DATABASE ADMINISTRATION

T2: DATABASE OPTIMIZATION [C6]

Physical Design and Tools

CHAPTER PLAN

- 1. Physical Design and Tools
  - 1.1 Physical Design - Optimization Objectives
  - 1.2 Physical Design Process
  - 1.3 Execution Plan Tool
  - 1.4 CBO Scopes
- 2. Table and Index Access Optimization
  - 2.1 Buffering & Table (scan) Access Opt. Techniques
  - 2.2 Index-based Optimization Techniques
- 3. JOIN and SORT Operation Optimization
  - 3.1 JOIN Optimization Techniques
  - 3.2 SORT Optimization Techniques
- 4. More advanced techniques
  - Transformed Subqueries: subquery refactoring
  - Subquery optimization: temporary tables and materialized views
  - Access optimization with Parallel Query

3. JOIN and SORT Optimization

- JOIN operations and tuning techniques.
- SORT operations and tuning techniques.

3.1 JOIN Operation Optimization

- SQL JOIN types [Logical JOINs]
  - INNER JOIN
  - OUTER JOIN
  - MERGE (natural join)
  - anti.JOIN [with subqueries NOT IN, NOT EXISTS, !=ALL()]
  - CROSS JOIN [CARTESIAN JOIN]

- Execution(Explain) Plan JOIN operation types (joining algorithms):
  - NL as Nested Loop Join;
  - HASH;
  - MERGE JOIN.

Practice C6_P1

Steps	Notes	SQL Script
SQL JOIN Types	INNER JOIN MERGE OUTER JOIN LATERAL* CROSS JOIN NOT IN [ANTI JOIN]	C6_P1.1.JOIN_Types_and_Operations.sql
OEP JOIN Operations	NL JOIN HASH JOIN SORT MERGE JOIN PARALLEL JOIN	C6_P1.1.JOIN_Types_and_Operations.sql

Nested-LOOP Join [T1 ⋈ (t1.a=t2.b) T2]

```
FOR rec_t1 IN (SELECT * FROM t1) LOOP
    FOR rec_t2 IN (SELECT * FROM t2) LOOP
        IF Join_Predicate(rec_t1.a, rec_t2.b) THEN
            Save_Joined(rec_t1, rec_t2);
        END IF;
    END LOOP;
END LOOP;
```

Sort-MERGE Join: [T1 ⋈ (t1.a=t2.b) T2]

```
-- Step 1:
FOR rec_t1 IN (SELECT * FROM t1) LOOP
        Save_Sorted(sort_result_of => 'T1_S', sort_key => 'T1.A', sort_val => rec_t1.a
                sorted_record => rec_t1);
END LOOP;
FOR rec_t2 IN (SELECT * FROM t2) LOOP
        Save_Sorted(sort_result _of => 'T2_S', sort_key => 'T2.B', sort_val => rec_t2.b
                sorted_record => rec_t2);
END LOOP;
-- Step 2
WHILE T1_S.COUNT > 0 AND T2_S.COUNT > 0 LOOP
        head_t1_s := t1_s(1).a;
        head_t2_s := t2_s(1).a;
        IF head_t1_s  > head_t2_s  THEN
                Remove_Head_Of('T2_S');
        ELSE IF head_t1_s  < head_t2_s  THEN
                Remove_Head_Of('T1_S');
        ELSE IF Join_Predicate(head_t1_s , head_t1_s )
                Save_Joined(t1_s(1), t2_s(1));
                Remove_Head_Of('T1_S');
                Remove_Head_Of('T2_S');
        END;
END LOOP;
```

HASH Join [T1 ⋈(t1.a=t2.b) T2]

- *Build phase (build in-memory hash table)*

```
For each row RW1 in small (left/build) table [T1] loop
    Calculate hash value on RW1 join key
    Insert RW1 in appropriate hash bucket.
End loop;
```

