Oracle SQL Tuning

3 parts (BASIC, INTERMEDIATE, ADVANCED)

www.lingaro.com

lingaro

The Value of Business Intelligence

Training Agenda

- BASIC
  - SQL Tuning Introduction
  - Instrumentation & Rewrite SQL Text
  - SQL Plan Reading
- INTERMEDIATE
  - Optimizer & Statistics
  - SQL Plan Operations
  - SQL Plan Tuning
- ADVANCED
  - SQL Processing & Cursor Sharing
  - Advanced Plan Operations
  - SQL Plan Transformations
  - SQL Plan Management

Training Agenda

- For All Topics
  - Theory
  - Quiz Comp
  - Workshop
  - Workshop Comp
- Winner Awards

Training Agenda

- Tuning Techniques (not covered yet)
  - Using Automatic SQL Tuning
  - Using SQL Tuning & SQL Access Advisors (dbms_sqltune or OEM)
  - Using Oracle Enterprise Manager Console OEM
  - Using DBA_HIST and V$ performance views (in BASIC and here)
  - Using Test Executions
  - Using SQL Decomposition

    Test Execution on SQL parts

Oracle SQL Tuning

INTERMEDIATE

www.lingaro.com

lingaro

The Value of Business Intelligence

Training Agenda

- Optimizer & Statistics
- SQL Plan Operations
- SQL Execution Plan Tuning

Topic Agenda

Oracle Optimizer

- Optimizer Introduction
- Optimizer Components
  - Calculator, Transformer, Estimator, Plan Generator
- Optimizer Statistics
  - Tables, Columns, Histograms and Indexes
- Execution Cost Estimation
  - Selectivity, Cardinality, Cost
  - Optimizer Estimation Examples
- Dynamic Sampling

Optimizer

Introduction

- SQL statements – description what to do with data
- Instruction how to do it - not needed inside SQL
- Optimizer – used to chooses SQL execution plan - from many possible
- Optimizer – components used during hard parse step
  - Calculator – to simplify expressions

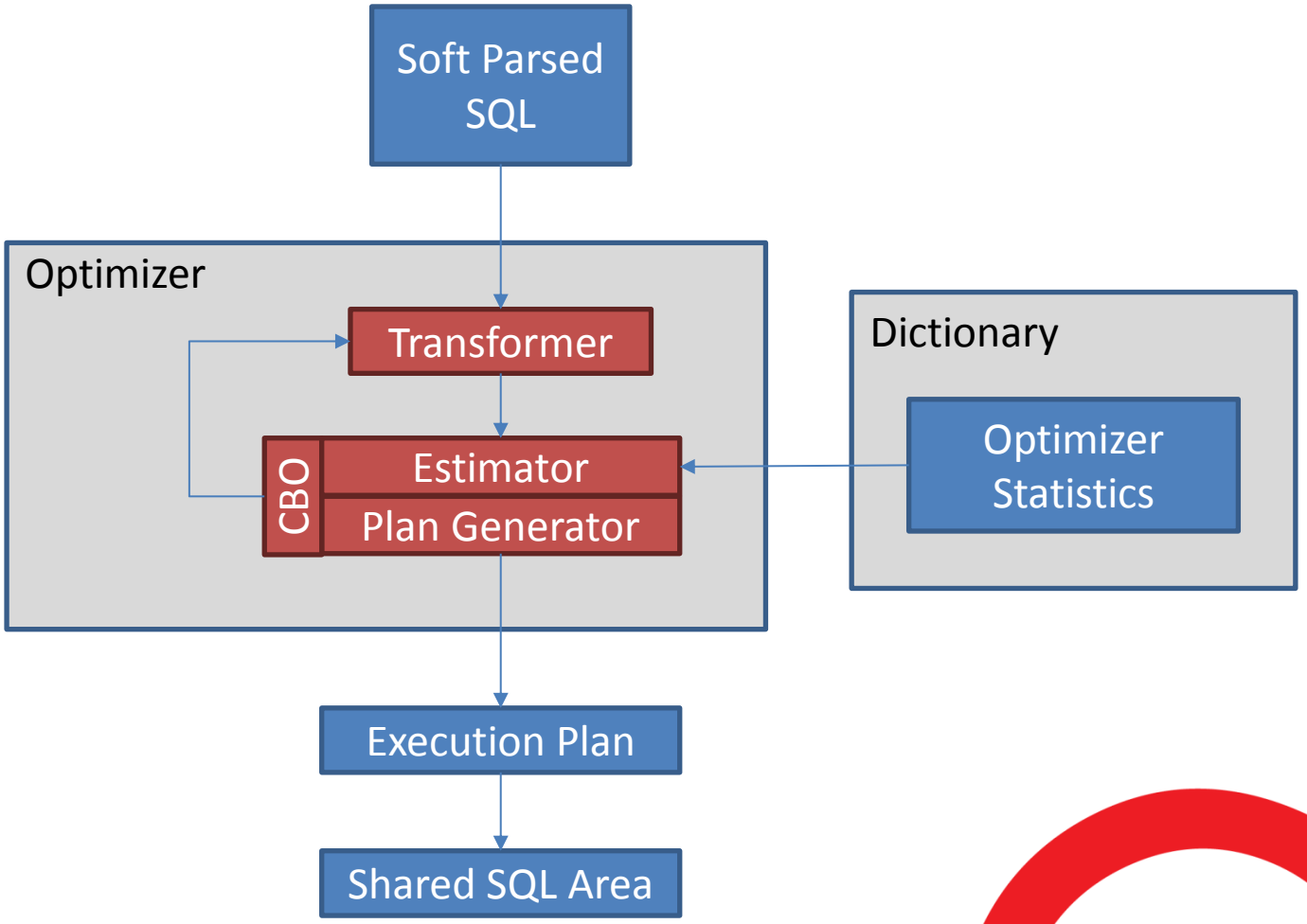
    e.g. `SELECT sales_amt * 100 / (47 + 3)  ->  SELECT sales_amt * 2`
  - Transformer – to rewrite statement on the fly

    e.g. changes subquery into join
  - Estimator – to estimate operations costs – visible in execution plan

    e.g. number of rows, memory, time estimates for particular statement operations
  - Plan Generator – to build SQL execution plan

Optimizer

Components



Optimizer Statistics

Introduction

- Metadata - describes data and its storage (in tables, indexes)
- Used by Optimizer Estimator to estimate SQL Plan Operations costs.
- **Are static – not changed when data are changed – refresh needed.**
  - Gathered using samples - random part of data - to reduce refresh cost.
  - Database has (default enabled) refresh job - disabled in most DW.
  - If defined percent of data is modified then statistic is marked as STALE.
- **ETL processes should refresh it just after data load or modification.**
- **Not up to date statistics can lead to suboptimal SQL execution plans.**
  - If statistic is high deviated then wrong plan can make execution costs.
  - Dynamic sampling can be used when optimizer statistics are not available in data dictionary during SQL parse phase.
- Object, partitions and sub partitions have individual statistics.
- Optionally column statistics and column histograms can be added.

Optimizer Statistics

Statistics Maintenance

- Use **dbms_stats** package **procedures**
  - Calculate/refresh

    gather_table_stats - for table & columns (optionally with indexes)

    gather_index_stats - for index

    gather_schema_stats - for all objects in schema
  - Delete and lock

    delete_table_stats, delete_column_stats, delete_index_stats, delete_schema_stats

    lock_table_stats, lock_schema_stats, unlock_table_stats, unlock_schema_stats
  - Manual entering and reading

    set_table_stats, seed_col_usag, set_index_stats

    get_table_stats, get_column_stats, get_index_stats
  - Restoring from history. - Get historical timestamps from USER_TAB_STATS_HISTORY view

    restore_table_stats, restore_schema_stats
  - Export & import

    create_stat_table, export_table_stats, import_table_stats

  Example:

  dbms_stats.**gather_table_stats**(USER, 'SALES_FCT', estimate_percent => 5)
- Default values for few parameters are configured as preferences

Optimizer Statistics

Gather Table Statistics Procedure Parameters

- partname - Name of partition or subpartition
- estimate_percent - Percentage of rows to estimate. (100% if over 20%)
  The default is DBMS_STATS.AUTO_SAMPLE_SIZE
- block_sample: TRUE - blocks sample, FALSE - rows sample
- method_opt - columns statistics and histogram buckets specification

```
– 'FOR ALL [INDEXED | HIDDEN]    COLUMNS [SIZE <buckets>] '
– 'FOR COLUMNS [SIZE <buckets>] <column> [SIZE <buckets>] [,
                                <column> [SIZE <buckets>] ... ] '
```

<buckets>:

- *integer* - Number of histogram buckets. Must be in the range [1 - 254].
- REPEAT - Collects histograms only on the columns that already have histograms.
- AUTO - Oracle determines the columns to collect histograms based on data distribution and the workload of the columns.
- SKEWONLY - Oracle determines the columns to collect histograms based on the data distribution.

The default is 'FOR ALL COLUMNS SIZE AUTO'.

Optimizer Statistics

Gather Table Statistics Procedure Parameters (continued)

- degree - Degree of parallelism.
- granularity:
  - '**ALL**' - Gathers all (sub partition, partition, and global) statistics
  - '**AUTO**‘ - Determines the granularity based on the partitioning. This is the default.
  - '**GLOBAL**' - Gathers global statistics
  - '**GLOBAL AND PARTITION**' - Gathers the global and partition level statistics. No sub partition level statistics are gathered.
  - '**PARTITION**‘ - Gathers partition-level statistics
  - '**SUBPARTITION**' - Gathers sub partition-level statistics.
- cascade - Gather statistics on the indexes for this table.
- stattab - User statistics table name
- statid - Identifier associate with these statistics within stattab
- statown - Schema containing stattab (if different than ownname)
- no_invalidate - TRUE - Does not invalidate the dependent cursors.
- force - Gather statistics of table even if it is locked

Optimizer Statistics
Statistics Maintenance Examples

- Delete table level or partition level or column level statistics

```
dbms_stats.delete_table_stats(USER, 'SALE_FCT', cascade_parts => FALSE, cascade_columns => TRUE)
dbms_stats.delete_table_stats(USER, 'SALE_FCT', partname => 'P201503', cascade_parts => FALSE)
dbms_stats.delete_column_stats(USER, 'SALE_FCT', 'PROD_ID', cascade_parts => FALSE,
                                                            col_stat_type => 'HISTOGRAM')
```

- Compute on sub partition without columns or without histogram

```
dbms_stats.gather_table_stats(USER, 'SALE_FCT', partname => 'P201503-WM',
                                                granularity => 'SUBPATITION', method_op => NULL)
dbms_stats.gather_table_stats(USER, 'SALE_FCT', method_op => method_op => 'FOR ALL COLUMNS SIZE 1')
```

- Lock statistic - `dbms_stats.lock_table_stats(USER, 'SALE_FCT')`
- Restore table statistics

```
SELECT table_name, partition_name, subpartition_name, stats_update_time FROM user_tab_stats_history;
dbms_stats.restore_table_stats(USER, 'SALE_FCT', as_of_timestamp => SYSDATE - 1)
```

- Create stat table - `dbms_stats.create_stat_table(USER, 'MY_STAT_TABLE')`
- Export/Import table statistics

```
dbms_stats.export_table_stats(USER, 'SALE_FCT', stattable =>  'MY_STAT_TABLE')
```

- Set/Get table stats

```
dbms_stats.set_table_stats(USER, 'SALE_FCT',  numrows => 1222000, numblks => 312000, avgrlen =>122300)
```

Optimizer Statistics
Statistics Preferences

- Used to balance accuracy and costs and define default gathering parameters
- Can be set on schema & table levels

```
dbms_stats.set_schema_prefs, dbms_stats.set_table_prefs
```

- Default values are set on database level by DBA

```
dbms_stats.get_database_prefs
```

- Preferences
  - CASCADE - Collect index statistics with table statistics.
  - DEGREE - Determines parallelism for gathering statistics.
  - ESTIMATE_PERCENT - Percentage of rows to estimate.
  - METHOD_OPT - Controls column statistics and histogram
  - NO_INVALIDATE - If FALSE - invalidation of dependent cursors
  - GRANULARITY - Gathering level for partitioned tables
  - PUBLISH - If FALSE then statistic is pending
  - INCREMENTAL - If TRUE Oracle update global statistics by scanning only changed partitions.
  - STALE_PERCENT - Percentage of rows have to change before the statistics are deemed stale. The default value is 10%.

