Oracle Database 11*g*: SQL Tuning Workshop

Electronic Presentation

D52163GC10

Edition 1.0

June 2008

ORACLE®

Author

Jean-François Verrier

Technical Contributors and Reviewers

Muriel Fry (Special thanks)
Joel Goodman
Harald van Breederode
Jörn Bartels
Pekka Siltala
Bernard Soleillant

James Spiller
Clay Fuller
Ira Singer
Christopher Andrews
Magnus Isaksson
Sean Kim
Trevor Bauwen
Yash Jain

Editors

Amitha Narayan
Raj Kumar

Graphic Designer

Priya Saxena

Publishers

Jothi Lakshmi
Veena Narasimhan

Disclaimer

This document contains proprietary information and is protected by copyright and other intellectual property laws. You may copy and print this document solely for your own use in an Oracle training course. The document may not be modified or altered in any way. Except where your use constitutes "fair use" under copyright law, you may not use, share, download, upload, copy, print, display, perform, reproduce, publish, license, post, transmit, or distribute this document in whole or in part without the express authorization of Oracle.

The information contained in this document is subject to change without notice. If you find any problems in the document, please report them in writing to: Oracle University, 500 Oracle Parkway, Redwood Shores, California 94065 USA. This document is not warranted to be error-free.

Restricted Rights Notice

If this documentation is delivered to the United States Government or anyone using the documentation on behalf of the United States Government, the following notice is applicable:

U.S. GOVERNMENT RIGHTS
The U.S. Government’s rights to use, modify, reproduce, release, perform, display, or disclose these training materials are restricted by the terms of the applicable Oracle license agreement and/or the applicable U.S. Government contract.

Trademark Notice

1

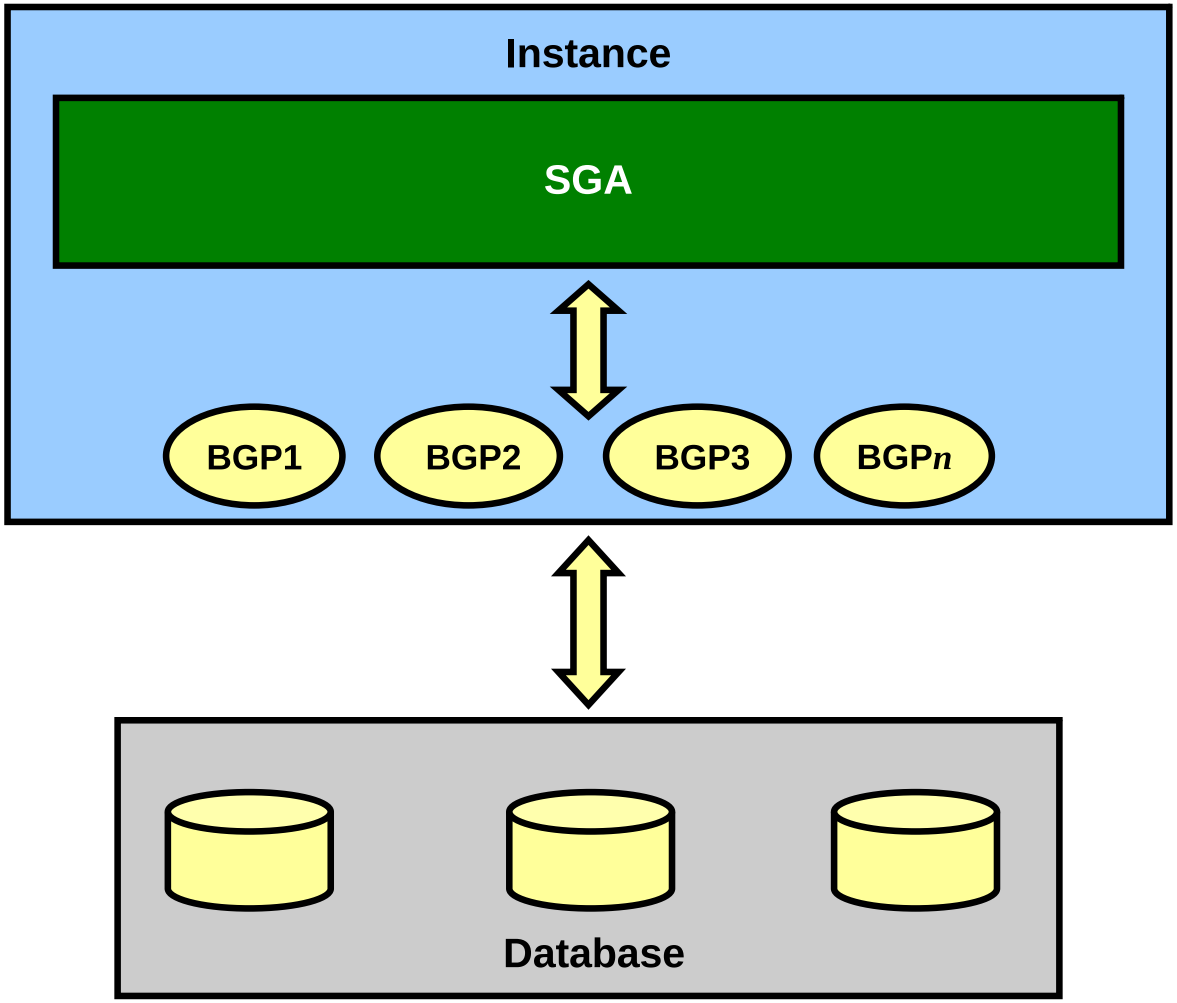
Exploring the Oracle Database Architecture

Objectives

After completing this lesson, you should be able to:

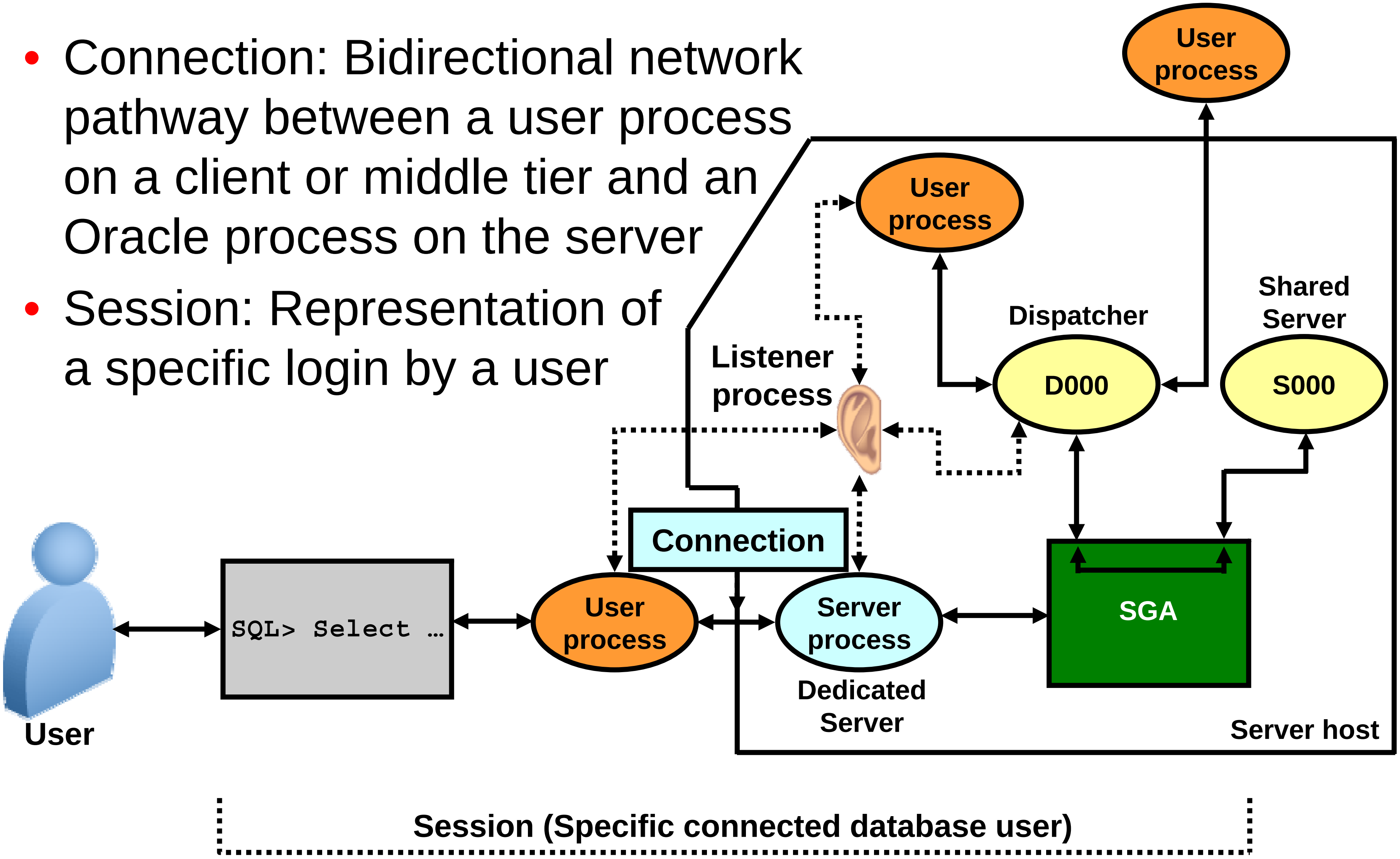
- List the major architectural components of the Oracle Database server
- Explain memory structures
- Describe background processes
- Correlate logical and physical storage structures

Oracle Database Server Architecture: Overview

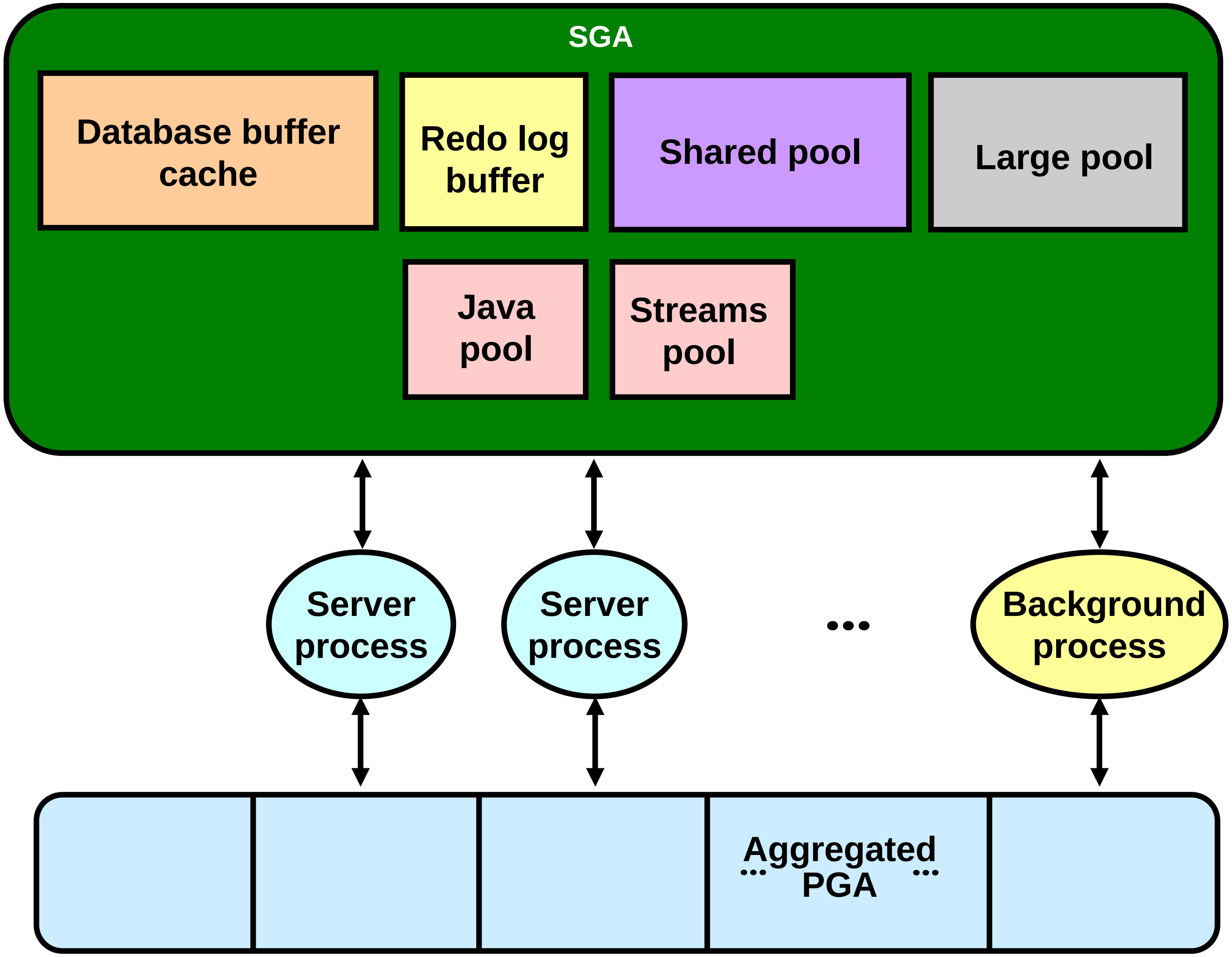


Connecting to the Database Instance

- Connection: Bidirectional network pathway between a user process on a client or middle tier and an Oracle process on the server
- Session: Representation of a specific login by a user

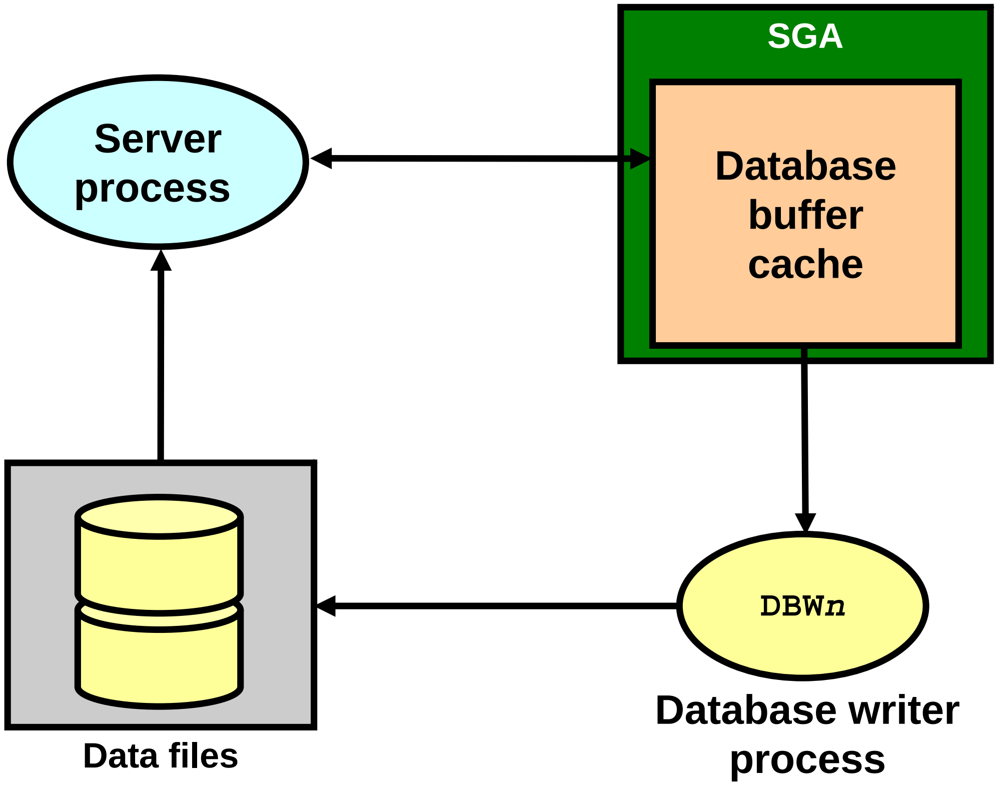


Oracle Database Memory Structures: Overview



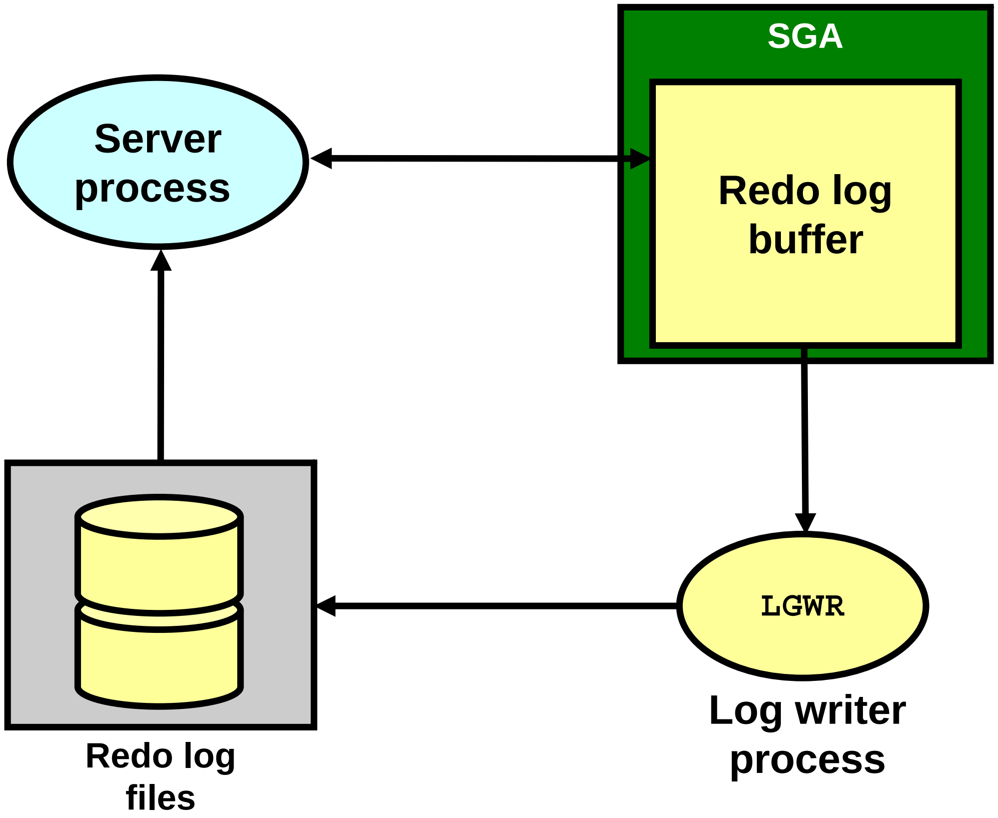
Database Buffer Cache

- Is a part of the SGA
- Holds copies of data blocks that are read from data files
- Is shared by all concurrent processes



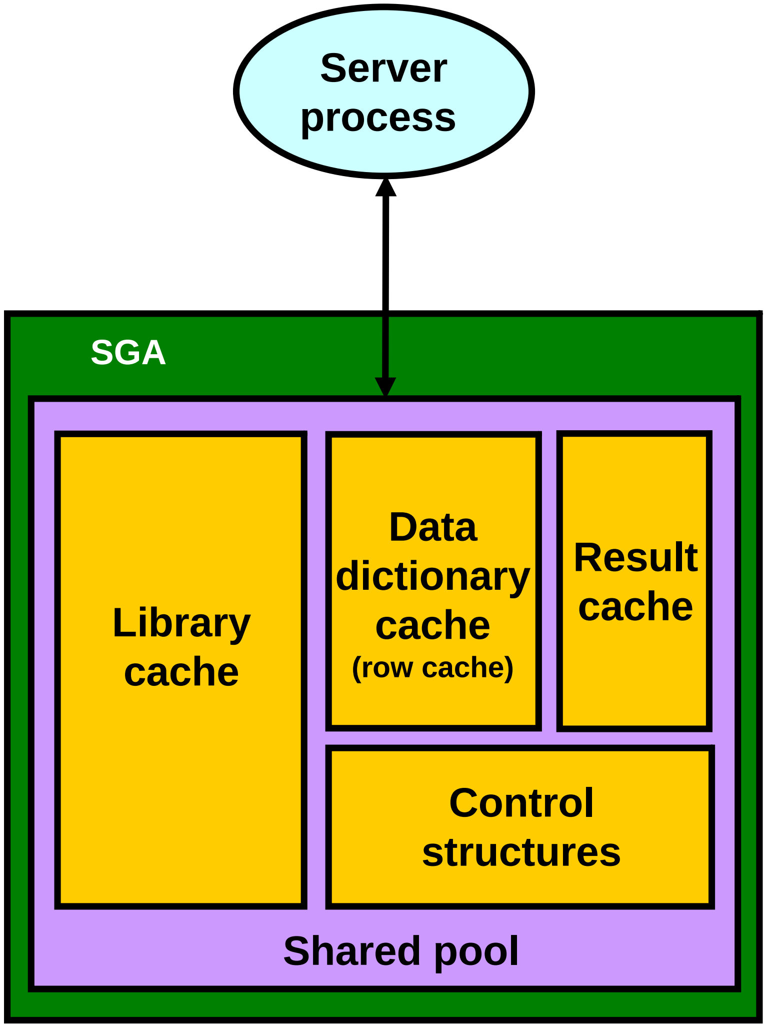
Redo Log Buffer

- Is a circular buffer in the SGA (based on the number of CPUs)
- Contains redo entries that have the information to redo changes made by operations, such as DML and DDL

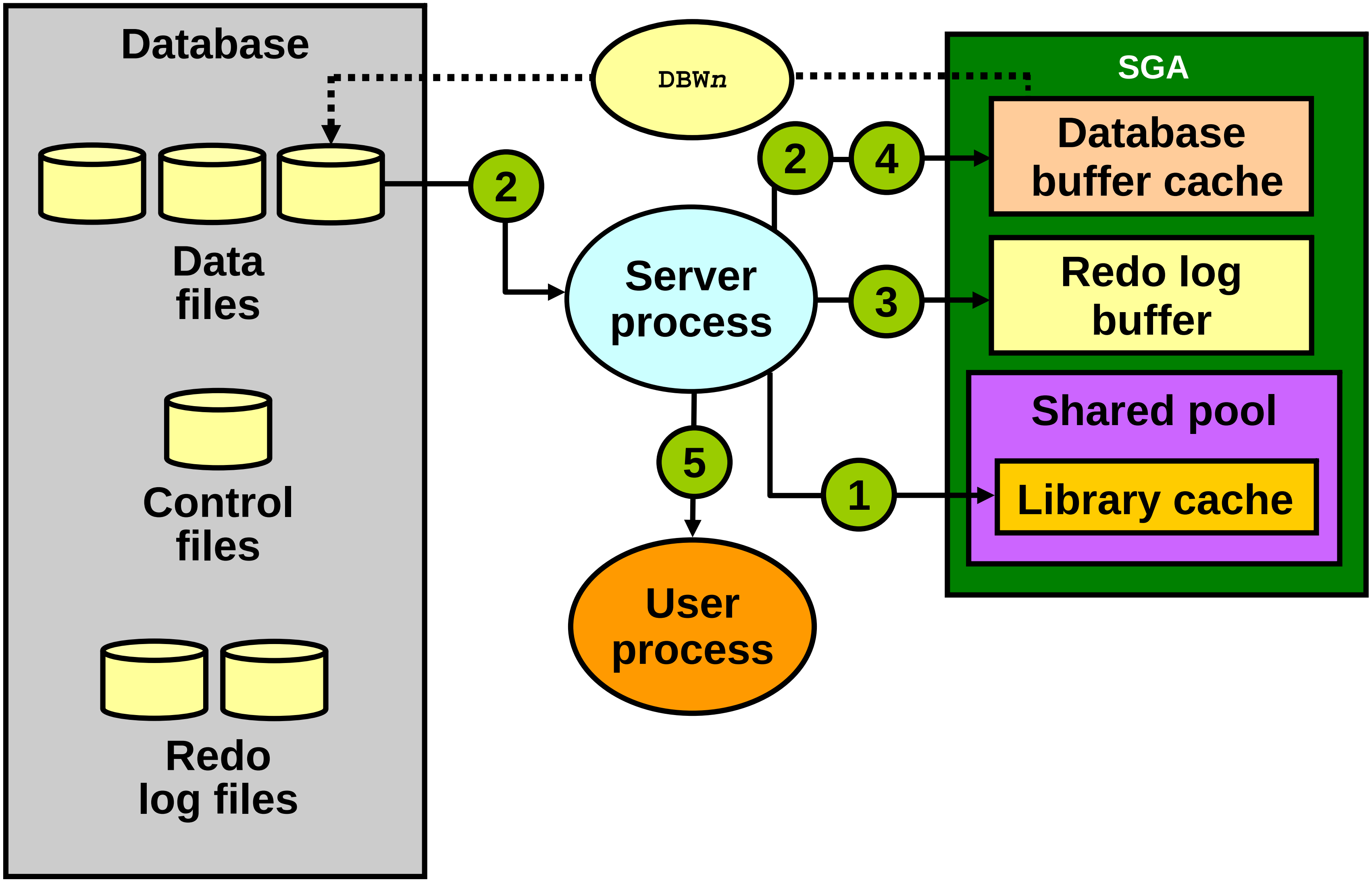


Shared Pool

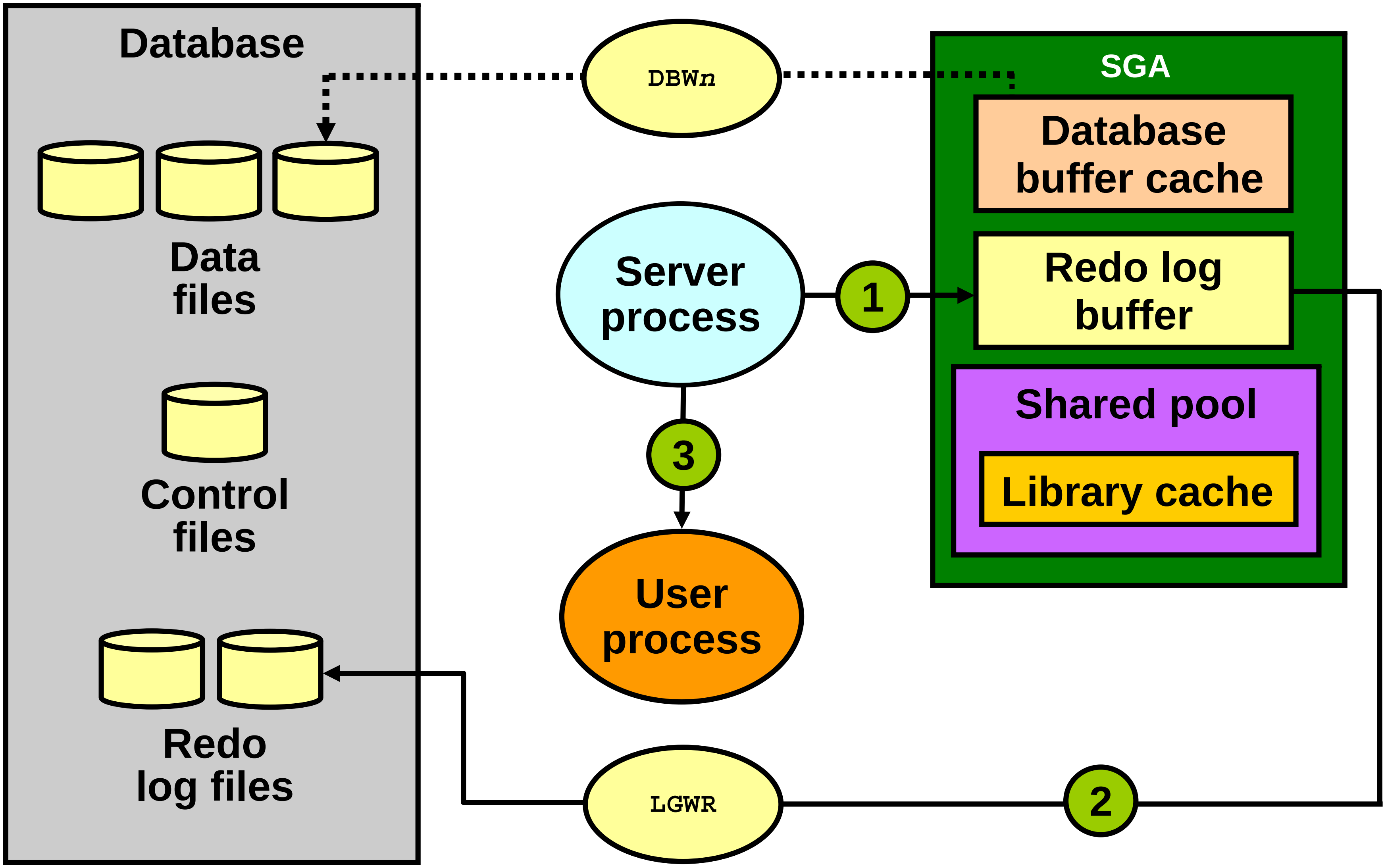
- Is part of the SGA
- Contains:
  - Library cache
    - Shared parts of SQL and PL/SQL statements
  - Data dictionary cache
  - Result cache:
    - SQL queries
    - PL/SQL functions
  - Control structures
    - Locks



