**Oracle SQL, PLSQL questions**

version 0.1



No

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Prepared By** | **Reviewed by** | **Approved By** |
| **Name** | Ankit Upadhyay |  |  |
| **Role** | Principal Consultant |  |  |
| **Date** |  |  |  |

Contents

**[Oracle Architecture and internals](#_Toc504749919)** [3](#_Toc504749919)

[**SQL** 6](#_Toc504749920)

[**PLSQL** 9](#_Toc504749921)

[**Performance Tuning** 12](#_Toc504749922)

## **Oracle Architecture and internals**

**Q: Explain Deadlocks and how Oracle resolves them?**

A: A deadlock occurs when two or more sessions are waiting for data locked by each other. Oracle automatically detects and resolves deadlocks by rolling back the statement associated with the transaction that detects the deadlock.

**Q: What is pessimistic locking (specify clause)?**

Oracle allows you to choose the strategy for locking, either pessimistic or optimistic, depending on your needs. When a user accesses an object to update it, the database locks the object until the update is completed. No other user can read or update the object until the first user releases the lock. This is achieved by SELECT xxx FOR UPDATE clause.

**Q: Explain difference between Consistent reads, Current reads and Physical reads.**

A: Consistent and current reads are logical I/O. Physical reads is physical I/O. Oracle makes multiple copies of a data block in the RAM buffers to support concurrent updates. Consistent get is any get from the RAM buffer. Current read is also called db block read which is a fetch from the current state of buffer.

**Q: How can we reduce REDO generated by Insert statements?**

A: Use

* APPEND hint in Insert statement.
* NOLOG option while creating a table, it will not skip redo but reduce it. Not suggested for Production environment but can be used for tables which hold temporary data.

**Q: What is UNDO log?**

A: When running DML like Updates and Deletes, Oracle will save a copy of non-modified data in Undo segment which can be used in case user rollback the transaction.

**Q: How does High water mark effects table scans?**

A: When Oracle does full table scan, it usually scans until high water mark set for the given table irrespective of how many blocks are occupied. So if we have deleted some data from a table, please note that high water mark will still be at a point where it was before delete so query to retrieve all rows from this table will still scan all the blocks.

**Q: In what scenario should we use Index Organized tables?**

A: Index organized tables store all columns of table in leaf node of the index structure. While in normal tables, indexes store only indexed keys and rowid. Useful when the data is accessed using the primary key. Data access based on range of primary key is also faster. As index and table data are stored in the same data structure, extra space is not needed for index(as in case of normal heap organized tables). Indexes are not in unusable state after table is reorganized.

**Q: How is Index skip scan different from normal index scan?**

A: Index skip scan applies when indexes contain more than one column (composite index), Oracle can opt for a index skip scan when

* the leading column of the index only one of the columns is there in the query.
* the leading column has low cardinality.

In above cases, Oracle uses distinct values of the leading column and use them to search for the next column in the index. In normal index scan, Oracle uses all columns and search their indexes.

**Q: What are various joins methods Oracle use to join datasets (not right, left and full)?**

A: Nested Loop join, Hash join, sort-merge join.

**Q: In what scenario Oracle will pick Hash join access path?**

A: Oracle will go for hash join when

* Enough hash area is available for query to use.
* Query uses equi-join.
* Nested loop join is inefficient (lack of indexes).

**Q: In what scenario Oracle will pick Nested loop join access path?**

A: Oracle will go for Nested loop join when

* Driving (outer) table is small and joined column has been indexed in the inner table.
* Highly useful when user can be presented with first few rows without waiting for the entire result set.

**Q: What could be possible reasons that CBO is not picking suitable plans?**

A:

* Table or index stats are stale.
* Data in the table is highly skewed i.e for one set of value you have very few rows while for other set you have huge amount of data. In such cases previosuly cached plan of the query may result in inefficient access path. Oracle introduced Adaptive cursor sharing in 11g to overcome this problem.
* Indexes are in unusable state.

**Q: When should be we use a Bitmap Index?**

A: Bitmap indexes are useful when

* Cardinality of the indexed column is low. e.g. gender, with only 2 values
* Primarily used in data warehouses, they should be avoided in OLTP applications.
* Very little dml on table.

**Q: Indexes helps us fetch data faster, explain why?**

A: Indexes are stored in sorted data structure, for example trees. When looking for a particular value, search is made faster using algorithms like binary search.

In case the query references to range of values for the indexed column, Oracle does Index range scan.