- *Probe Phase*

```
For each row RW2 in big (right/probe) table [T2] loop
    Calculate the hash value on RW2 join key
    For each row RW1 in hash table loop
        If RW1 joins with RW2
        Return RW1, RW2
    End loop;
End loop;
```

HASH Join [T1 ⋈(t1.a=t2.b) T2]

```
-- Step 1:
-- Hash driving table: optional, only if not fit in hash_area_size
FOR rec_t1 IN (SELECT * FROM t1) LOOP
        hash_value := hash_partitioning_function(rec_t1.a);
        IF NOT Exists_Partition (partition_hash = > hash_value, source => 'T1') THEN
                Create_Partition(partition_hash = > hash_value, source => 'T1');
        END IF;
        Add_rec_Into_Hash_Partition(hash_key => hash_value, source=> 'T1', record => rec_t1);
END LOOP;

FOR rec_t2 IN (SELECT * FROM t2) LOOP
        hash_value := hash_partitioning_function (rec_t2.b);
        IF NOT Exists_Partition (partition_hash = > hash_value, source => 'T2') THEN
                Create_Partition(partition_hash = > hash_value, source => 'T2');
        END IF;
        Add_rec_Into_Hash_Partition(hash_key => hash_value, source=> 'T2'', record => rec_t2);
END LOOP;
```

HASH Join [T1 ⋈(t1.a=t2.b) T2]

```
-- Step 2:
FOR i IN 1..Partition_Count(source => T1) LOOP
-- build [in memory] table for partition i of T1 --
        FOR rec_t1_part_i IN
        (SELECT * FROM Partition_Table(source=>t1, partition_no=>i))
        LOOP
                hash_ref_value := Hash_Probe_Function(rec_t1_part_i.a);
                In_Memory_Tbl(/*indexed_key =>*/ hash_ref_value) =
                        /*row_physical_pointer =>*/ rec_t1_part_i .rowid;
        END LOOP;
-- probe partition i of T2 against [in memory] table of partition i of T1 --
        FOR rec_t2_part_i IN
        (SELECT * FROM Partition_Table(source=>t2, partition_no=>i))
        LOOP
                hash_probe_value := Hash_Probe_Function(rec_t2_part_i.b);
                row_physical_pointer _t1 := In_Memory_Tbl(hashe_probe_value) ;
                IF row_physical_pointer _t1 IS NOT NULL THEN
                        rec_t1_i := read_record_from_physical_pointer(
                                row_physical_pointer _t1);
                        Save_Joined(rec_t1_i, rec_t2_part_i);
                END IF;
        END LOOP;
END LOOP;
```

JOIN Operation Optimization

OEP JOIN Operation	Tuning Technique	Notes
Nested Loop [NL] JOIN	Foreign-key based index Change JOIN type: by JOIN hint Change JOIN type: by OEP goal	
Hash [JOIN] JOIN	Change table JOIN order In-memory HASH Change JOIN Type	HASH_MEMORY SIZE
Merge-Sort [MERGE] JOIN	In-memory SORT Sorting index Sorting cluster Change JOIN type	SORT_MEMORY_SIZE
BITMAP JOIN Index	CREATE BITMAP INDEX ON FROM WHERE	

Practice C6_P1

Steps	Notes	SQL Script
NL JOIN Tuning	CREATE INDEX fk_idx	C6_P1.2.JOIN_Tuning.sql
HASH JOIN Tuning	ALTER SESSION SET hash_area_size /*+ LEADING */	C6_P1.2.JOIN_Tuning.sql
SORT JOIN Tuning	ALTER SESSION SET sort_area_size CREATE INDEX sort_idx	C6_P1.2.JOIN_Tuning.sql
NL JOIN Tuning: BITMAP JOINing	CREATE BITMAP INDEX ON FROM WHERE	C6_P1.2.JOIN_Tuning.sql

