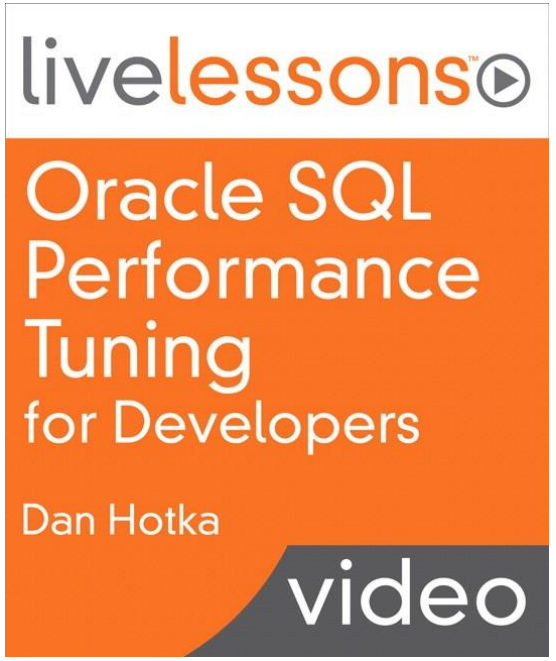

Pearson Webcast Series
informIT®
the trusted technology learning source
PEARSON

Understanding Oracle Explain Plans

Presented by **Dan Hotka**, Oracle ACE Director



Brought to you by InformIT – informit.com/webcasts

InformIT is the online presence of the family of information technology publishers and brands of Pearson

informIT

About Us

Pearson

We are the world's largest education company, with 40,000 employees in more than 70 countries helping people of all ages to make measurable progress in their lives. We have a simple mission: to help people make more of their lives through learning.

InformIT

We are an online community created by the authors and team members behind trusted Pearson learning brands...and you! Our passion is delivering quality content and resources from the creators, innovators, and leaders of technology.

Welcome. Our Agenda Today.

- Explain Plan Tools
  - Xplan.display
  - JS Tuner
- Understanding Explain Plans
  - Reading the Explain Plan
  - What does it mean
- Q&A
- Wrap-up & Resources

Explain Plan Tools

Explain Plan Tools

- DBMS_XPLAN.DISPLAY
  - Available with Oracle 8.1.7+
  - Comes with Database
  - Used via SQL*Plus
- JS Tuner
  - Available for Oracle8+
  - Download from www.DanHotka.com
  - Executable Jar File

Explain Plan Tools

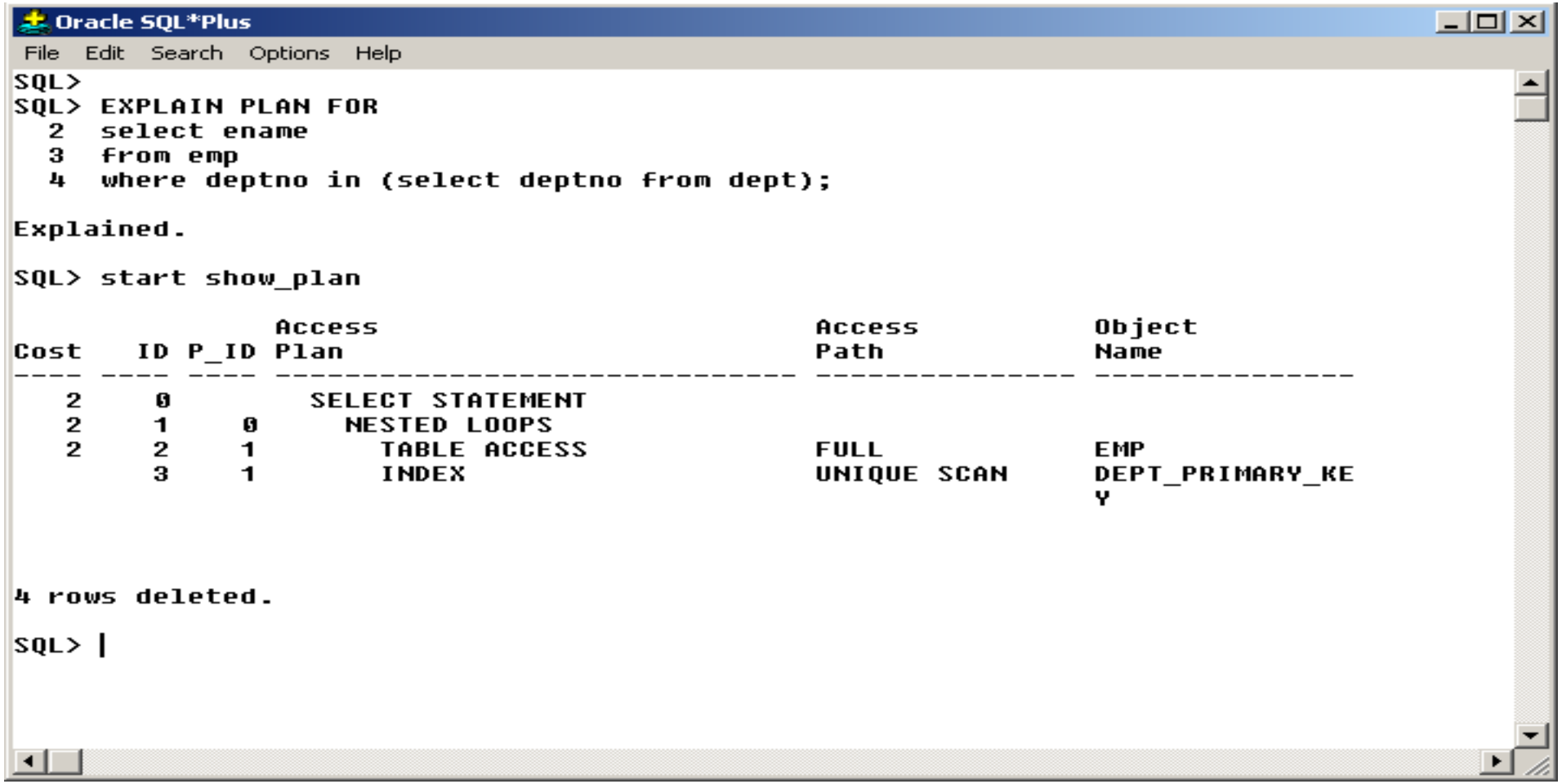
- PLAN_TABLE
  - Oracle10g+ - use PLAN_TABLE$
    - Owned by SYS
    - Have DBA make public synonym to: PLAN_TABLE
    - Make sure to drop your PLAN_TABLE
  - Use the ‘explain plan for’ syntax to populate this table
  - Use tools to populate this table (TOAD, SQL Developer)
  - Use SHOW_PLAN.sql to display contents
    - Available on my website