Processing a DML Statement: Example

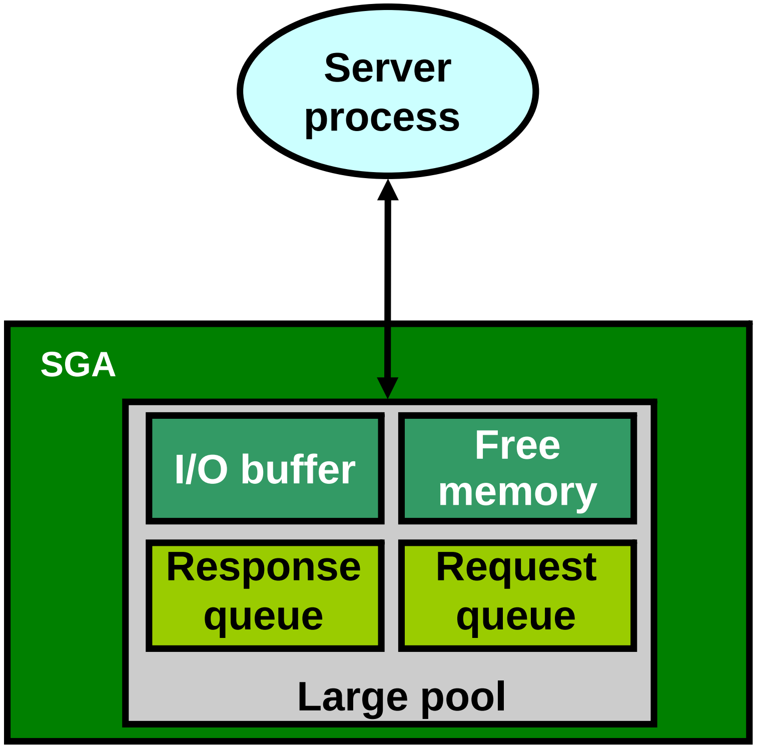


`COMMIT` Processing: Example



Large Pool

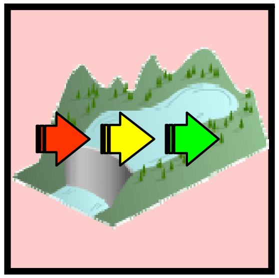
- Provides large memory allocations for:
  - Session memory for the shared server and Oracle XA interface
  - Parallel execution buffers
  - I/O server processes
  - Oracle Database backup and restore operations
- Optional pool better suited when using the following:
  - Parallel execution
  - Recovery Manager
  - Shared server



Java Pool and Streams Pool

- Java pool memory is used in server memory for all session-specific Java code and data in the JVM.
- Streams pool memory is used exclusively by Oracle Streams to:
  - Store buffered queue messages
  - Provide memory for Oracle Streams processes

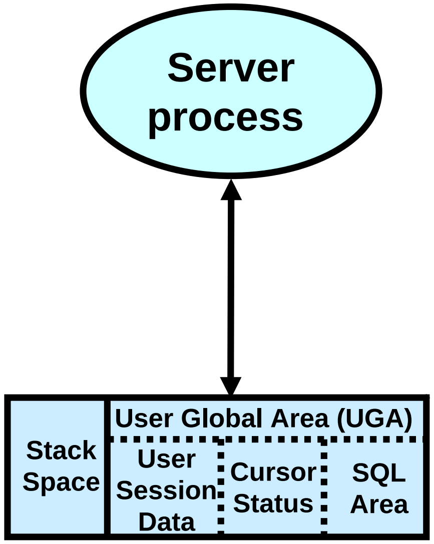
Java pool



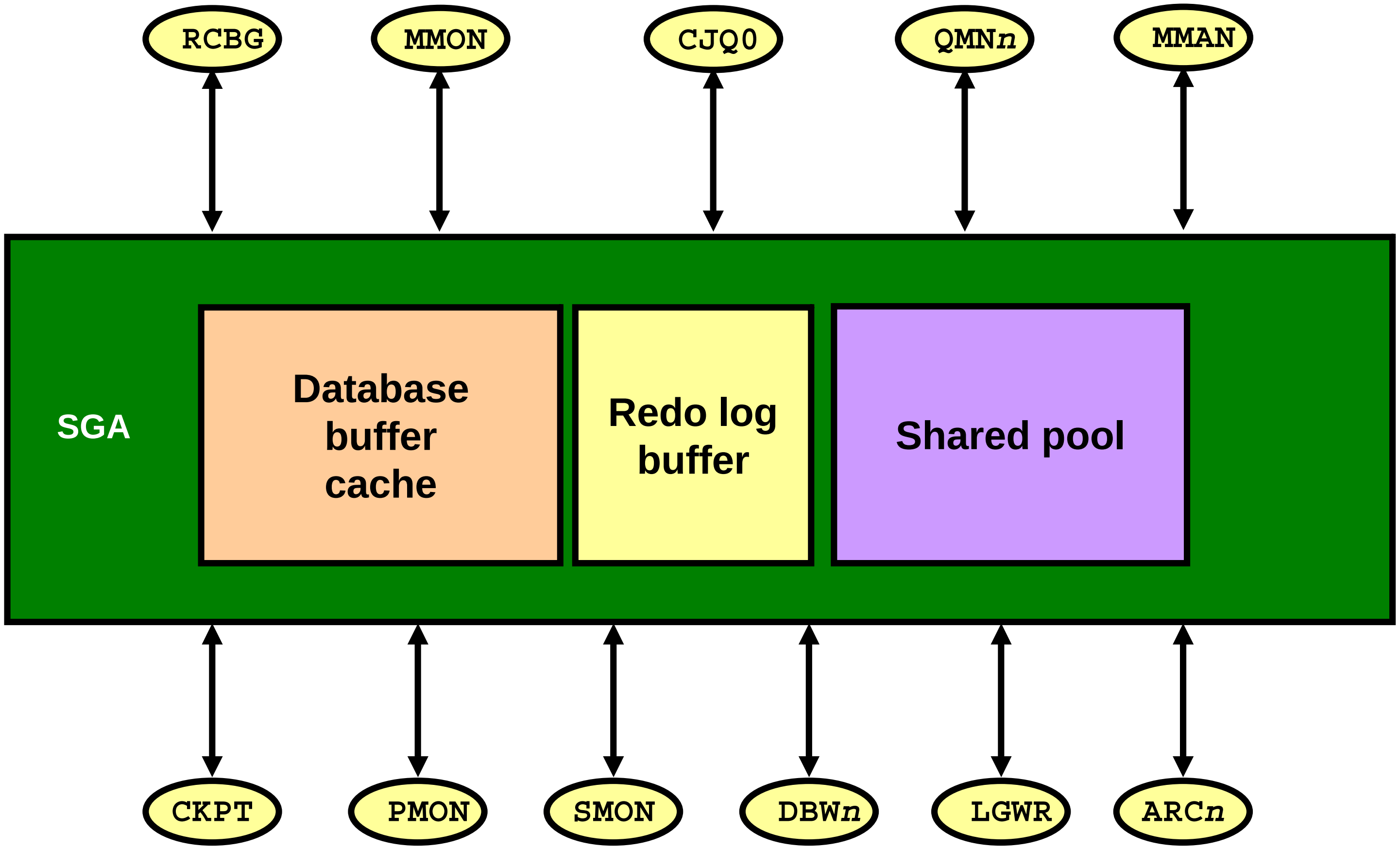
Streams pool

Program Global Area (PGA)

- PGA is a memory area that contains:
  - Session information
  - Cursor information
  - SQL execution work areas:
    - Sort area
    - Hash join area
    - Bitmap merge area
    - Bitmap create area
- Work area size influences SQL performance.
- Work areas can be automatically or manually managed.

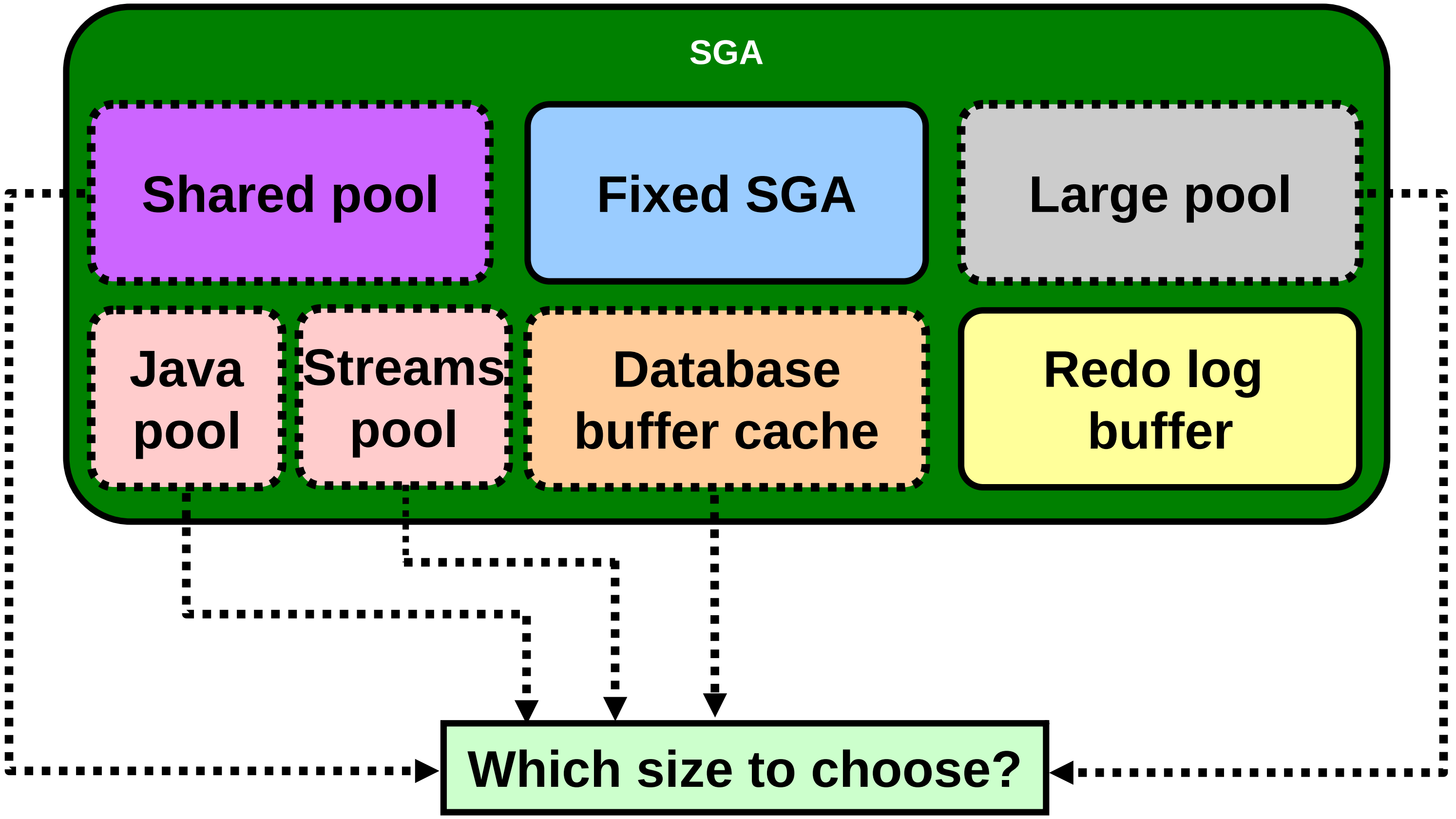


Background Process Roles



Automatic Shared Memory Management

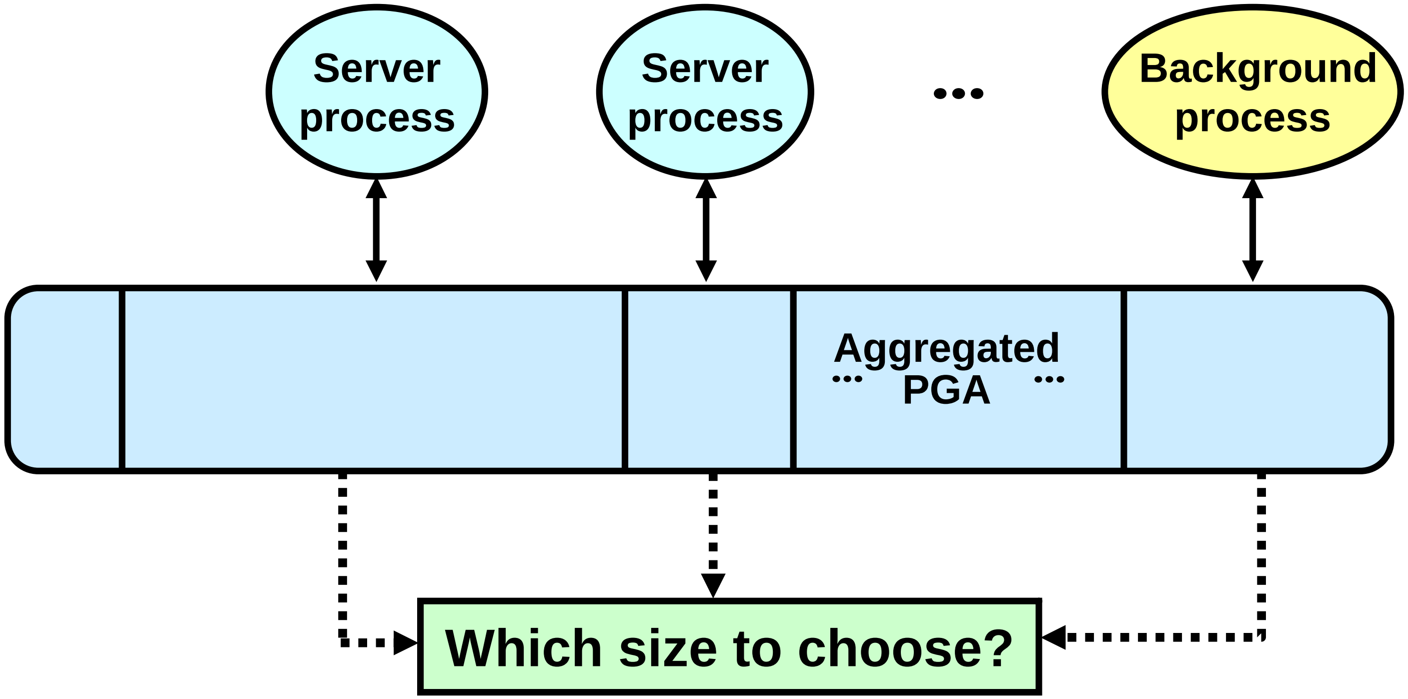
`SGA_TARGET + STATISTICS_LEVEL`



⬚ Automatically tuned SGA components

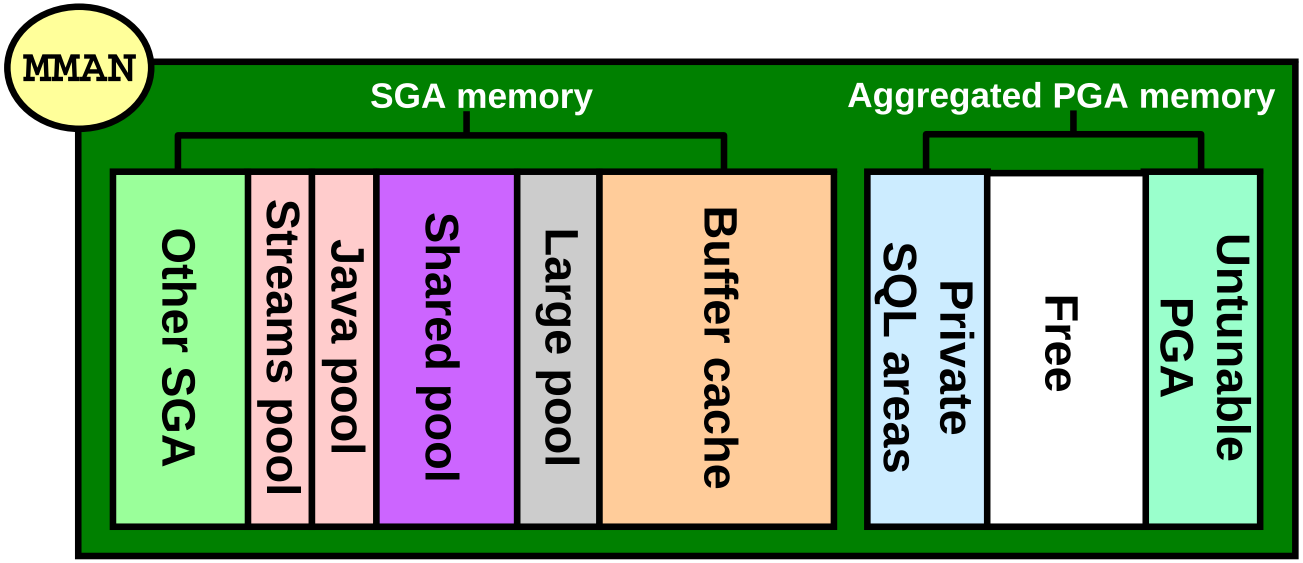
Automated SQL Execution Memory Management

`PGA_AGGREGATE_TARGET`



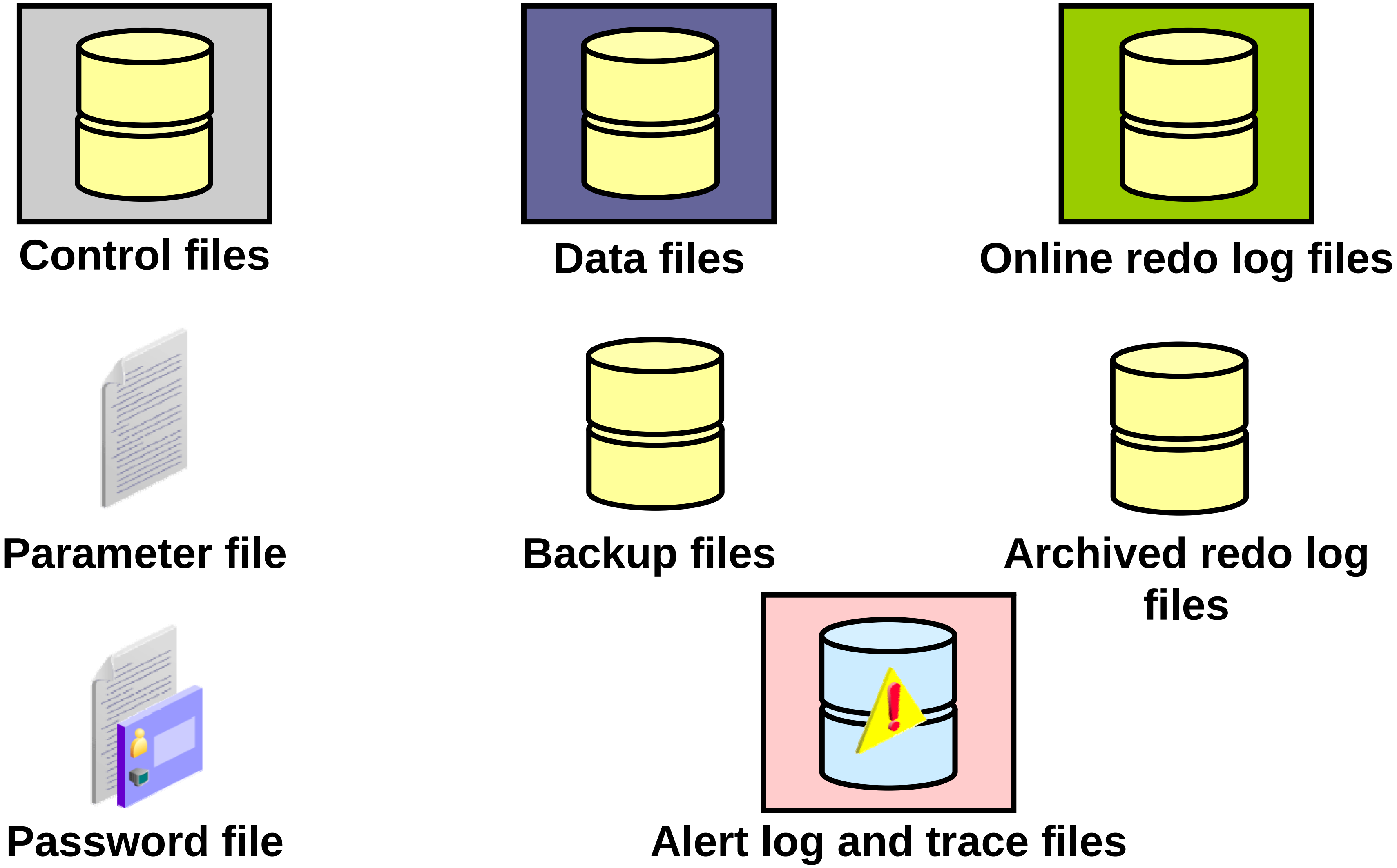
Automatic Memory Management

- Sizing of each memory component is vital for SQL execution performance.
- It is difficult to manually size each component.
- Automatic memory management automates memory allocation of each SGA component and aggregated PGA.



MEMORY_TARGET + STATISTICS_LEVEL

Database Storage Architecture



Control files

Data files

Online redo log files

Parameter file

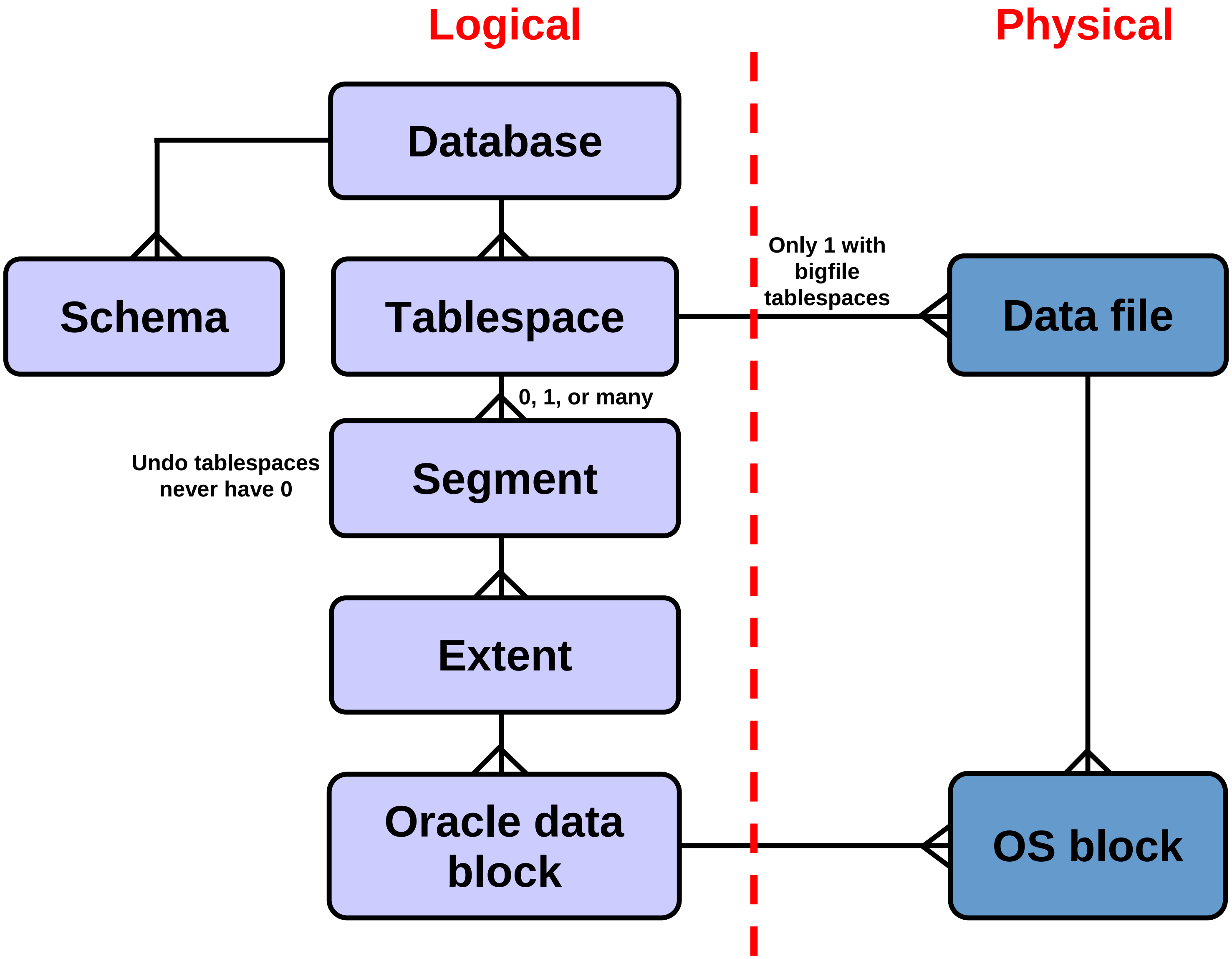
Backup files

Archived redo log files

Password file

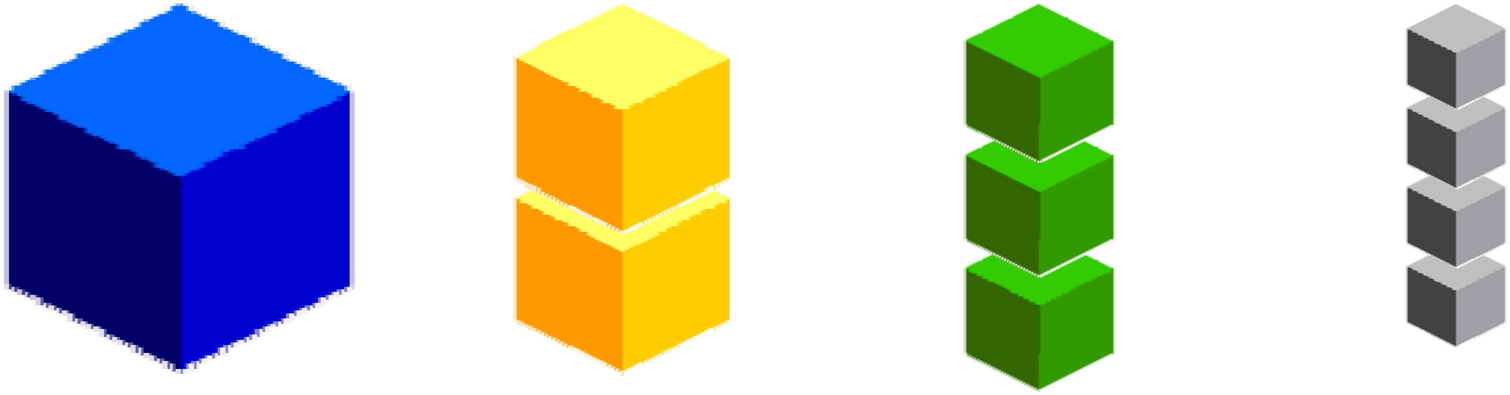
Alert log and trace files

Logical and Physical Database Structures



Segments, Extents, and Blocks

- Segments exist in a tablespace.
- Segments are collections of extents.
- Extents are collections of data blocks.
- Data blocks are mapped to disk blocks.



Segment

Extents

Data blocks

Disk blocks

`SYSTEM` and `SYSAUX` Tablespaces

- The `SYSTEM` and `SYSAUX` tablespaces are mandatory tablespaces that are created at the time of database creation. They must be online.
- The `SYSTEM` tablespace is used for core functionality (for example, data dictionary tables).
- The auxiliary `SYSAUX` tablespace is used for additional database components (such as the Enterprise Manager Repository).