Blue – available only in preferences

- Example

```
dbms_stats.set_table_prefs(USER, 'SALES_FCT', 'STALE_PERCENT', '15')
```

Optimizer Statistics

Table Statistics

- Dictionary Views
  - USER_TABLES — Only tables statistics
  - USER_TAB_STATISTICS — Tables, partitions, and subpartitions statistics in separate rows
- Dictionary View Columns
  - **NUM_ROWS** — Number of rows in the object
  - **BLOCKS** — Number of used blocks in the object – **EXACT VALUE**
  - **EMPTY_BLOCKS** — Number of empty blocks in the object – **EXACT VALUE**
  - **AVG_SPACE** — Average available free space in the object
  - **CHAIN_CNT** — Number of chained rows in the object
  - **AVG_ROW_LEN** — Average row length, including row overhead
  - AVG_SPACE_FREELIST_BLOCKS — Average freespace of all blocks on a freelist
  - NUM_FREELIST_BLOCKS — Number of blocks on the freelist
  - AVG_CACHED_BLOCKS — Average number of blocks in the buffer cache
  - AVG_CACHE_HIT_RATIO — Average cache hit ratio for the object
  - **SAMPLE_SIZE** — Sample size used in analyzing the table
  - **LAST_ANALYZED** — Date of the most recent time the table was analyzed
  - **GLOBAL_STATS** — Indicates calculated without merging underlying partitions YES/NO
  - USER_STATS — Indicates whether statistics were entered directly by the user YES/NO
  - **STATTYPE_LOCKED** — Type of lock: **<null>,** (DATA, CACHE) – internal lock, **ALL - locked**
  - **STALE_STATS** — Indicates whether statistics for the object are stale YES/NO

Optimizer Statistics

Table Columns Statistics

- Dictionary Views
  - USER_TAB_COL_STATISTICS or USER_COLUMNS — Column statistics for tables
  - USER_PART_COL_STATISTICS — Column statistics for table partitions
  - USER_SUBPART_COL_STATISTICS — Column statistics for table subpartitions
- Dictionary View Columns
  - **NUM_DISTINCT** — NDV - Number of distinct values in the column
  - **LOW_VALUE** — Low value in the column – **EXACT VALUE**
  - **HIGH_VALUE** — High value in the column – **EXACT VALUE**
  - DENSITY — Density of the column (similar to selectivity but not used from 10g)
  - **NUM_NULLS** — Number of nulls in the column
  - **NUM_BUCKETS** — Number of buckets in histogram for the column
  - LAST_ANALYZED — Date on which this column was most recently analyzed
  - SAMPLE_SIZE — Sample size used in analyzing this column
  - GLOBAL_STATS — Indicates whether collected as a whole (YES) or partitions (NO)
  - USER_STATS — Indicates whether entered directly by the user (YES) or not (NO)
  - AVG_COL_LEN — Average length of the column (in bytes)
  - **HISTOGRAM** — Indicates existence/type of histogram: NONE, FREQUENCY, HEIGHT BALANCED

Optimizer Statistics

Histograms

- Are optional column statistics used to describe skew distributed data
- Dictionary Views
  - USER_TAB_HISTOGRAMS — Column histograms for tables
  - USER_PART_HISTOGRAMS — Column histograms for table partitions
  - USER_SUBPART_HISTOGRAMS — Column histograms for table sub partitions
- Dictionary View Columns
  - ENDPOINT_NUMBER — Histogram bucket number
  - ENDPOINT_VALUE — Normalized endpoint value for this bucket
  - ENDPOINT_ACTUAL_VALUE — Actual string value of the endpoint for this bucket

Example Table Histogram.sql

```
dbms_stats.gather_table_stats(USER,'<table name>',
        method_opt => 'for columns <column> size <buckets>');
```

- Types – Oracle decision

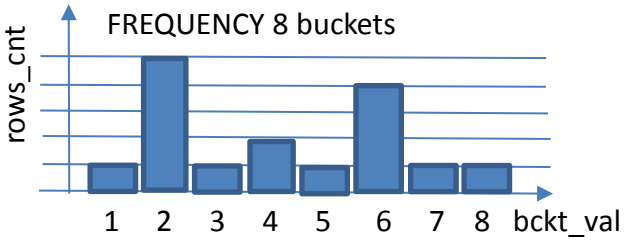
  - FREQUENCY

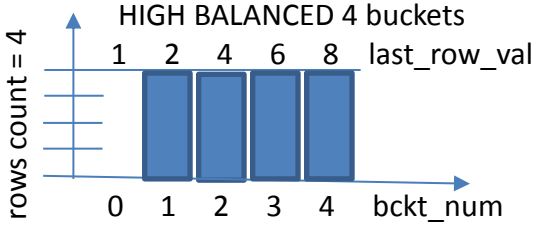
  Typically used when NDV = buckets count (small NDV)

  - HEIGHT BALANCED

  Typically used if NDV is higher then buckers count

Examples: 16 rows 8 values 1 - 8





Optimizer Statistics

Reading Histograms

- Checking histogram for columns

```
SELECT table_name, column_name, num_distinct, num_buckets, histogram
  FROM user_tab_col_statistics
  WHERE table_name = 'SEA_VALDN_FUNC_LKP' AND column_name = 'VALDN_ID' OR
        table_name = 'SEA_INTVN_FCT' AND column_name = 'INTVN_DESC' OR
        table_name = 'SEA_EXPSN_PI_AND_DV_DRIDN_IFCT' AND column_name = 'LAST_ONHND_QTY'
```

TABLE_NAME	COLUMN_NAME	NUM_DISTINCT	NUM_BUCKETS	HISTOGRAM
SEA_EXPSN_PI_AND_DV_DRIDN_IFCT	LAST_ONHND_QTY	214	71	FREQUENCY
SEA_INTVN_FCT	INTVN_DESC	13	147	HEIGHT BALANCED
SEA_VALDN_FUNC_LKP	VALDN_ID	14	14	FREQUENCY

- Display buckets values

FREQUENCY

```
SELECT endpoint_value                          AS bckt_val,
    endpoint_number - lag(endpoint_number, 1, 0)
       OVER (ORDER BY endpoint_number)         AS rows_cnt,
    endpoint_number                            AS rows_cnt_cum
 FROM user_histograms
 WHERE table_name='<table name>'
 AND column_name='<column name>'
 ORDER BY bckt_val;
```

HEIGHT BALANCED

```
SELECT endpoint_number                         AS bckt_num,
    <NDV>/<buckets> * endpoint_number AS bckt_last_row_num,
    endpoint_value                             AS last_row_val
 FROM user_histograms
 WHERE table_name='<table name>'
 AND column_name='<column name>'
 ORDER BY bckt_num;
```

- Popular value – bucked ending value for more then one bucket

Optimizer Statistics

Index Statistics

- Dictionary Views
  - USER_INDEXES — Only indexes statistics
  - USER_IND_STATISTICS — Indexes, index partitions and index subpartitions statistics in separate rows
- Dictionary View Columns
  - BLEVEL — B-Tree level – **EXACT VALUE**
  - LEAF_BLOCKS — Number of leaf blocks in the index
  - DISTINCT_KEYS — Number of distinct keys in the index
  - AVG_LEAF_BLOCKS_PER_KEY — Average number of leaf blocks per key
  - AVG_DATA_BLOCKS_PER_KEY — Average number of data blocks per key
  - CLUSTERING_FACTOR — Indicates order of the rows in the table related to index order. If the value is near the number of blocks, then the table is well ordered.
  - NUM_ROWS — Number of rows in the index
  - AVG_CACHED_BLOCKS — Average number of blocks in the buffer cache
  - AVG_CACHE_HIT_RATIO — Average cache hit ratio for the object
  - SAMPLE_SIZE — Sample size used in analyzing the index
  - LAST_ANALYZED — Date of the most recent time the index was analyzed
  - GLOBAL_STATS — Indicates calculated without merging underlying partitions YES/NO
  - USER_STATS — Indicates whether statistics were entered directly by the user YES/NO
  - STATTYPE_LOCKED — Type of statistics lock
  - STALE_STATS — Whether statistics for the object are stale or not

Q & A

+ Quiz

Optimizer Statistics
Workshop

- Fact table statistics creation and reading
  - Use SQL_Tuning_INTERM_workshop.sql script in SQL-Developer
  - Connect to database on VBOX using train/oracle@<vbox IP>/orcl
  - Create table SALES_FCT from SH_SALES_FCT (only chanl_id=4)

```
and bitmap index on prod_id
and create product_dim with PK from sh_product_dim
```

  - Compute statistics on tables using default preferences
  - Display table and columns statistics, histograms and index statistics
  - Display histogram buckets for selected column

Q & A

+ Workshop Comp

Optimizer Statistics

Refresh Strategy for DW

- Assumptions – typical DW workload
  - No data modifications, only load in ETL processes
  - Statistics calculated immediately after data load - in ETL process
- For small unpartitioned metadata, lookup and dimension tables
  - Recalculate all statistics on whole table if statistic not exists or STALE_STATS = 'YES'
  - Use columns statistics on all columns
- For large partitioned fact and dimension tables
  - Calculate only lowest level (e.g. sub partition) statistic after load and before exchange
  - Use one from two available strategy to have fresh statistics on higher levels
    1 – Cheap but problems with NDV quality – steps below for sub partitioned tables
    Delete statistics on table and partition level
    Keep all sub partitions statistics even are empty or not used
    2 – Accurate NDV but more expensive so use only for few tables with NDV problems
    Use INCREMENTAL = TRUE in table statistics preferences
    Calculate statistics using SKEWONLY in method_opt buckets count
- BEWARE maximum, minimum values in column statistics & buckets values in histograms
  - For dates and sequence numbers in columns stale statistics can make problems with estimates
  - Possible very low wrong cardinality estimation and very bad plans if statistics are not fresh
- Use SIZE = SKEWONLY (not AUTO) if no nominal workload was executed yet on tables

Optimizer Estimations

Terminology

- Estimated **Selectivity**
  - Fraction of rows to be processed by 1 operation (from many statement operations) estimation
  - Values between 0 and 1 where 1 means all rows
  - For non-unique column equality filtering: *Selectivity = 1 / NDV column stat*
  - For range filtering: *Selectivity = (high filter - low filter) / (highest val stat - lowest val stat)*
  - If histogram exists selectivity is calculated from histogram
  - For many column filters (AND) selectivity = multiply of columns selectivity

```
It is not correct if columns are related like product category and sub category
- if related use Extended (multicolumn-virtual column) Column Statistics
```