Using the Tools

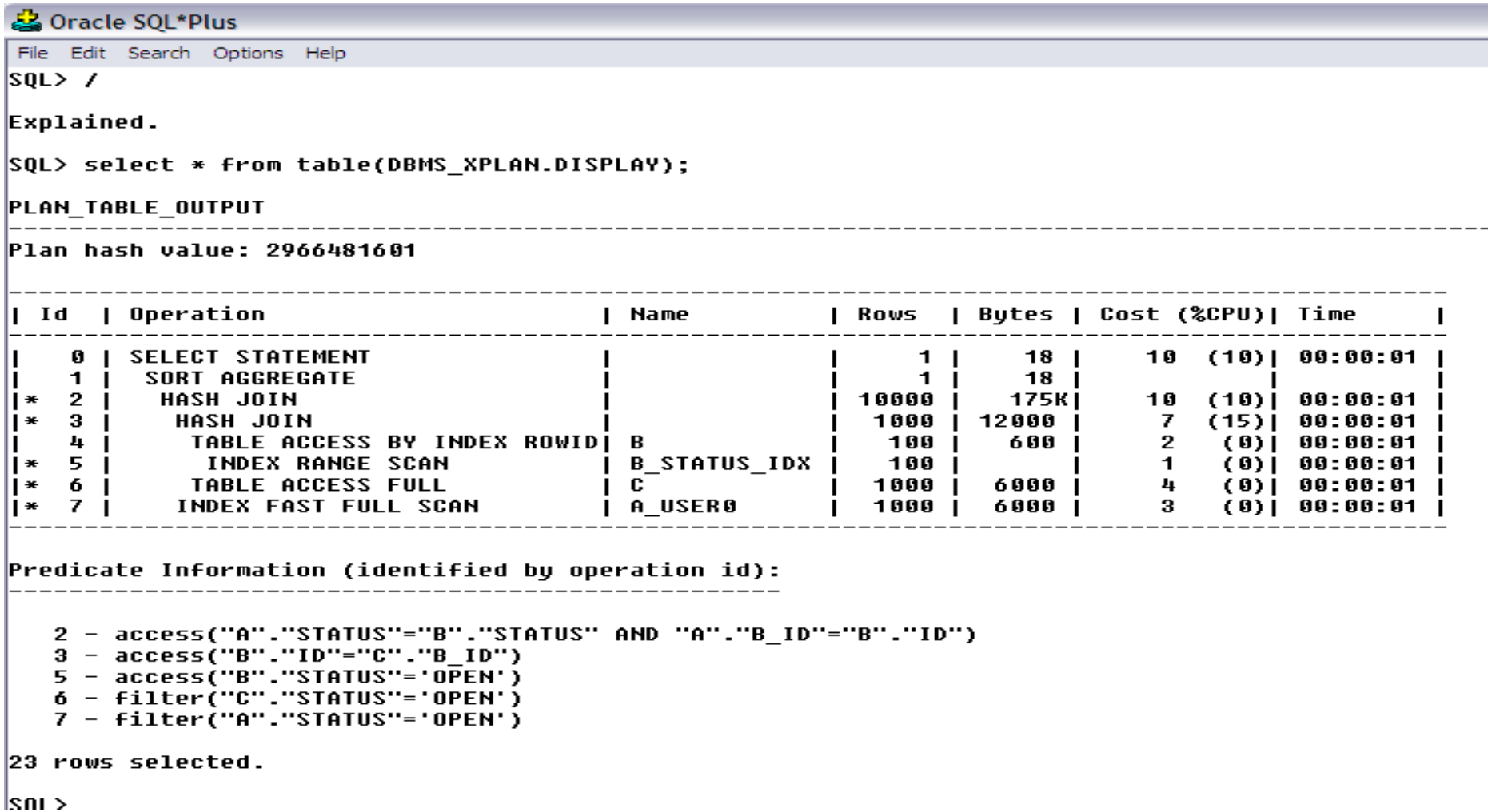
- All tools use the PLAN_TABLE
  - Oracle10g+
    - Automatically Created
  - Migration:
    - Use PLAN_TABLE$
    - Autotrace will give warning if not using correct PLAN_TABLE

