DATABASE ADMINISTRATION

T2: DATABASE OPTIMIZATION [C4]

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

1. Physical Design: Optimization Objectives

- Logical and Conceptual Design Objectives:
  - *Data rationalizing:*
    - concerning storage: redundancy control;
    - concerning update mode: avoiding INSERT/UPDATE/DELETE anomalies;
  - Data integrity and consistency:
    - clear, complete and coherent domain model;
    - enforce integrity constraints (business rules);
  - *Data availability (data completeness)*:
    - user-views integration within a global comprehensible conceptual schema.
- Final target: producing ***an independent logical model.***

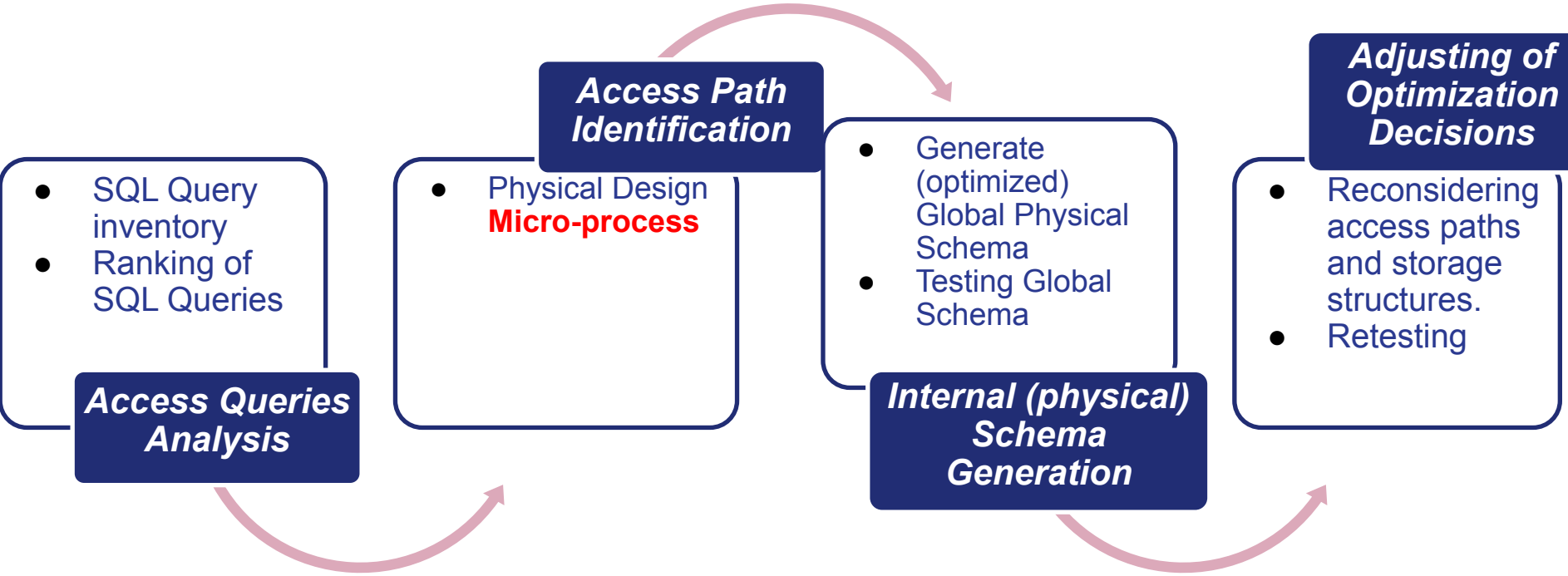
Physical design objectives

- Data access *performance* by:
  - optimizing access time (reducing access time);
  - optimizing resource cost (reducing costs like storage and CPU time).
- Data storage *rationalizing* by:
  - effective use of storage space usage;
  - effective use of internal memory (active space) usage.
- Data *availability* by:
  - minimizing downtimes;
  - minimizing unavailability of data amounts (or temporary unavailability);
  - minimizing data loss.
- Final target: ***an optimized internal data model.***