**Q: Are NULL values stored in Index?**

A: By default relational databases dont store NULL values in the index. However, starting 11g, Oracle introduced a feature to store NULL values.

**Q: What happens when we don’t index foreign keys?**

A: Query are much slower, Oracle avoids nested loop join and there is a possibility of deadlock.

**Q: How PLS\_INTEGER is better than NUMBER data type?**

A: PLS\_INTEGER uses less storage than NUMBER and uses machine arithmetic so it is faster than NUMBER.

**Q: Can we create indexes on External tables?**

A: No, external tables are stored in flat files and dont have rowids, so we cannot create indexes on external table.

**Q: Can we have two before-insert triggers on same table?**

A: Yes, we can use FOLLOWS keyword to control the order of execution of the triggers.

**Q: What is Interval partition?**

A: Kind of range partition where Oracle can automatically create new partitions if inserted partition key exceeds all other partition key values. Partition column must be NUMBER or DATE and only on one key.

**Q: What happens when we try to update a partition key and row movement is disabled?**

A: You receive an error ORA-14402: updating partition key column would cause a partition change

**Q: What are the different options to do Gather stats on a table?**

A: We can set the granularity of the stats to collect

* 'ALL' - gathers all (subpartition, partition, and global) statistics
* 'GLOBAL' - gathers global statistics
* 'GLOBAL AND PARTITION' - gathers the global and partition level statistics
* 'PARTITION '- gathers partition-level statistics
* 'SUBPARTITION' - gathers subpartition-level statistics.

## **SQL**

**Q: Employee table has details of employees including dept number. Write a SQL to print employee name, dept number and number of employees working in employee’s dept.**

A: We can use analytical function to print granular rows along with aggregate column.

SELECT emp\_id,   
       emp\_name,   
       dept\_no,   
       **Count**() OVER(PARTITION BY dept\_no)   
 FROM employee;

**Q: What are the options we can use to slide the window of any analytical function in Oracle?**

A: Analytical functions like RANK, COUNT have sliding windows.

RANGE BETWEEN start\_point AND end\_point

Default window is RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW

Possible values for "start\_point" and "end\_point" are:

* UNBOUNDED PRECEDING : The window starts at the first row of the partition, or the whole result set if no partitioning clause is used. Only available for start points.
* UNBOUNDED FOLLOWING : The window ends at the last row of the partition, or the whole result set if no partitioning clause is used. Only available for end points.
* CURRENT ROW : The window starts or ends at the current row. Can be used as start or end point.
* value\_expr PRECEDING : A physical or logical offset before the current row using a constant or expression that evaluates to a positive numerical value. When used with RANGE, it can also be an interval literal if the order\_by\_clause uses a DATE column.
* value\_expr FOLLOWING : As above, but an offset after the current row.

e.g

SELECT empno,   
       deptno,   
       sal,   
       **First\_value**(sal)   
         over (   
           ORDER BY sal ROWS BETWEEN 1 preceding AND CURRENT ROW) AS   
       previous\_sal,   
       **Last\_value**(sal)   
         over (   
          ORDER BY sal ROWS BETWEEN CURRENT ROW AND 1 following) AS next\_sal   
FROM   emp;

**Q: What are implicit cursors?**

A: Oracle creates implicit cursor which Oracle creates when we execute any select query in PLSQL. Using OPEN, FETCH and CLOSE creates explicit cursors.

**Q: Explain scenario in which we should use EXISTS not IN?**

A: If the subquery produces small number of rows, use IN else use EXISTS.

**Q: Implement rank in SQL without using RANK function.**

SELECT emp\_id

,sal

,(SELECT COUNT(distinct emp\_id) + 1

FROM emp e2

WHERE e2.sal > e1.sal

AND e2.emp\_id != e1.emp\_id

) rank\_of\_sal

FROM emp;

**Q: Can we use analytical function in Group By clause?**

A: No, all joins, where, group by clauses are executed before analytical function.