Column	Pk	Data Type
STATEMENT_ID		VARCHAR2 (30)
TIMESTAMP		DATE
REMARKS		VARCHAR2 (80)
OPERATION		VARCHAR2 (30)
OPTIONS		VARCHAR2 (255)
OBJECT_NODE		VARCHAR2 (128)
OBJECT_OWNER		VARCHAR2 (30)
OBJECT_NAME		VARCHAR2 (30)
OBJECT_INSTANCE		NUMBER
OBJECT_TYPE		VARCHAR2 (30)
OPTIMIZER		VARCHAR2 (255)
SEARCH_COLUMNS		NUMBER
ID		NUMBER
PARENT_ID		NUMBER
POSITION		NUMBER
COST		NUMBER
CARDINALITY		NUMBER
BYTES		NUMBER
OTHER_TAG		VARCHAR2 (255)
PARTITION_START		VARCHAR2 (255)
PARTITION_STOP		VARCHAR2 (255)
PARTITION_ID		NUMBER
OTHER		LONG
DISTRIBUTION		VARCHAR2 (30)
CPU_COST		NUMBER
IO_COST		NUMBER
TEMP_SPACE		NUMBER
ACCESS_PREDICATES		VARCHAR2 (4000)
FILTER_PREDICATES		VARCHAR2 (4000)

Explain Plan Tools



Explain Plan Tools



DBMS_XPLAN

Oracle SQL*Plus

File Edit Search Options Help

```
  1  SELECT /*+ GATHER_PLAN_STATISTICS */ count(*)
  2  from A, B, C
  3  WHERE A.STATUS = B.STATUS
  4  AND   A.B_ID = B.ID
  5  AND   B.STATUS = 'OPEN'
  6  AND   B.ID = C.B_ID
  7* AND   C.STATUS = 'OPEN'
SQL> /

  COUNT(*)
----------
     10000

SQL> Select plan_table_output
  2  From table(dbms_xplan.display_cursor(FORMAT=>'ALLSTATS LAST'));

PLAN_TABLE_OUTPUT
------------------------------------------------------------------------------------------------------------------------
SQL_ID  49s63xqn62buu, child number 0
-------------------------------------
SELECT /*+ GATHER_PLAN_STATISTICS */ count(*) from A, B, C WHERE
A.STATUS = B.STATUS AND   A.B_ID = B.ID AND   B.STATUS = 'OPEN' AND
B.ID = C.B_ID AND   C.STATUS = 'OPEN'

Plan hash value: 2966481601

------------------------------------------------------------------------------------------------------------------------
| Id  | Operation                     | Name          | Starts | E-Rows | A-Rows |   A-Time    | Buffers |  OMem |  1Mem | Used-Mem |
------------------------------------------------------------------------------------------------------------------------
|   1 |  SORT AGGREGATE               |               |      1 |      1 |      1 |00:00:00.01 |      15 |       |       |          |
|*  2 |   HASH JOIN                   |               |      1 |  10000 |  10000 |00:00:00.02 |      15 |   935K|   935K| 1171K (0)|
|*  3 |    HASH JOIN                  |               |      1 |   1000 |   1000 |00:00:00.01 |       9 |   935K|   935K| 1162K (0)|
|   4 |     TABLE ACCESS BY INDEX ROWID| B             |      1 |    100 |    100 |00:00:00.01 |       2 |       |       |          |
|*  5 |      INDEX RANGE SCAN         | B_STATUS_IDX  |      1 |    100 |    100 |00:00:00.01 |       1 |       |       |          |
|*  6 |     TABLE ACCESS FULL         | C             |      1 |   1000 |   1000 |00:00:00.01 |       7 |       |       |          |
|*  7 |    INDEX FAST FULL SCAN       | A_USER0       |      1 |   1000 |   1000 |00:00:00.01 |       6 |       |       |          |
------------------------------------------------------------------------------------------------------------------------
```

Explain Plan Tools

- JSTuner
  - Incorporates SQL Scripts with enhanced Trace
  - Index Info
  - Includes index statistics
  - Enhanced Explain Plan
    - Works with V8+
    - Works with rule-based optimizer!
- Available from www.DanHotka.com

JS Tuner Console

user0/*****@oraxp9i

SQL | Query Results | Explain Plan | Statistics | Enhanced Explain Plan

ID	P_ID	Access Plan	Access Path	Object Name	Where Clause
0		SELECT STATEMENT			
1	0	SORT	AGGREGATE		
2	1	TABLE ACCESS	BY INDEX ROWID	C	C.STATUS='OPEN'
3	2	NESTED LOOPS			
4	3	NESTED LOOPS			
5	4	TABLE ACCESS	BY INDEX ROWID	B	
6	5	INDEX	RANGE SCAN	B_STATUS_IDX	B.STATUS='OPEN'
7	4	TABLE ACCESS	BY INDEX ROWID	A	A.B_ID=B.ID
8	7	INDEX	RANGE SCAN	A_STATUS_IDX	A.STATUS=B.STATUS
9	3	INDEX	RANGE SCAN	C_B_ID_IDX	B.ID=C.B_ID

Login | Run | Explain | Desc | Index | Clear | Open | Save | Help | Hints | History | Exit

JS Tuner Console

user0/*****@robinxp

SQL | Query Results | Explain Plan | Statistics | Enhanced Explain Plan

```
-------------------------------------
SELECT /*+ GATHER_PLAN_STATISTICS */ count(*) from A, B, C WHERE
A.STATUS = B.STATUS AND   A.B_ID = B.ID AND   B.STATUS = 'OPEN' AND
B.ID = C.B_ID AND   C.STATUS = 'OPEN'

Plan hash value: 1523248325

-----------------------------------------------------------------------------------------------
| Id  | Operation                          | Name          | Starts | E-Rows | A-Rows |   A-Time
-----------------------------------------------------------------------------------------------

PLAN_TABLE_OUTPUT
-----------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                   |               |      1 |        |      1 |00:00:00.0
|   1 |  SORT AGGREGATE                    |               |      1 |      1 |      1 |00:00:00.0
|*  2 |   HASH JOIN                        |               |      1 |  10000 |  10000 |00:00:00.0
|*  3 |    TABLE ACCESS FULL               | C             |      1 |   1000 |   1000 |00:00:00.0
|*  4 |    HASH JOIN                       |               |      1 |   1000 |   1000 |00:00:00.0
|   5 |     TABLE ACCESS BY INDEX ROWID| B             |      1 |    100 |    100 |00:00:00.0
|*  6 |      INDEX RANGE SCAN              | B_STATUS_IDX  |      1 |    100 |    100 |00:00:00.0
```