- Estimated **Cardinality**
  - Number of rows processed by operation estimation
  - *Cardinality = Selectivity * Number of rows table stat*
  - *Join Cardinality = T1 Cardinality * T2 Cardinality * Join Selectivity*
- Estimated **Cost**

CPU speed ↓

$$\frac{\text{singe block reads cost} + \text{multi block reads cost} + \text{CPU cost}}{\text{1blk read cnt} * \text{1blk read time} + \text{mblk read cnt} * \text{mblk read time} + \text{CPU cycles} / \text{cycles per second}}{\text{1blk read time}}$$

  - Is used in plan generation too choose plan with lowest cost

Optimizer Statistics
Cardinality Validation

- You can validate cardinality by comparing estimated and actual rows count

1)
- Query with count(*) instead of resulting column list and
- CARDINALITY column of plan operation visible after F10 pressed in SQL*Developer

2)
- E-Rows and A-Rows columns in plan from DBMS_XPLAN with 'ALLSTATS LAST' option
- Use STATISTIC_LEVEL = ALL on session or GATHER_PLAN_STATISTICS hint in SQL

3)
- Columns 

from SQL Monitoring in:
- v$sql_monitor and v$sql_monitor_plan
- SQL Monitor Lingaro tool
- Enterprise Manager Console

Optimizer Statistics

Columns Extended Statistics

- If many related columns in filter
- Create implicit virtual column

```
SELECT dbms_stats.create_extended_stats(NULL,
       '<table name>', '(<column1>, <column2>)') FROM dual;
```

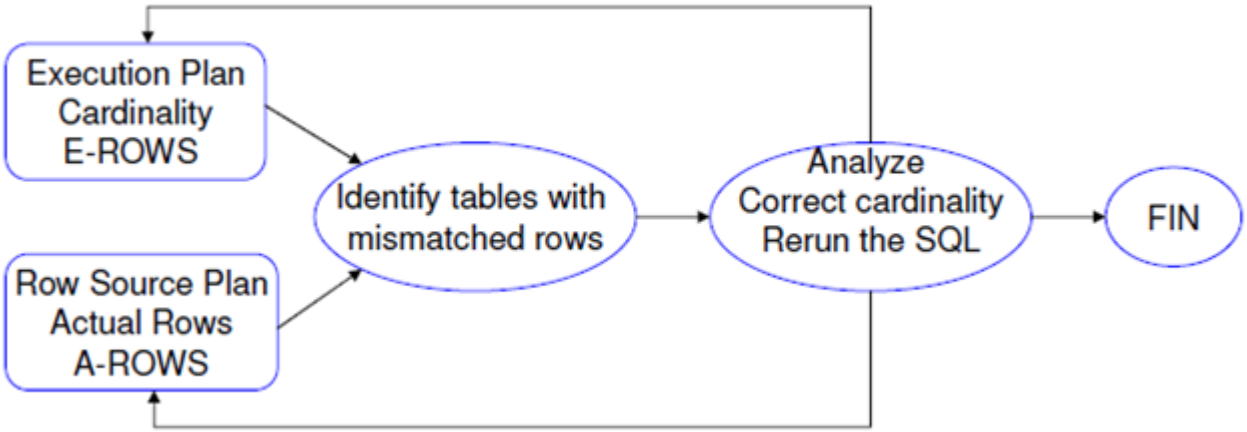
DBMS_STATS.CREATE_EXTENDED_STATS(NULL,'SH_CUST_DIM','(CUST_ST_PROV_NAME,CUST_CITY_NAME)')
SYS_STUTIT5GTXKWDPL518AHCNX_NY

- Compute all column statistics with SKEWONLY option
- Can create explicit virtual column with any expression
- Can calculate column statistics on any virtual columns
  - No expression based index needed

Optimizer Statistics

Cardinality Feedback

- Use Statistic Level ALL or GATHER_PLAN_STATISTICS hint
  - If you have problem with cardinality estimation
- Cardinality estimation will be automatically corrected
  - Correct value will take into account actual row number from first execution
- Plan will change on second execution



Dynamic Sampling

Overview

- When dictionary stored optimizer statistics not available (or insufficient)
- Calculate it during parse phase after recursive sample e.g. 10% table scans

```
SELECT /* OPT_DYN_SAMP */ ... FROM table SAMPLE(10) ...
```

- Criteria to us it and number of block sampled depends on sampling level
- Dynamic Sampling level can be modified:
  - On session - **OPTIMIZER_DYNAMIC_SAMPLING parameter** (default 2)
  - In statement - **DYNAMIC_SAMPLING[(level)]** query **hint**
- If used to build plan then plan is indicated by note

```
Note -----------------------------------------
- dynamic sampling level 8 used for this statement
```

- Can produce better cardinality estimates
  - Dictionary statistics can be stale
  - But if number of sampled block is smaller then during dictionary statistic gathering then can be worse
- Can lead to performance problems due to
  - Costly sample scan for large number of block
  - CPU cost to process large sampled data
  - Bugs possible for PQ (if default level 2 used) – “library cache lock”

Dynamic Sampling

Levels

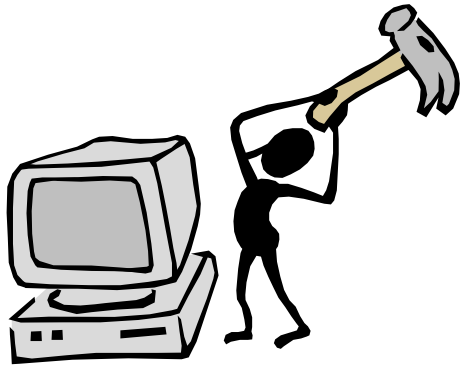
- Level 0: Do not use dynamic sampling.
- Level 1: Sample all tables not analyzed if the following criteria are met:
  - there is at least 1 unanalyzed table in the query;
  - this unanalyzed table is joined to another table or appears in a subquery or non-mergeable view;
  - this unanalyzed table has no indexes;
  - this unanalyzed table has more blocks than the number of blocks that would be used for dynamic sampling of this table. The number of blocks sampled is the **default number of dynamic sampling blocks = 32**.
- Level 2 (Default): Apply dynamic sampling to all unanalyzed tables.
  - The number of blocks sampled is two times the default number of dynamic sampling blocks.
- Level 3: Sampling to all tables that meet Level 2 criteria, plus all tables for which standard selectivity estimation used a guess for some predicate that is a potential dynamic sampling predicate.
  - The number of blocks sampled is the default number of dynamic sampling blocks. For unanalyzed tables, the number of blocks sampled is two times the default number of dynamic sampling blocks.
- Level 4: Sampling to all tables that meet Level 3 criteria, plus all tables that have single-table predicates that reference 2 or more columns.
  - The number of blocks sampled is the default number of dynamic sampling blocks. For unanalyzed tables, the number of blocks sampled is two times the default number of dynamic sampling blocks.
- Levels 5, 6, 7, 8, and 9: Previous level criteria
  - using 2, 4, 8, 32, or 128 times the default number of dynamic sampling blocks respectively.
- Level 10: Previous level criteria
  - using all blocks in the table.

Q & A

+ Quiz

Cardinality Validation

Workshop



- Cardinality Validation
  - Create table my_cust_dim as copy of sh_cust_dim
  - Compare rows count and cardinality in plan (F10 in SQLdeveloper) for SQL

```
SELECT count(*) FROM my_cust_dim
  WHERE cust_st_prov_name = 'CA'            -- filter f1
    AND cust_city_name LIKE 'Los Angeles'   -- filter f2
```

  - Enter cardinality results in following table

Cardinality	Count(*)	No stats	AUTO	SKEWONLY	Extended	Dynamic Sampling (2)
f1	3341	1213	3175	3455	3354	3064
f2	932	1213	874	874	1093	1108
f1 & f2	932	12	56	54	874	1108

Q & A

+ Workshop Comp

Topic Agenda

SQL Plan Operations

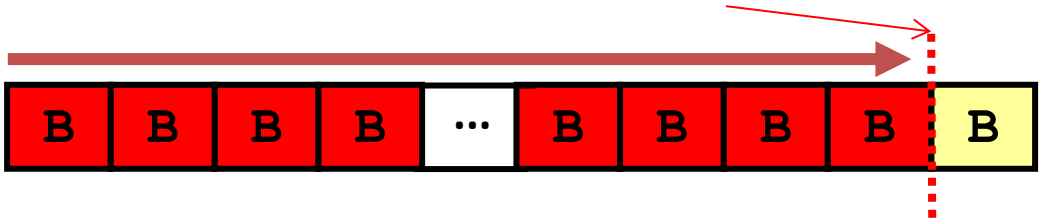
- Table Scans
  - FULL, BY ROWID, SAMPLE
- Index Scans
  - UNIQUE, RANGE, FULL, FAST FULL, JOIN
- Table Join Operations
  - HASH, SORT MERGE, NESTED LOOPS

Table Operations

Full Table Scan

Hint: FULL(<table alias>)

- Is faster than index range scans for large amount of data
- Performs multiblock reads (here `DB_FILE_MULTIBLOCK_READ_COUNT = 4`)
- Reads all formatted blocks below the high-water mark
- May filter rows
- Can be parallel
- Uses private pool



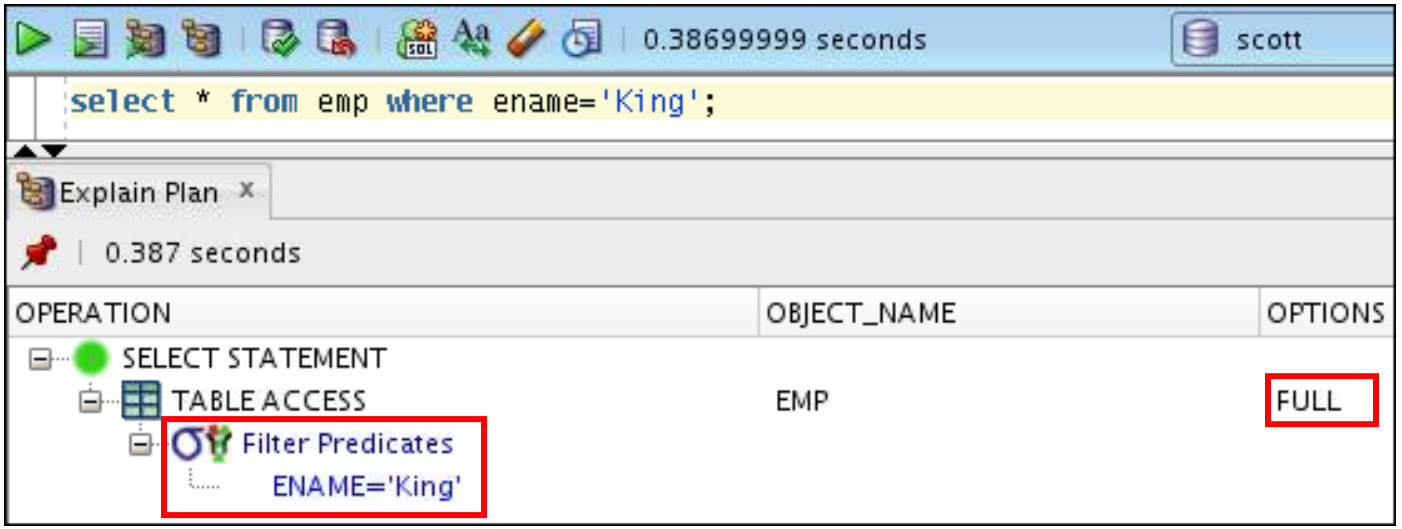


Table Operations

ROWID Scan

```
SELECT * FROM emp WHERE rowid='AAAQ+LAAEAAAAAfAAJ';

-------------------------------------------------------------------------
| Id  | Operation                   | Name | Rows  | Bytes | Cost |
-------------------------------------------------------------------------
|   0 | SELECT STATEMENT            |      |     1 |    37 |      1|
|   1 |  TABLE ACCESS BY USER ROWID| EMP  |     1 |    37 |      1|
-------------------------------------------------------------------------
```