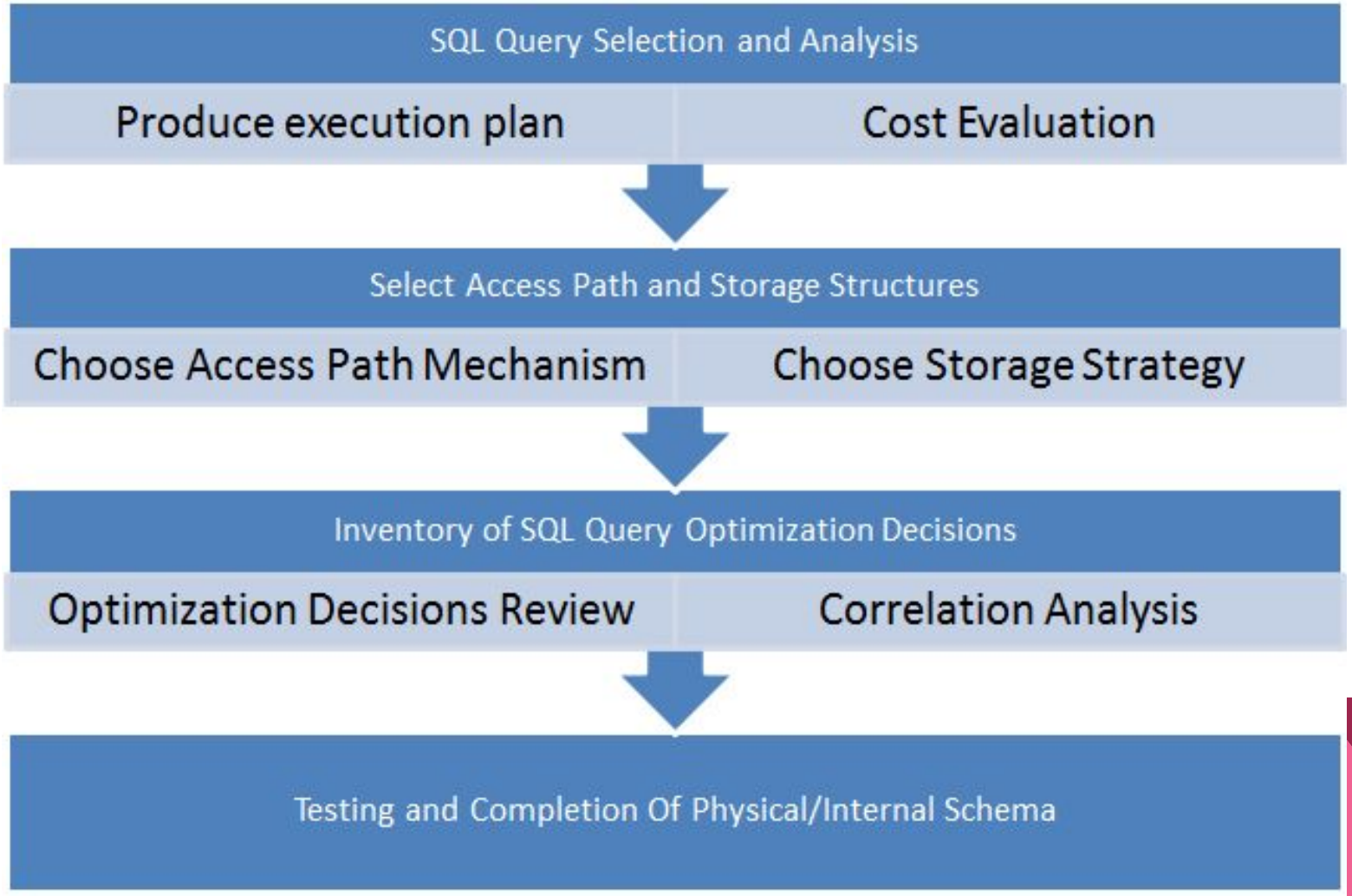
2. Optimizing: Physical Design Process for performance goals

- Two integrated perspectives:
  - *Global view* to integrate end-to-end the whole physical process: the incremental *macro-process* having as main *objective*:
    - producing an integrated and *optimized internal schema* by physical adjusting of logical schema structures.
  - *Granular focused view*: the micro-process of individual access query evaluation having as main objective:
    - producing an *optimized execution plan* of each individual SQL query by making the fundamental *decisions* concerning selecting and implementing the *data storage structures* and concerning selecting *data access strategies*.