3.2 SORT Operation optimization

- *Logical SQL query blocks:*
  - ORDER BY, GROUP BY
  - Aggregate functions: MIN, MAX
  - DISTINCT
  - JOIN
  - OVER (ORDER BY),
    - FIRST_VALUE, LAST_VALUE, RANK
- *OEP SORT operations:*
  - SORT ORDER BY
  - SORT GROUP BY
  - SORT AGGREGATE
  - SORT JOIN
  - WINDOW SORT

SQL Query blocks and OEP SORT Operations

SQL Query Block	OEP Operations
ORDER BY	SORT ORDER BY
GROUP BY	SORT GROUP BY HASH GROUP BY
/*+ use_merge */ JOIN	SORT JOIN
MIN/MAX	SORT AGGREGATE
RANK OVER(ORDER BY)	WINDOW SORT
SELECT DISTINCT	SORT UNIQUE NOSORT HASH UNIQUE

SORT Operation optimization

OEP JOIN Operation	Tuning Technique	Notes
SORT ORDER BY SORT GROUP BY SORT AGGREGATE MERGE SORT WINDOW SORT	In-memory sorting SORT_AREA_SIZE	WORKAREA_SIZE_POLICY PGA_AGGREGATE_TARGET
SORT ORDER BY MERGE SORT SORT AGGREGATE	Index-based sorting Cluster-based sorting	CREATE INDEX FK
SORT ORDER BY SORT AGGREGATE MERGE SORT	Rewrite sub-query	WITH subquery ROWID CREATE BITMAP INDEX

Practice C6_P2

Steps	Notes	SQL Scripts
SORT Operations	SORT ORDER BY SORT GROUP BY SORT AGGREGATE	C6_P2.1.SORT_Operations_and_Tuning.sql
SORT Optimize: adjust work area	SORT_AREA_SIZE	C6_P2.2.SORT_Tuning.sql
SORT Optimize: adding index	CREATE INDEX fk_btree	C6_P2.2.SORT_Tuning.sql
SORT Optimize: using ROWID table access	CREATE BITMAP INDEX TABLE ACCESS BY INDEX ROWID	C6_P2.2.SORT_Tuning.sql
SORT OPTIMIZE: cluster-based SORT	CREATE HASH CLUSTER SORT	C6_P2.2.SORT_Tuning.sql
Materialized VIEW	CREATE MATERIALIZED VIEW	C6_P2.2.SORT_Tuning.sql

Useful OEP /*+ HINTS*/

- To simulate workloads:
  - /*+ cardinality (tbl 1e9)*/
    - 1e9 = 1000 Million rows
  - /*+ leading (t1 t2 t3 t4)*/
    - control joining order

3.2 More Advanced Techniques

- Subqueries refactoring
- Temporary tables and materialized viewed
- Parallel queries

Subquery refactoring

- REWRITE subqueries:
  - using WITH clause preceding main SELECT statement;
  - adding /*+ materialize */ HINT to avoid VIEW operations within execution plan.
- Use Hints to avoid automatic subquery rewriting:
  - NO_MERGE to avoid outer and inner inline views to be combined into a single query;
  - NO_REWRITE to avoid query rewrite in favor of materialized views.

Materialized Views

- MVIEWs are persistent views stored in user schema.

- MVIEW could replace other denormalized tuning techniques:
  - computed tables (denormalized tables containing computed fields);
  - pre-joined tables (denormalized tables containing a pre-fetch JOIN operation or multi-JOIN operations result).

- MVIEW main benefit:
  - updating mechanism could be delegated to database engine instead of using computed-tables with user-defined update-triggers.

Materialized Views

- Server configuration:
  - parameters:
    - *query_rewrite_enabled = true*
    - *query_rewrite_integrity = enforced*
    - *optimizer_mode = ALL_ROWS*
  - user privileges to create and enable MVIEW features:
    - CREATE MATERIALIZED VIEW
    - QUERY REWRITE

Materialized Views

- MVIEW invocation from within query execution plan could be done:
  - by default if *query_rewrite_enabled* is set accordingly:
    - ALTER SESSION | SYSTEM query_rewrite_enabled = true;
  - explicitly– by using REWRITE hint:
    - SELECT /*+ REWRITE(factem_vm) */ ...