**Q: Display rows as columns for below data**

**SCENARIO CURRENCY       RATE**

**---------- -------- ----------**

**1 USD            1.00**

**1 CAD            1.45**

**1 MYR            3.34**

**1 NOK            6.08**

**2 USD            1.00**

**2 CAD            1.47**

**2 MYR            3.47**

**2 NOK            6.12**

**3 USD            1.00**

**3 CAD            1.51**

**3 MYR            2.98**

**3 NOK            5.96**

A: Use MAX and DECODE to transpose data

SELECT x.scenario,   
       **Max**(**Decode**(x.currency, 'CAD', x.rate)) CAD\_rate,   
       **Max**(**Decode**(x.currency, 'MYR', x.rate)) MYR\_rate,   
       **Max**(**Decode**(x.currency, 'NOK', x.rate)) NOK\_rate,   
       **Max**(**Decode**(x.currency, 'USD', x.rate)) USD\_rate   
FROM   (SELECT r.scenario,   
               r.currency,   
               r.rate   
        FROM   rates r) x   
GROUP  BY x.scenario

 SCENARIO   CAD\_RATE   MYR\_RATE   NOK\_RATE   USD\_RATE

---------- ---------- ---------- ---------- ----------

         1       1.45       3.34       6.08          1

         2       1.47       3.47       6.12          1

         3       1.51       2.98       5.96          1

## **PLSQL**

**Q: Why can’t Triggers have COMMIT inside body? What is the work-around?**

A: Trigger bodies are executed as a result of some DML. If we have COMMIT inside the trigger then it will complete the transaction/DML statement which will invalidate the purpose of the trigger (purpose might be to stop the DML from happening).

Work-around is to use AUTONOMOUS\_TRANSACTION which will only commit the transaction inside the trigger, not the main dml.

**Q: How is Index by table different from Nested tables?**

A: Index by table is a PLSQL object which is used inside a procedure or function to store the variables. While Nested table is a database object which can a store data in tables.

**Q: Can we create Indexes on Views? If yes, then how?**

A: Normal views cannot have indexes as they are stored queries not data. Materialized views can have indexes though.

**Q: How do we benefit from packaging functions and procedure in a Package?**

A:

* Since all the related code is in one module, they are compiled together and the bytecode is stored together thus reduces runtime.
* Gives modularity to your code.
* We can have functions with same name (with different parameters ofcourse) aka function overloading.

**Q: What is body-less package and explain its usage?**

A: Package can just have specification. They are used to declare session variables.

**Q: Can we have two functions with same name in a package?**

A: We can have functions with same name (with different parameters ofcourse) aka function overloading.

**Q: What is the fastest way to copy data from one table to another?**

A:

* Exchange partition if the table structures are same.
* INSERT INTO SELECT statements.
* BULK COLLECT statements.

**Q: Can we use DECODE in PLSQL? If yes, how?**

A: Yes, we need to use SELECT query

e.g.

SELECT supplier\_name,   
       **Decode**(supplier\_id, 10000, 'IBM',   
                           10001, 'Microsoft',   
                           10002, 'Hewlett Packard',   
                           'Gateway') result   
INTO   v\_supplier\_id   
FROM   suppliers; 

**Q: What are the restrictions on a function that is called from a SQL and why?**

A:

* Deterministic function required:  A function must have the deterministic keyword if used with SQL.
* No updates inserts or deletes from inside DML:  A PL/SQL function is restricted against writing to the database with DML if the function resides within a DML.
* No DDL allowed:  A function called from inside a SQL statement is restricted against DDL because DDL issues an implicit commit.  You cannot issue any DDL statements from within a PL/SQL function.
* Restrictions against constraints:  You cannot use a function in the check constraint of a create table DDL statement.
* No commits or IN, OUT parms:  When called from within a SQL query, a function cannot have OUT or IN parameters, and the function is restricted against using a commit.

**Q: What are Pipelined Table functions?**

A: Table functions are functions that produce a collection of rows (either a nested table or a varray) that can be queried like a physical database table. You use a table function like the name of a database table, in the FROM clause of a query.

e.g.

SELECT c.name,   
       Book.name,   
       Book.author,   
       Book.abstract   
FROM   catalogs c,   
       TABLE(**Getbooks**(c.cat)) Book;

**Q: What are the different types of Materialized Views?**

A: Generally we have two types REFRESH ON COMMIT and REFRESH ON DEMAND.

**Q: How do we catch and handle exceptions for BULK/FORALL statements?**

A: Using %BULK\_EXCEPTIONS.

All exceptions raised during the execution are saved in the cursor attribute %BULK\_EXCEPTIONS, which stores a collection of records.

Each record has two fields:

* + %BULK\_EXCEPTIONS(i).ERROR\_INDEX holds the "iteration" of the FORALL statement during which the exception was raised.
  + %BULK\_EXCEPTIONS(i).ERROR\_CODE holds the corresponding Oracle error code.