Summary

In this lesson, you should have learned how to:

- List the major architectural components of the Oracle Database server
- Explain memory structures
- Describe background processes
- Correlate logical and physical storage structures

Practice 1: Overview

This practice covers the following topics:

- Listing the different components of an Oracle Database server
- Looking at some instance and database components directly on your machine

2

Introduction to SQL Tuning

Objectives

After completing this lesson, you should be able to:

- Describe what attributes of a SQL statement can make it perform poorly
- List the Oracle tools that can be used to tune SQL
- List the tuning tasks

Reasons for Inefficient SQL Performance

- Stale or missing optimizer statistics
- Missing access structures
- Suboptimal execution plan selection
- Poorly constructed SQL

Inefficient SQL: Examples

1

```
SELECT COUNT(*) FROM products p
WHERE prod_list_price <
  1.15 * (SELECT avg(unit_cost) FROM costs c
          WHERE c.prod_id = p.prod_id)
```

2

```
SELECT * FROM job_history jh, employees e
  WHERE substr(to_char(e.employee_id),2) =
  substr(to_char(jh.employee_id),2)
```

3

```
SELECT * FROM orders WHERE order_id_char = 1205
```

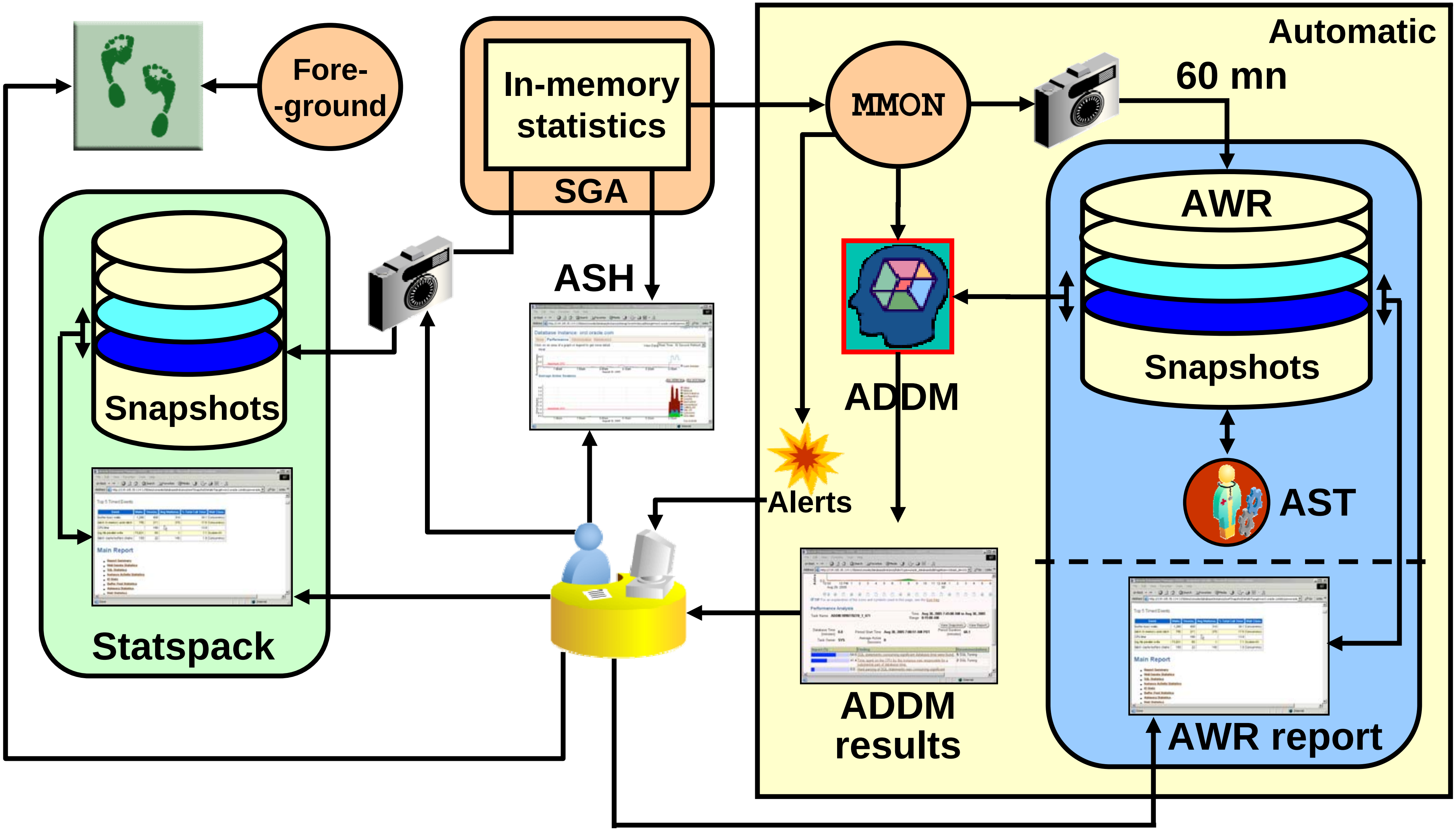
4

```
SELECT * FROM employees
   WHERE to_char(salary) = :sal
```

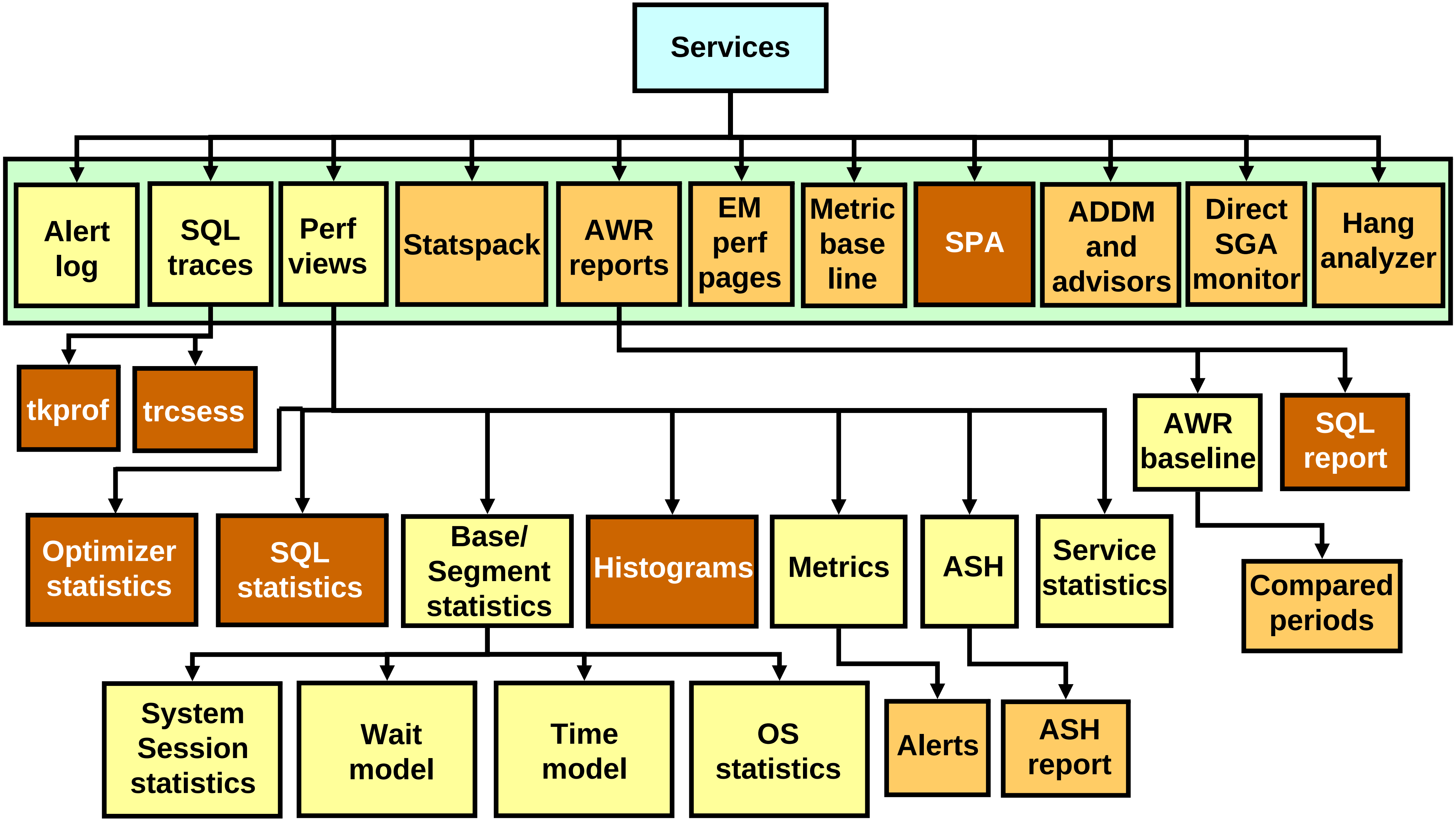
5

```
SELECT * FROM parts_old
UNION
SELECT * FROM parts_new
```

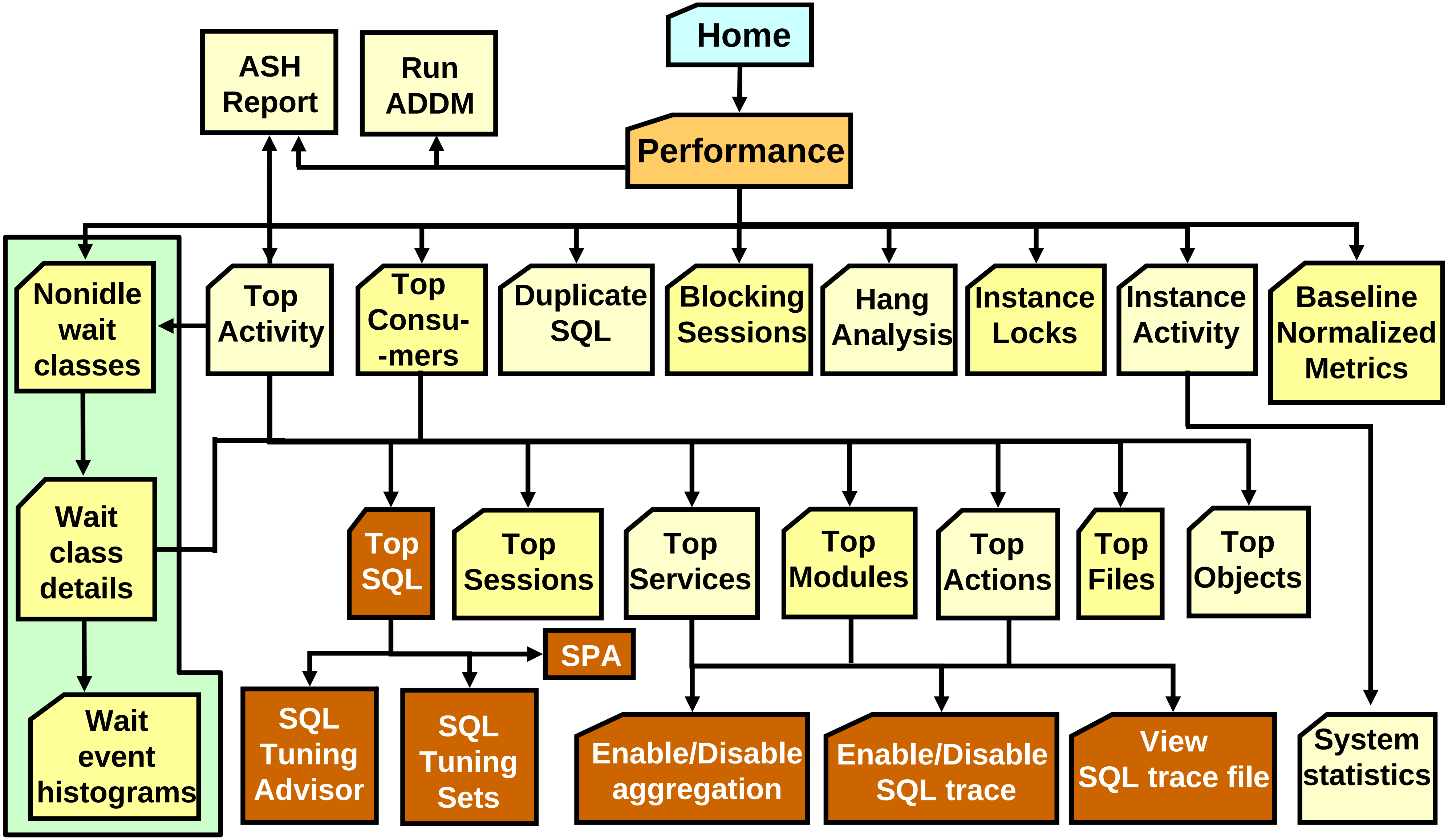
Performance Monitoring Solutions



Monitoring and Tuning Tools: Overview



EM Performance Pages for Reactive Tuning



Tuning Tools: Overview

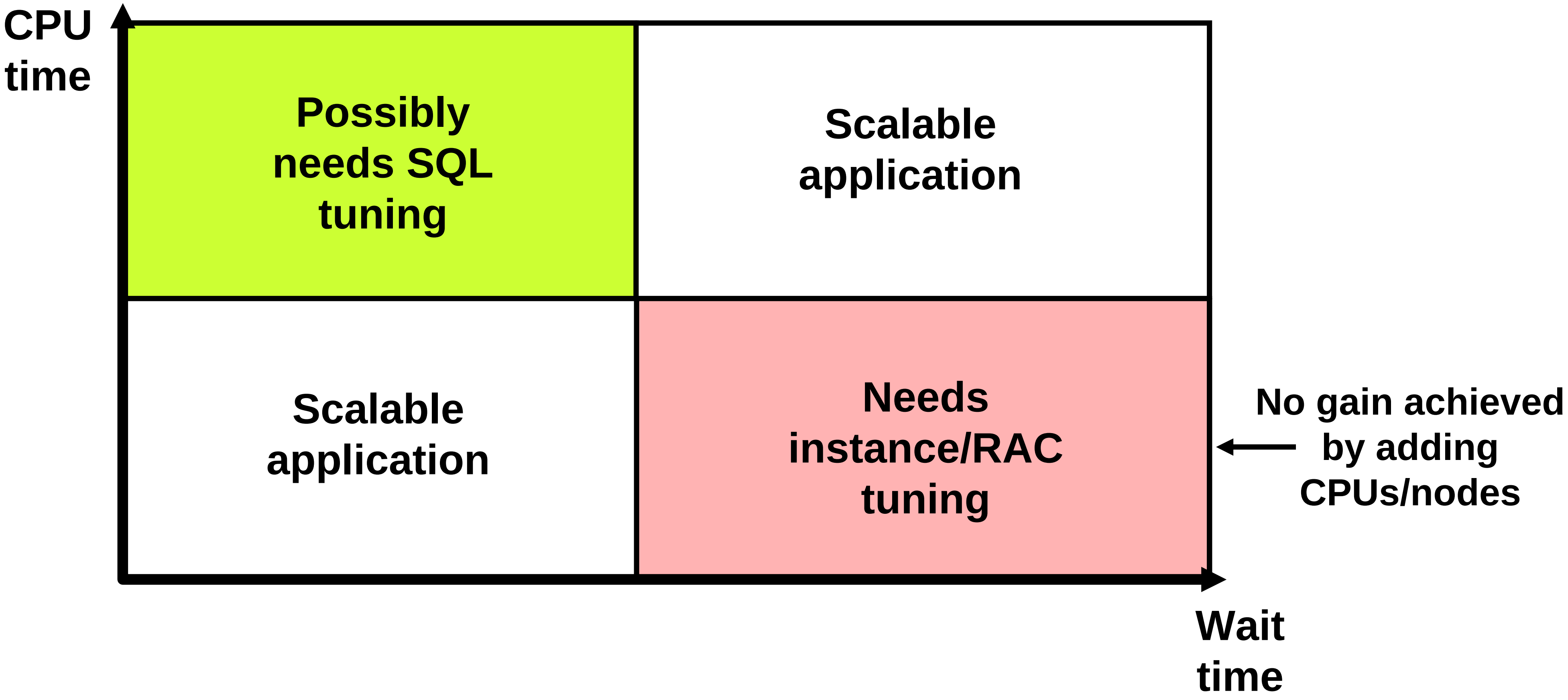
- Automatic Database Diagnostic Monitor (ADDM)
- SQL Tuning Advisor
- SQL Tuning Sets
- SQL Access Advisor
- SQL Performance Analyzer
- SQL Monitoring
- SQL Plan Management

SQL Tuning Tasks: Overview

- Identifying high-load SQL
- Gathering statistics
- Generating system statistics
- Rebuilding existing indexes
- Maintaining execution plans
- Creating new index strategies

CPU and Wait Time Tuning Dimensions

Scalability is a system’s ability to process more workload with a proportional increase in system resource use.



CPU time

Possibly needs SQL tuning

Scalable application

Scalable application

Needs instance/RAC tuning

No gain achieved by adding CPUs/nodes

Wait time

Scalability with Application Design, Implementation, and Configuration

Applications have a significant impact on scalability.

- Poor schema design can cause expensive SQL that does not scale.
- Poor transaction design can cause locking and serialization problems.
- Poor connection management can cause unsatisfactory response times.

Common Mistakes on Customer Systems

1. Bad connection management
2. Bad use of cursors and the shared pool
3. Excess of resources consuming SQL statements
4. Use of nonstandard initialization parameters
5. Poor database disk configuration
6. Redo log setup problems
7. Excessive serialization
8. Inappropriate full table scans
9. Large number of space-management or parse-related generated SQL statements
10. Deployment and migration errors

EDUCATE USERS

Proactive Tuning Methodology

- Simple design
- Data modeling
- Tables and indexes
- Using views
- Writing efficient SQL
- Cursor sharing
- Using bind variables

Simplicity in Application Design

- Simple tables
- Well-written SQL
- Indexing only as required
- Retrieving only required information

Data Modeling

- Accurately represent business practices
- Focus on the most frequent and important business transactions
- Use modeling tools
- Appropriately normalize data (OLTP versus DW)

Table Design

- Compromise between flexibility and performance:
  - Principally normalize
  - Selectively denormalize
- Use Oracle performance and management features:
  - Default values
  - Constraints
  - Materialized views
  - Clusters
  - Partitioning
- Focus on business-critical tables

Index Design

- Create indexes on the following:
  - Primary key (can be automatically created)
  - Unique key (can be automatically created)
  - Foreign keys (good candidates)
- Index data that is frequently queried (select list).
- Use SQL as a guide to index design.

Using Views

- Simplifies application design
- Is transparent to the developer
- Can cause suboptimal execution plans

SQL Execution Efficiency

- Good database connectivity
- Minimizing parsing
- Share cursors
- Using bind variables

Writing SQL to Share Cursors

- Create generic code using the following:
  - Stored procedures and packages
  - Database triggers
  - Any other library routines and procedures
- Write to format standards (improves readability):
  - Case
  - White space
  - Comments
  - Object references
  - Bind variables

Performance Checklist

- Set initialization parameters and storage options.
- Verify resource usage of SQL statements.
- Validate connections by middleware.
- Verify cursor sharing.
- Validate migration of all required objects.
- Verify validity and availability of optimizer statistics.

Summary

In this lesson, you should have learned how to:

- Describe what attributes of a SQL statement can make it perform poorly
- List the Oracle tools that can be used to tune SQL
- List the tuning tasks

Practice 2: Overview

This practice covers the following topics:

- Rewriting queries for better performance
- Rewriting applications for better performance

3

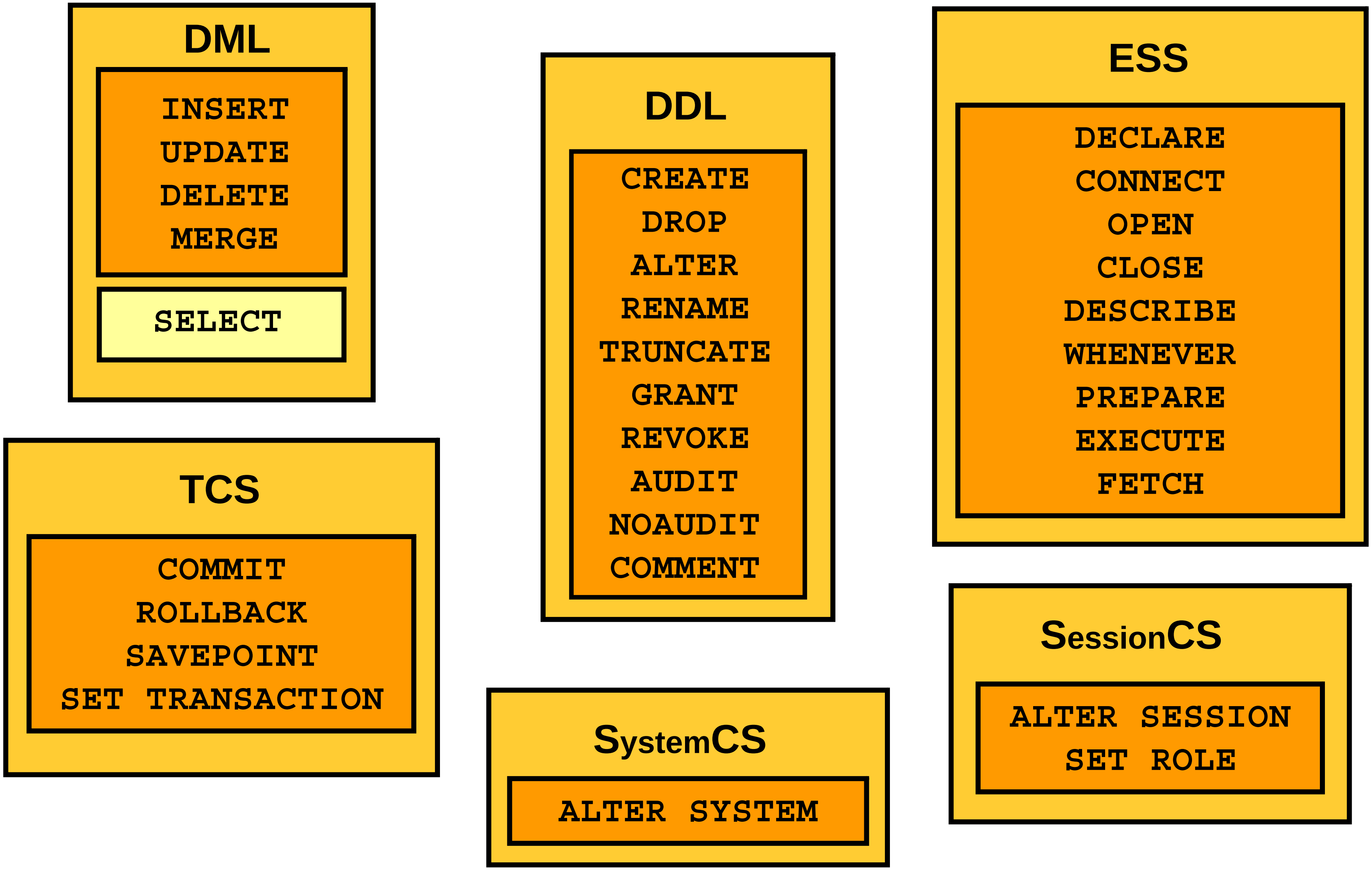
Introduction to the Optimizer

Objectives

After completing this lesson, you should be able to:

- Describe the execution steps of a SQL statement
- Discuss the need for an optimizer
- Explain the various phases of optimization
- Control the behavior of the optimizer