Hint: INDEX(<table alias>, [<index>])

```
SELECT * FROM emp WHERE ename = 'King';

-------------------------------------------------------------------------
| Id  | Operation                    | Name | Rows  | Bytes | Cost |
-------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |      |     1 |    37 |      1|
|   1 |  TABLE ACCESS BY INDEX ROWID| EMP  |     1 |    37 |      1|
|   2 |   INDEX RANGE SCAN           | EMPIX|     1 |    17 |      1|
-------------------------------------------------------------------------
```

Table Operations

Sample Table Scan

- Reads rows from random selected blocks sample
- Only specified percent of blocks are chosen
- Oracle uses it to gather optimizer statistics and for dynamic sampling

```
SELECT * FROM emp SAMPLE BLOCK (10) SEED (1);

--------------------------------------------------------------------------
| Id  | Operation            | Name | Rows  | Bytes | Cost (%CPU)|
--------------------------------------------------------------------------
|   0 | SELECT STATEMENT     |      |     4 |    99 |     2   (0)|
|   1 |  TABLE ACCESS SAMPLE| EMP  |     4 |    99 |     2   (0)|
--------------------------------------------------------------------------
```

Q & A

+ Table Scan Quiz

Index Types

- Storage techniques:
  - B*-tree indexes: The default and the most common
    - Normal (key on single or multiple columns)
    - Function based (key on expression)
    - Index-organized table (IOT)
  - Bitmap indexes
- Index attributes:
  - Key compression (COMPRESS)
  - Reverse key (REVERSE)
  - Descending sorted (DESC after key column name)
- Domain indexes: Specific to an application or cartridge

Index Types

Normal B-tree Index

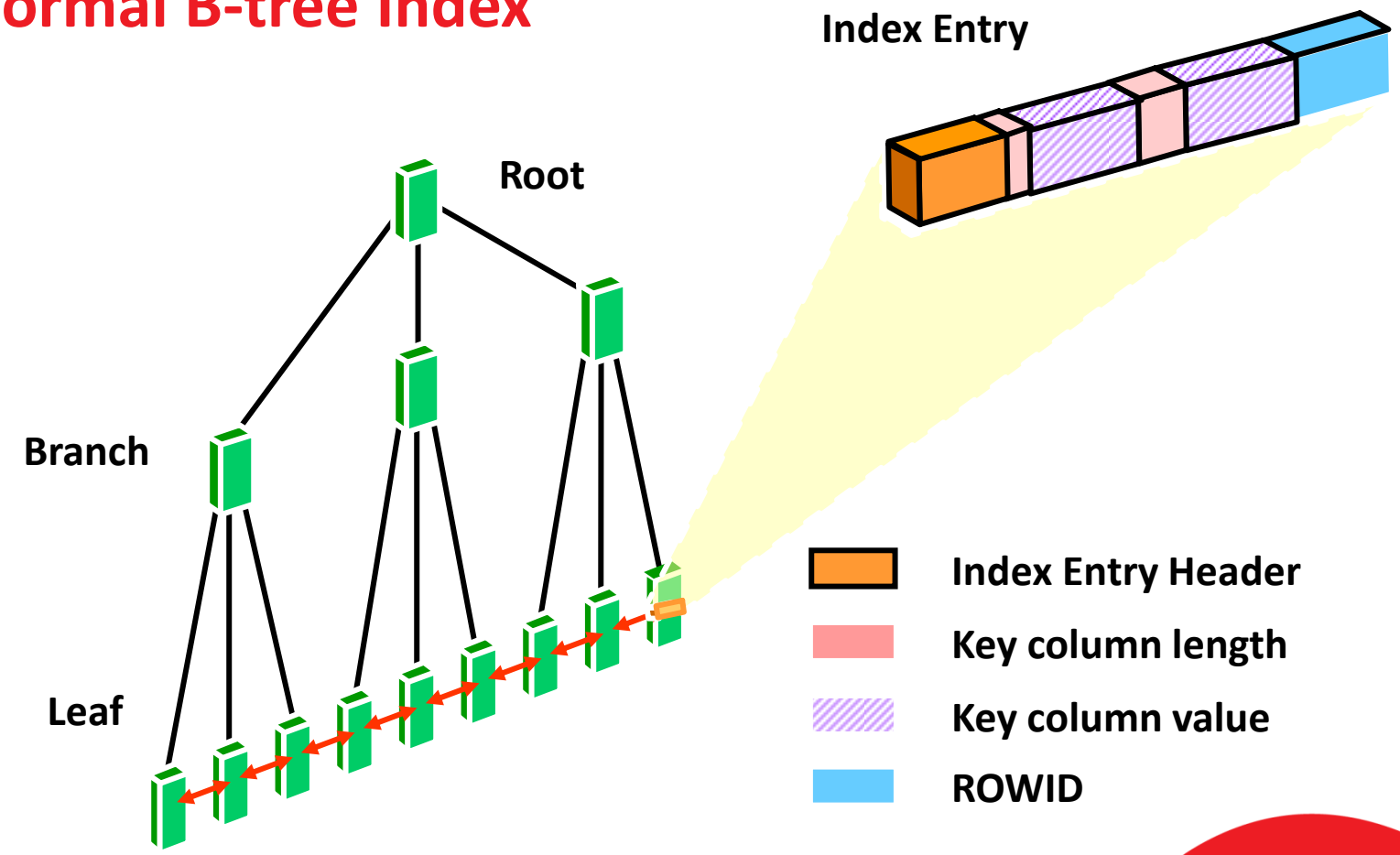


Table data retrieved by using ROWID

Index Operations

B-tree Index

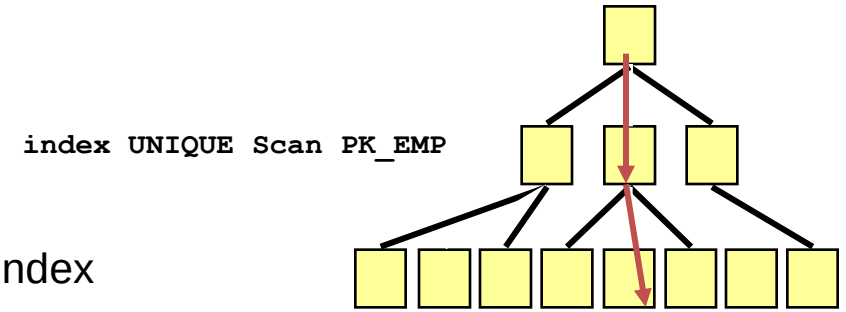
```
Hint: INDEX(tab [indx])
Opp.: NO_INDEX(tab [indx])
      FULL(tab)
```

- UNIQUE SCAN
- RANGE SCAN
- SKIP SCAN
- FULL SCAN
- FAST FULL SCAN
- JOIN SCAN
- NULL values in key
- Composite Indexes

B-tree Index Operations

UNIQUE SCAN

- Returns at most single `ROWID`
- Statement contains
  `UNIQUE`
  `PRIMARY KEY`
- All columns in unique (B*-tree) index
- Equality condition in filter




```
CREATE UNQUE INDEX pk_emp ON emp(empno)

SELECT * FROM emp WHERE empno = 7835;
```

Explain Plan

0.024 seconds

OPERATION	OBJECT_NAME	OPTIONS	COST	CARDINALITY
SELECT STATEMENT			1	1
TABLE ACCESS	EMP	BY INDEX ROWID	1	1
INDEX	PK_EMP	UNIQUE SCAN	0	1
Access Predicat				
EMPNO=7839				

Plan Operations

Index Operations

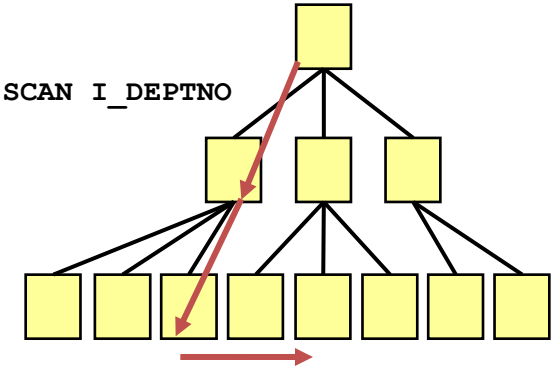
RANGE SCAN

Hint: INDEX_ASC

For filters:

col1 = :b1 (only none unique index)
col1 < :b1 col1 > :b2
col1 LIKE 'ABC%' .

- Can avoid sorting operation
  ORDER BY/GROUP BY
  Only when key columns are NOT NULL




```
CREATE INDEX i_deptno ON emp(deptno);

SELECT /*+ INDEX(emp i_deptno) */ * FROM emp
  WHERE deptno = 10 AND sal > 1000;
```

Script Output | Explain Plan

0.012 seconds

OPERATION	OBJECT_NAME	OPTIONS	COST	CARDINALITY
SELECT STATEMENT			2	4
TABLE ACCESS	EMP	BY INDEX ROWID	2	4
Filter Predicates				
SAL>1000				
INDEX	I_DEPTNO	RANGE SCAN	1	5
Access Predicat				
DEPTNO=10				

Plan Operations

Index Operations

RANGE SCAN DESCENDING

Hint: INDEX_DESC

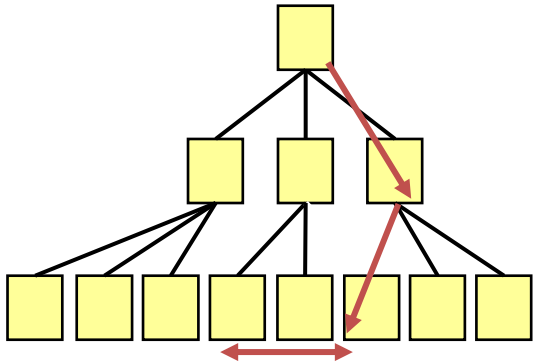
When scan order is different then index order

```
SELECT * FROM emp WHERE deptno > 20
  ORDER BY deptno DESC;
```

Script Output | Explain Plan

0.426 seconds

OPERATION	OBJECT_NAME	OPTIONS	COST	CARDINALITY
SELECT STATEMENT			2	7
TABLE ACCESS	EMP	BY INDEX ROWID	2	7
INDEX	I_DEPTNO	RANGE SCAN DESCENDING	1	7
Access Predicat				
DEPTNO>20				