Login | Run | Explain | Desc | Index | Clear | Open | Save | Help | Hints | History | Exit

Hints

Oracle SQL*Plus

File Edit Search Options Help

```
SQL>
SQL> EXPLAIN PLAN FOR
  2  select ename, job, sal
  3  from emp
  4  where job in (select /*+ QB_NAME(EMP1_SUBQUERY) */ job from emp1 where deptno = 20)
  5  and sal > (select /*+ QB_NAME(EMP2_SUBQUERY) */ avg(sal) from emp2 where deptno = 20)
  6  /

Explained.

SQL> start SHOW_PLAN_QB

                Access                          Access        Object
Cost    ID P_ID Plan                            Path          Name          QB Name
---- ---- ---- ------------------------------ ------------ ------------ -------------
  10    0        SELECT STATEMENT
   7    1    0     HASH JOIN                  SEMI
   3    2    1       TABLE ACCESS             FULL          EMP
        3    2         SORT                   AGGREGATE                   EMP2_SUBQUERY
   3    4    3           TABLE ACCESS         FULL          EMP2          EMP2_SUBQUERY
   3    5    1       TABLE ACCESS             FULL          EMP1

6 rows selected.

SQL>
SQL>
```

JS Tuner Console

user0/*****@robinxp

SQL | Query Results | Explain Plan | Statistics | Enhanced Explain Plan

Sub Query	Access Plan	Access Path	Object Name	Where Clause
	SELECT STATEMENT			
	HASH JOIN	SEMI		JOB=JOB
	TABLE ACCESS	FULL	EMP	SAL> (SELECT /*+ Q
				2) */ AVG(SAL) FROM
				ERE DEPTNO=20)
EMP2	SORT	AGGREGATE		
EMP2	TABLE ACCESS	FULL	EMP	DEPTNO=20
	TABLE ACCESS	FULL	EMP	DEPTNO=20

Login | Run | Explain | Desc | Index | Clear | Open | Save | Help | Hints | History | Exit

Understanding Explain Plans

Reading Explain Plans

- Oracle:
  - Hard Parses the SQL
    - Checks SQL syntax
    - Checks for available indexes/stats
    - Reads from bottom to the top
  - Arrives at an Execution Plan
    - Decides how it will access the tables and indexes
  - Executes the SQL
    - Oracle9/10 – peeks once at bind variables
    - Oracle11 – tracks explain plans with various bind vars
      - Called ‘Adaptive Cursor Sharing’
    - Runs the Execution plan

Reading Explain Plans

- Explain Plan
  - Visualizes the execution plan
  - Uses the Plan_Table
    - Which varies slightly from release to release
    - Utlxplan.sql
    - <Oracle Home>/RDBMS/ADMIN
  - Used heavily to tune SQL

Reading Explain Plans

- How does it work?
  - Reads from bottom up
  - Syntax checks/tracks available indexes
  - ALL queries begin with a table access
  - Regular queries
    - generally does the table joins first
    - Utilizes WHERE clause predicates
  - Partitioned queries
    - generally accesses the partitions first utilizing WHERE clause predicates
    - Then performs the table joins
  - Sometimes the tables are again
    - Including remaining WHERE clause predicates

Reading Explain Plans

- How does it work?
  - RBO - follows rules for index selection
    - Arrives at an execution plan in single pass
  - CBO - tries a variety of combinations of table order/where clause predicates
    - Combinations called ‘permutations’
      - Generally # perms = # tables * # where predicates
    - Regular queries:
      - Uses lowest cost method
      - *** Queries in this course are regular queries unless otherwise noted
    - Partitioned queries (covered in separate section)
      - SQL that accesses 1 or more partitioned objects
      - Uses fastest access method (elapse time)

Reading Explain Plans

- RBO – produces explain plans in a single pass
  - Hard parsing was not the issue
  - Size of the library cache was the issue
- CBO – produces multiple iterations looking for the lowest cost
  - These iterations are called permutations and query transformation

Reading Explain Plans

- CBO – Brute Force
  - CBO is a code set
  - Gets a SQL
  - Returns Cost #'s and Execution Plans
  - Oracle passes next permutation to CBO
  - Oracle passes query rewrite to CBO
  - IS overall cost (cost # at line 0) < prior permutation?
    - True – tosses prior and this one becomes the one to beat
- Permutations
  - CBO tries various table combinations
  - CBO costs out the 3 join types
    - Nested Loops
    - Merge Join
    - Hash Join
  - CBO costs out the various where clause predicates associated with each table

Reading Explain Plans

- Query Transformations
  - Introduced in Oracle9
    - Complex View Merging
      - Converts views to joins
    - Subquery Unnesting
      - Converts subqueries to inline views
    - Join Predicate Push Down
      - Moves where clause predicates into subquery
  - -Hints for each item above
    - - No_Query_Transformation prevents this behavior
    - -Discussed later in this chapter