Structured Query Language



DML

INSERT
UPDATE
DELETE
MERGE

SELECT

TCS

COMMIT
ROLLBACK
SAVEPOINT
SET TRANSACTION

DDL

CREATE
DROP
ALTER
RENAME
TRUNCATE
GRANT
REVOKE
AUDIT
NOAUDIT
COMMENT

SystemCS

ALTER SYSTEM

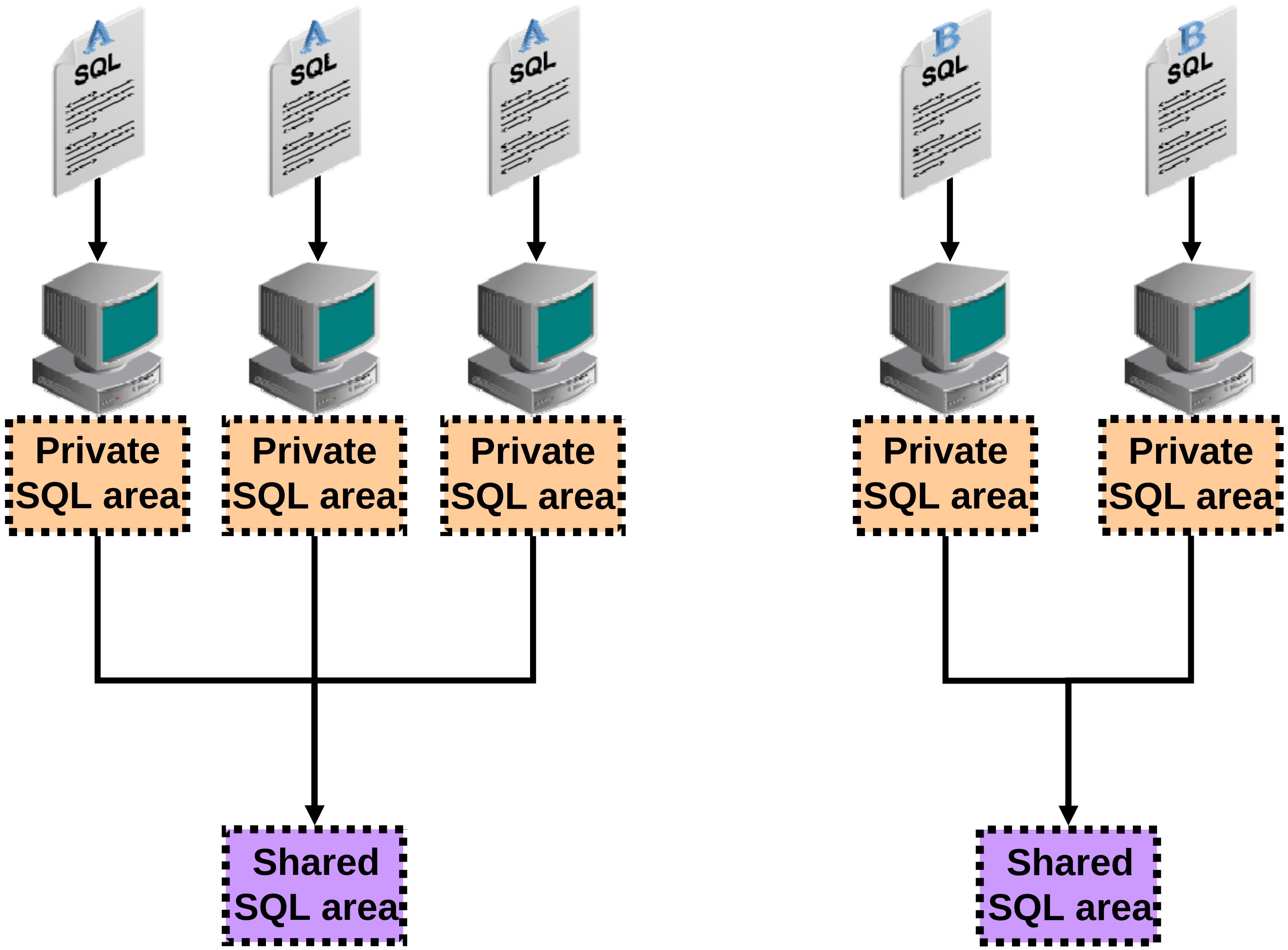
ESS

DECLARE
CONNECT
OPEN
CLOSE
DESCRIBE
WHENEVER
PREPARE
EXECUTE
FETCH

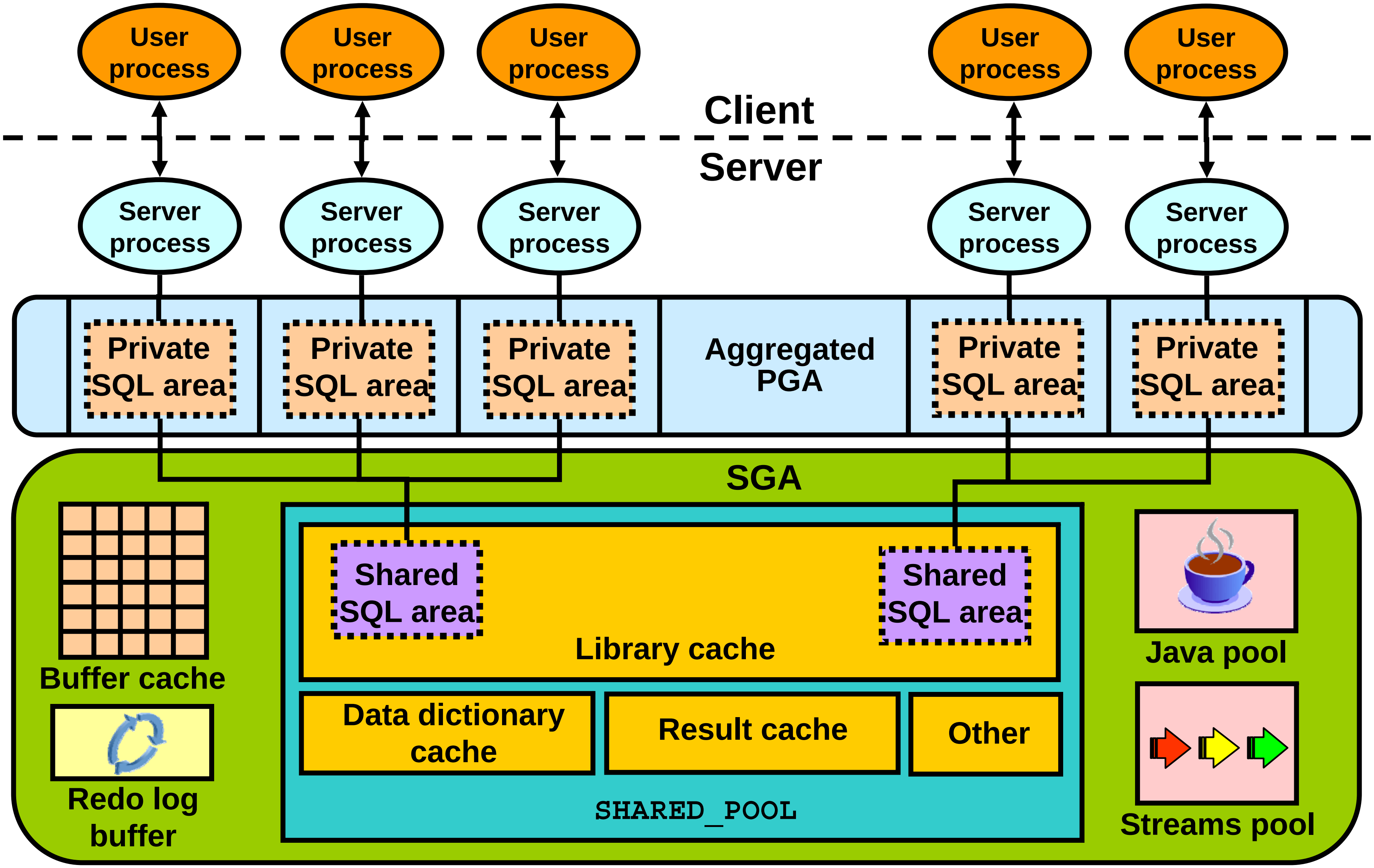
SessionCS

ALTER SESSION
SET ROLE

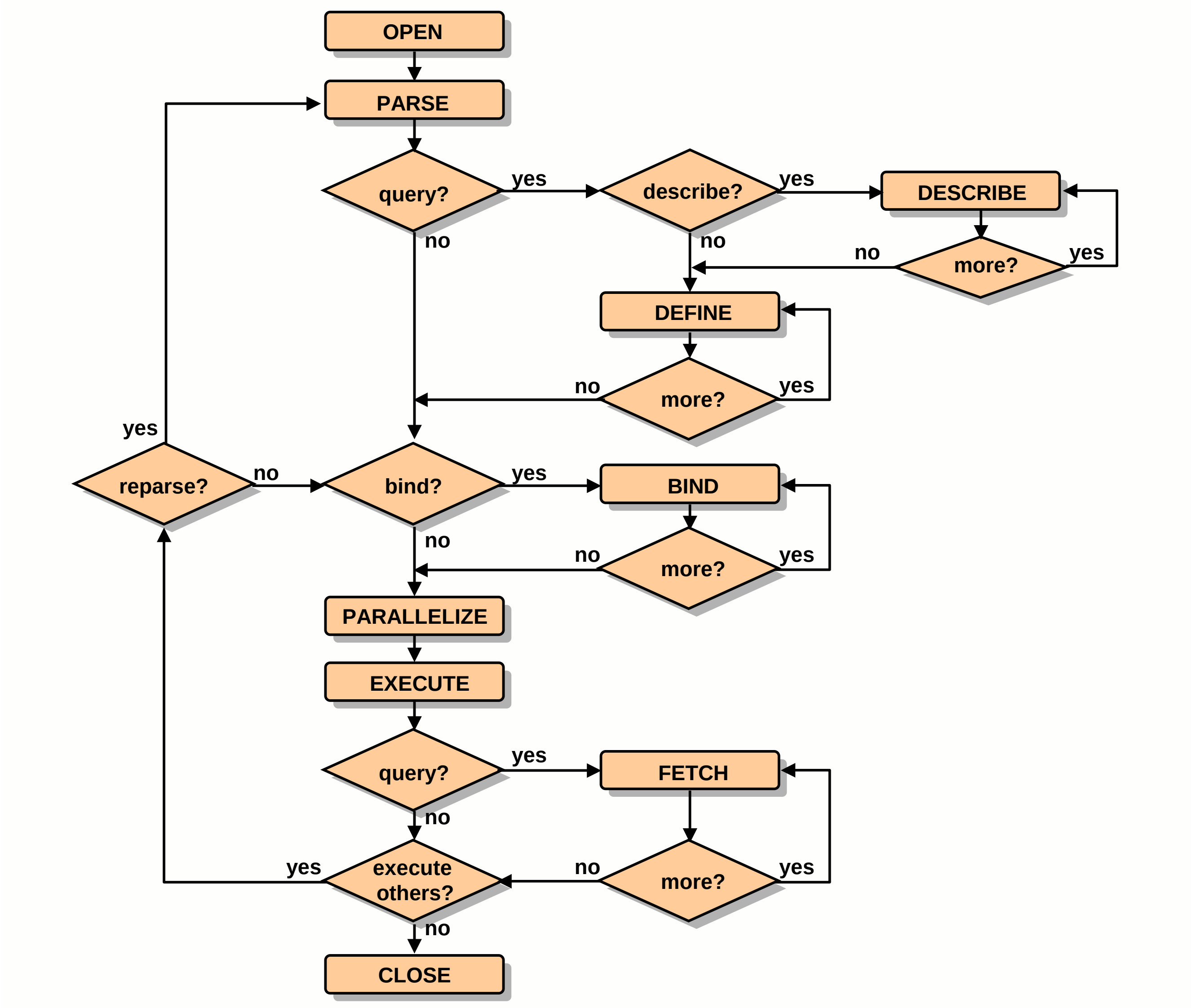
SQL Statement Representation



SQL Statement Implementation



SQL Statement Processing: Overview



SQL Statement Processing: Steps

1. Create a cursor.
2. Parse the statement.
3. Describe query results.
4. Define query output.
5. Bind variables.
6. Parallelize the statement.
7. Execute the statement.
8. Fetch rows of a query.
9. Close the cursor.

Step 1: Create a Cursor

- A cursor is a handle or name for a private SQL area.
- It contains information for statement processing.
- It is created by a program interface call in expectation of a SQL statement.
- The cursor structure is independent of the SQL statement that it contains.

Step 2: Parse the Statement

- Statement passed from the user process to the Oracle instance
- Parsed representation of SQL created and moved into the shared SQL area if there is no identical SQL in the shared SQL area
- Can be reused if identical SQL exists

Steps 3 and 4: Describe and Define

- The describe step provides information about the select list items; it is relevant when entering dynamic queries through an OCI application.
- The define step defines location, size, and data type information required to store fetched values in variables.

Steps 5 and 6: Bind and Parallelize

- Bind any bind values:
  - Enables memory address to store data values
  - Allows shared SQL even though bind values may change
- Parallelize the statement:
  - `SELECT`
  - `INSERT`
  - `UPDATE`
  - `MERGE`
  - `DELETE`
  - `CREATE`
  - `ALTER`

Steps 7 Through 9

- Execute:
  - Drives the SQL statement to produce the desired results
- Fetch rows:
  - Into defined output variables
  - Query results returned in table format
  - Array fetch mechanism
- Close the cursor.

SQL Statement Processing PL/SQL: Example

```
SQL> variable c1 number
SQL> execute :c1 := dbms_sql.open_cursor;
```

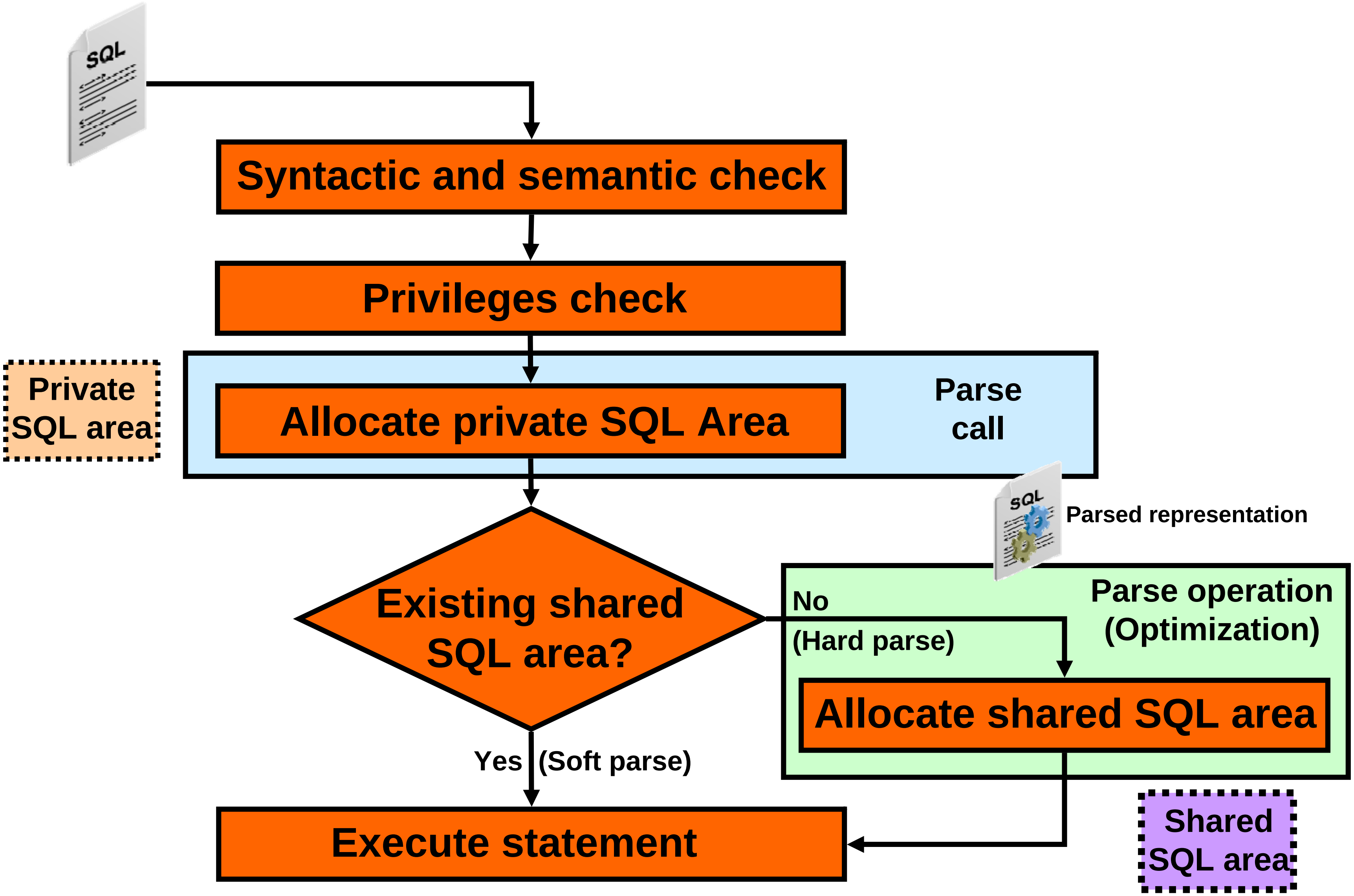
```
SQL> variable b1 varchar2
SQL> execute dbms_sql.parse
  2   (:c1
  3   ,'select null from dual where dummy = :b1'
  4   ,dbms_sql.native);
```

```
SQL> execute :b1:='Y';
SQL> exec dbms_sql.bind_variable(:c1,':b1',:b1);
```

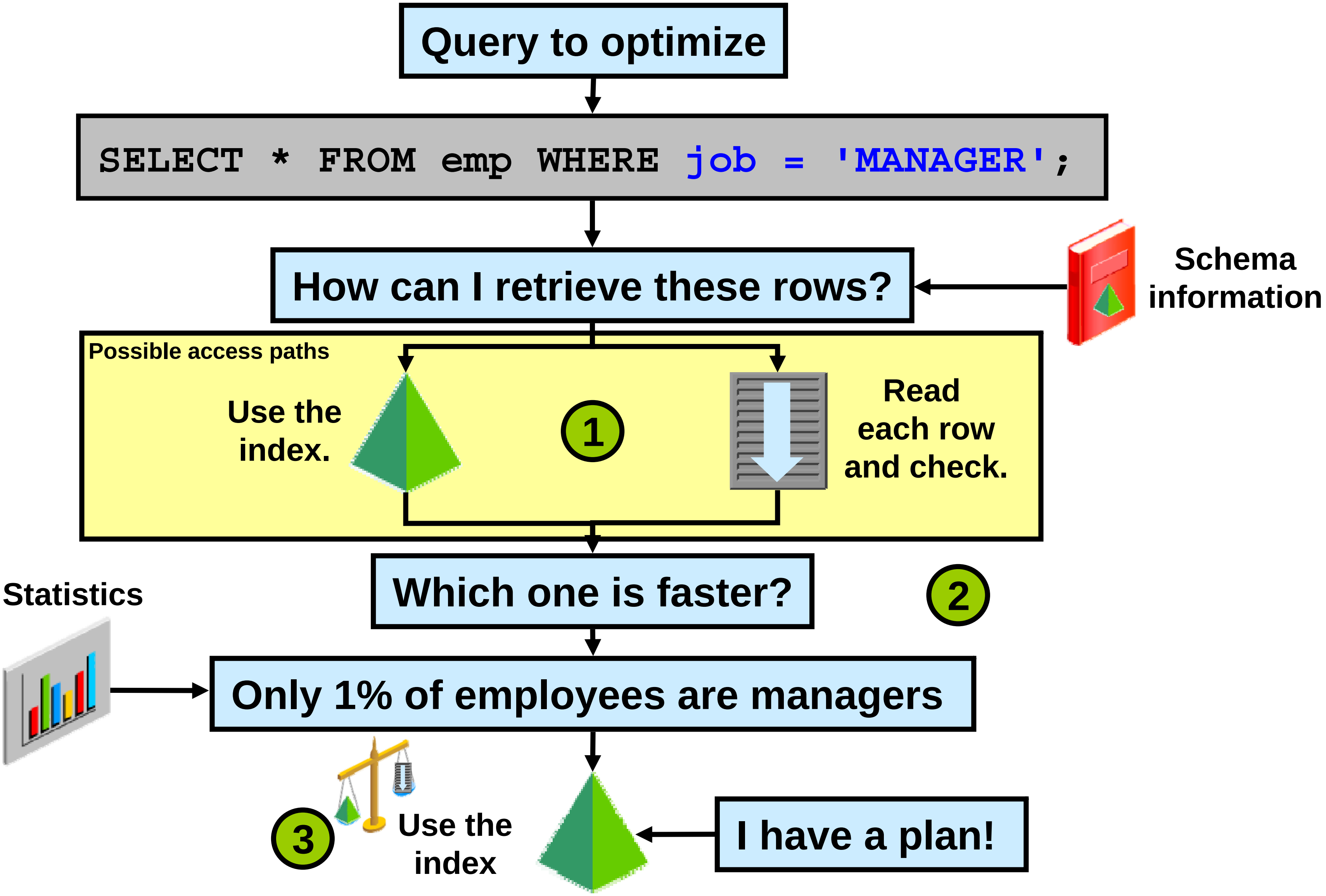
```
SQL> variable r number
SQL> execute :r := dbms_sql.execute(:c1);
```

```
SQL> variable r number
SQL> execute :r := dbms_sql.close_cursor(:c1);
```

SQL Statement Parsing: Overview



Why Do You Need an Optimizer?



Query to optimize

```
SELECT * FROM emp WHERE job = 'MANAGER';
```

How can I retrieve these rows?

Schema information

Possible access paths

Use the index.

1

Read each row and check.

Which one is faster?

2

Statistics

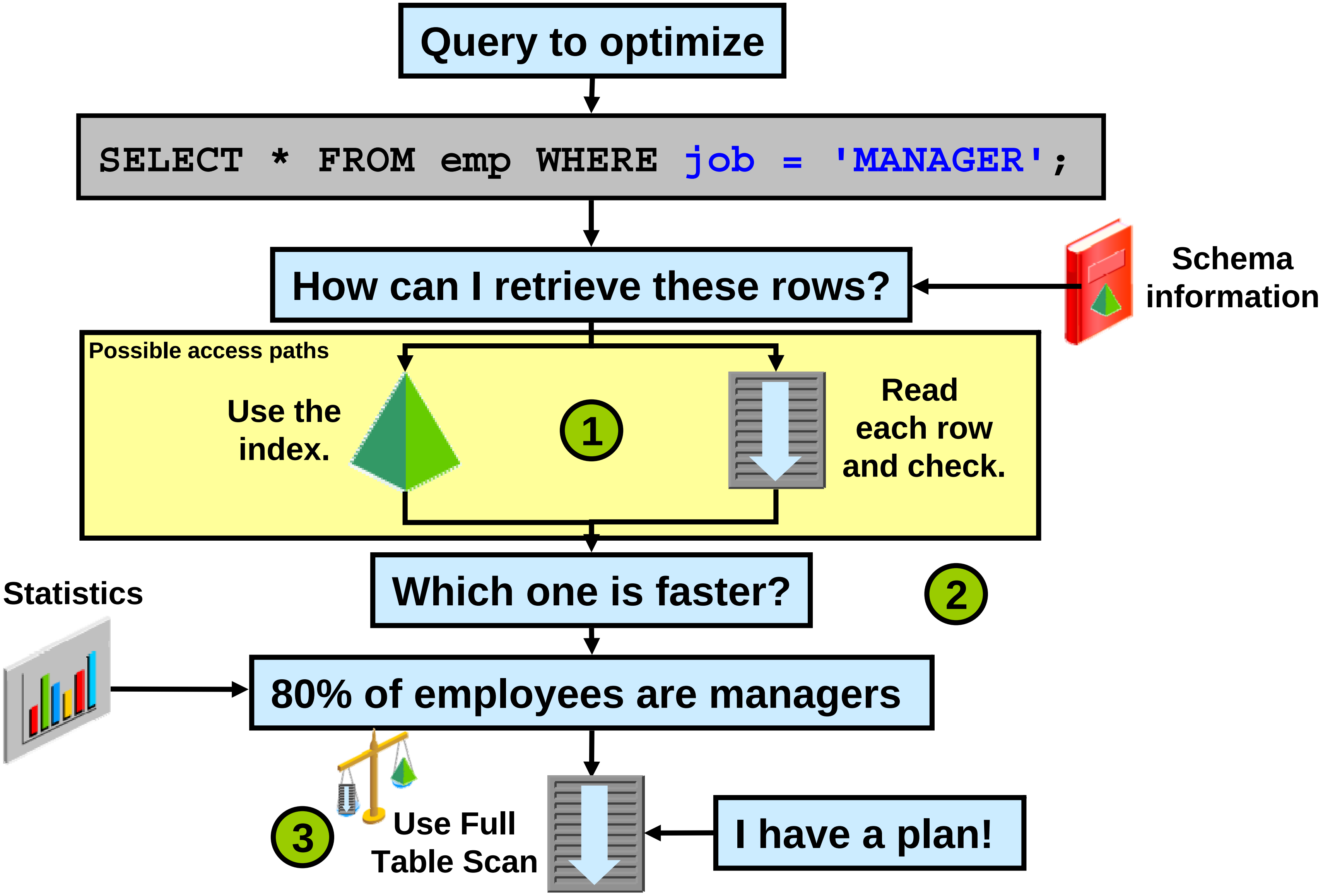
Only 1% of employees are managers

3

Use the index

I have a plan!

Why Do You Need an Optimizer?



Query to optimize

```
SELECT * FROM emp WHERE job = 'MANAGER';
```

How can I retrieve these rows?

Schema information

Possible access paths

1

Use the index.

Read each row and check.

Which one is faster?

2

Statistics

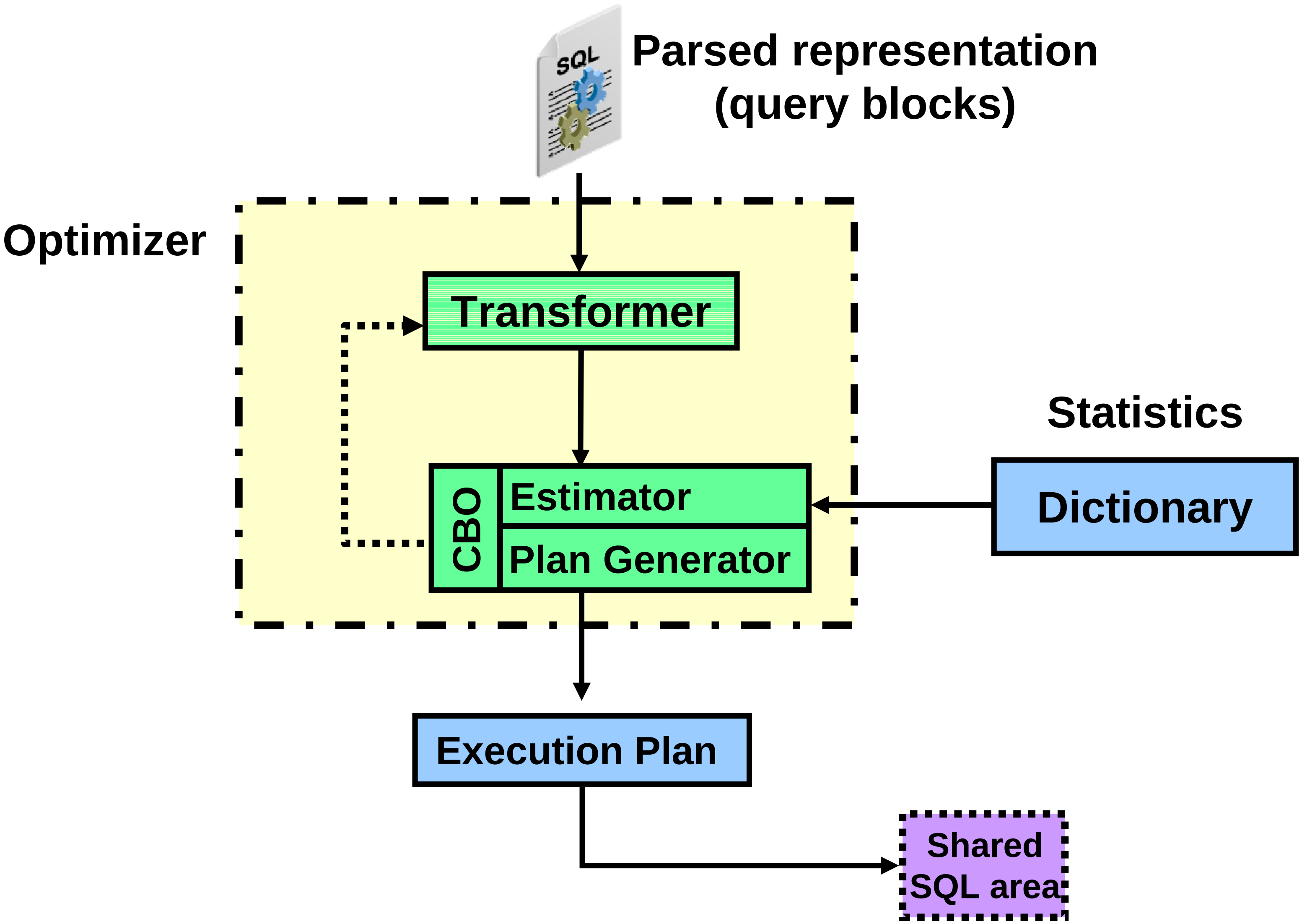
80% of employees are managers

3

Use Full Table Scan

I have a plan!

Optimization During Hard Parse Operation



Transformer: OR Expansion Example

▲ **B*-tree Index**

- Original query:

```
SELECT *
   FROM emp
   WHERE job = 'CLERK' OR deptno = 10;
```

- Equivalent transformed query:

```
SELECT *
   FROM emp
   WHERE job = 'CLERK'
UNION ALL
SELECT *
   FROM emp
   WHERE deptno = 10 AND job <> 'CLERK';
```

Transformer: Subquery Unnesting Example

- Original query:

```
SELECT *
  FROM accounts
  WHERE custno IN
       (SELECT custno FROM customers);
```

- Equivalent transformed query:

```
SELECT accounts.*
  FROM accounts, customers
  WHERE accounts.custno = customers.custno;
```

Primary or unique key

Transformer: View Merging Example

▲ Index

- Original query:

```
CREATE VIEW emp_10 AS
     SELECT empno, ename, job, sal, comm, deptno
     FROM emp
     WHERE deptno = 10;
```

```
SELECT empno FROM emp_10 WHERE empno > 7800;
```

- Equivalent transformed query:

```
SELECT empno
  FROM emp
  WHERE deptno = 10 AND empno > 7800;
```

Transformer: Predicate Pushing Example

- Original query:

▲ Index

```
CREATE VIEW two_emp_tables AS
SELECT empno, ename, job, sal, comm, deptno▲ FROM emp1
 UNION
SELECT empno, ename, job, sal, comm, deptno▲ FROM emp2;
```

```
SELECT ename FROM two_emp_tables WHERE deptno = 20;
```

- Equivalent transformed query:

```
SELECT ename
  FROM ( SELECT empno, ename, job,sal, comm, deptno
           FROM emp1 WHERE deptno = 20
          UNION
         SELECT empno, ename, job,sal, comm, deptno
           FROM emp2 WHERE deptno = 20 );
```

Transformer: Transitivity Example

- Original query:

▲ **Index**

```
SELECT *
 FROM emp, dept
 WHERE emp.deptno = 20 AND emp.deptno = dept.deptno;
```

- Equivalent transformed query:

```
SELECT *
 FROM emp, dept
 WHERE emp.deptno = 20 AND emp.deptno = dept.deptno
   AND dept.deptno ▲ = 20;
```

Cost-Based Optimizer

- Piece of code:
  - Estimator
  - Plan generator
- Estimator determines cost of optimization suggestions made by the plan generator:
  - Cost: Optimizer's best estimate of the number of standardized I/Os made to execute a particular statement optimization
- Plan generator:
  - Tries out different statement optimization techniques
  - Uses the estimator to cost each optimization suggestion
  - Chooses the best optimization suggestion based on cost
  - Generates an execution plan for best optimization

Estimator: Selectivity

$$\text{Selectivity} = \frac{\text{Number of rows satisfying a condition}}{\text{Total number of rows}}$$

- Selectivity is the estimated proportion of a row set retrieved by a particular predicate or combination of predicates.
- It is expressed as a value between 0.0 and 1.0:
  - High selectivity: Small proportion of rows
  - Low selectivity: Big proportion of rows
- Selectivity computation:
  - If no statistics: Use dynamic sampling
  - If no histograms: Assume even distribution of rows
- Statistic information:
  - `DBA_TABLES` and `DBA_TAB_STATISTICS` (`NUM_ROWS`)
  - `DBA_TAB_COL_STATISTICS` (`NUM_DISTINCT`, `DENSITY`, `HIGH/LOW_VALUE`,...)

