees.employee_id%TYPE;  
g_emp_ids empid_tab := empid_tab();
```

```
BEFORE EACH ROW IS  
BEGIN  
-- collect the affected employee ids (could be :OLD or :NEW)  
IF INSERTING OR UPDATING THEN
```

```
g_emp_ids.extend;  
  
g_emp_ids(g_emp_ids.count) := :NEW.employee_id;  
  
ELSIF DELETING THEN  
  
g_emp_ids.extend;  
  
g_emp_ids(g_emp_ids.count) := :OLD.employee_id;  
  
END IF;  
  
END BEFORE EACH ROW;
```

```
AFTER STATEMENT IS  
  
BEGIN  
  
-- safe to query/update employees here (statement-level)  
  
FOR i IN 1 .. g_emp_ids.count LOOP  
  
-- example: write a log or update a summary table (no mutating error here)  
  
NULL; -- replace with real logic  
  
END LOOP;  
  
END AFTER STATEMENT;
```

```
END trg_emp_compound;  
/
```

Why it works: the per-row section only collects data; the after-statement part runs when the DML completes and the table is stable — so queries are allowed.

B) Use an application/package variable to accumulate rows, and a statement-level trigger to process

Create a package with a collection variable.

Row-level trigger pushes keys into the package collection.

After-statement trigger reads package collection and does processing.

(Conceptually same as compound trigger, but split across package + triggers — older approach prior to compound triggers.)

C) Avoid using triggers for this logic at all

Move logic to the application layer or use scheduled jobs (DBMS\_SCHEDULER), or use materialized views / streams depending on use-case.

Triggers are good for auditing or quick enforcement, but complex logic (multi-row aggregate updates) is often better handled outside triggers.

Recommendation for Infosys-style interviews: Explain the mutating cause, then present a compound-trigger solution (or package+statement triggers) and mention trade-offs (readability, testability, performance).

4) PL/SQL block using collections (nested table / VARRAY) to process IDs and perform bulk updates

Goals shown:

Use PL/SQL collections

Use BULK COLLECT and FORALL for performance

Handle large datasets in batches

Show error handling with SAVE EXCEPTIONS

Example: batch update of salary for a given list of employee IDs

```
SET SERVEROUTPUT ON;
```

```
DECLARE
```

```
-- collection type (nested table of employee ids)
```

```
TYPE t_empid_tab IS TABLE OF employees.employee_id%TYPE;
```

```
l_emp_ids t_empid_tab := t_empid_tab(101, 102, 103, 104); -- sample IDs to update
```

```
BEGIN
```

```
-- bulk update using FORALL (very efficient)
```

```
FORALL i IN 1 .. l_emp_ids.COUNT SAVE EXCEPTIONS
```

```
UPDATE employees
```

```
SET salary = salary * 1.10 -- give 10% raise
```

```
WHERE employee_id = l_emp_ids(i);
```

```
COMMIT;
```

```
DBMS_OUTPUT.PUT_LINE('Bulk update complete. Processed ' || l_emp_ids.COUNT || ' employees.');
```

```
EXCEPTION
```

```
WHEN OTHERS THEN
```

```
-- If SAVE EXCEPTIONS used, SQL%BULK_EXCEPTIONS tells where the errors happened
```

```
IF SQLCODE = -24400 THEN
```

```
-- handle cases where no exceptions are saved? (safe guard)
```

```

DBMS_OUTPUT.PUT_LINE('Unexpected bulk exception state.');

END IF;

IF SQL%BULK_EXCEPTIONS.COUNT > 0 THEN

  FOR i IN 1 .. SQL%BULK_EXCEPTIONS.COUNT LOOP

    DBMS_OUTPUT.PUT_LINE(
      'Error at index ' || SQL%BULK_EXCEPTIONS(i).ERROR_INDEX ||
      ' SQLCODE=' || SQL%BULK_EXCEPTIONS(i).ERROR_CODE ||
      ' message=' || SQLERRM(-SQL%BULK_EXCEPTIONS(i).ERROR_CODE)

    );
  END LOOP;