Physical design: MACRO-PROCESS



Physical Design: Micro-process



3. Execution Plan Tool

- The Execution-plan is the main tool to analyze SQL processing flows in order to make physical design decisions: changing storage types and parameters or changing access paths.
- The Execution plan could be:
  - *inferred* taking into consideration the relational algebra operations: this way we could be generated *the estimated execution plan*;
  - *acquired* from DBMS context (the plan produced by DBMS) that will take into consideration the actual physical schema: this way will be generated *the actual execution plan*.

Execution plan: operations from relational algebra

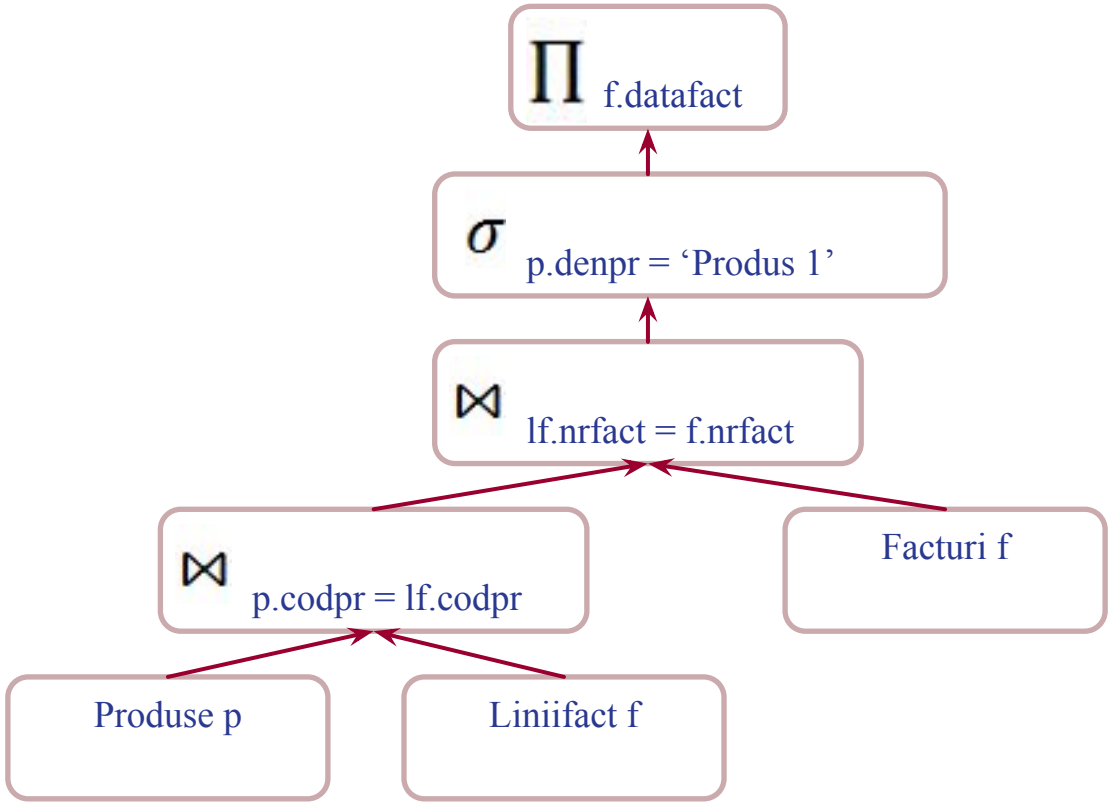
Operations	Symbols
Selection	σ
Projection	Π
Join	$\bowtie$
Division	
Union	$\cup$
Intersection	$\cap$
Difference	--
Cartesian Product	$\otimes$