Estimator: Cardinality

Cardinality = Selectivity * Total number of rows

- Expected number of rows retrieved by a particular operation in the execution plan
- Vital figure to determine join, filters, and sort costs
- Simple example:

```
SELECT days FROM courses WHERE dev_name = 'ANGEL';
```

  – The number of distinct values in `DEV_NAME` is 203.
  – The number of rows in `COURSES` (original cardinality) is 1018.
  – Selectivity = 1/203 = 4.926*e-03
  – Cardinality = (1/203)*1018 = 5.01 (rounded off to 6)

Estimator: Cost

- Cost is the optimizer’s best estimate of the number of standardized I/Os it takes to execute a particular statement.
- Cost unit is a standardized single block random read:
  - 1 cost unit = 1 SRds
- The cost formula combines three different costs units into standard cost units.

$$\texttt{Cost=}\frac{\overset{\text{Single block I/O cost}}{\boxed{\texttt{\#SRds*sreadtim}}} + \overset{\text{Multiblock I/O cost}}{\boxed{\texttt{\#MRds*mreadtim}}} + \overset{\text{CPU cost}}{\boxed{\texttt{\#CPUCycles/cpuspeed}}}}{\texttt{sreadtim}}$$

`#SRds`: Number of single block reads
`#MRds`: Number of multiblock reads
`#CPUCycles`: Number of CPU Cycles

`Sreadtim`: Single block read time
`Mreadtim`: Multiblock read time
`Cpuspeed`: Millions instructions per second

Plan Generator

```
select e.last_name, c.loc_id
from   employees e, classes c  where  e.emp_id = c.instr_id;
```

```
Join order[1]:  DEPARTMENTS[D]#0  EMPLOYEES[E]#1
 NL Join:  Cost: 41.13  Resp: 41.13  Degree: 1
 SM cost: 8.01
 HA cost: 6.51
Best:: JoinMethod: Hash
Cost: 6.51  Degree: 1  Resp: 6.51  Card: 106.00
Join order[2]:  EMPLOYEES[E]#1  DEPARTMENTS[D]#0
 NL Join:  Cost: 121.24  Resp: 121.24  Degree: 1
 SM cost: 8.01
 HA cost: 6.51
Join order aborted
Final cost for query block SEL$1 (#0)
All Rows Plan:
Best join order: 1
+-----------------------------------------------------------------+
| Id  | Operation            | Name        | Rows  | Bytes | Cost  |
+-----------------------------------------------------------------+
| 0   | SELECT STATEMENT     |             |       |       |     7 |
| 1   |  HASH JOIN           |             |   106 |  6042 |     7 |
| 2   |   TABLE ACCESS FULL  | DEPARTMENTS|    27 |   810 |     3 |
| 3   |   TABLE ACCESS FULL  | EMPLOYEES   |   107 |  2889 |     3 |
+-----------------------------------------------------------------+
```

Controlling the Behavior of the Optimizer

- **CURSOR_SHARING**: SIMILAR, EXACT, FORCE
- **DB_FILE_MULTIBLOCK_READ_COUNT**
- **PGA_AGGREGATE_TARGET**
- **STAR_TRANSFORMATION_ENABLED**
- **RESULT_CACHE_MODE**: MANUAL, FORCE
- **RESULT_CACHE_MAX_SIZE**
- **RESULT_CACHE_MAX_RESULT**
- **RESULT_CACHE_REMOTE_EXPIRATION**

Controlling the Behavior of the Optimizer

- **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**

Optimizer Features and Oracle Database Releases

OPTIMIZER_FEATURES_ENABLED

Features	9.0.0 to 9.2.0	10.1.0 to 10.1.0.5	10.2.0 to 10.2.0.2	11.1.0.6
Index fast full scan	✓	✓	✓	✓
Consideration of bitmap access to paths for tables with only B-tree indexes	✓	✓	✓	✓
Complex view merging	✓	✓	✓	✓
Peeking into user-defined bind variables	✓	✓	✓	✓
Index joins	✓	✓	✓	✓
Dynamic sampling		✓	✓	✓
Query rewrite enables		✓	✓	✓
Skip unusable indexes		✓	✓	✓
Automatically compute index statistics as part of creation		✓	✓	✓
Cost-based query transformations		✓	✓	✓
Allow rewrites with multiple MVs and/or base tables			✓	✓
Adaptive cursor sharing				✓
Use extended statistics to estimate selectivity				✓
Use native implementation for full outer joins				✓
Partition pruning using join filtering				✓
Group by placement optimization				✓
Null aware antijoins				✓

Summary

In this lesson, you should have learned how to:

- Describe the execution steps of a SQL statement
- Describe the need for an optimizer
- Explain the various phases of optimization
- Control the behavior of the optimizer

Practice 3: Overview

This practice covers exploring a trace file to understand the optimizer’s decisions.

4

Optimizer Operators

Objectives

After completing this lesson, you should be able to:

- Describe most of the SQL operators
- List the possible access paths
- Explain how join operations are performed

Row Source Operations

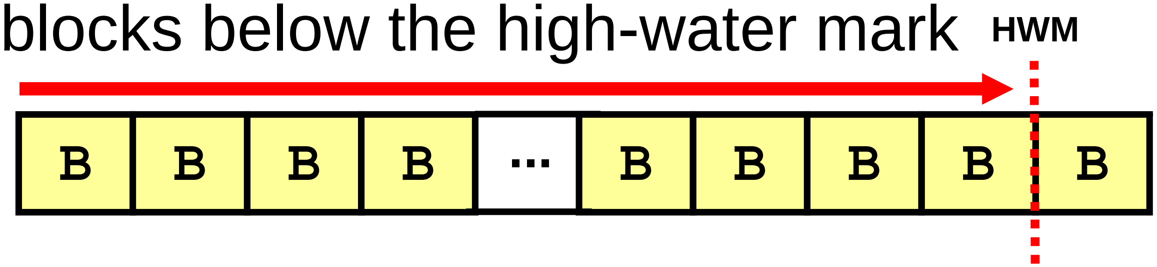
- Unary operations
  - Access Path
- Binary operations
  - Joins
- N-ary operations

Main Structures and Access Paths

Structures	Access Paths
Tables	1. Full Table Scan
	2. Rowid Scan
	3. Sample Table Scan
Indexes	4. Index Scan (Unique)
	5. Index Scan (Range)
	6. Index Scan (Full)
	7. Index Scan (Fast Full)
	8. Index Scan (Skip)
	9. Index Scan (Index Join)
	10. Using Bitmap Indexes
	11. Combining Bitmap Indexes

Full Table Scan

- Performs multiblock reads
  (here `DB_FILE_MULTIBLOCK_READ_COUNT = 4`)
- Reads all formatted blocks below the high-water mark
- May filter rows
- Faster than index range scans for large amount of data



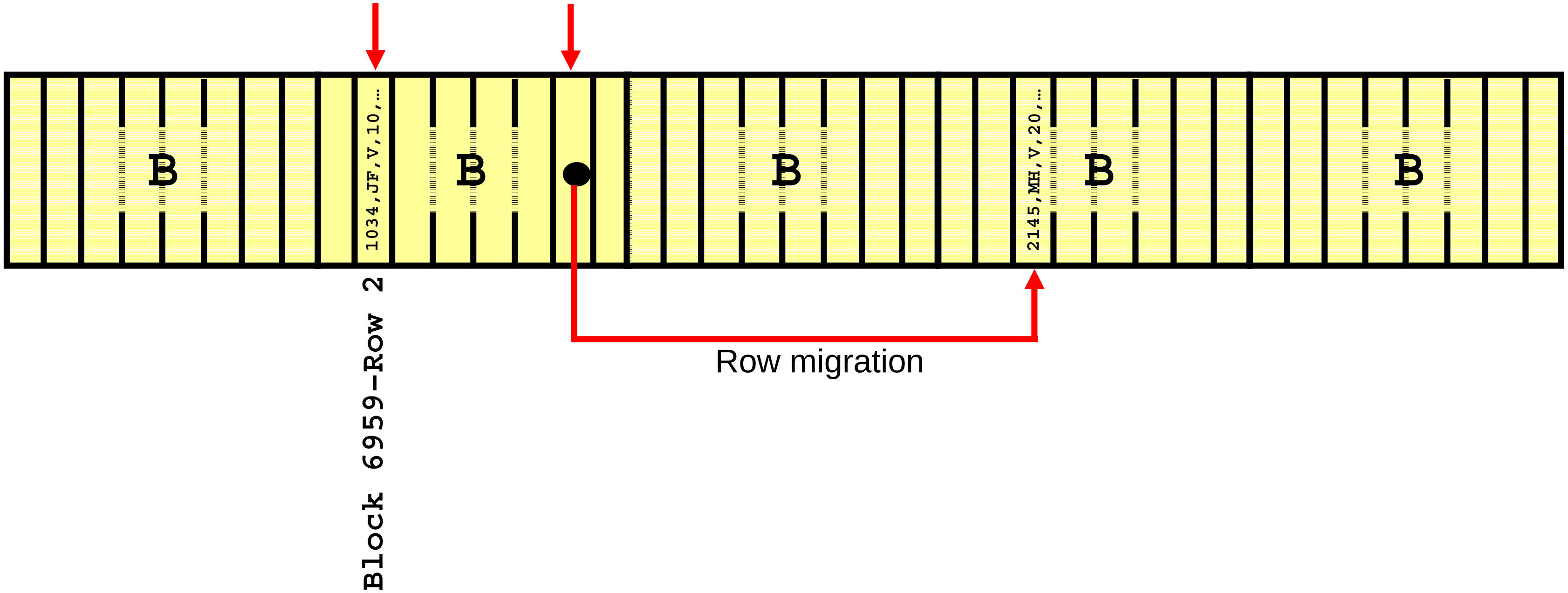
```
select * from emp where ename='King';

---------------------------------------------------------------
| Id  | Operation         | Name | Rows  | Bytes | Cost  (%CPU)|
---------------------------------------------------------------
|   0 | SELECT STATEMENT  |      |     1 |    37 |     3    (0)|
|*  1 |  TABLE ACCESS FULL| EMP  |     1 |    37 |     3    (0)|
---------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   1 - filter("ENAME"='King')
```

Full Table Scans: Use Cases

- No suitable index
- Low selectivity filters (or no filters)
- Small table
- High degree of parallelism
- Full table scan hint: `FULL (<table name>)`

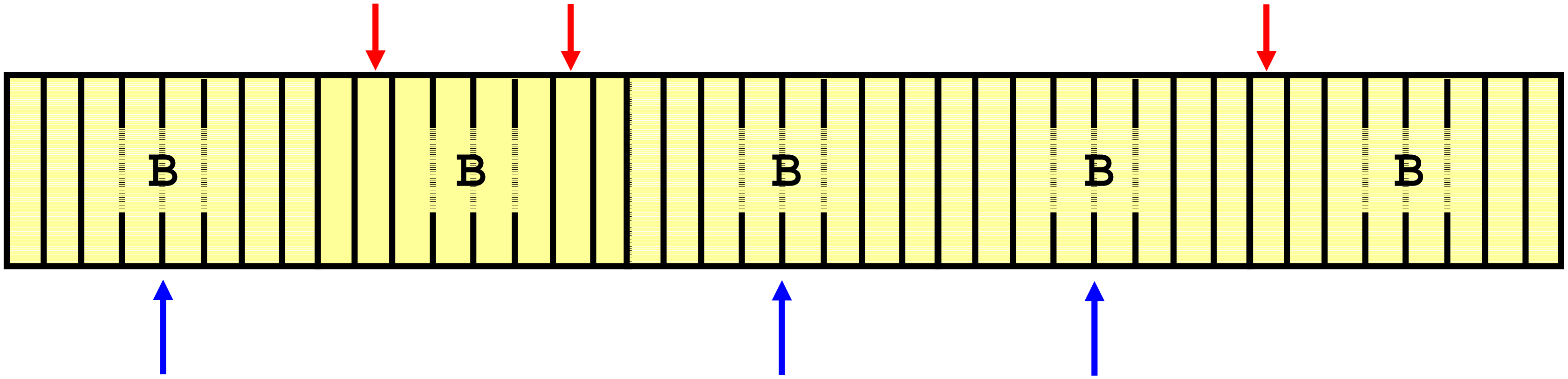
ROWID Scan

```
select * from scott.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|
-------------------------------------------------------------------
```



Sample Table Scans

```
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)|
---------------------------------------------------------------------
```

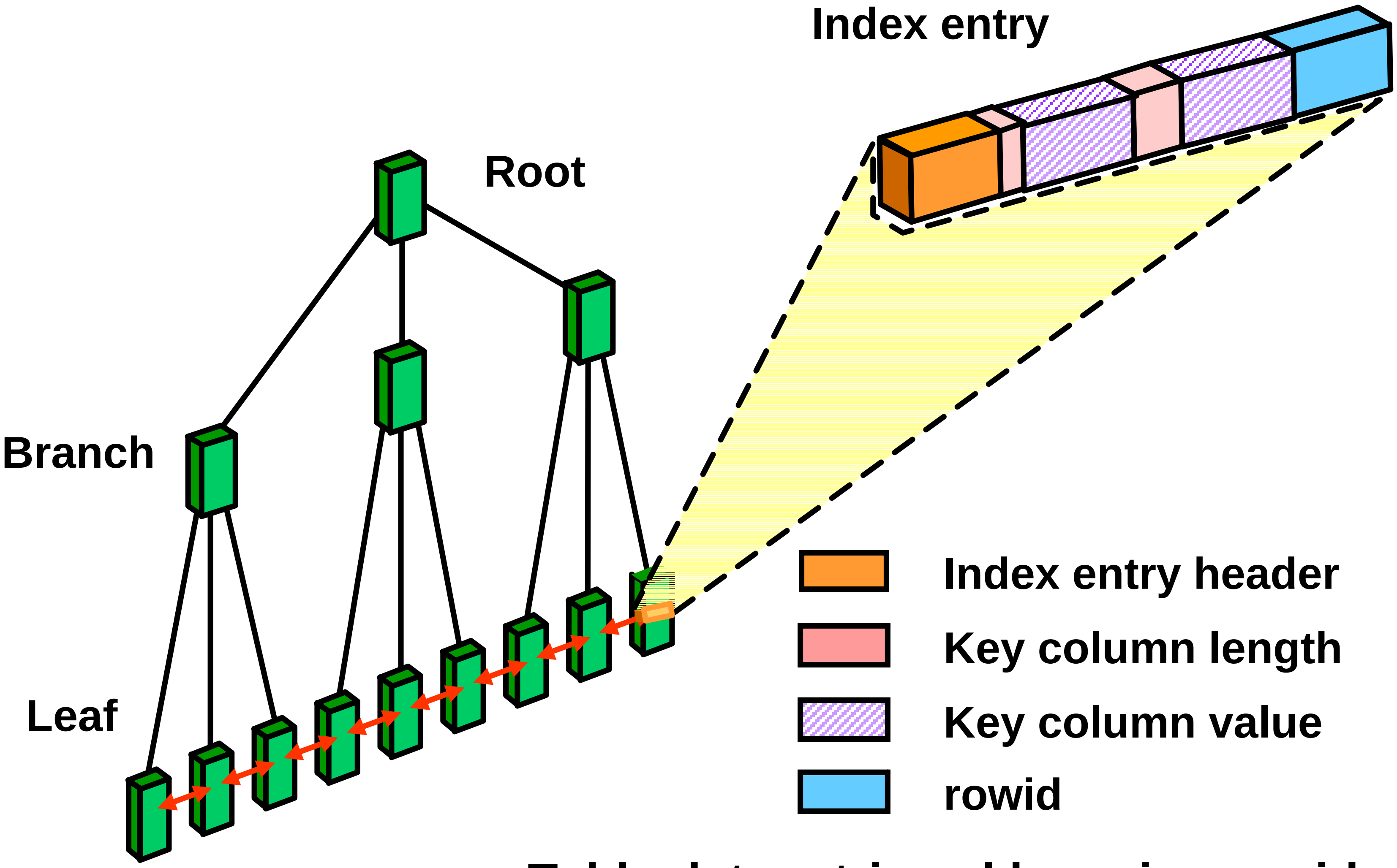


Indexes: Overview

Index storage techniques:

- B*-tree indexes: The default and the most common
  - Normal
  - Function based: Precomputed value of a function or expression
  - Index-organized table (IOT)
  - Bitmap indexes
  - Cluster indexes: Defined specifically for cluster
- Index attributes:
  - Key compression
  - Reverse key
  - Ascending, descending
- Domain indexes: Specific to an application or cartridge

Normal B*-tree Indexes



Index entry

Root

Branch

Leaf

🟧 Index entry header

🟥 Key column length

▨ Key column value

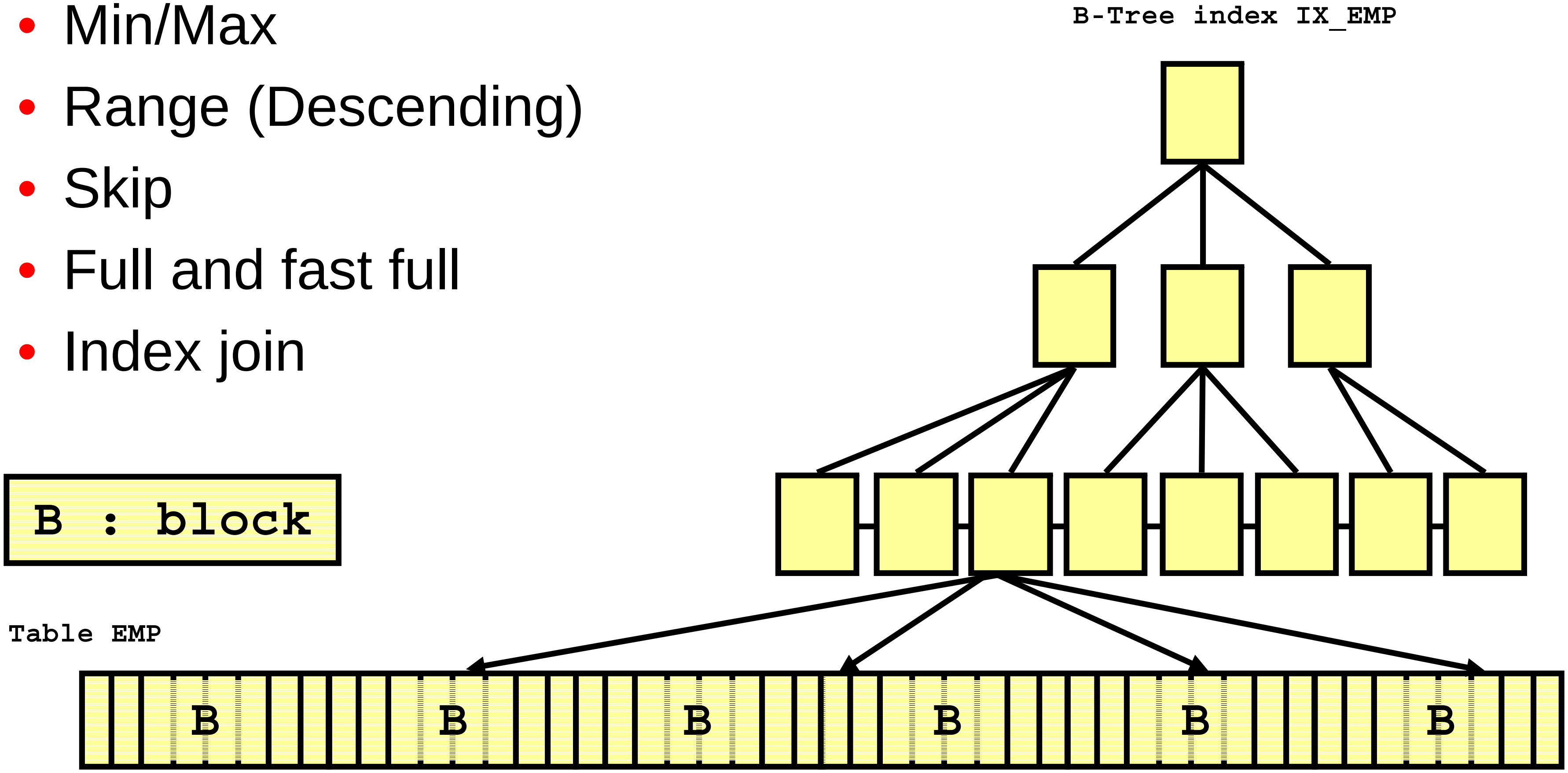
🟦 rowid

Table data retrieved by using rowid

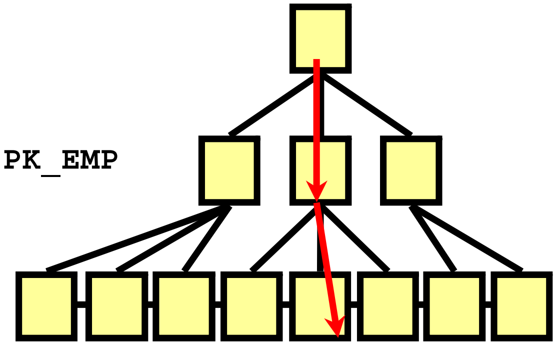
Index Scans

Types of index scans:

- Unique
- Min/Max
- Range (Descending)
- Skip
- Full and fast full
- Index join



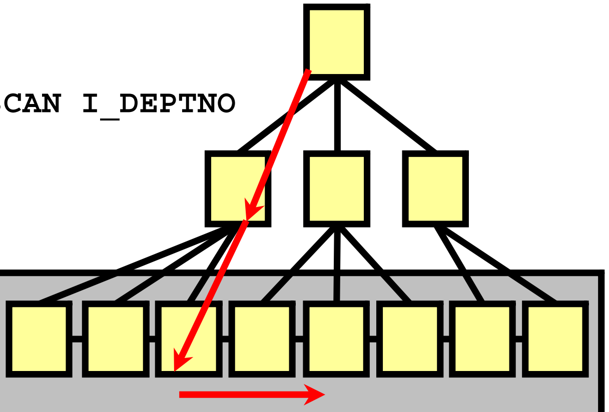
Index Unique Scan



index UNIQUE Scan PK_EMP

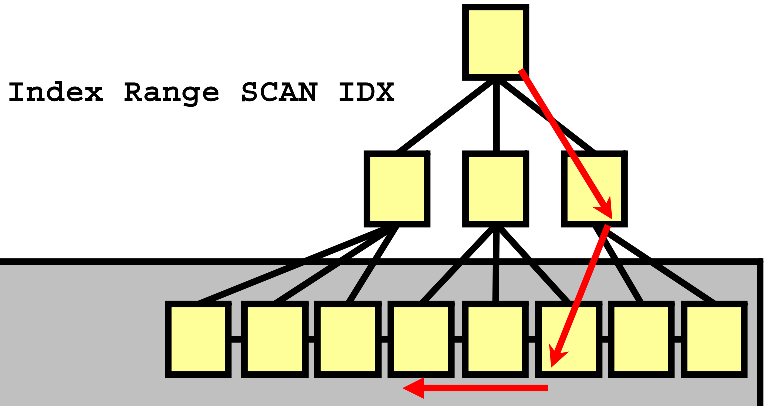
```
create unique index PK_EMP on EMP(empno)

select * from emp where empno = 9999;

---------------------------------------------------------------------
| Id  | Operation                    | Name    | Rows  | Bytes | Cost|
---------------------------------------------------------------------
|   0 | SELECT STATEMENT             |         |     1 |    37 |    1|
|   1 |  TABLE ACCESS BY INDEX ROWID| EMP     |     1 |    37 |    1|
|   2 |   INDEX UNIQUE SCAN          | PK_EMP  |     1 |       |    0|
---------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------------
   2 - access("EMPNO"=9999)
```

Index Range Scan



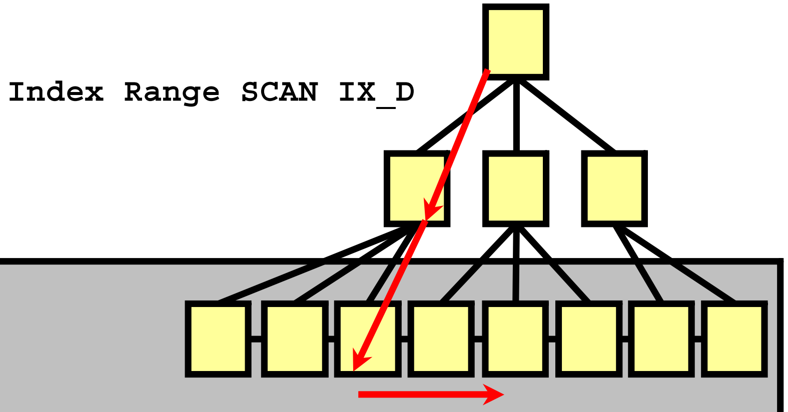
```
create index I_DEPTNO on EMP(deptno);

select /*+ INDEX(EMP I_DEPTNO) */ *
from emp where deptno = 10 and sal > 1000;


---------------------------------------------------------------------
| Id  | Operation                     | Name     | Rows  | Bytes | Cost
---------------------------------------------------------------------
|   0 | SELECT STATEMENT              |          |     3 |   261 |    2
|   1 |  TABLE ACCESS BY INDEX ROWID| EMP      |     3 |   261 |    2
|   2 |   INDEX RANGE SCAN            | I_DEPTNO |     3 |       |    1
---------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   1 - filter("SAL">1000)
   2 - access("DEPTNO"=10)
```

Index Range Scan: Descending



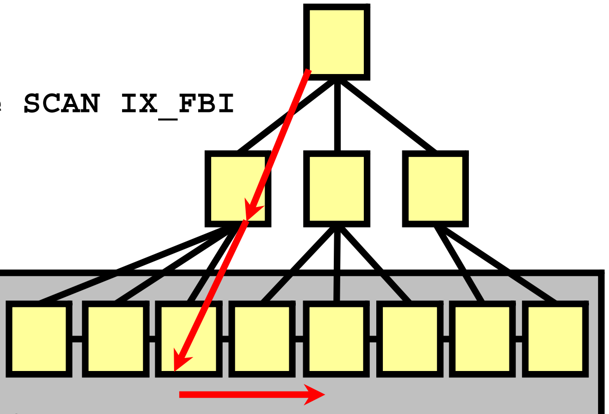
```
create index IDX on EMP(deptno);


select * from emp where deptno>20 order by deptno desc;

---------------------------------------------------------------------
| Id  | Operation                    | Name | Rows    | Bytes   | Cost     |
---------------------------------------------------------------------
|   0 | SELECT STATEMENT             |      |       6 |     522 |         2|
|   1 |  TABLE ACCESS BY INDEX ROWID| EMP  |       6 |     522 |         2|
|   2 |  INDEX RANGE SCAN DESCENDING| IDX  |       6 |         |         1|
---------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
  2 - access("DEPTNO">20)
```

Descending Index Range Scan



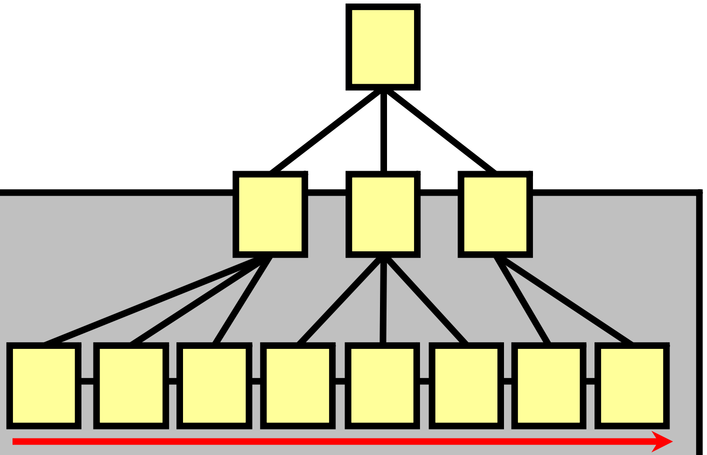
```
create index IX_D on EMP(deptno desc);

select * from emp where deptno <30;

---------------------------------------------------------------------
| Id  | Operation                    | Name | Rows  | Bytes | Cost   |
---------------------------------------------------------------------
|   0 | SELECT STATEMENT             |      |     9 |   333 |       2|
|   1 |  TABLE ACCESS BY INDEX ROWID| EMP  |     9 |   333 |       2|
|  2 |   INDEX RANGE SCAN          | IX_D |     1 |       |       1|
---------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   2 - access(SYS_OP_DESCEND("DEPTNO")>HEXTORAW('3EE0FF') )
       filter(SYS_OP_UNDESCEND(SYS_OP_DESCEND("DEPTNO"))<30)
```

Index Range Scan: Function-Based



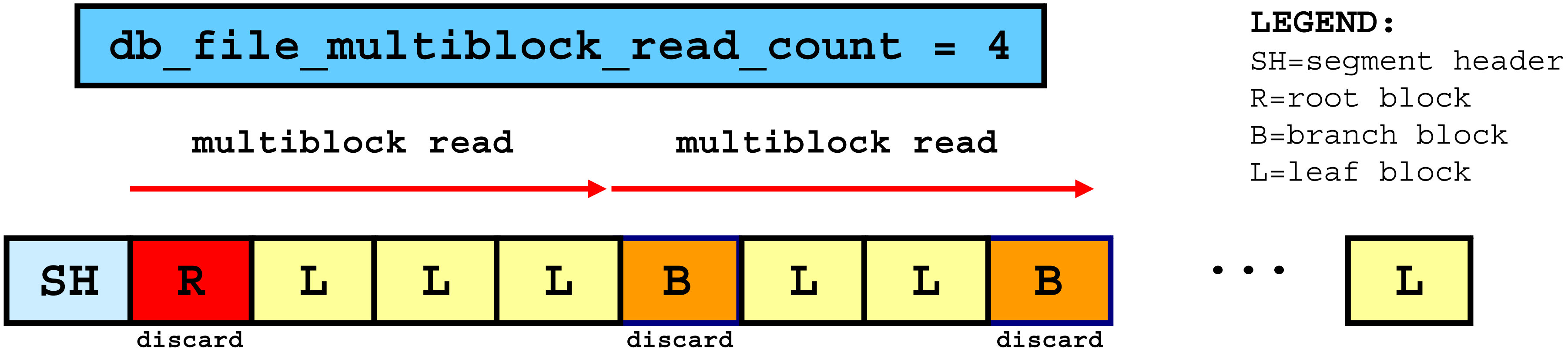
```
create index IX_FBI on EMP(UPPER(ename));

select * from emp where upper(ENAME) like 'A%';

---------------------------------------------------------------------
| Id  | Operation                    | Name   | Rows  | Bytes | Cost |
---------------------------------------------------------------------
|   0 | SELECT STATEMENT             |        |     1 |    37 |     2|
|   1 |  TABLE ACCESS BY INDEX ROWID| EMP    |     1 |    37 |     2|
|   2 |   INDEX RANGE SCAN           | IX_FBI |     1 |       |     1|
---------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   2 - access(UPPER("ENAME") LIKE 'A%')
       filter(UPPER("ENAME") LIKE 'A%')
```