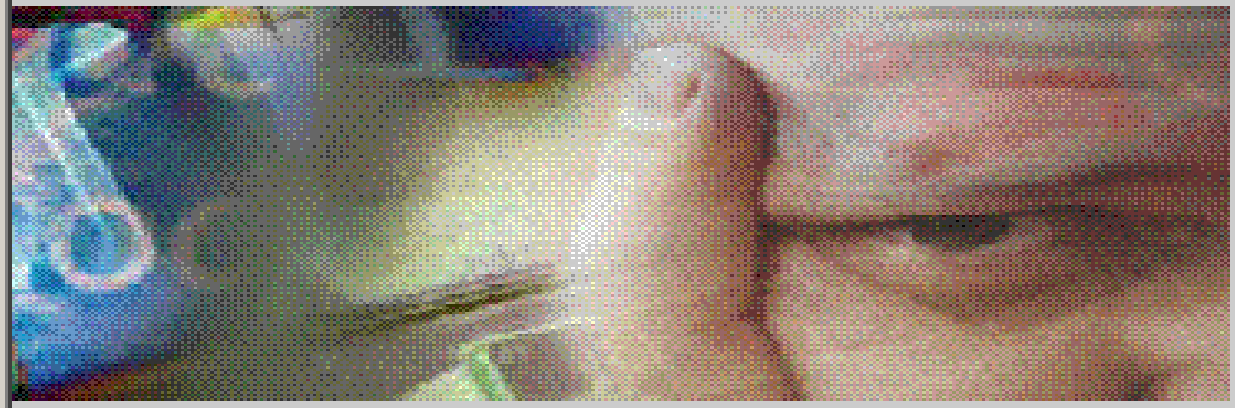
Reading Explain Plans

- Inner steps produce result sets
  - These ‘intermediate’ result sets are not visible
  - They are like temporary tables
  - They are passed to the next step in the execution plan
  - IF passed to a join step
    - They become the Outer Table
  - The result set at Step 0 is the final result set
    - This result set is then passed to the cursor area

Current SQL

```
SELECT count(*)
from A, B, C
WHERE A.STATUS = B.STATUS
AND   A.B_ID = B.ID
AND   B.STATUS = 'OPEN'
AND   B.ID = C.B_ID
AND   C.STATUS = 'OPEN';
```

Close



user0/*****@oraxp9i

Enhanced Explain Plan

Rows	Bytes	ID	P_ID	Access Plan	Access Path	Object Name
		0		SELECT STATEMENT		
		1	0	SORT	AGGREGATE	
		2	1	TABLE ACCESS	BY INDEX ROWID	A
		3	2	NESTED LOOPS		
		4	3	NESTED LOOPS		
		5	4	TABLE ACCESS	BY INDEX ROWID	B
		6	5	INDEX	RANGE SCAN	B_STATUS_IDX
		7	4	TABLE ACCESS	BY INDEX ROWID	C
		8	7	INDEX	RANGE SCAN	C_B_ID_IDX
		9	3	INDEX	RANGE SCAN	A_STATUS_IDX

Current SQL

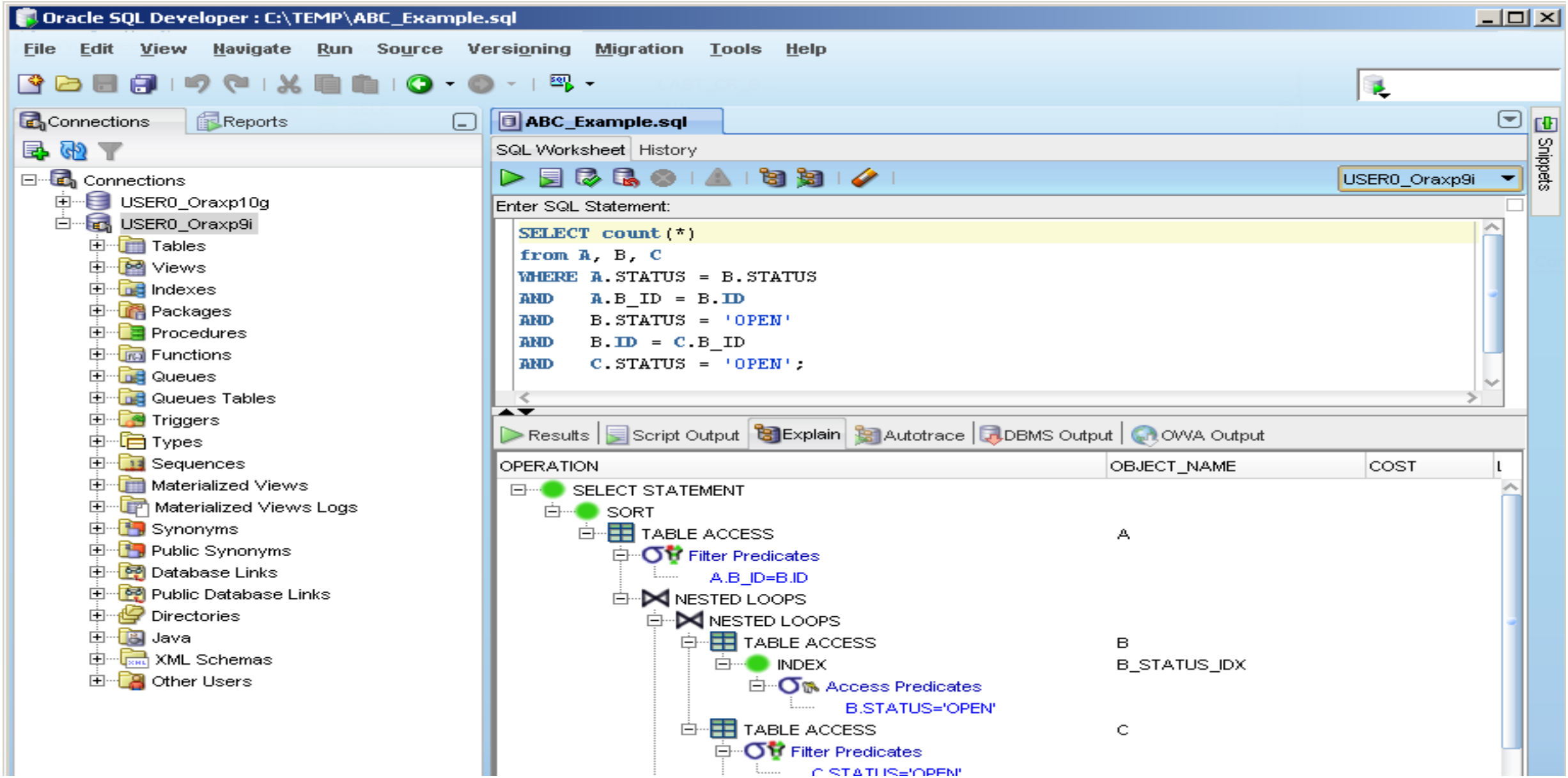
```
SELECT count(*)
from A, B, C
WHERE A.STATUS = B.STATUS
AND   A.B_ID = B.ID
AND   B.STATUS = 'OPEN'
AND   B.ID = C.B_ID
AND   C.STATUS = 'OPEN';
```