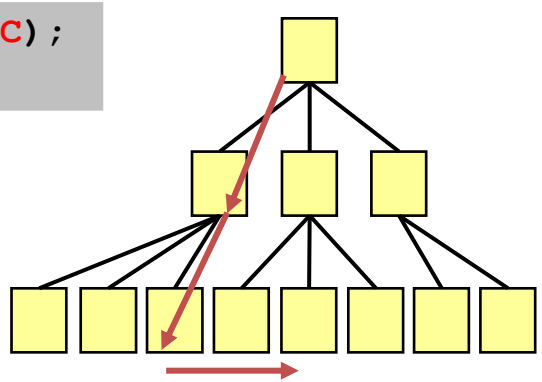
Index Range SCAN I_DEPTNO

```
DROP INDEX i_deptno; CREATE INDEX ix_d ON emp(deptno DESC);
SELECT * FROM emp WHERE deptno < 30;
```

Script Output | Explain Plan

0.011 seconds

OPERATION	OBJECT_NA...	OPTIONS	COST
SELECT STATEMENT			2
TABLE ACCESS	EMP	BY INDEX ROWID	2
INDEX	IX_D	RANGE SCAN	1
Access Predicates			
SYS_OP_DESCEND(DEPTNO)>HEXTORAW('3EE0FF')			
Filter Predicates			
SYS_OP_UNDESCEND(SYS_OP_DESCEND(DEPTNO))<30			

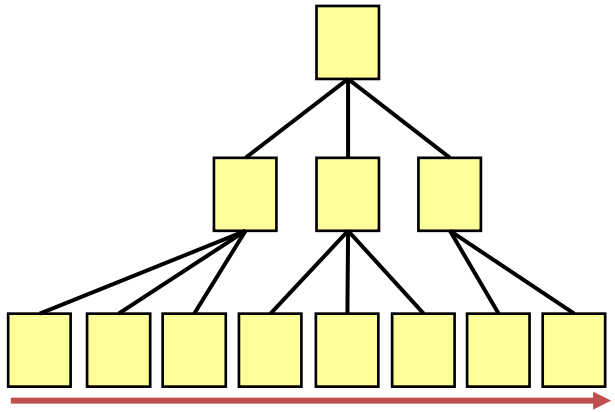


Index Range SCAN IX_D

B-tree Index Operations

FULL SCAN

All leafs read in index order (eliminate sort) when:
- predicate references column from index.
- no predicate if both the conditions are met
  all referenced columns included in the index
  at least one of the index columns is not null




```
CREATE INDEX i_deptno ON emp(deptno);

SELECT * FROM emp
  WHERE sal > 1000 AND deptno IS NOT NULL
  ORDER BY deptno;
```

Script Output | Explain Plan

0.012 seconds

OPERATION	OBJECT_NAME	OPTIONS	COST	CARDINALITY
SELECT STATEMENT			2	13
TABLE ACCESS	EMP	BY INDEX RO...	2	13
Filter Predicates				
SAL>1000				
INDEX	I_DEPTNO	FULL SCAN	1	14
Filter Predicates				
DEPTNO IS NOT NULL				

Plan Operations

Index Operations

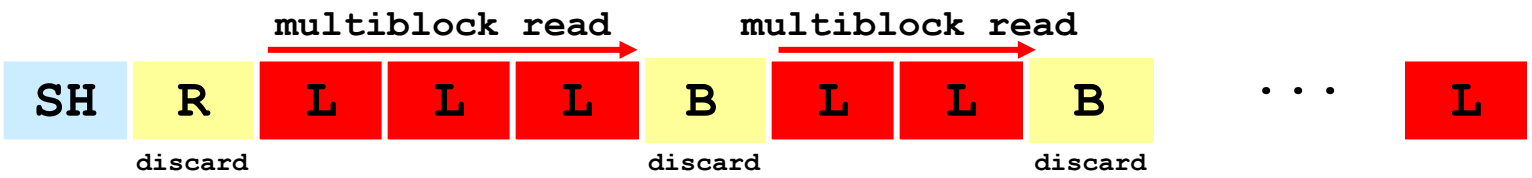
FAST FULL SCAN

```
Hint: INDEX_FFS
Opp.: NO_INDEX_FFS
```

- All leafs read in storage order.
- Not eliminate sort operation.
- Alternative to full table scans – multi-block and parallel
- When index contains all the columns needed.
  - at least one column in key has a `NOT NULL`.
- Without accessing the table (index only scan).

```
db_file_multiblock_read_count = 4
```

```
LEGEND:
SH=segment header
R=root block
B=branch block
L=leaf block
```




```
SELECT /*+ INDEX_FFS(EMP I_DEPTNO)  */ deptno
  FROM emp WHERE deptno IS NOT NULL;
```

Script Output | Explain Plan

0.01 seconds

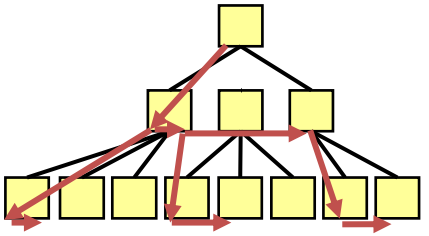
OPERATION	OBJECT_NAME	OPTIONS	COST	CARDINALITY
SELECT STATEMENT			2	14
INDEX	I_DEPTNO	FAST FULL SCAN	2	14
Filter Predicates				
DEPTNO IS NOT NULL				

B-tree Index Operations

SKIP SCAN

```
Hints:  INDEX_SS
        INDEX_SS_ASC
        INDEX_SS_DESC
Opp.:   NO_INDEX_SS
```

- If predicate exists on not first column in index key.
  and first column has small NDV (number of distinct values)
- Not interesting leafs are skipped
  after checked in brunch blocks



```
CREATE INDEX ix_ss ON emp(deptno, sal);

SELECT /*+ INDEX_SS(emp ix_ss) */ * FROM emp WHERE sal < 1500;
```

Script Output | Explain Plan

0.01 seconds

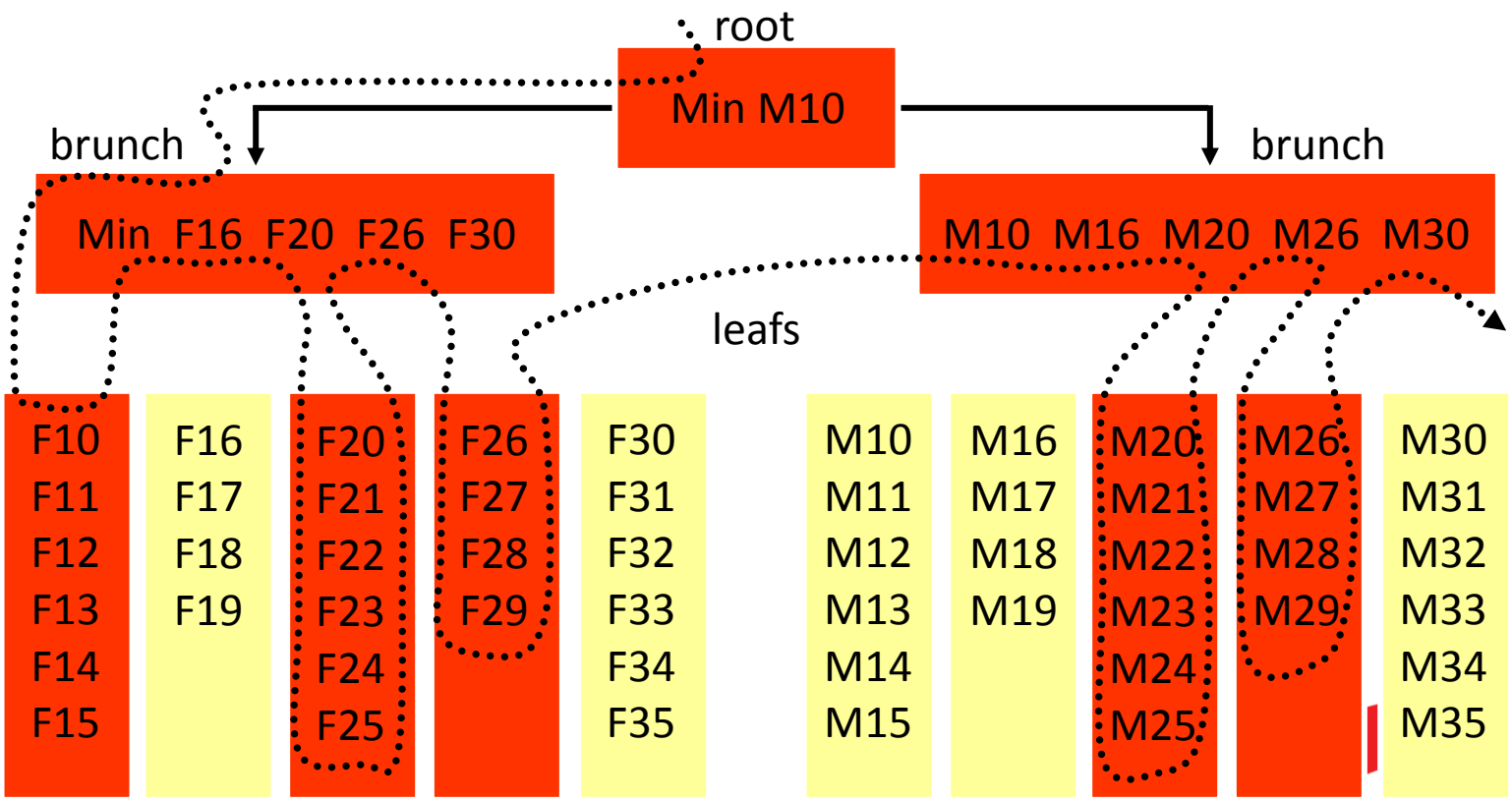
OPERATION	OBJECT_NAME	OPTIONS	COST	CARDINALITY
SELECT STATEMENT			4	2
TABLE ACCESS	EMP	BY INDEX ROWID	4	2
INDEX	IX_SS	SKIP SCAN	3	2
Access Predicates				
SAL<1500				
Filter Predicates				
SAL<1500				