*

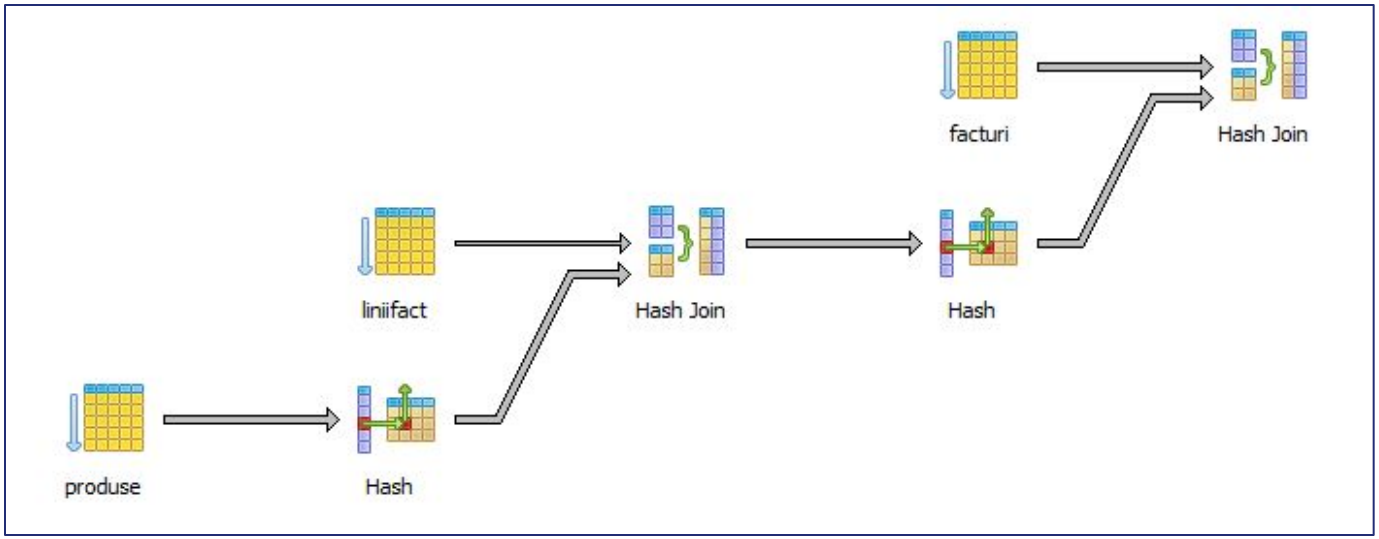
Estimated Execution Plan

```
SELECT f.datafact
FROM Produse p INNER JOIN Liniifact lf ON p.codpr = lf.codpr
INNER JOIN Facturi f ON lf.nrfact = f.nrfact
WHERE p.denpr = 'Produs 1';
```

Relational Algebra Operations
R1 ← JOIN(Produse, Liniifact; codpr)
R2 ← JOIN(R1, Facturi; nrfact)
R3 ← SELECTION(R2; denpr = ‘Produs 1’)
R ← PROJECTION(R3; datafact)



The Actual Execution Plan: PostgreSQL vs Oracle



OPERATION	OBJECT_NAME	OPTIONS	COST
SELECT STATEMENT			10
HASH JOIN			10
Access Predicates			
AND			
LF.CODPR=P.CODPR			
LF.NRFACT=F.NRFACT			
MERGE JOIN		CARTESIAN	6
TABLE ACCESS	PRODUSE	FULL	3
Filter Predicates			
P.DENPR='Produs 1'			
BUFFER		SORT	3
TABLE ACCESS	FACTURI	FULL	3
TABLE ACCESS	LINIIFACT	FULL	3
Filter Predicates			
LF.CODPR>0			

*

OEP: Oracle Execution Plan

- What is the OEP?
- What is the Optimizer?
- What is the Runtime?
- What are the optimizer hints?