Index Full Scan



```
create index I_DEPTNO on EMP(deptno);

select *
from emp
where sal > 1000 and deptno is not null
order by deptno;


---------------------------------------------------------------------
| Id  | Operation                    | Name     | Rows  | Bytes  |Cost|
---------------------------------------------------------------------
|   0 | SELECT STATEMENT             |          |    12 |    444 |   2|
|   1 |  TABLE ACCESS BY INDEX ROWID| EMP      |    12 |    444 |   2|
|   2 |   INDEX FULL SCAN            | I_DEPTNO |    14 |        |   1|
---------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   1 - filter("SAL">1000)
   2 - filter("DEPTNO" IS NOT NULL)
```

Index Fast Full Scan

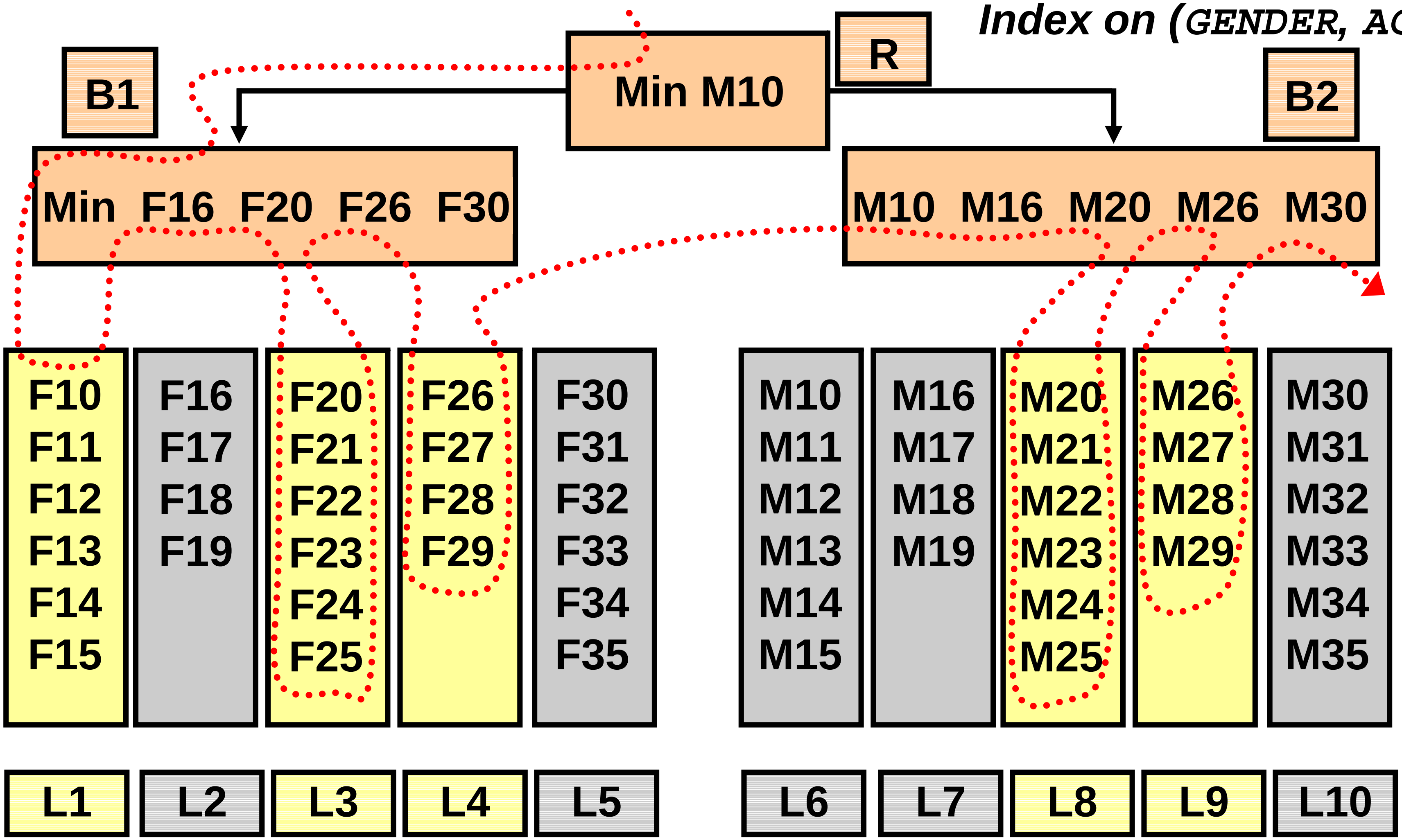


```
create index I_DEPTNO on EMP(deptno);
select /*+ INDEX_FFS(EMP I_DEPTNO)  */ deptno from emp
where deptno is not null;
-----------------------------------------------------------------
| Id  | Operation             | Name       | Rows  | Bytes  | Cost  |
-----------------------------------------------------------------
|   0 | SELECT STATEMENT      |            |    14 |     42 |      2|
|   1 |  INDEX FAST FULL SCAN| I_DEPTNO |    14 |     42 |      2|
-----------------------------------------------------------------
Predicate Information (identified by operation id):
----------------------------------------------------
   1 - filter("DEPTNO" IS NOT NULL)
```

Index Skip Scan

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

*Index on (**GENDER**, **AGE**)*



R

Min M10

B1

Min F16 F20 F26 F30

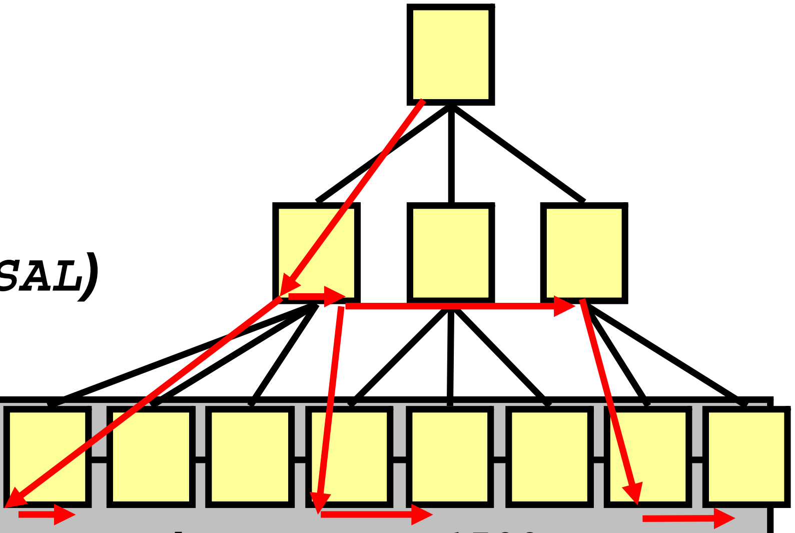
B2

M10 M16 M20 M26 M30

L1	L2	L3	L4	L5	L6	L7	L8	L9	L10
F10 F11 F12 F13 F14 F15	F16 F17 F18 F19	F20 F21 F22 F23 F24 F25	F26 F27 F28 F29	F30 F31 F32 F33 F34 F35	M10 M11 M12 M13 M14 M15	M16 M17 M18 M19	M20 M21 M22 M23 M24 M25	M26 M27 M28 M29	M30 M31 M32 M33 M34 M35

Index Skip Scan: Example

Index on (DEPTNO, SAL)



```
create index IX_SS on EMP(DEPTNO,SAL);

select /*+ index_ss(EMP IX_SS) */ * from emp where SAL < 1500;


---------------------------------------------------------------------
| Id  | Operation                   | Name  | Rows  | Bytes | Cost  |
---------------------------------------------------------------------
|   0 | SELECT STATEMENT            |       |     6 |   222 |     6 |
|   1 |  TABLE ACCESS BY INDEX ROWID| EMP   |     6 |   222 |     6 |
|   2 |   INDEX SKIP SCAN           | IX_SS |     6 |       |     5 |
---------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   2 - access("SAL"<1500)
       filter("SAL"<1500)
```

Index Join Scan

```
alter table emp modify (SAL not null, ENAME not null);
create index I_ENAME on EMP(ename);
create index I_SAL on EMP(sal);

select /*+ INDEX_JOIN(e)  */ ename, sal from emp e;

---------------------------------------------------------------------
| Id  | Operation                | Name             | Rows  | Bytes |
---------------------------------------------------------------------
|   0 | SELECT STATEMENT         |                  |    14 |   140 |
|   1 |  VIEW                    | index$_join$_001 |    14 |   140 |
|   2 |   HASH JOIN              |                  |       |       |
|   3 |    INDEX FAST FULL SCAN| IX_SS            |    14 |   140 |
|   4 |    INDEX FAST FULL SCAN| I_ENAME          |    14 |   140 |
--------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------
   2 - access(ROWID=ROWID)
```

The AND-EQUAL Operation

```
SELECT /*+ AND_EQUAL(emp isal ijob) */  *
FROM   emp
WHERE  sal=1000 and job='CLERK';

---------------------------------------------------------------------
| Id  | Operation                   | Name | Rows  | Bytes | Cost    |
---------------------------------------------------------------------
|    0 | SELECT STATEMENT            |      |     1 |    87 |     2   |
|    1 |  TABLE ACCESS BY INDEX ROWID| EMP  |     1 |    87 |     2   |
|    2 |   AND-EQUAL                 |      |       |       |         |
|    3 |    INDEX RANGE SCAN         | ISAL |     1 |       |     1   |
|    4 |    INDEX RANGE SCAN         | IJOB |     4 |       |     1   |
---------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   1 - filter("SAL"=1000 AND "JOB"='CLERK')
   3 - access("SAL"=1000)
   4 - access("JOB"='CLERK')
```

B*-tree Indexes and Nulls

```
create table nulltest ( col1 number, col2 number not null);
create index nullind1 on nulltest (col1);
create index notnullind2 on nulltest (col2);
```

```
select /*+ index(t nullind1) */ col1 from nulltest t;
--------------------------------------------------------------------
| Id  | Operation          | Name      | Rows  | Bytes | Cost (%CPU)|
--------------------------------------------------------------------
|   0 | SELECT STATEMENT   |           | 10000 |   126K|    11   (0)|
|   1 |  TABLE ACCESS FULL| NULLTEST  | 10000 |   126K|    11   (0)|
```

```
select col1 from nulltest t where col1=10;
-------------------------------------------------------------------
| Id  | Operation         | Name      | Rows  | Bytes | Cost (%CPU)|
-------------------------------------------------------------------
|   0 | SELECT STATEMENT  |           |     1 |    13 |     1   (0)|
|   1 |  INDEX RANGE SCAN| NULLIND1  |     1 |    13 |     1   (0)|
```

```
select /*+ index(t notnullind2) */ col2 from nulltest t;
---------------------------------------------------------------------
| Id  | Operation         | Name          | Rows  | Bytes | Cost (%CPU)|
---------------------------------------------------------------------
|   0 | SELECT STATEMENT  |               |     1 |    13 |     2   (0)|
|   1 |  INDEX FULL SCAN  | NOTNULLIND2   |     1 |    13 |     2   (0)|
```

Using Indexes: Considering Nullable Columns

Column	Null?
SSN	Y
FNAME	Y
LNAME	N
⋮	

PERSON

```
CREATE UNIQUE INDEX person_ssn_ix
    ON person(ssn);
```

```
SELECT COUNT(*) FROM person;

  SELECT STATEMENT    |
   SORT AGGREGATE     |
    TABLE ACCESS FULL |  PERSON
```

```
DROP INDEX person_ssn_ix;
```

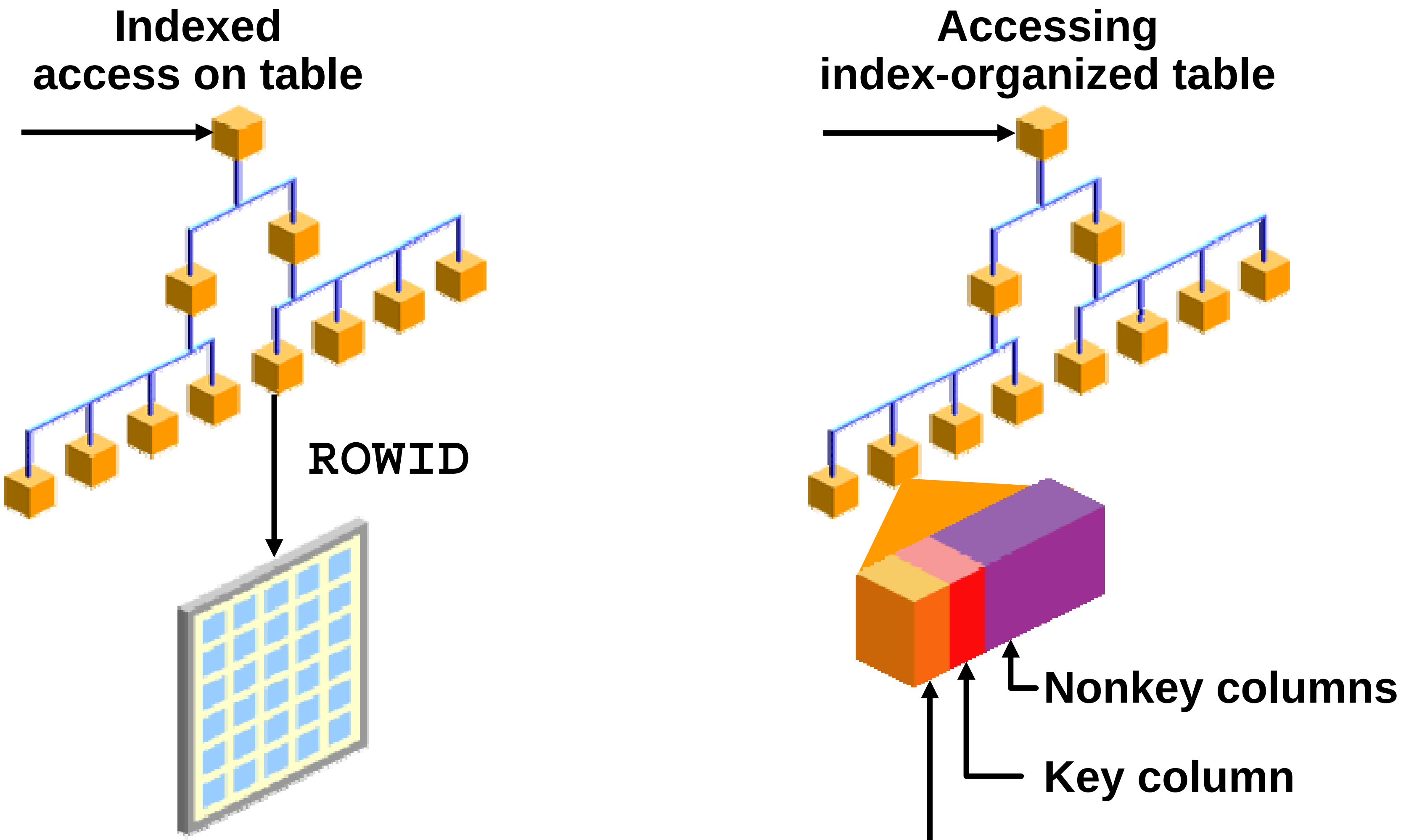
Column	Null?
SSN	N
FNAME	Y
LNAME	N
⋮	

PERSON

```
ALTER TABLE person ADD CONSTRAINT pk_ssn
    PRIMARY KEY (ssn);
```

```
SELECT /*+ INDEX(person) */ COUNT(*) FROM
    person;

  SELECT STATEMENT       |
   SORT AGGREGATE        |
    INDEX FAST FULL SCAN |  PK_SSN
```

Index-Organized Tables

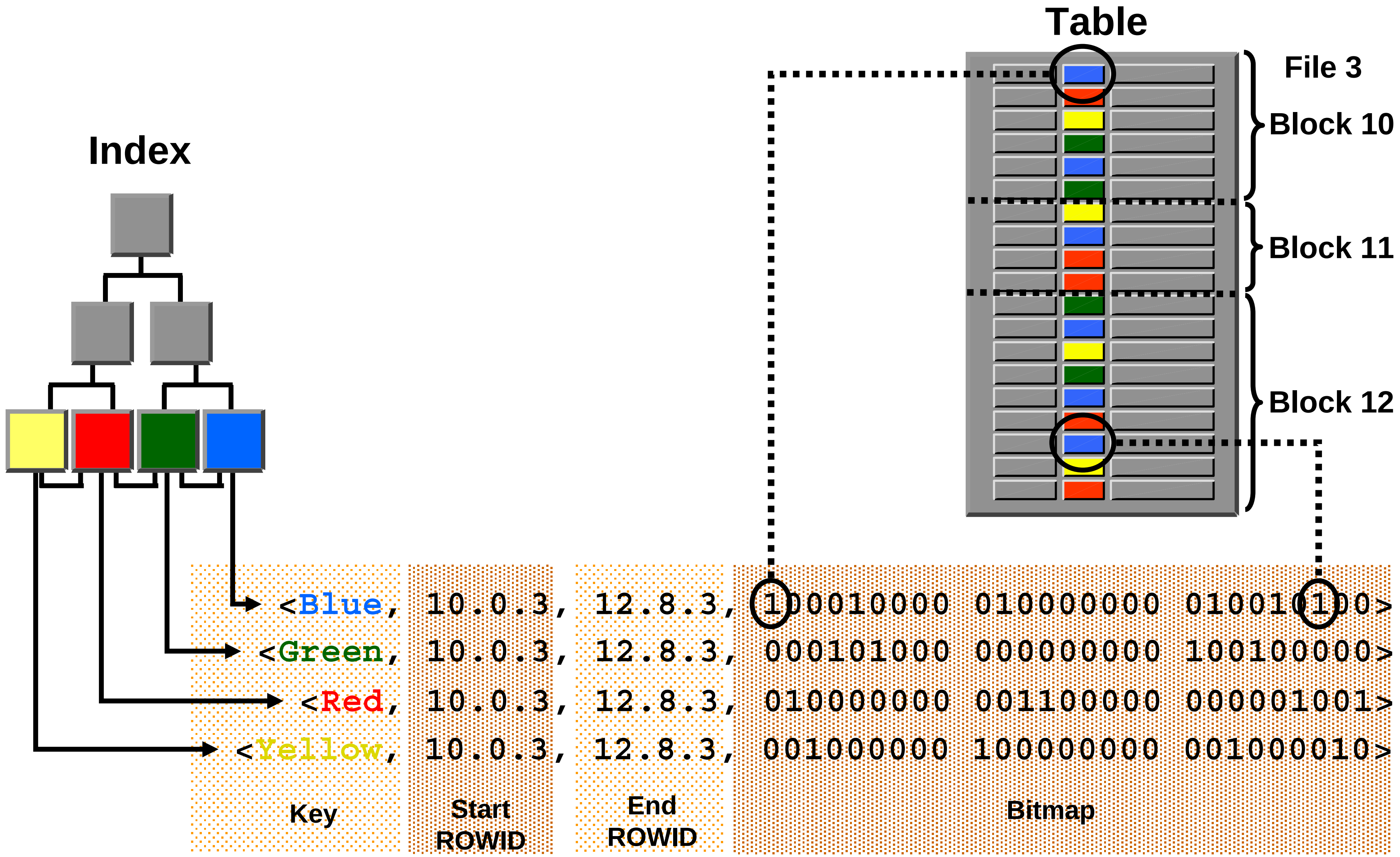


Index-Organized Table Scans

```
select * from iotemp where empno=9999;
---------------------------------------------------------------------
| Id  | Operation         | Name              | Rows | Bytes | Cost|
---------------------------------------------------------------------
|    0 | SELECT STATEMENT  |                   |     1 |    87 |    1|
|    1 |  INDEX UNIQUE SCAN| SYS_IOT_TOP_75664 |     1 |    87 |    1|
---------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   1 - access("EMPNO"=9999)
```

```
select * from iotemp where sal>1000;
---------------------------------------------------------------------
| Id  | Operation            | Name              | Rows | Bytes |
---------------------------------------------------------------------
|    0 | SELECT STATEMENT     |                   |    12 |  1044 |
|    1 |  INDEX FAST FULL SCAN| SYS_IOT_TOP_75664 |    12 |  1044 |
---------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   1 - filter("SAL">1000)
```

Bitmap Indexes



Key	Start ROWID	End ROWID	Bitmap
<Blue,	10.0.3,	12.8.3,	100010000 010000000 010010100>
<Green,	10.0.3,	12.8.3,	000101000 000000000 100100000>
<Red,	10.0.3,	12.8.3,	010000000 001100000 000001001>
<Yellow,	10.0.3,	12.8.3,	001000000 100000000 001000010>

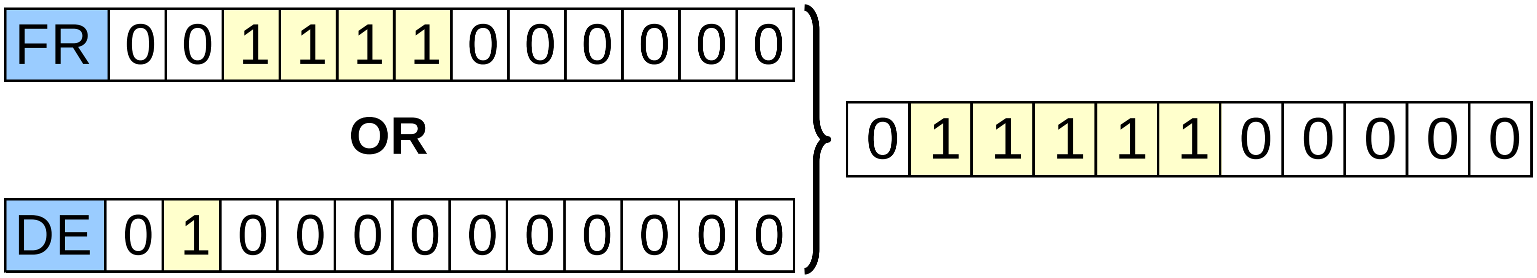
Bitmap Index Access: Examples

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

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

Combining Bitmap Indexes: Examples

```
SELECT * FROM PERF_TEAM WHERE country in('FR','DE');
```

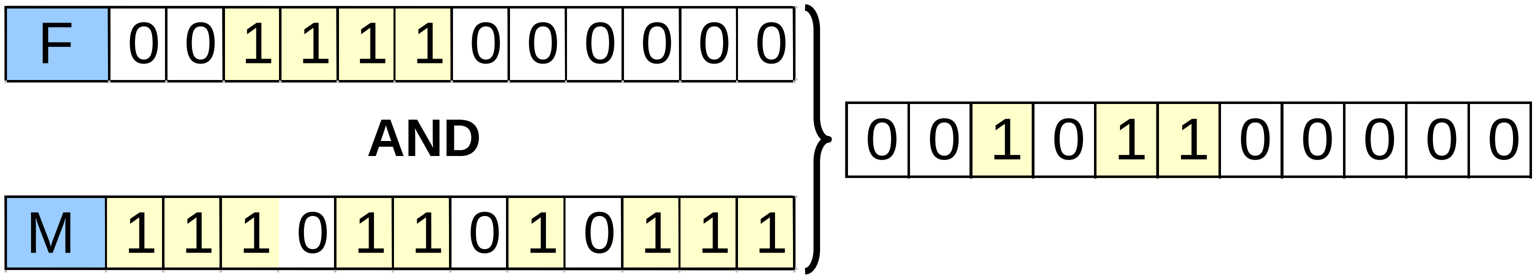


FR	0	0	1	1	1	1	0	0	0	0	0	0

OR

DE	0	1	0	0	0	0	0	0	0	0	0	0

0	1	1	1	1	1	0	0	0	0	0



F	0	0	1	1	1	1	0	0	0	0	0	0

AND

M	1	1	1	0	1	1	0	1	0	1	1	1

0	0	1	0	1	1	0	0	0	0	0

```
SELECT * FROM EMEA_PERF_TEAM T WHERE country='FR' and gender='M';
```

Combining Bitmap Index Access Paths

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

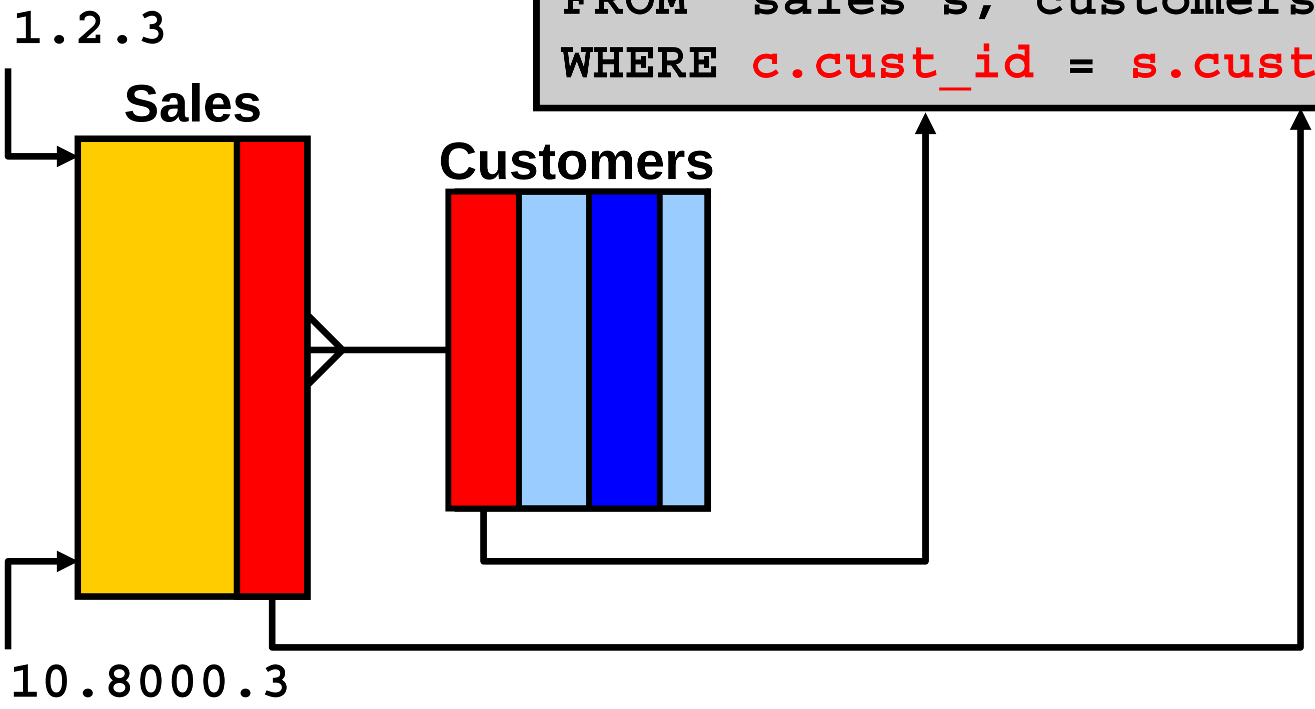
```
SELECT * FROM PERF_TEAM WHERE country='FR' and gender='M';
----------------------------------------------------------------------
| Id  | Operation                  | Name      | Rows  | Bytes |
|   0 | SELECT STATEMENT           |           |     1 |    45 |
|   1 |  TABLE ACCESS BY INDEX ROWID | PERF_TEAM |     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("GENDER"='M') 5 - access("COUNTRY"='FR')
```

Bitmap Operations

- BITMAP CONVERSION:
  - TO ROWIDS
  - FROM ROWIDS
  - COUNT
- BITMAP INDEX:
  - SINGLE VALUE
  - RANGE SCAN
  - FULL SCAN
- BITMAP MERGE
- BITMAP AND/OR
- BITMAP MINUS
- BITMAP KEY ITERATION