## **Performance Tuning**

**Q: How do we approach to tune the performance of a SQL query?**

A:

* Check query is using only required data
  + check if any unnecessary table is there which can be avoided.
  + it should not use any unnecessary outer join.
  + check if a required join condition is missing.
  + check for repeated use of a table (use WITH clause in such conditions)

* Check the semantics of the query
  + should not use HAVING when we can use WHERE and then HAVING.
  + check the use of IN vs EXISTS
  + check whether we can filter data before joining it with another table
  + use appropriate join conditions (specially columns which have indexes)
  + check if there is any function applied in the join condition (e.g. TO\_NUMBER(e.dept\_id) = d.dept\_id, this will avoid index on dept\_id column in emp table)
  + check for unnessary string operations, mathematical operations
  + check for hints like INDEX, FULL SCAN etc which might force to compiler to take wrong access path

* Use Explain plan
  + check the access path of the query
  + look for any CARTESIAN joins
  + some FULL TABLE SCANS are beneficial (when we are querying more than 15% of data)
  + Sometimes plan at execution is different, use dbms\_xplan.display\_cursor with SQL ID to find actual plan used. SQL ID can be found in v$sql views.
  + We can also use v$sql to find out most time taking queries.

* Use DBA\_HIST\_SQL\_PLAN, displays the SQL execution plans.
* Use Active session History: V$ACTIVE\_SESSION\_HISTORY, DBA\_HIST\_ACTIVE\_SESS\_HISTORY
* Use SQL Trace facility with TKPROF utility.
* Use DBMS\_PROFILER, it gathers and save run-time statistics.

**Q: How do we approach to tune the performance of a Stored Procedure?**

A: Manually time the various statements inside the procedure to find out whether it is SQL or PLSQL which is taking time. If is PLSQL,

* Check for unoptimized looping conditions.
* check for unnecessary COMMITs inside the procedure.
* Use bind variables in queries, EXECUTE IMMEDIATE statements.
* Check for unnecessary string operations, mathematical operations, large variables.
* Check if BULK operations can be used instead of row by row operations.
* Check if built-in functions can be used in place of user defined functions.
* Put least expensive/most occurring conditions first in conditional statments.
* Avoid data type conversions.
* Use PLS\_INTEGER instead of NUMBER/INTEGER wherever possible.
* Group related functions and procedures in a package.

**Q: In AWR reports, how do we identify Index scans, Full table scans?**

A: db file sequential read will generally identify Index scans. db file scattered read will generally identify full table or index scans.

**Q: In what scenarios we would partition an index globally and locally?**

A:

* Local indexes are suitable for data warehouses and OLAP. They are ideal when they are prefixed with the key which has been used to partition the table. One partition of local index points to only one partition in table.
* Global indexes are useful for OLTP applications where index probes are lesser. They can be used to enforce uniqueness on columns which are not used to partition the table. One partition of a global index can point to multiple partitions of table.

**Q: How to generate plan for a query and how do we read its output?**

A: Execution plans can be generated using statement

EXPLAIN PLAN FOR <query>;

Output of the plan will be stored in a plan table.

Example

SQL> explain plan for select \* from emp

     where empno not in (select min(empno)

     from big\_emp group by empno);

Query Plan

------------------

SELECT STATEMENT [CHOOSE]  Cost=1

  FILTER

    TABLE ACCESS FULL EMP

     FILTER

        SORT GROUP BY NOSORT

          INDEX FULL SCAN BE\_IX

Most indented row(s) will be the first ones to be processed. In above plan big\_emp table is accessed using Index scan and emp table is being accessed using full table scan.

**Q: In case of slow query performance, what are the things to look in Explain Plan?**

A: Basically we look for

* + access paths used (index scans, full table scans etc)
  + join method used (nested loop, hash join etc)
  + Sorting, merging, filters of data
  + Cost of various steps (we may concentrate on higher costs steps in the plan to tune the query)

**Q: In what scenario Full table scan good for the performance of the query?**

A: When we are accessing more than 15% of the data. Full tables scans are useful PARALLEL queries. If the tables are very small generally full table scans are faster than index scans.

**Q: In what scenario Index scan is good for the performance of the query?**

A: When the query is using very selective data from the table (usually less than 15%).

**Q: What is the use of APPEND Hint?**

A: APPEND hint is used in INSERT statements. It is also called direct path insert which means that data is written directly into datafiles, bypassing the buffer cache. Oracle appends the inserted data after existing data in the table over high water mark.