Plan Operations

B-tree Index Operations

SKIP SCAN - Example

```
SELECT * FROM employees WHERE age BETWEEN 20 AND 29
```



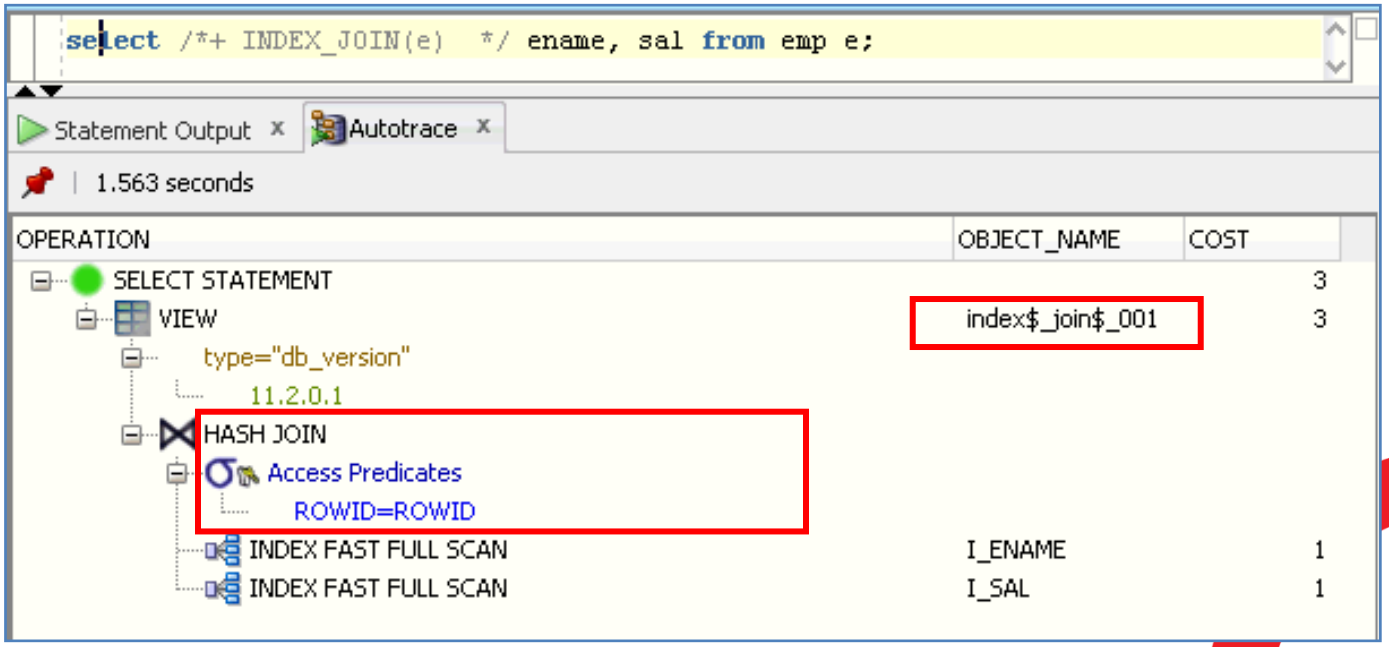
Index Operations

JOIN SCAN

Hint: INDEX_JOIN

- Hash join of several indexes to gather all the table columns referenced
- No table access needed
- Cannot be used to eliminate a sort operation.

```
ALTER TABLE emp MODIFY (sal NOT NULL, ename NOT NULL);
CREATE INDEX i_ename ON emp(ename);
CREATE INDEX i_sal ON emp(sal);
```



B-tree Index Operations

B-tree Do Not Sore NULL Values

Null elimination needed to use index in plan

```
CREATE TABLE nulltest (col1 NUMBER NULL, col2 NUMBER NOT NULL);
CREATE INDEX nullind1    ON nulltest (col1);
CREATE INDEX notnullind2 ON nulltest (col2);
```

COL1 is defined NULL
index cannot be used

```
select /*+ index(t nullind1) */ col1 from nulltest t;
```

Script Output | Explain Plan | 0.864 seconds

OPERATION	OBJECT_NAME	OPTIONS	COST	CARDINALITY
SELECT STATEMENT			2	1
TABLE ACCESS	NULLTEST	FULL	2	1

Predicate against COL1
this eliminates NULLs
index is used.

```
select col1 from nulltest t where col1=10;
```

Script Output | Explain Plan | 1.105 seconds

OPERATION	OBJECT_NAME	OPTIONS	COST	CARDINALITY
SELECT STATEMENT			1	1
INDEX	NULLIND1	RANGE SCAN	1	1
Access Predicates				
COL1=10				

COL2 is NOT NULL
index is used

```
select /*+ index(t notnullind2) */ col2 from nulltest t;
```

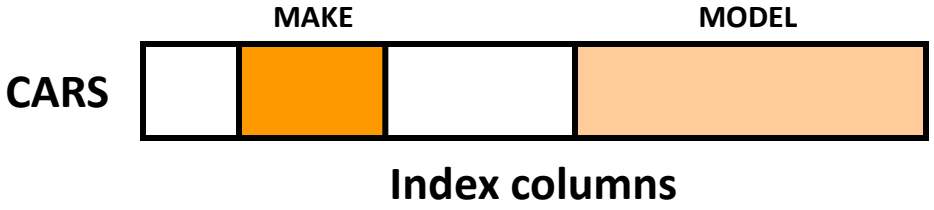
Script Output | Explain Plan | 0.034 seconds

OPERATION	OBJECT_NAME	OPTIONS	COST	CARDINALITY
SELECT STATEMENT			1	1
INDEX	NOTNULLIND2	FULL SCAN	1	1

Plan Operations

Index Operations

Composite Indexes



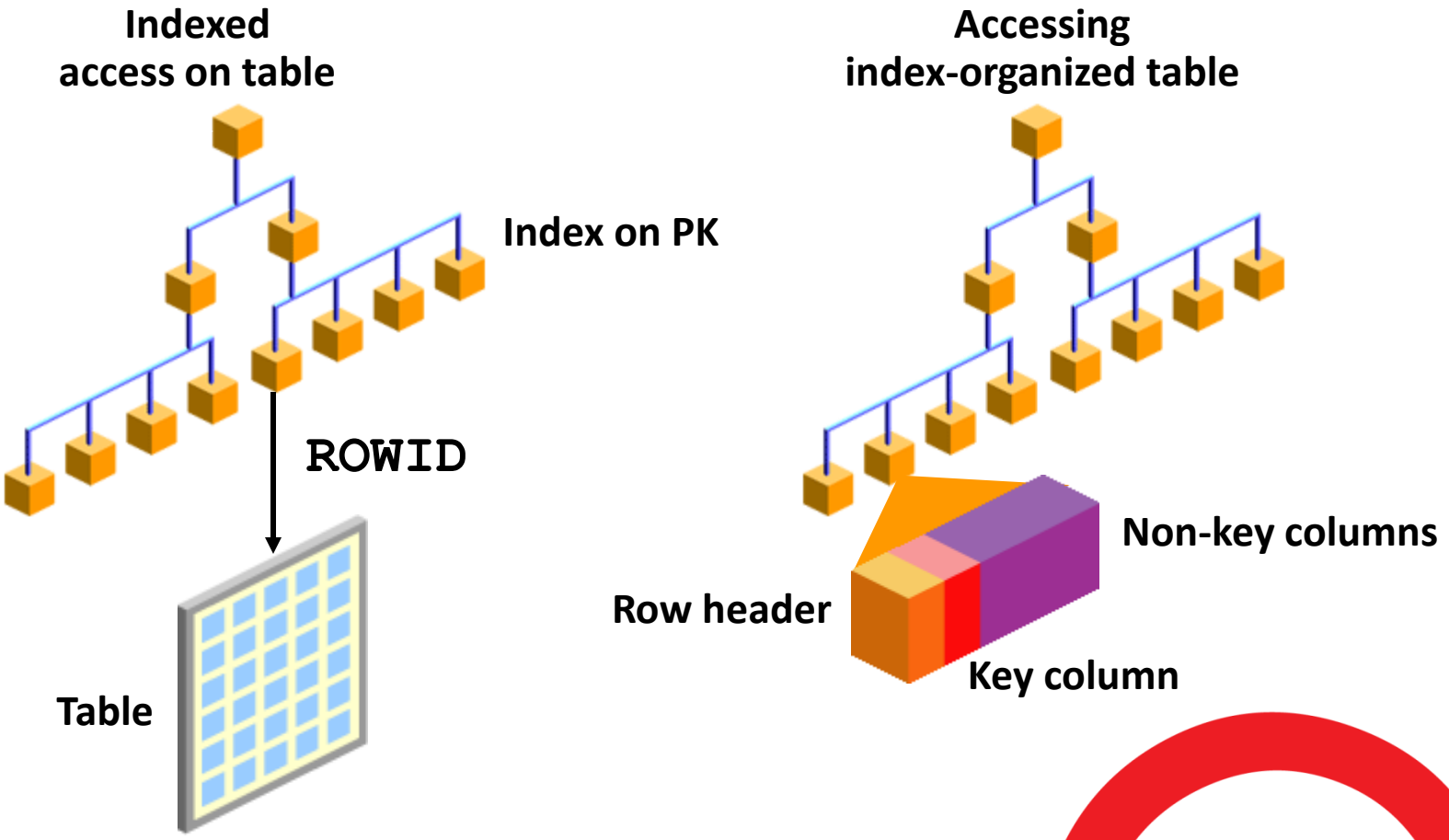

```
CREATE INDEX cars_make_model_idx ON cars(make, model);
```

```
SELECT *
  FROM cars
  WHERE make = 'CITROEN' AND model = '2CV';
```

```
-----------------------------------------------------------------
| Id  | Operation                    | Name                     |
-----------------------------------------------------------------
|   0 | SELECT STATEMENT             |                          |
|   1 |  TABLE ACCESS BY INDEX ROWID| CUSTOMERS                 |
|*  2 |   INDEX RANGE SCAN           | CARS_MAKE_MODEL_IDX      |
-----------------------------------------------------------------
```

Index Organized Table

IOT Overview



Plan Operations

Index Organized Table

IOT Operations

```
CREATE TABLE iotemp
 ( empno NUMBER(4) PRIMARY KEY, ...)
 ORGANIZATION INDEX;
```

```
select * from iotemp where empno = 9999;
```

Script Output | Explain Plan

0.734 seconds

OPERATION	OBJECT_NAME	OPTIONS	COST	CARDINALITY
SELECT STATEMENT			1	1
INDEX	SYS_IOT_TOP_7..	UNIQUE SCAN	1	1
Access Predicates				
EMPNO=9999				

```
select * from iotemp where sal > 1000;
```