ELSE

  DBMS_OUTPUT.PUT_LINE('Error: ' || SQLERRM);

END IF;

ROLLBACK;

END;
/

```

Example: iterate over a large dataset in batches (BULK COLLECT ... LIMIT + FORALL)

```

DECLARE

CURSOR c IS

SELECT employee_id, salary

FROM employees

WHERE department_id = 10

ORDER BY employee_id;

```

```
TYPE t_empid_tab IS TABLE OF employees.employee_id%TYPE;
TYPE t_salary_tab IS TABLE OF employees.salary%TYPE;

l_ids  t_empid_tab := t_empid_tab();
l_salaries t_salary_tab := t_salary_tab();

l_limit CONSTANT PLS_INTEGER := 500; -- batch size

BEGIN
  OPEN c;
  LOOP
    FETCH c BULK COLLECT INTO l_ids, l_salaries LIMIT l_limit;
    EXIT WHEN l_ids.COUNT = 0;

    -- bulk update salaries (for example, 5% increment)
    FORALL i IN 1..l_ids.COUNT SAVE EXCEPTIONS
      UPDATE employees
        SET salary = l_salaries(i) * 1.05
        WHERE employee_id = l_ids(i);

    COMMIT; -- commit per batch to avoid undo growth
  END LOOP;
  CLOSE c;
END;
/
```

## Explanation & best practices

BULK COLLECT reduces context switches when fetching many rows.

FORALL reduces context switches for DML.

Use LIMIT when processing very large datasets to avoid memory issues.

SAVE EXCEPTIONS allows FORALL to continue after some row-level errors and collect them in SQL%BULK\_EXCEPTIONS.

Commit frequency: committing once in very large batches helps manage undo retention vs performance tradeoff.

## 5) Analytic functions: find 2nd highest or top N per group — examples & explanation

Analytic/window functions (e.g., ROW\_NUMBER(), RANK(), DENSE\_RANK()) are ideal for “top N per group” problems. I’ll show multiple queries and explain tie behavior.

### Sample data (conceptual)

employee\_id | first\_name | department\_id | salary

employee_id	first_name	department_id	salary
-----			
101	John	10	9000
102	Jane	10	8000
103	Bob	10	8000
104	Alice	20	10000
105	Mary	20	9500

106	Tom	20	9000
-----	-----	----	------

(A) Get the 2nd highest salary per department — using ROW\_NUMBER() (picks one row if ties)

```
SELECT department_id, employee_id, first_name, salary  
FROM (  
    SELECT department_id, employee_id, first_name, salary,  
        ROW_NUMBER() OVER (PARTITION BY department_id ORDER BY salary DESC) AS rn  
    FROM employees  
) t  
WHERE rn = 2;
```

#### Behavior

ROW\_NUMBER() assigns a unique sequence per partition ordered by salary. If there are ties, ROW\_NUMBER arbitrarily orders tied rows and assigns different numbers; so only one row will be returned as the "2nd" even if there are ties.

Using our sample:

Dept 10 salaries: 9000 (rn=1), 8000 (rn=2), 8000 (rn=3) — only one of the 8000s returns.

(B) Get the 2nd highest salary per department — using RANK() (returns all ties at same rank)

```
SELECT department_id, employee_id, first_name, salary  
FROM (  
    SELECT department_id, employee_id, first_name, salary,  
        RANK() OVER (PARTITION BY department_id ORDER BY salary DESC) AS rnk
```

```
FROM employees
```

```
) t
```

```
WHERE rnk = 2;
```

## Behavior

RANK() gives the same rank to ties, but leaves gaps after ties.

In dept 10: ranks -> 9000 (rnk=1), 8000 (rnk=2), 8000 (rnk=2) — both 8000 rows are returned.

(C) DENSE\_RANK() — like RANK() but without gaps after ties

```
SELECT department_id, employee_id, first_name, salary
```

```
FROM (
```

```
    SELECT department_id, employee_id, first_name, salary,
```

```
        DENSE_RANK() OVER (PARTITION BY department_id ORDER BY salary DESC) AS dr
```