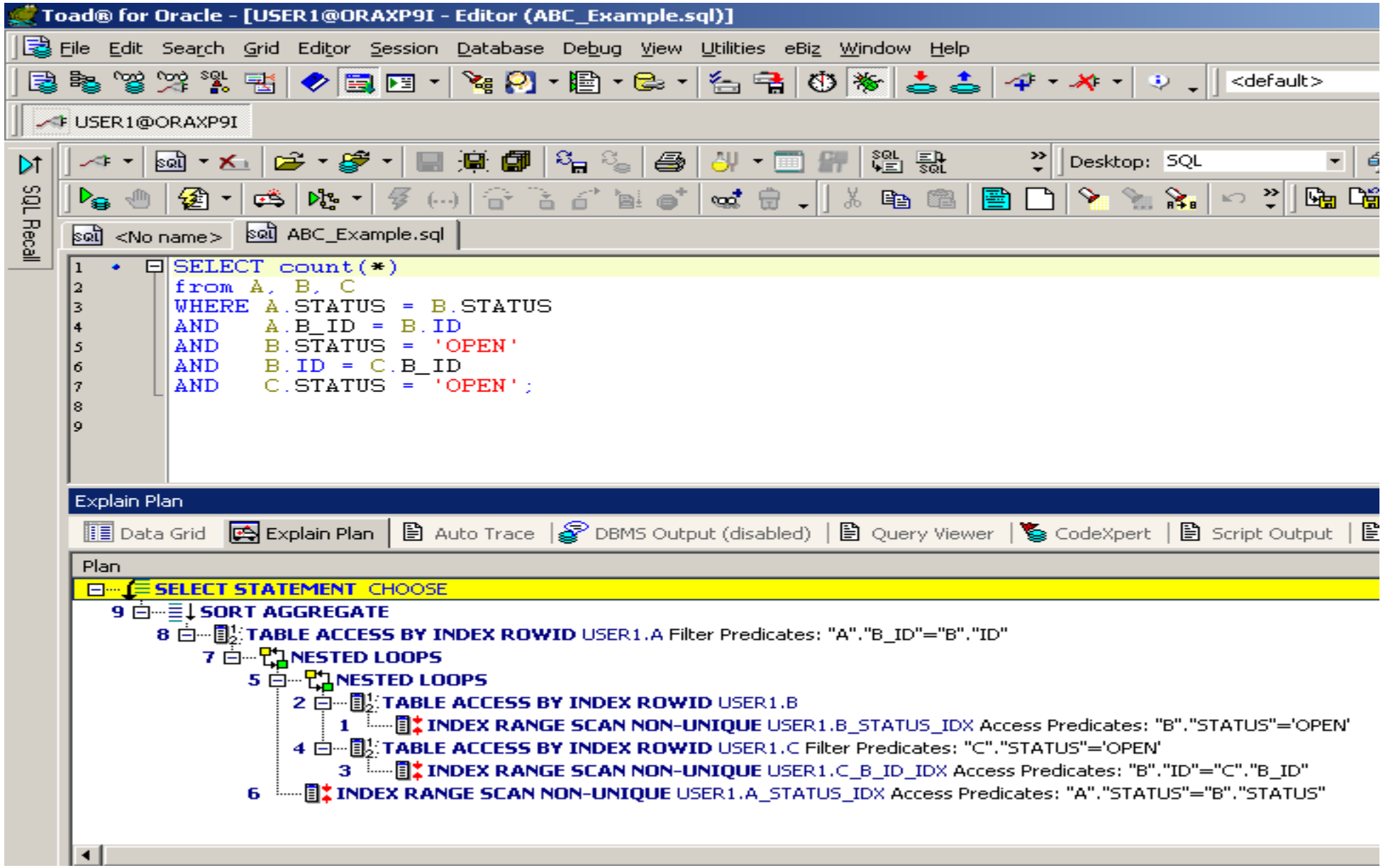
Close

user0/*****@oraxp9i

Enhanced Explain Plan

ID	P_ID	Access Plan	Access Path	Object Name	Where Clause
0		SELECT STATEMENT			
1	0	SORT	AGGREGATE		
2	1	TABLE ACCESS	BY INDEX ROWID	A	A.B_ID=B.ID
3	2	NESTED LOOPS			
4	3	NESTED LOOPS			
5	4	TABLE ACCESS	BY INDEX ROWID	B	
6	5	INDEX	RANGE SCAN	B_STATUS_IDX	B.STATUS='OPEN'
7	4	TABLE ACCESS	BY INDEX ROWID	C	C.STATUS='OPEN'
8	7	INDEX	RANGE SCAN	C_B_ID_IDX	B.ID=C.B_ID
9	3	INDEX	RANGE SCAN	A_STATUS_IDX	A.STATUS=B.STATUS