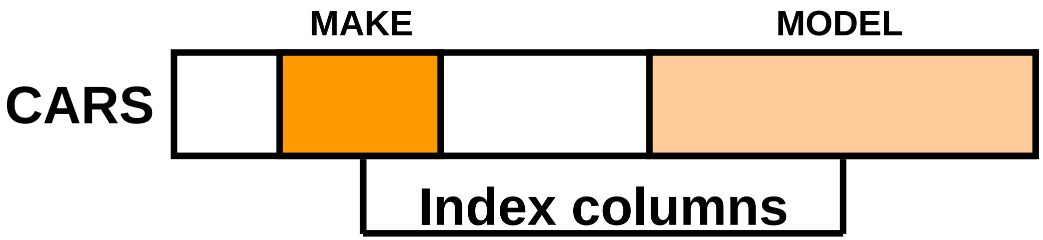
Bitmap Join Index

```
CREATE BITMAP INDEX cust_sales_bji
ON     sales(c.cust_city)
FROM   sales s, customers c
WHERE  c.cust_id = s.cust_id;
```



```
      <Rognes, 1.2.3, 10.8000.3, 100010010010100…>
<Aix-en-Provence, 1.2.3, 10.8000.3, 000101000100000…>
   <Marseille, 1.2.3, 10.8000.3, 010000001000001…>
```

Composite Indexes

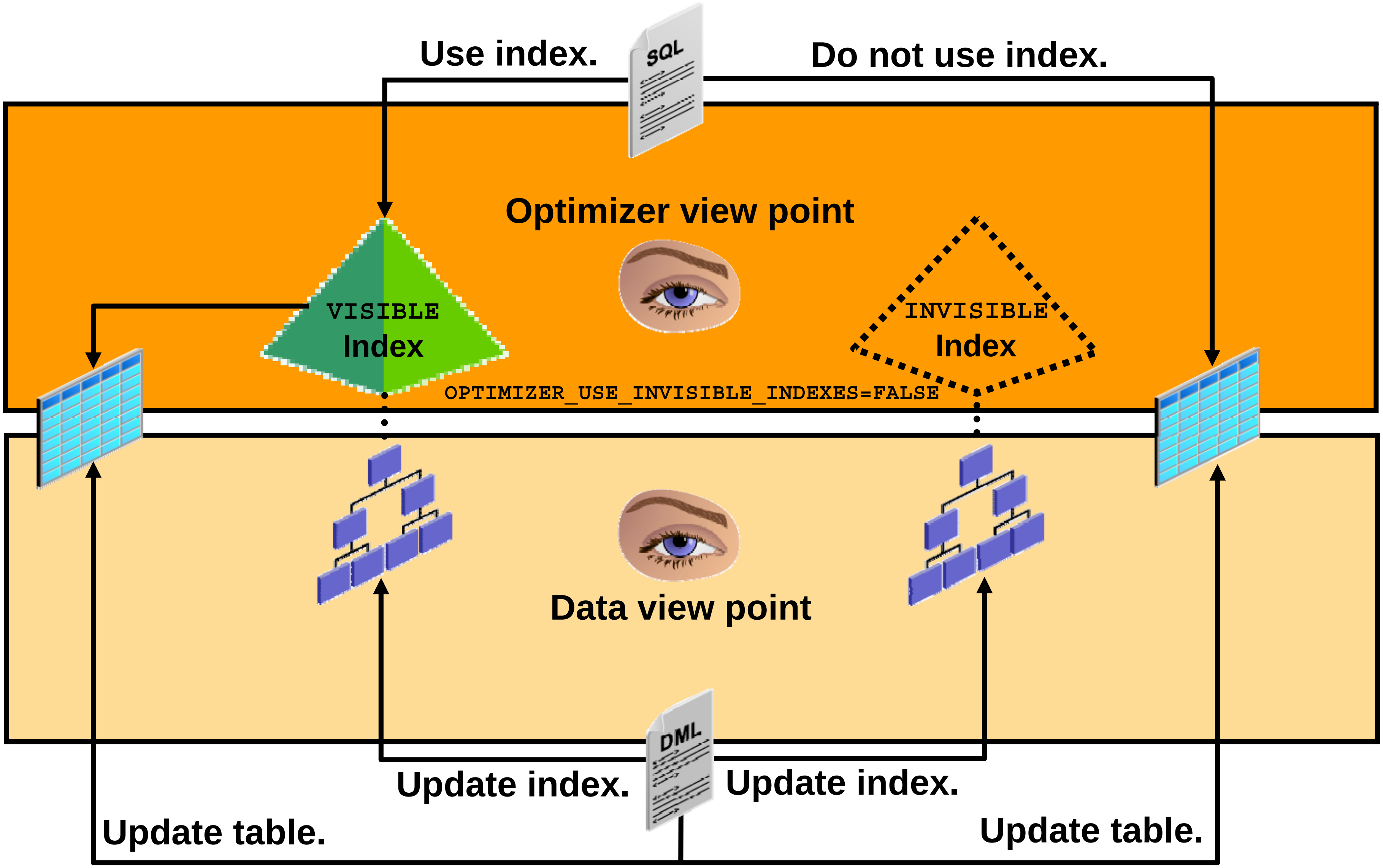


```
create index cars_make_model_idx on cars(make, model);
```

```
select *
from cars
where make = 'CITROËN' and model = '2CV';
```

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

Invisible Index: Overview



Invisible Indexes: Examples

- 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;
```

Guidelines for Managing Indexes

- 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.
- Drop indexes that are no longer required.
- Specify the tablespace for each index.
- Consider parallelizing index creation.
- Consider creating indexes with `NOLOGGING`.
- Consider costs and benefits of coalescing or rebuilding indexes.
- Consider cost before disabling or dropping constraints.

Investigating Index Usage

An index may not be used for one of many reasons:

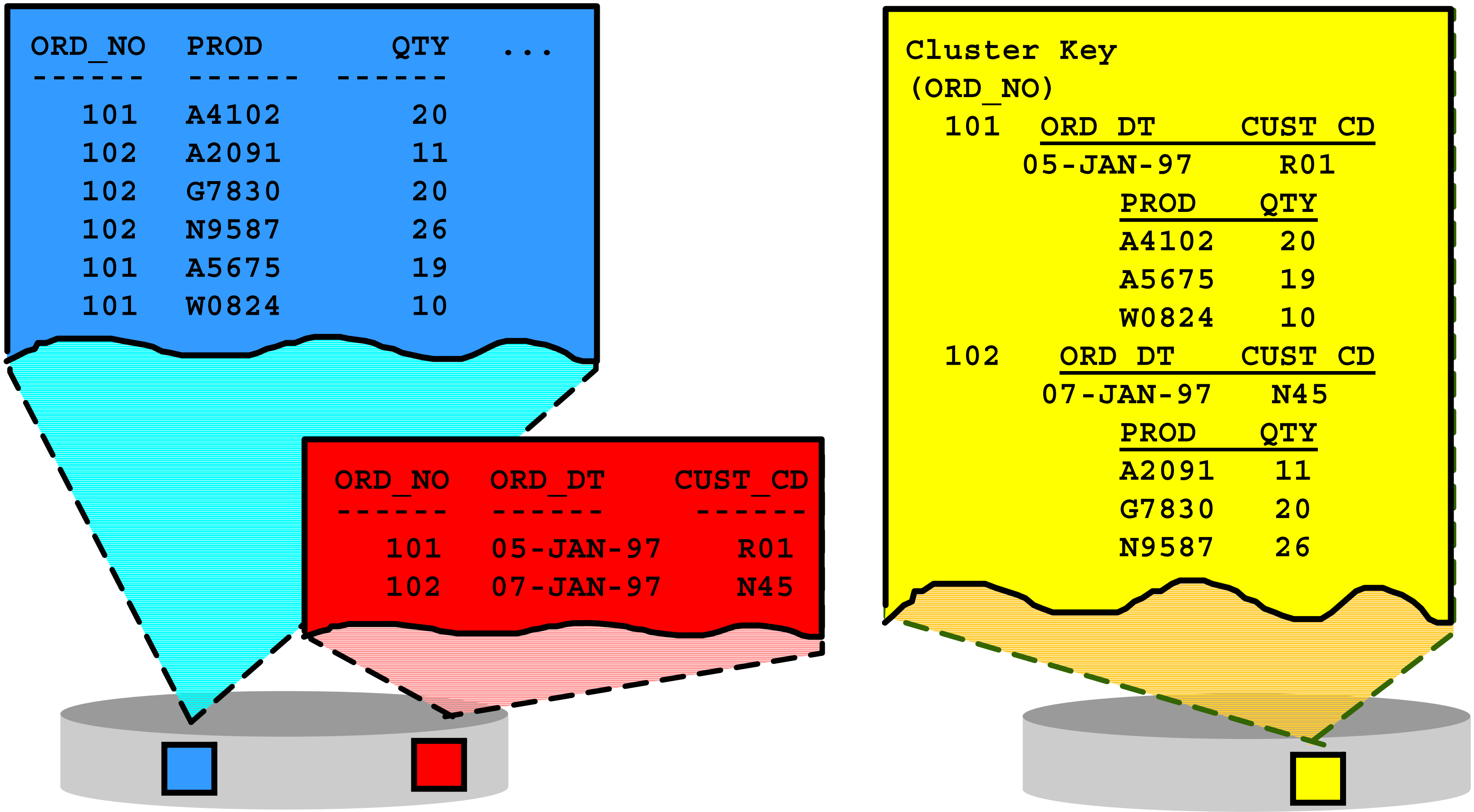
- There are functions being applied to the predicate.
- There is a data type mismatch.
- Statistics are old.
- The column can contain null.
- Using the index would actually be slower than not using it.

Practice 4: Overview

This practice covers using different access paths for better optimization.

- Case 1 through case 13

Clusters



```
ORD_NO   PROD        QTY   ...
------   ------    ------
   101   A4102        20
   102   A2091        11
   102   G7830        20
   102   N9587        26
   101   A5675        19
   101   W0824        10
```

```
ORD_NO   ORD_DT      CUST_CD
------   ------       ------
   101   05-JAN-97       R01
   102   07-JAN-97       N45
```

```
Cluster Key
(ORD_NO)
  101   ORD_DT      CUST_CD
      05-JAN-97      R01
             PROD     QTY
             A4102    20
             A5675    19
             W0824    10
  102    ORD_DT     CUST_CD
       07-JAN-97     N45
             PROD     QTY
             A2091    11
             G7830    20
             N9587    26
```

Unclustered `ORDERS` and `ORDER_ITEMS` tables

Clustered `ORDERS` and `ORDER_ITEMS` tables

When Are Clusters Useful?

- Index cluster:
  - Tables always joined on the same keys
  - The size of the table is not known
  - In any type of searches
- Hash cluster:
  - Tables always joined on the same keys
  - Storage for all cluster keys allocated initially
  - In either equality (=) or nonequality (<>) searches

When Are Clusters Useful?

- Single-table hash cluster:
  - Fastest way to access a large table with an equality search
- Sorted hash cluster:
  - Only used for equality search
  - Avoid sorts on batch reporting
  - Avoid overhead probe on the branch blocks of an IOT

Cluster Access Path: Examples

```
SELECT * FROM calls WHERE origin_number=33442395322;

----------------------------------------------------------------
| Id  | Operation         | Name  | Rows  | Bytes | Cost (%CPU)|
----------------------------------------------------------------
|   0 | SELECT STATEMENT  |       |     1 |    56 |     0   (0)|
|   1 |  TABLE ACCESS HASH| CALLS |     1 |    56 |            |
----------------------------------------------------------------
   1 - access("ORIGIN_NUMBER"=33442395322)
```

```
SELECT * FROM emp,dept WHER emp.deptno=dept.deptno;

-----------------------------------------------------------------
| Id  | Operation             | Name | Rows  | Bytes | Cost     |
-----------------------------------------------------------------
|   0 | SELECT STATEMENT      |      |     1 |   117 |     3    |
|   1 |  NESTED LOOPS         |      |     1 |   117 |     3    |
|   2 |   TABLE ACCESS FULL   | EMP  |     1 |    87 |     2    |
|   3 |   TABLE ACCESS CLUSTER| DEPT |     1 |    30 |     1    |
-----------------------------------------------------------------
   3 - filter("EMP"."DEPTNO"="DEPT"."DEPTNO")
```

Sorting Operators

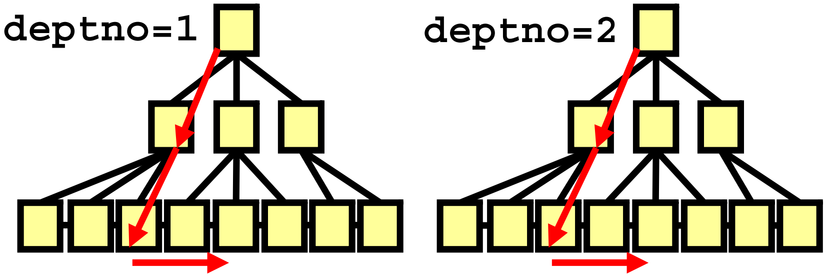
- `SORT` operator:
  - `AGGREGATE`: Single row from group function
  - `UNIQUE`: To eliminate duplicates
  - `JOIN`: Precedes a merge join
  - `GROUP BY`, `ORDER BY`: For these operators
- `HASH` operator:
  - `GROUP BY`: For this operator
  - `UNIQUE`: Equivalent to `SORT UNIQUE`
- If you want ordered results, *always* use `ORDER BY`.

Buffer Sort Operator

```
select ename, emp.deptno, dept.deptno, dname
from   emp, dept
where ename like 'A%';

---------------------------------------------------------------------
| Id  | Operation                    | Name    | Rows  | Bytes | Cost
---------------------------------------------------------------------
|    0 | SELECT STATEMENT             |         |     4 |   124 |    5
|    1 |  MERGE JOIN CARTESIAN        |         |     4 |   124 |    5
|    2 |   TABLE ACCESS BY INDEX ROWID| EMP     |     1 |     9 |    2
|    3 |    INDEX RANGE SCAN          | I_ENAME |     1 |       |    1
|    4 |   BUFFER SORT                |         |     4 |    88 |    3
|    5 |    TABLE ACCESS FULL         | DEPT    |     4 |    88 |    3
---------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   3 - access("ENAME" LIKE 'A%')
       filter("ENAME" LIKE 'A%')
```

Inlist Iterator

Every value executed separately



```
select * from emp where deptno in (1,2);
select * from emp where deptno = 1 or deptno =2 ;
---------------------------------------------------------------------
| Id  | Operation                      | Name   | Rows  | Bytes | Cost |
---------------------------------------------------------------------
|   0 | SELECT STATEMENT               |        |     2 |    78 |     2|
|   1 |  INLIST ITERATOR               |        |       |       |      |
|   2 |   TABLE ACCESS BY INDEX ROWID| EMP    |     2 |    78 |     2|
|   3 |    INDEX RANGE SCAN            | IX_SS  |     2 |       |     1|
---------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   3 - access("DEPTNO"=1 OR "DEPTNO"=2)
```

View Operator

```
create view V as select /*+ NO_MERGE */ DEPTNO, sal  from emp ;
select * from V;
--------------------------------------------------------------------
| Id  | Operation          | Name    | Rows   | Bytes  | Cost  (%CPU)| Time
|   0 | SELECT STATEMENT   |         |     14 |    364 |      1    (0)| 0:01
|   1 |  VIEW              | V       |     14 |    364 |      1    (0)| 0:01
|   2 |   INDEX FULL SCAN| IX_SS   |     14 |     98 |      1    (0)| 0:01
--------------------------------------------------------------------
```

```
select v.*,d.dname from (select DEPTNO, sum(sal) SUM_SAL
from emp group by deptno) v, dept d  where v.deptno=d.deptno;
--------------------------------------------------------------------
| Id  | Operation            | Name    | Rows   | Bytes  | Cost  (%CPU)|
|   0 | SELECT STATEMENT     |         |      3 |    144 |      5   (20)|
|   1 |  HASH JOIN           |         |      3 |    144 |      5   (20)|
|   2 |   VIEW               |         |      3 |     78 |      1    (0)|
|   3 |    HASH GROUP BY     |         |      3 |     21 |      1    (0)|
|   4 |     INDEX FULL SCAN| IX_SS   |     14 |     98 |      1    (0)|
|   5 |   TABLE ACCESS FULL| DEPT    |      4 |     88 |      3    (0)|
--------------------------------------------------------------------
Predicate: 1 - access("V"."DEPTNO"="D"."DEPTNO")
```

Count Stop Key Operator

```
select count(*)
from (select /*+ NO_MERGE */ *
      from TC where C1 ='1' and rownum < 10);


---------------------------------------------------------------------
| Id  | Operation            | Name | Rows   | Bytes | Cost  (%CPU)|
---------------------------------------------------------------------
|   0 | SELECT STATEMENT     |      |      1 |       |     4    (0)|
|   1 |  SORT AGGREGATE      |      |      1 |       |             |
|   2 |   VIEW               |      |      9 |       |     4    (0)|
|   3 |    COUNT STOPKEY     |      |        |       |             |
|   4 |     TABLE ACCESS FULL| TC   |   4282 |  4190K|     4    (0)|
---------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   3 - filter(ROWNUM<10)
   4 - filter("C1"='1')
```

Min/Max and First Row Operators

```
select min(id) FROM t WHERE id > 500000;

---------------------------------------------------------------------
| Id  | Operation                    | Name | Rows  | Bytes | Cost |
---------------------------------------------------------------------
|   0 | SELECT STATEMENT             |      |     1 |    13 |     3|
|   1 |  SORT AGGREGATE              |      |     1 |    13 |      |
|   2 |   FIRST ROW                  |      |   717K|  9113K|     3|
|   3 |    INDEX RANGE SCAN (MIN/MAX)| IXT  |   717K|  9113K|     3|
---------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   3 - access("ID">500000)
```

Join Methods

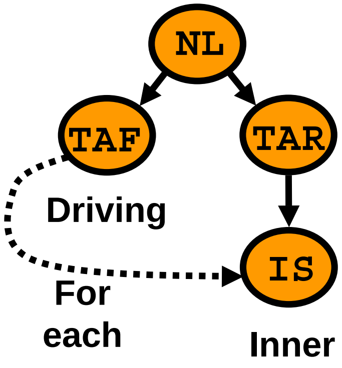
- A join defines the relationship between two row sources.
- A join is a method of combining data from two data sources.
- It is controlled by join predicates, which define how the objects are related.
- Join methods:
  - Nested loops
  - Sort-merge join
  - Hash join

```
SELECT e.ename, d.dname
FROM dept d JOIN emp e USING (deptno)            ← Join predicate
WHERE e.job = 'ANALYST' OR e.empno = 9999;       ← Nonjoin predicate
```

```
SELECT e.ename,d.dname
FROM   emp e, dept d
WHERE  e.deptno = d.deptno AND                   ← Join predicate
       (e.job = 'ANALYST' OR e.empno = 9999);    ← Nonjoin predicate
```

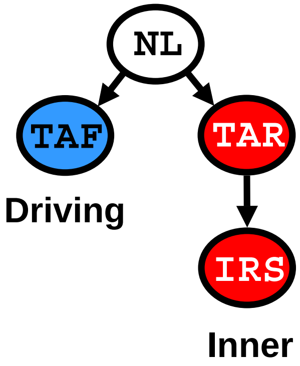
Nested Loops Join

- Driving row source is scanned
- Each row returned drives a lookup in inner row source
- Joining rows are then returned



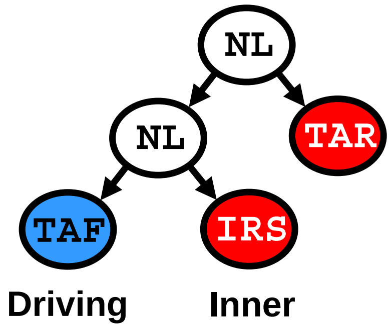
```
select ename, e.deptno, d.deptno, d.dname
from emp e, dept d
where e.deptno = d.deptno and ename like 'A%';
---------------------------------------------------------------------
| Id  | Operation                    | Name       | Rows  |Cost  |
---------------------------------------------------------------------
|   0 | SELECT STATEMENT             |            |     2 |     4 |
|   1 |  NESTED LOOPS                |            |     2 |     4 |
|   2 |   TABLE ACCESS FULL          | EMP        |     2 |     2 |
|   3 |   TABLE ACCESS BY INDEX ROWID| DEPT       |     1 |     1 |
|   4 |    INDEX UNIQUE SCAN         | PK_DEPT    |     1 |       |
---------------------------------------------------------------------
   2 - filter("E"."ENAME" LIKE 'A%')
   4 - access("E"."DEPTNO"="D"."DEPTNO")
```

Nested Loops Join: Prefetching



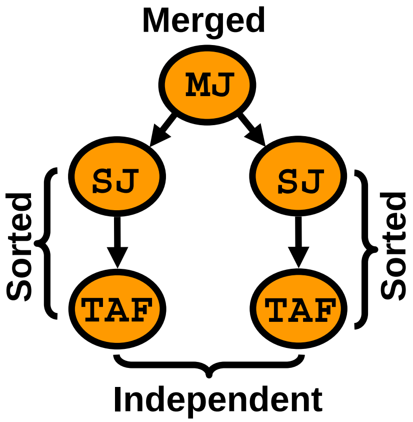
```
select ename, e.deptno, d.deptno, d.dname
from emp e, dept d
where e.deptno = d.deptno and ename like 'A%';
---------------------------------------------------------------------
|   0 | SELECT STATEMENT             |       |      2 |    84 |     5
|   1 |  TABLE ACCESS BY INDEX ROWID| DEPT  |      1 |    22 |     1
|   2 |   NESTED LOOPS               |       |      2 |    84 |     5
|*  3 |    TABLE ACCESS FULL         | EMP   |      2 |    40 |     3
|*  4 |    INDEX RANGE SCAN          | IDEPT |      1 |       |     0
---------------------------------------------------------------------
   3 - filter("E"."ENAME" LIKE 'A%')
   4 - access("E"."DEPTNO"="D"."DEPTNO")
```

Nested Loops Join: 11*g* Implementation



```
select ename, e.deptno, d.deptno, d.dname
from emp e, dept d
where e.deptno = d.deptno and ename like 'A%';
----------------------------------------------------------------------
|   0 | SELECT STATEMENT             |       |       2 |      84 |      5
|   1 |  NESTED LOOPS                |       |         |         |
|   2 |   NESTED LOOPS               |       |       2 |      84 |      5
|*  3 |    TABLE ACCESS FULL         | EMP   |       2 |      40 |      3
|*  4 |    INDEX RANGE SCAN          | DDEPT |       1 |         |      0
|   5 |   TABLE ACCESS BY INDEX ROWID| DEPT  |       1 |      22 |      1
----------------------------------------------------------------------
   3 - filter("E"."ENAME" LIKE 'A%')
   4 - access("E"."DEPTNO"="D"."DEPTNO")
```

Sort Merge Join

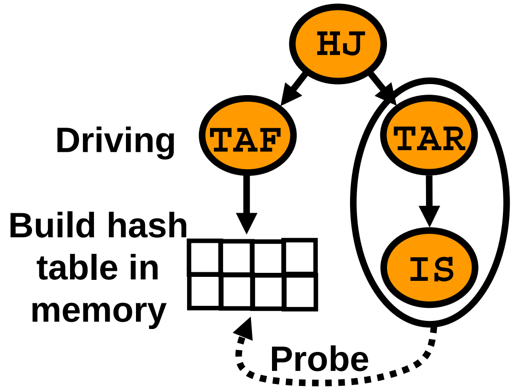


- First and second row sources are sorted by same sort key.
- Sorted rows from both side are merged.

```
select ename, e.deptno, d.deptno, dname
from emp e, dept d
where e.deptno = d.deptno and ename > 'A';
---------------------------------------------------------------------
| Id  | Operation             | Name | Rows  | Bytes | Cost (%CPU)|
|   0 | SELECT STATEMENT      |      |     2 |    84 |     8  (25)|
|   1 |  MERGE JOIN           |      |     2 |    84 |     8  (25)|
|   2 |   SORT JOIN           |      |     2 |    40 |     4  (25)|
|   3 |    TABLE ACCESS FULL| EMP  |     2 |    40 |     3   (0)|
|   4 |   SORT JOIN           |      |     4 |    88 |     4  (25)|
|   5 |    TABLE ACCESS FULL| DEPT |     4 |    88 |     3   (0)|
---------------------------------------------------------------------
Predicate:    3 - filter("ENAME">'A')
              4 - access("E"."DEPTNO"="D"."DEPTNO")
                  filter("E"."DEPTNO"="D"."DEPTNO")
```

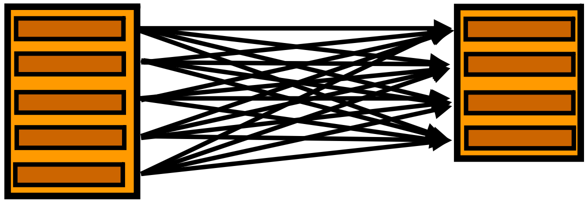
Hash Join

- The smallest row source is used to build a hash table.
- The second row source is hashed and checked against the hash table.