Displaying the execution plan. Tools

- Data Dictionary Dynamic Views:
  - V$PLAN_TABLE
  - VSQL, VSQL_PLAN, VSQL_AREA, VSESSION_HISTORY
- DBMS packages:
  - DBMS_XPLAN.display(format_prms)
  - DBMS_XPLAN.display_cursor(format_prms)
  - DBMS_XPLAN.display_awr(format_prms)
- SQL Autotrace Tool
- SQLDeveloper tools
  - [SQLWorksheet → Explain plan…F10]
- SQL Tune HTML Report
  - DBMS_SQLTUNE.report_sql_monitor()

Practice C4_1

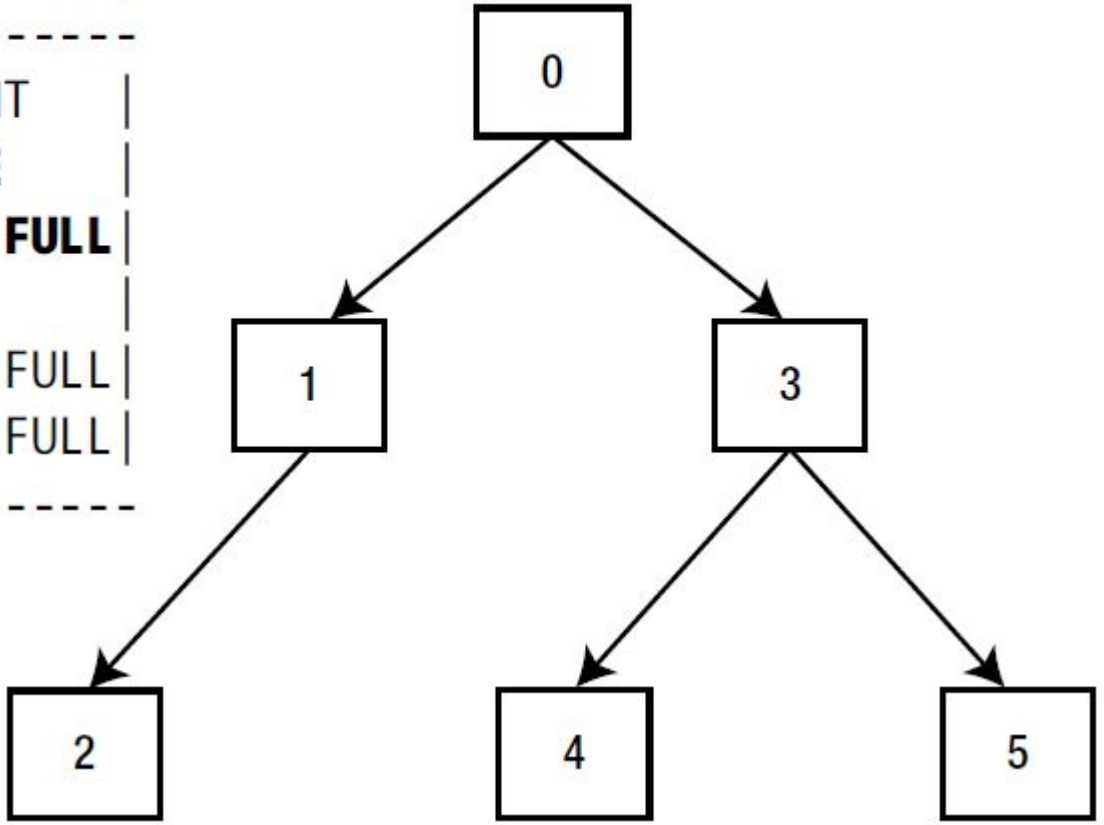
Steps	Notes	SQL Scripts
1. Get execution plan from V$SQL_PLAN	SELECT source SELECT sql_id FROM v$sqlarea SELECT v$sql_plan	C4_P1.3.Get_SQL_Explain.sql
2. Get execution plan from PLAN_TABLE	EXPLAIN PLAN FOR SELECT SELECT DBMS_EXPLAIN	C4_P1.3.Get_SQL_Explain.sql
3. Get execution plan with SQL TRACE	SET AUTOTRACE ON SELECT source -- check Script output	C4_P1.3.Get_SQL_Explain.sql
4. Get execution plan as HTML report	SELECT /*+monitor*/ source SELECT sql_id FROM v$sql_monitor DBMS_SQLTUNE.report_sql_monitor	C4_P1.3.Get_SQL_Explain.sql
5. Get execution plan from SQL Worksheet toolbar	→ F10: Explain plan… → F6: Autotrace...	