Script Output | Explain Plan

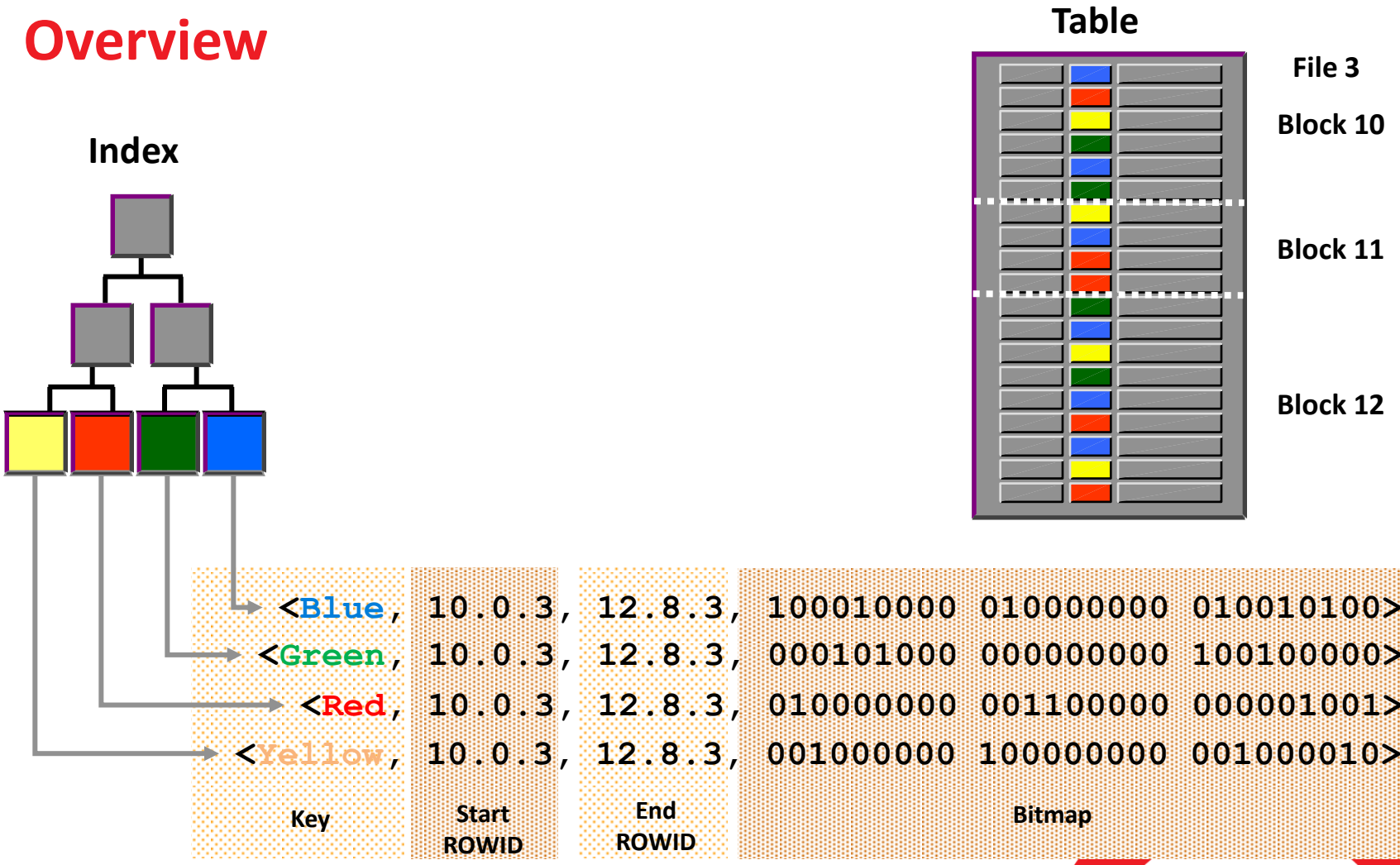
0.007 seconds

OPERATION	OBJECT_NAME	OPTIONS	COST	CARDINALITY
SELECT STATEMENT			2	1
INDEX	SYS_IOT_TOP_7...	FAST FULL SCAN	2	1
Filter Predicates				
SAL>1000				

Plan Operations

Bitmap Index

Overview



Plan Operations

Bitmap Index

Operations

- BITMAP CONVERSION:
  - TO ROWIDS
  - FROM ROWIDS
  - COUNT
- BITMAP INDEX:
  - SINGLE VALUE (similar to a B-tree unique scan)
  - RANGE SCAN (similar to a B-tree range scan)
  - FULL SCAN (all used columns are in index, NOT NULL not needed)
- Bitmap Combining
  - BITMAP AND / OR / MINUS
- Star Transformation (in next topic)
  - BITMAP MERGE
  - BITMAP KEY ITERATION

Bitmap Index

Counting Rows

```
SQL> CREATE TABLE t1 AS select * from all_objects;
SQL> CREATE BITMAP INDEX t1_b1 on t1(owner);

SELECT COUNT(*) FROM t1
-------------------------------------------------------------------------------
| Id  | Operation                          | Name  | Starts | E-Rows | A-Rows
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                   |       |      1 |        |      1
|   1 |  SORT AGGREGATE                    |       |      1 |      1 |      1
|   2 |   BITMAP CONVERSION COUNT          |       |      1 |  84499 |     31
|   3 |    BITMAP INDEX FAST FULL SCAN     | T1_B1 |      1 |        |     31
-------------------------------------------------------------------------------
```

Plan Operations

Bitmap Index

Access

```
SELECT * FROM cust_dim WHERE cntry_code = 'FR';
---------------------------------------------------------------------------
| Id  | Operation                     | Name       | Rows  | Bytes |
---------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |            |     1 |    45 |
|   1 |  TABLE ACCESS BY INDEX ROWID  | CUST_DIM   |     1 |    45 |
|   2 |   BITMAP CONVERSION TO ROWIDS |            |       |       |
|   3 |    BITMAP INDEX SINGLE VALUE  | IX_B2      |       |       |
---------------------------------------------------------------------------
Predicate:  3 - access("COUNTRY"='FR')
```

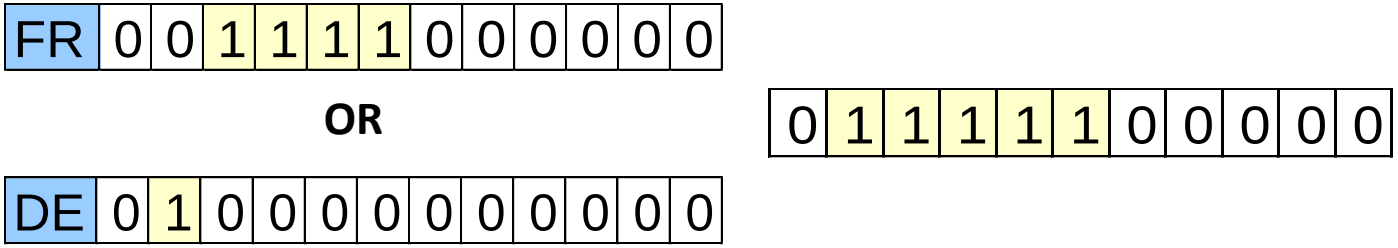
```
SELECT * FROM cust_dim WHERE cntry_code > 'FR';
---------------------------------------------------------------------------
| Id  | Operation                     | Name       | Rows  | Bytes |
---------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |            |     1 |    45 |
|   1 |  TABLE ACCESS BY INDEX ROWID  | CUST_DIM   |     1 |    45 |
|   2 |   BITMAP CONVERSION TO ROWIDS |            |       |       |
|   3 |    BITMAP INDEX RANGE SCAN    | IX_B2      |       |       |
---------------------------------------------------------------------------
Predicate: 3 - access("CNTRY_CODE">'FR') filter("CNTRY_CODE">'FR')
```

Plan Operations

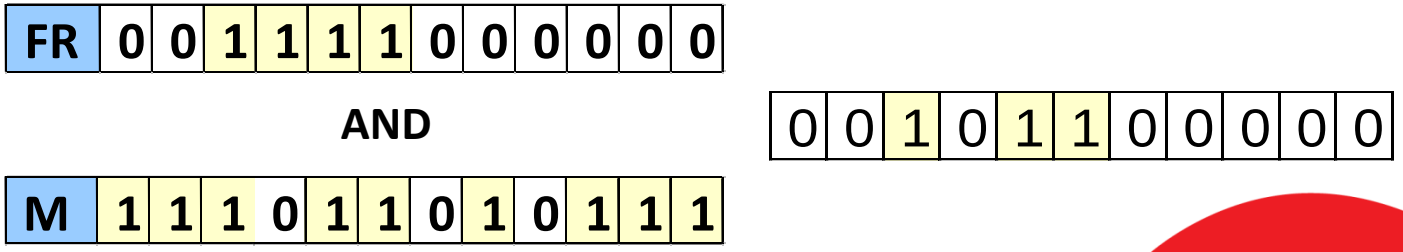
Bitmap Index

Combining - Overview

```
SELECT * FROM cust_dim WHERE cntry_code in ('FR','DE');
```




```
SELECT * FROM cust_dim WHERE cntry_code = 'FR' and gndr_code = 'M';
```



Plan Operations

Bitmap Index

Combining - Example

Hint: INDEX_COMBINE

```
SELECT * FROM cust_dim WHERE cntry_code IN ('FR','DE');
-------------------------------------------------------------------------
| Id  | Operation                      | Name       | Rows  | Bytes |
|   0 | SELECT STATEMENT               |            |     1 |    45 |
|   1 |  INLIST ITERATOR               |            |       |       |
|   2 |   TABLE ACCESS BY INDEX ROWID  | CUST_DIM   |     1 |    45 |
|   3 |    BITMAP CONVERSION TO ROWIDS |            |       |       |
|   4 |     BITMAP INDEX SINGLE VALUE  | IX_B2      |       |       |
Predicate: 4 - access("CNTRY_CODE"='DE' OR "CNTRY_CODE"='FR')
```

OR

```
SELECT * FROM cust_dim WHERE cntry_code = 'FR' AND gndr_code = 'M';
-------------------------------------------------------------------------
| Id  | Operation                      | Name       | Rows  | Bytes |
|   0 | SELECT STATEMENT               |            |     1 |    45 |
|   1 |  TABLE ACCESS BY INDEX ROWID   | CUST_DIM   |     1 |    45 |
|   2 |   BITMAP CONVERSION TO ROWIDS  |            |       |       |
|   3 |    BITMAP AND                  |            |       |       |
|   4 |     BITMAP INDEX SINGLE VALUE  | IX_B1      |       |       |
|   5 |     BITMAP INDEX SINGLE VALUE  | IX_B2      |       |       |
Predicate: 4 - access("GNDR_CODE"='M') 5 - access("CNTRY_CODE"='FR')
```

Invisible Indexes

- Index is altered as not visible to the optimizer:

```
ALTER INDEX ind1 INVISIBLE;
```

- Optimizer does not consider this index:

```
SELECT /*+ index(TAB1 IND1) */ COL1 FROM TAB1 WHERE …;
```

- Optimizer considers this index:

```
ALTER INDEX ind1 VISIBLE;
```

- Create an index as invisible initially:

```
CREATE INDEX IND1 ON TAB1(COL1) INVISIBLE;
```

Indexes Recommendations

- Index should be used on large tables filter columns.
- Are not recommended when processing over 10% rows.
- Create indexes after inserting table data.
- Index the correct tables and columns.
- Order index columns for performance.
- Limit the number of indexes for each table (DMLs).
- Drop indexes that are no longer required.
- Consider parallelizing index creation.
- Consider creating indexes with `NOLOGGING`.
- Consider costs and benefits of coalesce or rebuild.
- Consider cost before off or drop PK, UK constraints.

Index Investigation

Why Is Not Used ?

- There are functions being applied to the predicate.
- There is a data type mismatch.
- Statistics are old.
- The column can contain NULLs.
- Using index would actually be slower than not using it.

Why Is Used When Not Needed?