Oracle SQL Developer : C:\TEMP\ABC_Example.sql
File Edit View Navigate Run Source Versioning Migration Tools Help
Connections
Reports
Connections
USER0_Oraxp10g
USER0_Oraxp9i
Tables
Views
Indexes
Packages
Procedures
Functions
Queues
Queues Tables
Triggers
Types
Sequences
Materialized Views
Materialized Views Logs
Synonyms
Public Synonyms
Database Links
Public Database Links
Directories
Java
XML Schemas
Other Users
ABC_Example.sql
SQL Worksheet
History
USER0_Oraxp9i
Enter SQL Statement:
SELECT count(*)
from A, B, C
WHERE A.STATUS = B.STATUS
AND A.B_ID = B.ID
AND B.STATUS = 'OPEN'
AND B.ID = C.B_ID
AND C.STATUS = 'OPEN';
Results
Script Output
Explain
Autotrace
DBMS Output
OWA Output
OPERATION
OBJECT_NAME
COST
SELECT STATEMENT
SORT
TABLE ACCESS
A
Filter Predicates
A.B_ID=B.ID
NESTED LOOPS
NESTED LOOPS
TABLE ACCESS
B
INDEX
B_STATUS_IDX
Access Predicates
B.STATUS='OPEN'
TABLE ACCESS
C
Filter Predicates
Snippets


Toad® for Oracle - [USER1@ORAXP9I - Editor (ABC_Example.sql)]
File Edit Search Grid Editor Session Database Debug View Utilities eBiz Window Help
<default>
USER1@ORAXP9I
SQL Recall
Desktop: SQL
<No name>
ABC_Example.sql
1 SELECT count(*)
2 from A, B, C
3 WHERE A.STATUS = B.STATUS
4 AND A.B_ID = B.ID
5 AND B.STATUS = 'OPEN'
6 AND B.ID = C.B_ID
7 AND C.STATUS = 'OPEN';
8
9
Explain Plan
Data Grid
Explain Plan
Auto Trace
DBMS Output (disabled)
Query Viewer
CodeXpert
Script Output
Plan
SELECT STATEMENT CHOOSE
9 SORT AGGREGATE
8 TABLE ACCESS BY INDEX ROWID USER1.A Filter Predicates: "A"."B_ID"="B"."ID"
7 NESTED LOOPS
5 NESTED LOOPS
2 TABLE ACCESS BY INDEX ROWID USER1.B
1 INDEX RANGE SCAN NON-UNIQUE USER1.B_STATUS_IDX Access Predicates: "B"."STATUS"='OPEN'
4 TABLE ACCESS BY INDEX ROWID USER1.C Filter Predicates: "C"."STATUS"='OPEN'
3 INDEX RANGE SCAN NON-UNIQUE USER1.C_B_ID_IDX Access Predicates: "B"."ID"="C"."B_ID"
6 INDEX RANGE SCAN NON-UNIQUE USER1.A_STATUS_IDX Access Predicates: "A"."STATUS"="B"."STATUS"

Reading Explain Plans

- Access Rule Description
- AND-EQUAL Index values will be used to join rows.
- Access Predicate works with ROWID's, tyically from indexes.
- Filter Predicate FILTERs apply 'other criteria' in the query to further qualify the matching rows. The 'other criteria' include correlated sub queries, and HAVING clause.
- TABLE ACCESS When not associated with a join condition, they act like Filter…processing additional Where Clause predicates.
- VIEW OF Processed SQL from a view. **IF on a join condition, Oracle converted it to a view/subquery during query transformation
- INTERNAL_FUNCTION This typically means that you have a data type mismatch.

Reading Explain Plans

- Access Rule Description
- CONCATENATOIN statement has a union clause
- INDEX (UNIQUE) SQL statement utilized a unique index to search for a specific value.
- INDEX (RANGE SCAN) SQL statement contains a non-equality or BETWEEN condition.
- INLIST ITERATOR SQL statement has an IN clause, or, values being treated as an IN clause
- TABLE ACCESS (FULL) All rows are retrieved from the table without using an index.
- TABLE ACCESS (BY ROWID) A row was retrieved from a table based on the ROWID of the row.

Reading Explain Plans

- Access Rule Description
- HASH JOIN SQL statement initiated a hash-join operation.
- MERGE JOIN SQL statement references two or more tables, sorting the two result sets being joined over the join columns and then merging the results via the join columns.
- MERGE JOIN (CARTESIAN) SQL statement references two or more tables but without a joining column (generally not a good thing)
- NESTED LOOPS This operation is one form of joining tables, as opposed to a merge join. One row is retrieved from the row source identified by the first child operation, and then joined to all matching rows in the other table, identified in the second child operation.
- NONUNIQUE INDEX (RANGE SCAN) The RANGE SCAN option indicates that ORACLE expects to return multiple matches (ROWIDs) from the index search

Reading Explain Plans

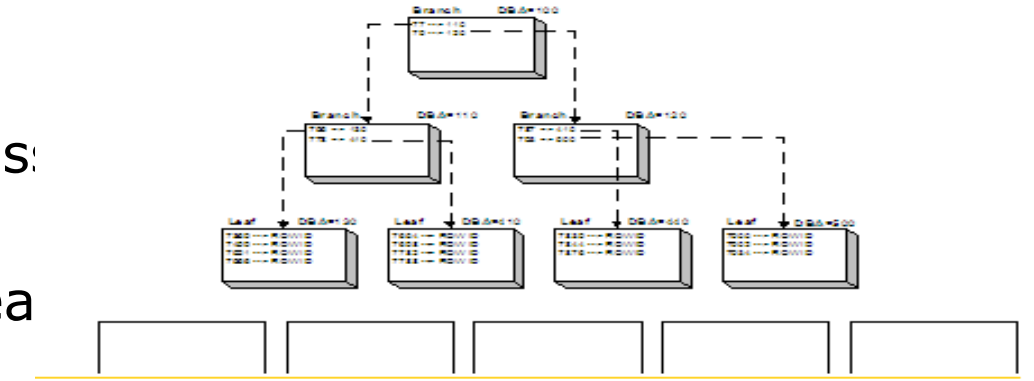
- Access Rule Description
- BITMAP CONVERSION Bitmap Index being merged
- BITMAP MERGE Generally used with Bitmap Range Scan
- BITMAP MINUS Bitmap Index handling a not = condition.
- BITMAP INDEX SINGLE VALUE Bitmap index being used for single value lookup.
- BITMAP INDEX (RANGE SCAN) Bitmap index being used for multiple value lookup.