Materialized Views

- **MVIEWs content UPDATE could be done:**
  - manually by execution of the admin procedure DBMS_MVIEW.***refresh()***:
    - using COMPLETE mode (if the MVIEW object was created with REFRESH COMPLETE option);
    - using the incremental mode (if the MVIEW object was created with REFRESH FAST option);
  - *automatically*:
    - on each transactional commit,
      - if the MVIEW object was created with REFRESH ON COMMIT option;
      - If there are define MVIEW LOGS on base tables;
    - by using DBMS_JOBS scheduler.

Materialized Views

- MVIEWs content UPDATE will be done by using some specific back-reference mechanisms (back reference to original ROWS within view base tables) as:
  - *ROWID-s* (WITH ROWID clause) - that:
    - has to be included in MVIEW structure;
    - will be included in some specific structures named MVIEW LOGS (CREATE MATERIALIZED VIEW LOG WITH ROWID);
  - *Primary keys* (WITH PRIMARY KEY clause) - that:
    - has to be included in MVIEW structure definition;
    - will be included in MVIEW logs structures (CREATE MATERIALIZED VIEW LOG WITH PRIMARY KEY).

Parallel Queries

- Parallel options could be used to multiply reading operations on multiple server-process threads.
- Parallel options could be efficient:
  - if there are multiple processing units on server;
  - database storage space and physical database design (see partitioning options) took into consideration the use of a multiple disk architecture.
- Parallel options could be enable by using some specific hints:
  - /*+ PARALLEL(table_name parallel_factor) */ hint on tables;
  - /*+ PARALLEL(table_name (index_key_definition) parallel_factor) */ hint on indexes;
- Parallel operations:
  - PX_COORDINATOR;
    - PX_SEND_QC
      - PX_BLOCK_ITERATOR
      - PX_PARTITION_RANGE_ITERATOR

Practice C6_P3

Steps	Notes	SQL Script
1. Query REWRITE: subquery refactoring	LOAD AS SELECT	C6_P3.1.SubQuery_Refact_and_GTT_Tunning .sql
2. Using temporary tables	CREATE TEMPORARY TABLE	C6_P3.1.SubQuery_Refact_and_GTT_Tunning .sql
3. MVIEW refresh forced-COMPLETE	CREATE MATERIALIZED VIEW MAT VIEW ACCESS FULL	C6_P3.2.MVIEW_Tunning.sql
4. MVIEW refresh FAST	CREATE MATERIALIZED VIEW LOG CREATE MATERIALIZED VIEW	C6_P3.2.MVIEW_Tunning.sql

References

- Craig S. Mullins, *Database Administration The Complete Guide to DBA Practices and Procedures* Second Edition, Addison-Wesley, 2013
- Tony Hasler, *Expert Oracle SQL Optimization, Deployment, and Statistics*, Apress, 2014
- Christian Antognini, *Troubleshooting Oracle Performance*, Apress, 2014
- Donald Burleson, *Oracle High-Performance SQL Tuning* 1st Edition, McGraw-Hill Education; 1 edition (August 17, 2001)

References

- Lahdenmaki, Tapio, Leach, Michael, *Relational database index design and optimizers: DB2, Oracle, SQL server et al*, John Wiley & Sons, 2005
- Harrison, Guy, *Oracle performance survival guide: a systematic approach to database optimization,* Prentice Hall, 2009
- Allen, Grant, Bryla, Bob, Kuhn, Darl, *Oracle SQL Recipes*: *A Problem-Solution Approach*, Apress, 2009
- Caffrey, Mellanie et.al. *Expert Oracle Practices: Oracle Database Administration from the Oak Table,* Apress, 2010