- Wrong low number of rows estimation computed
  - Index statistics are wrong

Q & A

+ Index Scan Quiz

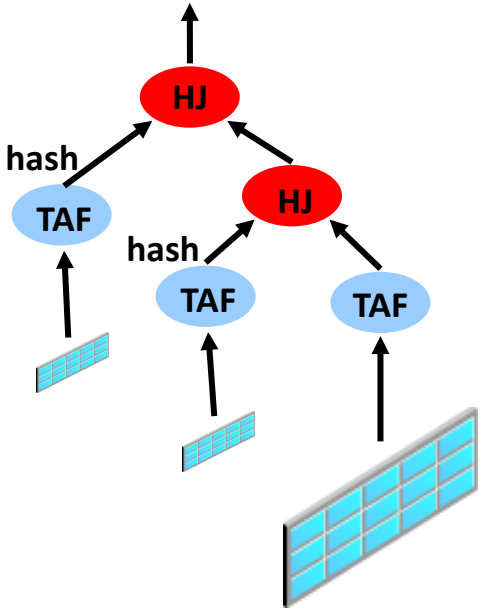
Table Join Operations

HASH JOIN

`Hint: USE_HASH(tab1 tab2 …)`

- Good in DW star schema – fact join with dimensions
- The small row source (dim) is used to build a hash table.
- The second source (fct) is hash checked against the hash table.

```
SELECT /*+ USE_HASH(f p t) */
    p.prod_name, t.fisc_yr_num, f.sold_amt
  FROM sh_sales_fct f
    INNER JOIN sh_time_dim t
      ON t.time_id  = f.time_id
    INNER JOIN sh_prod_dim p
      ON p.prod_id  = f.prod_id
```



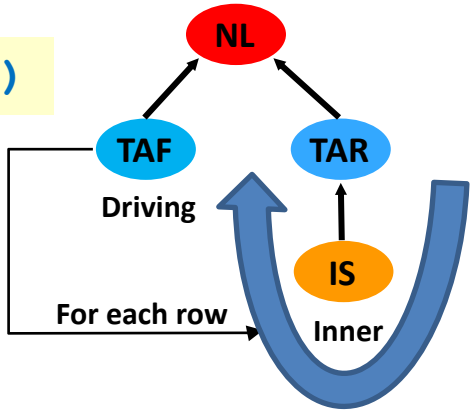
OPERATION	OBJECT_NAME	CARDINALITY	COST
SELECT STATEMENT		918843	1414
HASH JOIN		918843	1414
Access Predicates			
T.TIME_ID=F.TIME_ID			
PART JOIN FILTER (CREATE)	:BF0000	1826	17
TABLE ACCESS (FULL)	SH_TIME_DIM	1826	17
HASH JOIN		918843	1392
Access Predicates			
P.PROD_ID=F.PROD_ID			
TABLE ACCESS (FULL)	SH_PROD_DIM	72	3
PARTITION RANGE (JOIN-FILTER)		918843	1385
TABLE ACCESS (FULL)	SH_SALES_FCT	918843	1385

Plan Operations

Table Join Operations

NESTED LOOPS

`Hint: USE_NL(tab1 tab2)`



- Good when driving row source is very small.
- All driving row source is scanned.
- Each row returned drives a lookup in inner row source
  using needed index on join key column
- If driving source is large - many index scans in loop !

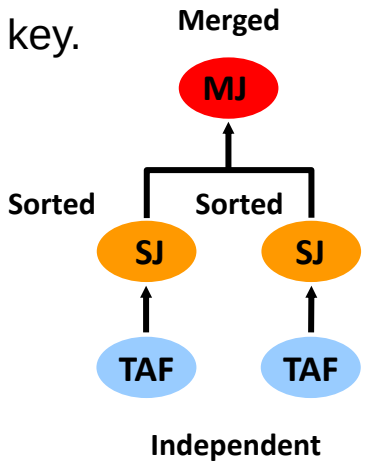
```
SELECT /*+ USE_NL(e d) */ e.employee_id, d.department_name
  FROM employees e, departments d
  WHERE e.department_id = d.department_id AND first_name LIKE 'A%'
-------------------------------------------------------------------------
| Id  | Operation                    | Name         | Starts | A-Rows |
-------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |              |      1 |     10 |
|   1 |  NESTED LOOPS                |              |      1 |     10 |
|   2 |   NESTED LOOPS               |              |      1 |     10 |
|*  3 |    TABLE ACCESS FULL         | EMPLOYEES    |      1 |     10 |
|*  4 |    INDEX UNIQUE SCAN         | DEPT_ID_PK   |     10 |     10 |
|   5 |   TABLE ACCESS BY INDEX ROWID| DEPARTMENTS  |     10 |     10 |
-------------------------------------------------------------------------
   3 - filter("FIRST_NAME" LIKE 'A%')
   4 - access("E"."DEPARTMENT_ID"="D"."DEPARTMENT_ID")
```

Table Join Operations

SORT MERGE JOIN

`Hint: USE_MARGE(tab1 tab2)`

- Good when join condition is not equality or results are ordered by join key.
- Can generate large and expensive sort.
- First and second row sources are sorted by same join key.
- Sorted rows from both tables are merged.




```
SELECT /*+ USE_MERGE(d e) NO_INDEX(d) */
    ename, e.deptno, d.deptno, dname
  FROM emp e, dept d
  WHERE e.deptno = d.deptno AND ename > 'A'
```

OPERATION	OBJECT_NAME	OPTIONS	COST	CARDINALI...
SELECT STATEMENT			8	14
MERGE JOIN			8	14
SORT		JOIN	4	4
TABLE ACCESS	DEPT	FULL	3	4
SORT		JOIN	4	14
Access Predicates				
AND				
E.DEPTNO=D.DEPTNO				
SYS_OP_DESCEND(E.DEPT				
Filter Predicates				
E.DEPTNO=D.DEPTNO				
TABLE ACCESS	EMP	FULL	3	14
Filter Predicates				
ENAME>'A'				

Plan Operations

Q & A

+ Table Join Quiz

Topic Agenda

Execution Plan Tuning

- Plan Generation
- Plan Evaluation
- Influence on Plan

Execution Plan

Optimizer Decisions

- Selectivity and cardinality derived from statistics or sampling
- Cost calculated for possible plan variants and smallest cost variant used
- Choosing objects to scan:
  - Tables – for large results volume
  - Indexes – for small results volume from big tables and to avoid sorts or table scans
- Choosing objects scan methods
  - Full scan for large volume reads
- Partition Pruning needed ? - static or dynamic if filtering on partition key exists ?
- Choosing Access Predicate - first filter - most restrictive
- Choosing Filter Predicates Order - most restrictive first
- PQ plan needed? Degree of parallelism choice - on execution time
  - Depends on time estimation and number of rows processed
- Row Distribution Methods (PQ plan only) - broadcast for small row sources
- Sorting and Grouping Methods - hash for large volume
- Join Methods – hash join for large volume but nested loop for very small
- Join Order – small row sources first

Evaluation

Where is the problem?

- Check if plan is changed to worse
- Make SQL Operations ranking (by CPU, time)
- Look for the following:
  - Driving table has the best filter
  - Fewest number of rows are returned to the next step
  - The join method is correct for number of rows
  - Parallel row distribution is good for number of rows
  - Views are correctly used
  - Unintentional Cartesian products
  - Tables/indexes accessed efficiently (also pruning)

Evaluation
Top SQL Plan Operations

```
SELECT r.samples, u.username, r.sql_id, r.event, r.in_parse,
    r.sql_exec_id, r.sql_exec_start, r.sql_plan_hash_value,
    r.sql_plan_operation, r.sql_plan_options,
    p.object_owner, p.object_name, p.cardinality
  FROM ( SELECT * FROM ( SELECT
         count(*) samples, user_id, sql_id, event, in_parse,
         row_number() OVER (PARTITION BY sql_id ORDER BY count(*) DESC) AS rn,
         sql_exec_id, sql_exec_start, sql_plan_hash_value,
         sql_plan_line_id, sql_plan_operation, sql_plan_options
       FROM dba_hist_active_sess_history
       WHERE sample_time > SYSDATE - 8
         AND sql_id = '6mm9w58yznv1k'
         AND sql_plan_hash_value = 1652160185
         AND session_state = 'ON CPU'
       --AND session_state = 'WAITING' and event <> 'resmgr:cpu quantum'
       GROUP BY sql_id, session_state, event, in_parse,
                sql_plan_operation, sql_plan_options, sql_plan_line_id,
                sql_exec_id, sql_exec_start, sql_plan_hash_value, user_id
       ORDER BY samples DESC
     ) WHERE ROWNUM <= 10 ) r
  LEFT JOIN dba_hist_sql_plan p ON p.ID = r.sql_plan_line_id
         AND p.sql_id = '6mm9w58yznv1k'
         AND p.plan_hash_value = 1652160185
  JOIN all_users u ON r.user_id = u.user_id
  ORDER BY samples DESC
```

Plan Change History

```
SELECT COUNT(*) samples,
       sql_exec_start,
       sql_plan_hash_value
  FROM dba_hist_active_sess_history
  WHERE sample_time > SYSDATE - 8
    AND sql_id = '6mm9w58yznv1k'
    AND session_state = 'ON CPU'
  GROUP BY sql_exec_start,
           sql_plan_hash_value
  ORDER BY sql_exec_start
```

SMPS	SQL_EXEC_S	HASH_VALUE
3	29 13:13:56	1652160185
6	29 13:24:07	1652160185
4	29 14:05:38	1652160185
5	29 14:13:44	1652160185
4	29 17:12:49	1652160185
3	29 17:19:02	1652160185
4	29 17:25:12	1652160185
4	30 12:40:08	1481012807
3	30 15:32:57	3896590179

Plan Modification

- What has influence on Execution Plan ?
  - **Optimizer Statistics (described in previous topic)**
  - **Optimizer Sensitive Parameters (change on session)**
  - Stored Outline (depreciated)
  - SQL Profile (main method)
  - Plan Baseline (new method)
  - Hints (as last resort)

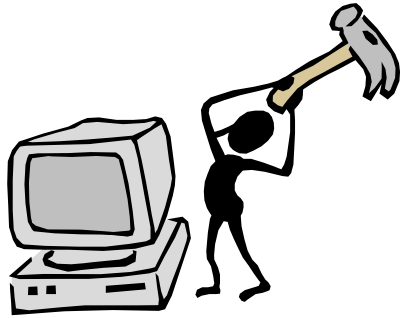
SQL Plan Management topic

Optymizer Sensitive Parameters

- **CURSOR_SHARING:** SIMILAR, EXACT, FORCE
- **DB_FILE_MULTIBLOCK_READ_COUNT**
- **PGA_AGGREGATE_TARGET** (for DBA but *_AREA_SIZE can change on session)
- STAR_TRANSFORMATION_ENABLED
- RESULT_CACHE_MODE: MANUAL, FORCE
- RESULT_CACHE_MAX_SIZE (for DBA)
- RESULT_CACHE_MAX_RESULT (for DBA)
- RESULT_CACHE_REMOTE_EXPIRATION (for DBA)
- **OPTIMIZER_INDEX_CACHING**
- **OPTIMIZER_INDEX_COST_ADJ**
- **OPTIMIZER_FEATURES_ENABLED**
- **OPTIMIZER_MODE:** ALL_ROWS, FIRST_ROWS, FIRST_ROWS_*n*
- OPTIMIZER_CAPTURE_SQL_PLAN_BASELINES
- OPTIMIZER_USE_SQL_PLAN_BASELINES
- OPTIMIZER_DYNAMIC_SAMPLING
- **OPTIMIZER_USE_INVISIBLE_INDEXES**
- **OPTIMIZER_USE_PENDING_STATISTICS**

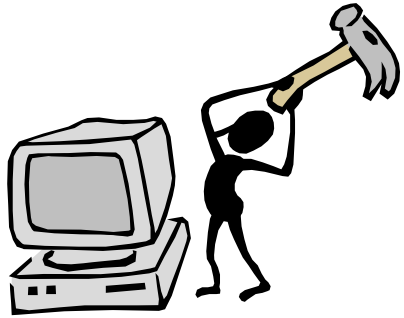
Q & A

Plan Tuning
Workshop



- Run Workload
- Check operation ranking

Plan Tuning
Workshop Comp Round 1-3



- Run statement from script
- Answer quiz questions

SQL Tuning Resources

- Oracle Database Documentation Library
  http://docs.oracle.com/cd/E11882_01/index.htm

- Database Performance Tuning Guide
  http://docs.oracle.com/cd/E11882_01/server.112/e41573/toc.htm

- Oracle SQL Tuning
  http://docs.oracle.com/cd/E28271_01/server.1111/e16638/sql_overview.htm

- Optimizer Statistics

http://www.oracle.com/technetwork/database/bi-datawarehousing/twp-optimizer-stats-concepts-110711-1354477.pdf