Interpreting OEP: operation-tree reading rules

- OEP hierarchy:
  - each inner (child) node is subordinate to a parent node [id → parent_id];
  - each parent node subordinates one or many child nodes within a determined hierarchy where data objects from parent nodes result from processing operation of inner nodes;
- OEP reading rules:
  1. Start from the first most indented node (vertically) - the most indented to the right;
     - if there are multiple indented nodes at the same level, select the uppermost one.
  2. Continue (down) with the inner nodes covered by the same parent node.
  3. Continue (up) with the parent node (after looping on all child nodes).
  4. Continue (down) with the next parent on the same (horizontal) level as the last parent node processed.
  5. Continue with the next horizontal level, repeating 1, 2, 3, 4 step sequence.

OEP: operation tree

Id	Operation
0	SELECT STATEMENT
1	**SORT AGGREGATE**
* **2**	**TABLE ACCESS FULL**
* 3	HASH JOIN
4	TABLE ACCESS FULL
5	TABLE ACCESS FULL



OEP Operation Types

- Table access operations.
- Index access operations.
- JOIN operations.
- SORT operations.
- Aggregation operations.
- VIEWs operations.

OEP: Table access operations and hints

- TABLE ACCESS FULL|HASH|CLUSTER
  - `full(table)`
  - `hash(table)`
  - `cluster(table)`
  - `no_index(table)`
- TABLE ACCESS BY USER ROWID
  - `rowid(table)`
- TABLE ACCESS BY INDEX ROWID
  - `index(table index)`

OEP: Index access operations and hits

- INDEX - UNIQUE SCAN <row>
  - `index(table index)`
- INDEX - RANGE SCAN <row>
- INDEX - FULL SCAN <row>
  - `index_ffs(table index)`
- BITMAP INDEX – SINGLE VALUE
- BITMAP INDEX – RANGE SCAN
- BITMAP INDEX – FULL SCAN
- BITMAP CONVERSION
- INDEX JOIN
  - `index_join(table index)`

OEP: JOIN operations and hints

- NESTED LOOPS <row>
  - `use_nl(tables)`
- HASH JOIN <set>
  - `use_hash(tables)`
- SORT-MERGE JOIN <set>
  - `use_merge(tables)`

OEP: Sort Operations

- Sort operations:
  - SORT AGGREGATE
  - SORT GROUP BY
  - SORT ORDER BY
- Other operations
  - HASH GROUP BY
  - VIEW
    - `merge, no_merge`
  - UNION
  - MINUS
  - INTERSECTION

Practice C4_P2

Steps	Notes	SQL Scripts
1. Generate SQL Object statistics	DBMS_STATS	C4_P2.1.OEP_Operations_and_Hints.sql
2. Check Execution Plan instances: table access	TABLE ACCESS	C4_P2.1.OEP_Operations_and_Hints.sql
3. Check Execution Plan instances: index access	INDEX ACCESS	C4_P2.1.OEP_Operations_and_Hints.sql
4. Check Execution Plan instances: table joins	JOINS	C4_P2.1.OEP_Operations_and_Hints.sql
5. Check Execution Plan instances: sorting and aggregating	SORT and HASH	C4_P2.1.OEP_Operations_and_Hints.sql
6. Check Execution Plan instances: view and view transformation	VIEW Query Refactored	C4_P2.1.OEP_Operations_and_Hints.sql

4. Oracle Execution Plan Scopes: CBO

- Optimizer Types/Modes:
  - RBO Rule-Based optimizer (deprecated) – ranks access rules by using generic criteria, but not taking into account database physical state
  - CBO Cost-Based Optimizer (default) – ponders access rules' ranking by taking into account the statistics concerning database physical state.