```
    FROM employees
```

```
) t
```

```
WHERE dr = 2;
```

## Behavior

DENSE\_RANK() maps equal values to same rank and next distinct value gets the next consecutive rank (no gaps).

Example with salaries 100, 90, 90, 80: DENSE\_RANK -> 1,2,2,3; RANK -> 1,2,2,4.

(D) Top N per group (e.g., top 3 salaries per department) — using ROW\_NUMBER():

```
SELECT department_id, employee_id, first_name, salary  
FROM (  
    SELECT department_id, employee_id, first_name, salary,  
        ROW_NUMBER() OVER (PARTITION BY department_id ORDER BY salary DESC) rn  
    FROM employees  
) t  
WHERE rn <= 3  
ORDER BY department_id, rn;
```

#### Notes

ROW\_NUMBER() is appropriate when you want a fixed number of rows per group (exactly N rows), even if ties exist — it will arbitrarily break ties.

If you want to include all tied rows at the Nth rank, use RANK() or DENSE\_RANK() and <= that rank value.

(E) Get the “Nth highest salary” value per department (without returning rows) — using DENSE\_RANK():

To get the 2nd highest salary value per department:

```
SELECT department_id, salary AS second_highest_salary  
FROM (  
    SELECT department_id, salary,  
        DENSE_RANK() OVER (PARTITION BY department_id ORDER BY salary DESC) dr
```

```
FROM employees
```

```
) t
```

```
WHERE dr = 2;
```

This returns the salary value(s) that are second highest. If multiple employees share that salary, all rows with that salary are returned — you can wrap with DISTINCT or aggregate if you want just one value per department.

#### (F) Performance tips

Window functions require sorting within partitions. For large tables, ensure appropriate indexing (on partition key and/or order by column) and consider predicate filtering before windowing.

Use ROW\_NUMBER() when you must limit to exact N rows per group (deterministic).

Use RANK()/DENSE\_RANK() when tie-handling matters and you may want multiple rows for same rank.

Final notes / quick checklist for interview-ready answers

When presenting trigger solutions, mention mutating-table pitfalls and when triggers are appropriate (audit vs heavy logic).

For stored procedures/functions, explain trade-offs: REF CURSOR (application friendly) vs pipelined functions (SQL-friendly).

For bulk operations, always show BULK COLLECT + FORALL + LIMIT for production scale and mention SAVE EXCEPTIONS for error handling.

For analytic queries, explain tie behavior differences between ROW\_NUMBER, RANK, and DENSE\_RANK with small example data — interviewers often ask that nuance.

Mention testing: test triggers & bulk logic with edge cases (NULLs, large volume, constraints, transaction rollbacks).

*I have used PL/SQL mainly for handling business logic in the database layer. For example, in one of my projects, I worked on a Pathology Reporting module where we had to enhance the report generation for Gross and Micro Correlation. We added a feature called 'Show All Animals', which allows users to see all animals in the report even if no gross or microscopic data is available. To achieve this: I wrote PL/SQL functions like IS\_SHOW\_ALL\_ANIMALS to check if the checkbox option was selected. Used IF conditions and loops inside PL/SQL to control when to insert records. Implemented INSERT statements into RPT\_GROSS\_MICRO\_CORRELATIONS table with default values like 'No Data Available' when data was missing. Added logic so that when the option was not checked, only animals with available data were inserted into the report. This way, I used PL/SQL to implement conditional data handling, improve performance, and ensure correct reporting."*

## What is a Cursor?

In Oracle, when you execute a SQL query, Oracle allocates a work area in memory to process it.

A cursor is a pointer to that memory/work area which stores the query result.

In simple words: A cursor lets you fetch rows one by one from a query result set inside PL/SQL.

### 2. Types of Cursors

#### Implicit Cursor

Created automatically by Oracle for DML statements (INSERT, UPDATE, DELETE, SELECT INTO).

You don't declare it manually.

Example:

```
BEGIN  
    UPDATE employees SET salary = salary * 1.1 WHERE department_id = 10;  
    DBMS_OUTPUT.PUT_LINE(SQL%ROWCOUNT || ' rows updated.');
```

END;

Here, Oracle internally uses an implicit cursor.

SQL%ROWCOUNT → tells how many rows were affected.

#### Explicit Cursor

Declared by the programmer for queries that return multiple rows.

You control when to OPEN, FETCH, and CLOSE it.

### 3. Explicit Cursor Example (Step by Step)

Suppose we want to display employees of department 10:

DECLARE

-- Step 1: Declare the cursor

CURSOR emp\_cur IS

SELECT employee\_id, first\_name, salary

FROM employees

WHERE department\_id = 10;

-- Step 2: Declare variables to hold fetched values

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

v\_name employees.first\_name%TYPE;

v\_salary employees.salary%TYPE;

BEGIN

-- Step 3: Open the cursor

OPEN emp\_cur;

-- Step 4: Fetch rows one by one

LOOP

FETCH emp\_cur INTO v\_emp\_id, v\_name, v\_salary;

EXIT WHEN emp\_cur%NOTFOUND; -- exit when no more rows

DBMS\_OUTPUT.PUT\_LINE('ID: ' || v\_emp\_id ||

' Name: ' || v\_name ||

' Salary: ' || v\_salary);

END LOOP;

-- Step 5: Close the cursor

CLOSE emp\_cur;

```
END;
```

#### 4. Cursor FOR Loop (Simplified)

Oracle gives a shorthand:

```
BEGIN
```

```
FOR rec IN (SELECT employee_id, first_name, salary  
            FROM employees  
            WHERE department_id = 10) LOOP  
    DBMS_OUTPUT.PUT_LINE('ID: ' || rec.employee_id ||  
                         ' Name: ' || rec.first_name ||  
                         ' Salary: ' || rec.salary);
```

```
END LOOP;
```

```
END;
```

✓ This automatically opens, fetches, and closes the cursor. Very commonly used.

#### 5. Cursor Attributes

Every cursor (implicit or explicit) has attributes:

%FOUND → TRUE if last fetch returned a row.

%NOTFOUND → TRUE if last fetch did not return a row.

%ROWCOUNT → Number of rows fetched so far.

%ISOPEN → TRUE if cursor is still open.

Example:

```
IF emp_cur%ISOPEN THEN  
    DBMS_OUTPUT.PUT_LINE('Cursor is open');  
END IF;
```

Cursor = pointer to result set.

Types = implicit (auto by Oracle) & explicit (defined by user).

Explicit cursor requires OPEN → FETCH → CLOSE.

Cursor FOR loop simplifies handling.

Attributes like %FOUND, %ROWCOUNT are important.

## **What is a trigger?**

A trigger is a stored PL/SQL block that Oracle executes automatically when a specified event occurs on a table, view or the database (for example: INSERT, UPDATE, DELETE, DDL events, or database events like LOGON).

Common uses: enforcing business rules, auditing, maintaining derived/summary data, validating/modifying data, and protecting referential logic.

1. How can you optimize a BigQuery query?
2. Difference between ROWNUM and ROW\_NUMBER().
3. Have you used ROW\_NUMBER() in your application? Example scenario.
4. Different types of Analytical Functions.
5. What is an Index?
6. Types of Indexes.
7. What is a Constraint?
8. Difference between Primary Key and Unique Key.
9. Difference between Procedure and Function.
10. Why can't a Procedure return data directly? Role of OUT parameters.
11. Can a Function return multiple values? How?
12. Types of Cursors.
13. What is an Exception in PL/SQL?
14. What is an Autonomous Transaction?
15. Given two tables (T1 & T2), what will be the row counts for Inner Join, Left Join, Right Join, and Full Outer Join?
16. Is an Index a Database Object?
17. Usage of SQLLoader or SQLStar Loader.
18. Types of Collections in PL/SQL.
19. What is a Trigger?

20. Real-time scenarios where Triggers are used.

Company: Virtusa

Round: Technical Interview – 1

Role: Senior PL/SQL Developer

Focus areas: Performance tuning, SQL/PLSQL optimization, Joins, Analytical functions, Database objects, and real-world use cases