Reading Explain Plans

- Access Rule Description
- PARTITIONING covered in another ppt…ask for it next time!
- SORT (ORDER BY) SQL statement contains an ORDER BY SORT (AGGREGATE) SQL statement initiated a sort to resolve a MIN or MAX type function.
- SORT (GROUP BY) SQL statement contains a GROUP BY

Reading Explain Plans

- Index Scans
  - **Unique Scan**
    - via root -> branch -> leaf for a single row acces
  - **Range Scan**
    - via root -> branch -> leaf for first row, then lea to leaf for remaining rows
  - **Full Scan (Index-Full)**
    - Scans leaf blocks using single-block access
  - **Fast Full Scan (Index-FFS)**
    - scans all index leaf blocks using multi-block read
  - **Index Skip Scan (Index-SS)**
    - Useful for multi-column indexes, accessing only 2nd column and first column has low cardinality



Reading Explain Plans

- Oracle only joins 2 tables at a time
- The smaller the initial result sets, the faster the whole query runs
  - Drive off the item that will eliminate the most rows first

Reading Explain Plans

- Nested Loops
  - Inner table looped for each row returned in outer table
  - Lg table should be outer
  - Sm table (or unique indexed lookup) should be inner
  - Rows returned to the result set that qualify the driving WHERE clause
  - Cost = outer access + (inner table access * outer cardinality)
- Merge Scan Join
  - Both tables are sorted
  - Rows are inserted into result set based on key value
  - THEN WHERE clause applied
  - Cost = outer access + inner access + sort costs
- Hash Join
  - Hashes join keys and caches object into a hash table
  - Driving table should be smaller of the 2
  - Cost = inner cost + (outer cost * inner cardinality/hash partitions)

Reading Explain Plans

- Nested Loop Join
  - driving table
  - Default order(rule)
- Merge Scan Join
  - sort & match
- Hash Join
  - Full scans with no sorts
  - Join column to row address

Reading Explain Plans

- Nested Loop Join
  - If join condition is ‘ANDed’, make a compound index on the inner table
  - Inner and Outer join column should have same data type
  - Outer Table: Larger of Result set
  - Inner Table: Smaller of Result set
  - Foreign key indexes helps CBO choose between nested loops and hash joins

Reading Explain Plans

- Merge Scan Join
  - large portion of rows are being joined
  - Both Tables have larger result sets
  - Helpful if using indexes on merged columns (rows returned in proper order)
- Hash Join
  - Smaller tables being joined on '=' condition
  - Outer Table: Smaller of Result Set
  - Inner Table: Larger of Result Set
  - Use Trace Event 10104 (Hash Area Trace) or 10053 (CBO Trace) to size correctly
    - HASH_AREA_SIZE

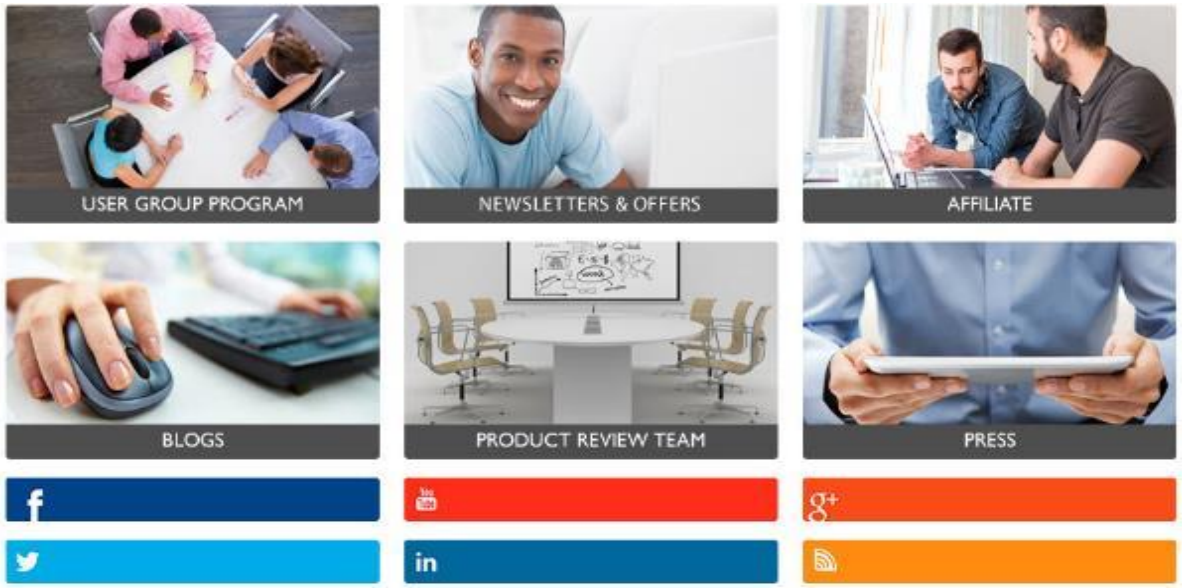
Additional Resources

Dan Hotka

- Dan@DanHotka.com
- Website: www.DanHotka.com
- Twitter: @DanHotka

Thank you!

- Sign-up for more webcasts at informit.com/webcasts
- Connect with InformIT at informit.com/community



informIT