CBO: Cost Based Optimizer Goals

- CBO:
  - computes OEP operation effectiveness relative to physical parameters (SQL object stats) from data dictionary.
- Goals: FIRST_ROWS and ALL_ROWS
  - as (session parameters):
    - ALTER SESSION SET OPTIMIZER_MODE = first_row;
  - as (SQL phrase) hints:
    - SELECT /*+ first_row*/
- Object statistics and schema analysis:
  - ANALYZE TABLE … COMPUTE STATISTICS;
  - DBMS_UTILITY.ANALYZE_SCHEMA('VINZARI01', 'COMPUTE')
  - DBMS_STATS.GATHER_SCHEMA_STATS
  - ALTER SESSION SET statistics_level = TYPICAL | ALL;

COST BASED OPTIMIZER

- The cost-based optimizer features:
  - makes access decisions by taking into consideration data object stats from data dictionary;
  - generates an optimized execution plan by making some performance (cost) computations on actual number of rows and column selectivity.
- Statistics are produced and saved in data dictionary
  - by using ANALYZE commands:
  - By using DBMS package:
    - DBMS_UTILITY.ANALYZE_SCHEMA
    - DBMS_STATS.GATHER_TABLE_STATS
    - DBMS_STATS.GATHER_INDEX_STATS

```
ANALYZE TABLE table COMPUTE STATISTICS;
ANALYZE TABLE table ESTIMATE STATISTICS  SAMPLE 10 PERCENT;
ANALYZE TABLE table ESTIMATE STATISTICS  SAMPLE 1000 ROWS;
DBMS_UTILITY.ANALYZE_SCHEMA('VINZARI01', 'COMPUTE');
DBMS_STATS.GATHER_TABLE_STATS ( OWNNAME=>'VINZARI01', TABNAME=>'CLIENTI');
DBMS_STATS.GATHER_INDEX_STATS ( OWNNAME=>'VINZARI01', INDNAME=>'UK_DENCL');
```

Choosing Optimizer Mode

- ***Set OPTIMIZER_MODE Initialization Parameter with one of the values:***
  - RULE – to enable RBO;
  - CHOOSE – to enable CBO if statistics are already present in the data dictionary, or setting a more specific CBO scopes by one of the values:
    - FIRST_ROWS
    - ALL_ROWS

- ***Set OPTIMIZER_MODE at session level:***

```
ALTER SESSION SET OPTIMIZER_MODE = {CHOOSE | RULE}
```

- ***Set OPTIMIZER_MODE at call/statement level:***

```
select /*+RULE*/ codcl, nrcom, data from comenzi
where nrcom = 610;
```

CBO: Cost Based Optimizer
Goals and HINTS

/*+ HINT */	Scope
FIRST_ROWS_n	Use CBO to return first *n* rows as soon as possible.
ALL_ROWS	Use CBO to return the entire rowset as soon as possible.
CHOOSE	Use CBO as ALL_ROWS by default in the presence of statistics in data dictionary.
RULE	Force RBO usage.

Practice C4_P3

Steps	Notes	SQL Scripts
1. SET CBO Session Goal	ALTER SESSION SET optimizer_mode SELECT FROM v$parameter	C4_P3.1.CBO_Goals.sql
2. Set CBO Statement Goal	SELECT /*+ first_row*/ FROM source SELECT optimizer_mode FROM v$sql	C4_P3.1.CBO_Goals.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