```
select ename, e.deptno, d.deptno, dname from emp e, dept d
where e.deptno = d.deptno and ename like 'A%';


---------------------------------------------------------------------
| Id  | Operation                      | Name  | Rows | Bytes | Cost
---------------------------------------------------------------------
|   0 | SELECT STATEMENT               |       |    3 |    66 |     6
|   1 |  HASH JOIN                     |       |    3 |    66 |     6
|   2 |   TABLE ACCESS BY INDEX ROWID| EMP   |    3 |    27 |     2
|   3 |    INDEX FULL SCAN             | EDEPT |   14 |       |     1
|   4 |   TABLE ACCESS FULL            | DEPT  |    4 |    52 |     3
---------------------------------------------------------------------
Predicate:   1 - access("E"."DEPTNO"="D"."DEPTNO")
             2 - filter("ENAME" LIKE 'A%')
```

Cartesian Join

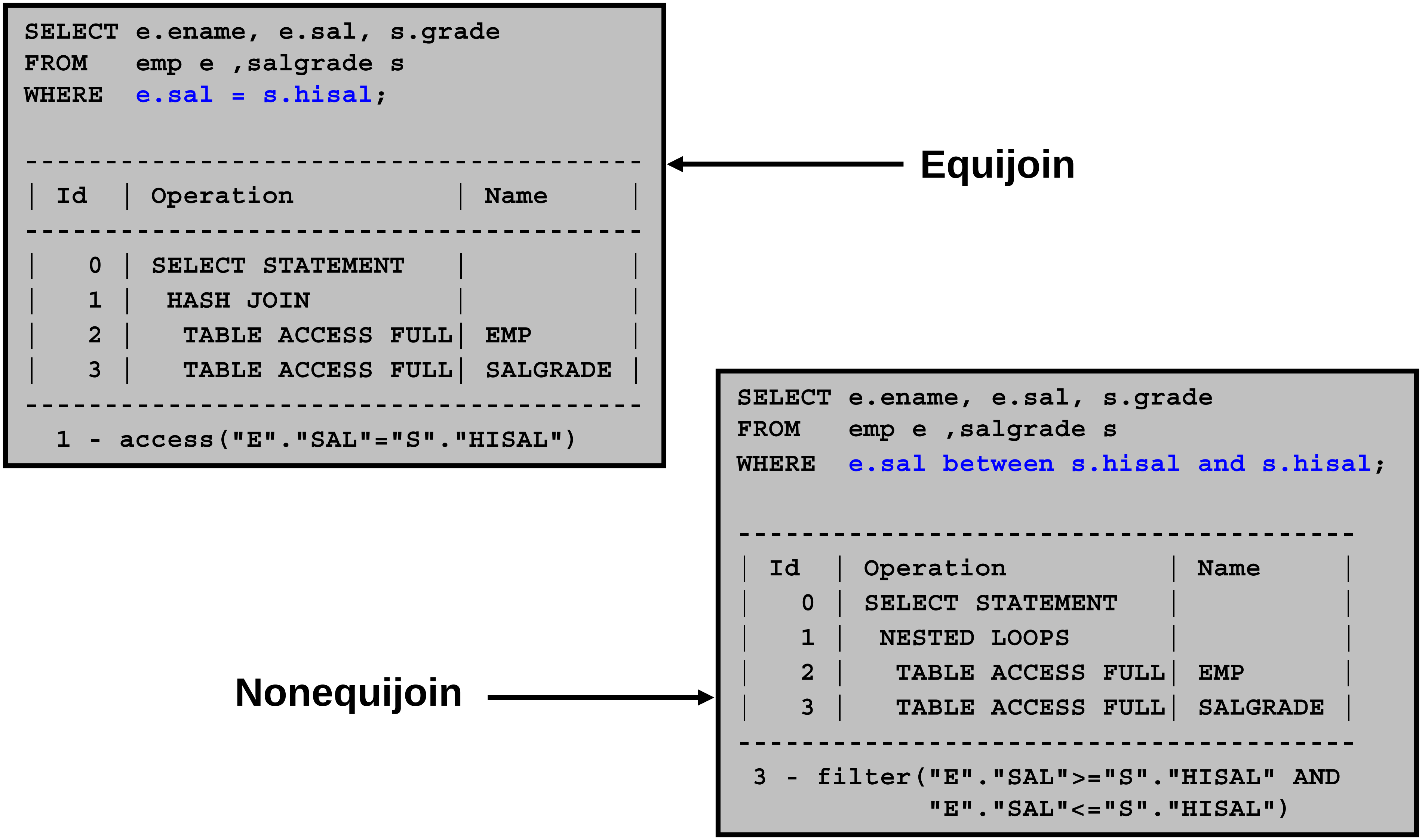


```
select ename, e.deptno, d.deptno, dname
from emp e, dept d where ename like 'A%';


---------------------------------------------------------------------
| Id  | Operation               | Name | Rows   | Bytes  | Cost (%CPU)|
---------------------------------------------------------------------
|   0 | SELECT STATEMENT        |      |     11 |    242 |       8   (0)|
|   1 |  MERGE JOIN CARTESIAN|      |     11 |    242 |       8   (0)|
|   2 |   TABLE ACCESS FULL  | EMP  |      3 |     27 |       3   (0)|
|   3 |   BUFFER SORT          |      |      4 |     52 |       5   (0)|
|   4 |    TABLE ACCESS FULL | DEPT |      4 |     52 |       2   (0)|
---------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   2 - filter("ENAME" LIKE 'A%')
```

Join Types

- A join operation combines the output from two row sources and returns one resulting row source.
- Join operation types include the following :
  - Join (Equijoin/Natural – Nonequijoin)
  - Outer join (Full, Left, and Right)
  - Semi join: `EXISTS` subquery
  - Anti join: `NOT IN` subquery
  - Star join (Optimization)

Equijoins and Nonequijoins



```
SELECT e.ename, e.sal, s.grade
FROM   emp e ,salgrade s
WHERE  e.sal = s.hisal;


---------------------------------------
| Id  | Operation            | Name     |
---------------------------------------
|   0 | SELECT STATEMENT     |          |
|   1 |  HASH JOIN           |          |
|   2 |   TABLE ACCESS FULL| EMP      |
|   3 |   TABLE ACCESS FULL| SALGRADE |
---------------------------------------
  1 - access("E"."SAL"="S"."HISAL")
```

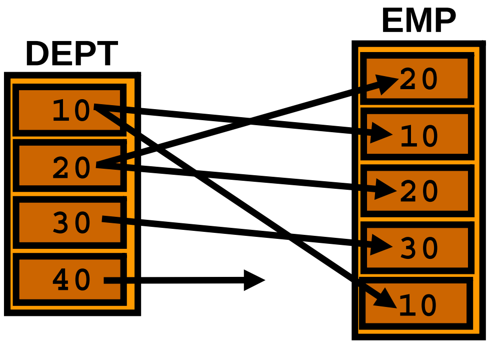
Equijoin

```
SELECT e.ename, e.sal, s.grade
FROM   emp e ,salgrade s
WHERE  e.sal between s.hisal and s.hisal;


---------------------------------------
| Id  | Operation            | Name     |
|   0 | SELECT STATEMENT     |          |
|   1 |  NESTED LOOPS        |          |
|   2 |   TABLE ACCESS FULL| EMP      |
|   3 |   TABLE ACCESS FULL| SALGRADE |
---------------------------------------
 3 - filter("E"."SAL">="S"."HISAL" AND
            "E"."SAL"<="S"."HISAL")
```

Nonequijoin

Outer Joins

An outer join also returns a row if no match is found.



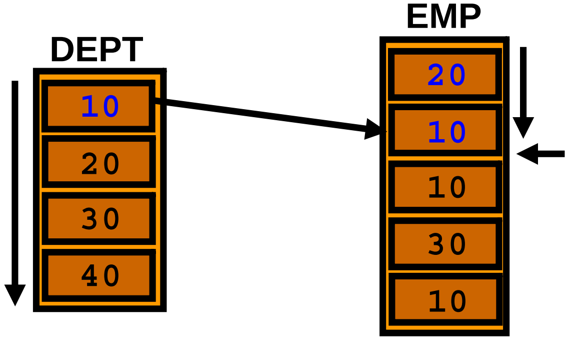
```
SELECT d.deptno,d.dname,e.empno,e.ename
FROM   emp e, dept d
WHERE  e.deptno(+)=d.deptno;

----------------------------------------------
| Id  | Operation                    | Name  |
----------------------------------------------
|   0 | SELECT STATEMENT             |       |
|   1 |  NESTED LOOPS OUTER          |       |
|   2 |   TABLE ACCESS FULL          | DEPT  |
|   3 |   TABLE ACCESS BY INDEX ROWID| EMP   |
|   4 |    INDEX RANGE SCAN          | EDEPT |
----------------------------------------------
   4 - access("E"."DEPTNO"(+)="D"."DEPTNO")
```

```
SELECT d.deptno,d.dname,e.empno,e.ename
FROM   emp e, dept d
WHERE  e.deptno(+)=d.deptno;

---------------------------------------
| Id | Operation               | Name |
---------------------------------------
|  0 | SELECT STATEMENT        |      |
|  1 |  HASH JOIN RIGHT OUTER  |      |
|  2 |   TABLE ACCESS FULL     | EMP  |
|  3 |   TABLE ACCESS FULL     | DEPT |
-----------------------------------
1 - access("E"."DEPTNO"(+)="D"."DEPTNO")
```

```
SELECT d.deptno,d.dname,e.empno,e.ename
FROM   emp e, dept d
WHERE  e.deptno(+)=d.deptno;

-----------------------------------
| Id | Operation          | Name |
-----------------------------------
|  0 | SELECT STATEMENT   |      |
|  1 |  HASH JOIN OUTER   |      |
|  2 |   TABLE ACCESS FULL| DEPT |
|  3 |   TABLE ACCESS FULL| EMP  |
-----------------------------------
1 - access("E"."DEPTNO"(+)="D"."DEPTNO")
```

Semijoins

Semijoins only look for the first match.

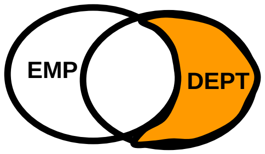


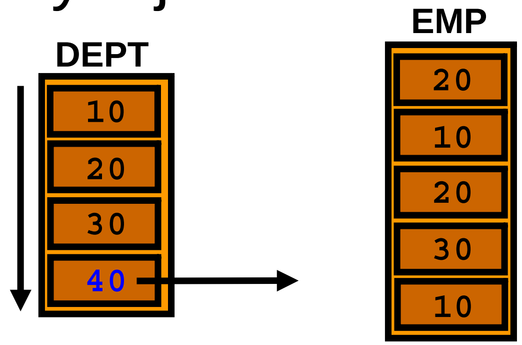
```
SELECT deptno, dname
FROM   dept
WHERE  EXISTS (SELECT 1 FROM emp WHERE emp.deptno=dept.deptno);

--------------------------------------------------------------------
| Id  | Operation            | Name | Rows  | Bytes | Cost (%CPU)|
--------------------------------------------------------------------
|   0 | SELECT STATEMENT     |      |     3 |   105 |      7  (15)|
|   1 |  HASH JOIN SEMI      |      |     3 |   105 |      7  (15)|
|   2 |   TABLE ACCESS FULL| DEPT |     4 |    88 |      3   (0)|
|   3 |   TABLE ACCESS FULL| EMP  |    14 |   182 |      3   (0)|
--------------------------------------------------------------------
   1 - access("EMP"."DEPTNO"="DEPT"."DEPTNO")
```

Antijoins

Reverse of what would have been returned by a join

```
SELECT deptno, dname
FROM   dept
WHERE  deptno not in
      (SELECT deptno FROM emp);
---------------------------------------
| Id  | Operation             | Name   |
---------------------------------------
|   0 | SELECT STATEMENT      |        |
|   1 |  NESTED LOOPS ANTI    |        |
|   2 |   TABLE ACCESS FULL   | DEPT   |
|   3 |   INDEX RANGE SCAN    | I_DEPTNO |
---------------------------------------
   3 - access("DEPTNO"="DEPTNO")
```





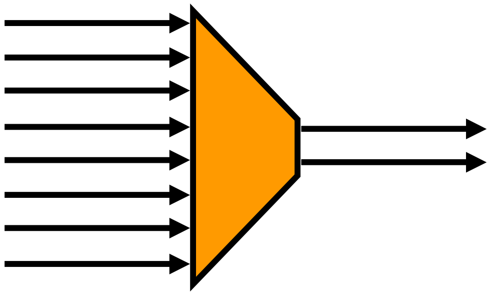
```
SELECT deptno, dname
FROM   dept
WHERE  deptno not in
      (SELECT deptno FROM emp);
-----------------------------------
| Id  | Operation           | Name |
-----------------------------------
|   0 | SELECT STATEMENT    |      |
|   1 |  HASH JOIN ANTI     |      |
|   2 |   TABLE ACCESS FULL| DEPT |
|   3 |   TABLE ACCESS FULL| EMP  |
-----------------------------------
```

Other N-Array Operations

- `FILTER`
- `CONCATENATION`
- `UNION ALL`/`UNION`
- `INTERSECT`
- `MINUS`

Filter Operations

- Accepts a set of rows
- Eliminates some of them
- Returns the rest



```
SELECT   deptno, sum(sal) SUM_SAL
FROM     emp
GROUP BY deptno
HAVING sum(sal) > 9000;

------------------------------------
| Id  | Operation            | Name |
------------------------------------
|   0 | SELECT STATEMENT     |      |
|   1 |  FILTER              |      |
|   2 |   HASH GROUP BY      |      |
|   3 |    TABLE ACCESS FULL| EMP  |
------------------------------------
   1 - filter(SUM("SAL")>9000)
```

```
SELECT deptno, dname
FROM dept d WHERE NOT EXISTS
(select 1 from emp e
 where e.deptno=d.deptno);

------------------------------------
| Id  | Operation             | Name  |
------------------------------------
|   0 | SELECT STATEMENT      |
|   1 |  FILTER               |
|   2 |   TABLE ACCESS FULL| DEPT
|   3 |   INDEX RANGE SCAN   |I_DEPTNO
------------------------------------
  1 - filter( NOT EXISTS
      (SELECT 0 FROM "EMP" "E" WHERE
      "E"."DEPTNO"=:B1))
  3 - access("E"."DEPTNO"=:B1)
```

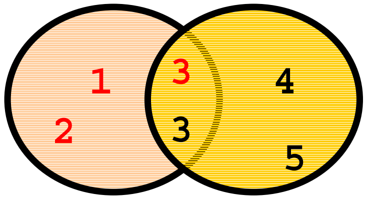
Concatenation Operation

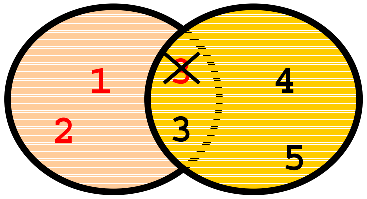
```
SELECT * FROM emp WHERE deptno=1 or sal=2;

--------------------------------------------------------------------
| Id  | Operation                     | Name     | Rows  | Bytes |
--------------------------------------------------------------------
|   0 | SELECT STATEMENT              |          |     8 |   696 |
|   1 |  CONCATENATION                |          |       |       |
|   2 |   TABLE ACCESS BY INDEX ROWID| EMP      |     4 |   348 |
|   3 |    INDEX RANGE SCAN           | I_SAL    |     2 |       |
|   4 |   TABLE ACCESS BY INDEX ROWID| EMP      |     4 |   348 |
|   5 |    INDEX RANGE SCAN           | I_DEPTNO |     2 |       |
--------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   3 - access("SAL"=2)
   4 - filter(LNNVL("SAL"=2))
   5 - access("DEPTNO"=1)
```

UNION [ALL], INTERSECT, MINUS

UNION
UNION ALL

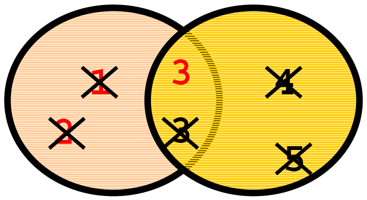
```
SORT UNIQUE
  UNION-ALL
   INDEX FULL SCAN
   INDEX FAST FULL SCAN
```





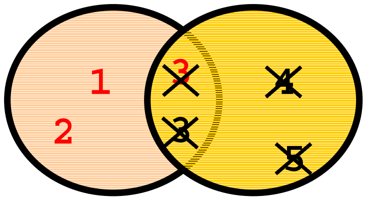
INTERSECT

```
INTERSECTION
  SORT UNIQUE NOSORT
   INDEX FULL SCAN
  SORT UNIQUE
   INDEX FAST FULL SCAN
```



MINUS

```
MINUS
  SORT UNIQUE NOSORT
   INDEX FULL SCAN
  SORT UNIQUE
   INDEX FAST FULL SCAN
```



Result Cache Operator

```
EXPLAIN PLAN FOR
SELECT /*+ RESULT_CACHE */ department_id, AVG(salary)
FROM employees
GROUP BY department_id;
```

```
--------------------------------------------------------------
| Id  | Operation           | Name                       |Rows
--------------------------------------------------------------
|    0 | SELECT STATEMENT    |                            |  11
|    1 |  RESULT CACHE       | 8fpza04gtwsfr6n595au15yj4y |
|    2 |   HASH GROUP BY     |                            |  11
|    3 |    TABLE ACCESS FULL| EMPLOYEES                  | 107
--------------------------------------------------------------
```

Summary

In this lesson, you should have learned to:

- Describe most of the SQL operators
- List the possible access paths
- Explain how join operations are performed

Practice 4: Overview

This practice covers the following topics:

- Using different access paths for better optimization
  - Case 14 to case 16
- Using the result cache

5

Interpreting Execution Plans

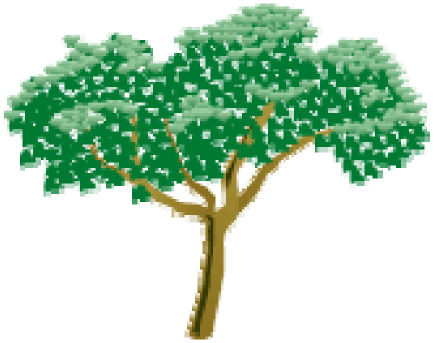
Objectives

After completing this lesson, you should be able to:

- Gather execution plans
- Display execution plans
- Interpret execution plans

What Is an Execution Plan?

- The execution plan of a SQL statement is composed of small building blocks called row sources for serial execution plans.
- The combination of row sources for a statement is called the execution plan.
- By using parent-child relationships, the execution plan can be displayed in a tree-like structure (text or graphical).



Where to Find Execution Plans?

- `PLAN_TABLE` (`EXPLAIN PLAN` or SQL*Plus autotrace)
- `V$SQL_PLAN` (Library Cache)
- `V$SQL_PLAN_MONITOR` (11*g*)
- `DBA_HIST_SQL_PLAN` (AWR)
- `STATS$SQL_PLAN` (Statspack)
- SQL Management Base (SQL Plan Management Baselines)
- SQL tuning set
- Trace files generated by `DBMS_MONITOR`
- Event 10053 trace file
- Process state dump trace file since 10*g*R2

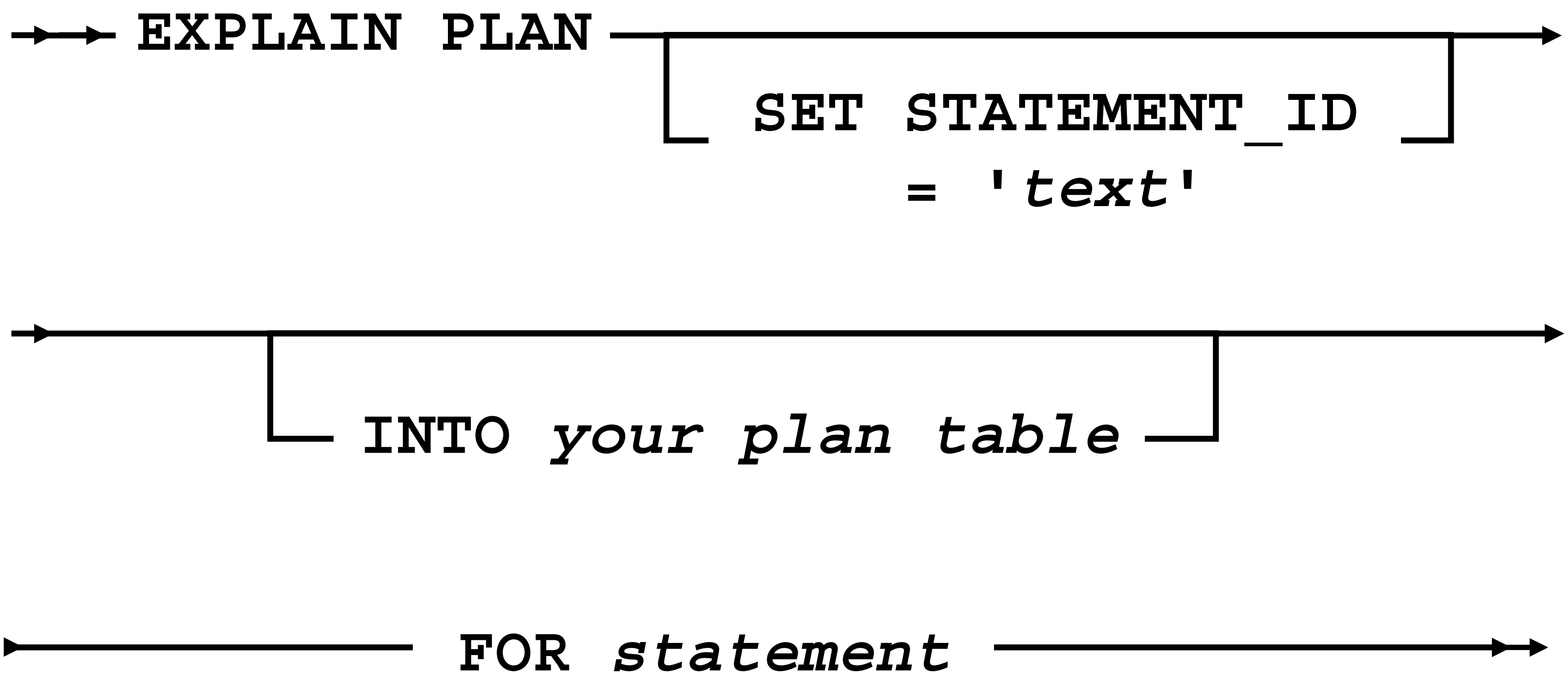
Viewing Execution Plans

- The `EXPLAIN PLAN` command followed by:
  - `SELECT from PLAN_TABLE`
  - `DBMS_XPLAN.DISPLAY()`
- SQL*Plus Autotrace: `SET AUTOTRACE ON`
- `DBMS_XPLAN.DISPLAY_CURSOR()`
- `DBMS_XPLAN.DISPLAY_AWR()`
- `DBMS_XPLAN.DISPLAY_SQLSET()`
- `DBMS_XPLAN.DISPLAY_SQL_PLAN_BASELINE()`

The EXPLAIN PLAN Command

- Generates an optimizer execution plan
- Stores the plan in PLAN_TABLE
- Does not execute the statement itself

The EXPLAIN PLAN Command



EXPLAIN PLAN

SET STATEMENT_ID = '*text*'

INTO *your plan table*

FOR *statement*

The EXPLAIN PLAN Command: Example

```
SQL> EXPLAIN PLAN
  2  SET STATEMENT_ID = 'demo01' FOR
  3  SELECT e.last_name, d.department_name
  4  FROM hr.employees e, hr.departments d
  5  WHERE e.department_id = d.department_id;

Explained.

SQL>
```

Note: The `EXPLAIN PLAN` command does not actually execute the statement.

PLAN_TABLE

- PLAN_TABLE:
  - Is automatically created to hold the EXPLAIN PLAN output.
  - You can create your own using utlxplan.sql.
  - Advantage: SQL is not executed
  - Disadvantage: May not be the actual execution plan
- PLAN_TABLE is hierarchical.
- Hierarchy is established with the ID and PARENT_ID columns.

Displaying from PLAN_TABLE: Typical

```
SQL> EXPLAIN PLAN SET STATEMENT_ID = 'demo01' FOR SELECT * FROM emp
  2  WHERE ename = 'KING';

Explained.

SQL> SET LINESIZE 130
SQL> SET PAGESIZE 0
SQL> select * from table(DBMS_XPLAN.DISPLAY());

Plan hash value: 3956160932
---------------------------------------------------------------------------
| Id  | Operation         | Name | Rows  | Bytes | Cost (%CPU)| Time     |
---------------------------------------------------------------------------
|   0 | SELECT STATEMENT  |      |     1 |    37 |     3   (0)| 00:00:01 |
|*  1 |  TABLE ACCESS FULL| EMP  |     1 |    37 |     3   (0)| 00:00:01 |
---------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------
   1 - filter("ENAME"='KING')
```

Displaying from PLAN_TABLE: ALL

```
SQL> select * from table(DBMS_XPLAN.DISPLAY(null,null,'ALL'));

Plan hash value: 3956160932
---------------------------------------------------------------------------
| Id  | Operation         | Name | Rows  | Bytes | Cost (%CPU)| Time     |
---------------------------------------------------------------------------
|   0 | SELECT STATEMENT  |      |     1 |    37 |     3   (0)| 00:00:01 |
|*  1 |  TABLE ACCESS FULL| EMP  |     1 |    37 |     3   (0)| 00:00:01 |
---------------------------------------------------------------------------
Query Block Name / Object Alias (identified by operation id):
--------------------------------------------------------------
   1 - SEL$1 / EMP@SEL$1
Predicate Information (identified by operation id):
----------------------------------------------------
   1 - filter("ENAME"='KING')
Column Projection Information (identified by operation id):
------------------------------------------------------------
 1 - "EMP"."EMPNO"[NUMBER,22], "ENAME"[VARCHAR2,10], "EMP"."JOB"[VARCHAR2,9],
     "EMP"."MGR"[NUMBER,22], "EMP"."HIREDATE"[DATE,7], "EMP"."SAL"[NUMBER,22],
     "EMP"."COMM"[NUMBER,22], "EMP"."DEPTNO"[NUMBER,22]
```

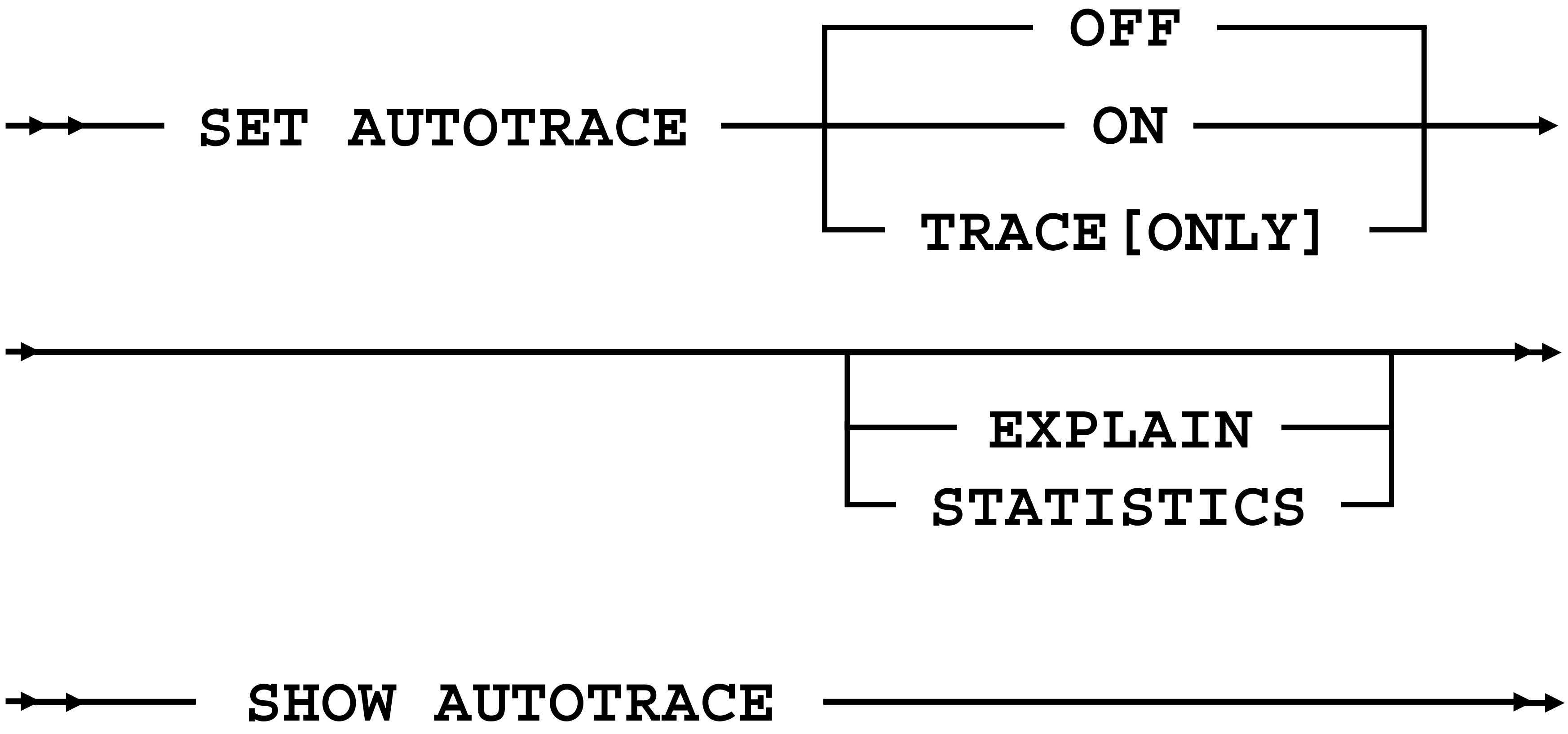
Displaying from PLAN_TABLE: ADVANCED

```
select plan_table_output from table(DBMS_XPLAN.DISPLAY(null,null,'ADVANCED
 -PROJECTION -PREDICATE -ALIAS'));
Plan hash value: 3956160932
---------------------------------------------------------------------------
| Id  | Operation         | Name | Rows  | Bytes | Cost (%CPU)| Time     |
---------------------------------------------------------------------------
|   0 | SELECT STATEMENT  |      |     1 |    37 |     3   (0)| 00:00:01 |
|   1 |  TABLE ACCESS FULL| EMP  |     1 |    37 |     3   (0)| 00:00:01 |
---------------------------------------------------------------------------
Outline Data
-------------
  /*+
      BEGIN_OUTLINE_DATA
      FULL(@"SEL$1" "EMP"@"SEL$1")
      OUTLINE_LEAF(@"SEL$1")
      ALL_ROWS
      DB_VERSION('11.1.0.6')
      OPTIMIZER_FEATURES_ENABLE('11.1.0.6')
      IGNORE_OPTIM_EMBEDDED_HINTS
      END_OUTLINE_DATA
  */
```

AUTOTRACE

- AUTOTRACE is a SQL*Plus facility.
- Introduced with Oracle7.3
- Needs a PLAN_TABLE
- Needs the PLUSTRACE role to retrieve statistics from some V$ views
- By default, it produces the execution plan and statistics after running the query.
- May not be the actual plan when using bind peeking (recursive EXPLAIN PLAN)

The AUTOTRACE Syntax



SET AUTOTRACE

OFF

ON

TRACE [ONLY]

EXPLAIN

STATISTICS

SHOW AUTOTRACE

AUTOTRACE: Examples

- To start tracing statements using AUTOTRACE:

```
SQL> set autotrace on
```

- To display the execution plan only without execution:

```
SQL> set autotrace traceonly explain
```

- To display rows and statistics:

```
SQL> set autotrace on statistics
```

- To get the plan and the statistics only (suppress rows):

```
SQL> set autotrace traceonly
```

AUTOTRACE: Statistics

```
SQL> show autotrace
autotrace OFF
SQL> set autotrace traceonly statistics
SQL> SELECT * FROM oe.products;

288 rows selected.

Statistics
--------------------------------------------------------
       1334  recursive calls
          0  db block gets
        686  consistent gets
        394  physical reads
          0  redo size
     103919  bytes sent via SQL*Net to client
        629  bytes received via SQL*Net from client
         21  SQL*Net roundtrips to/from client
         22  sorts (memory)
          0  sorts (disk)
        288  rows processed
```

Using the V$SQL_PLAN View

- V$SQL_PLAN provides a way of examining the execution plan for cursors that are still in the library cache.
- V$SQL_PLAN is very similar to PLAN_TABLE:
  - PLAN_TABLE shows a theoretical plan that can be used if this statement were to be executed.
  - V$SQL_PLAN contains the actual plan used.
- It contains the execution plan of every cursor in the library cache (including child).
- Link to V$SQL:
  - ADDRESS, HASH_VALUE, and CHILD_NUMBER

The V$SQL_PLAN Columns

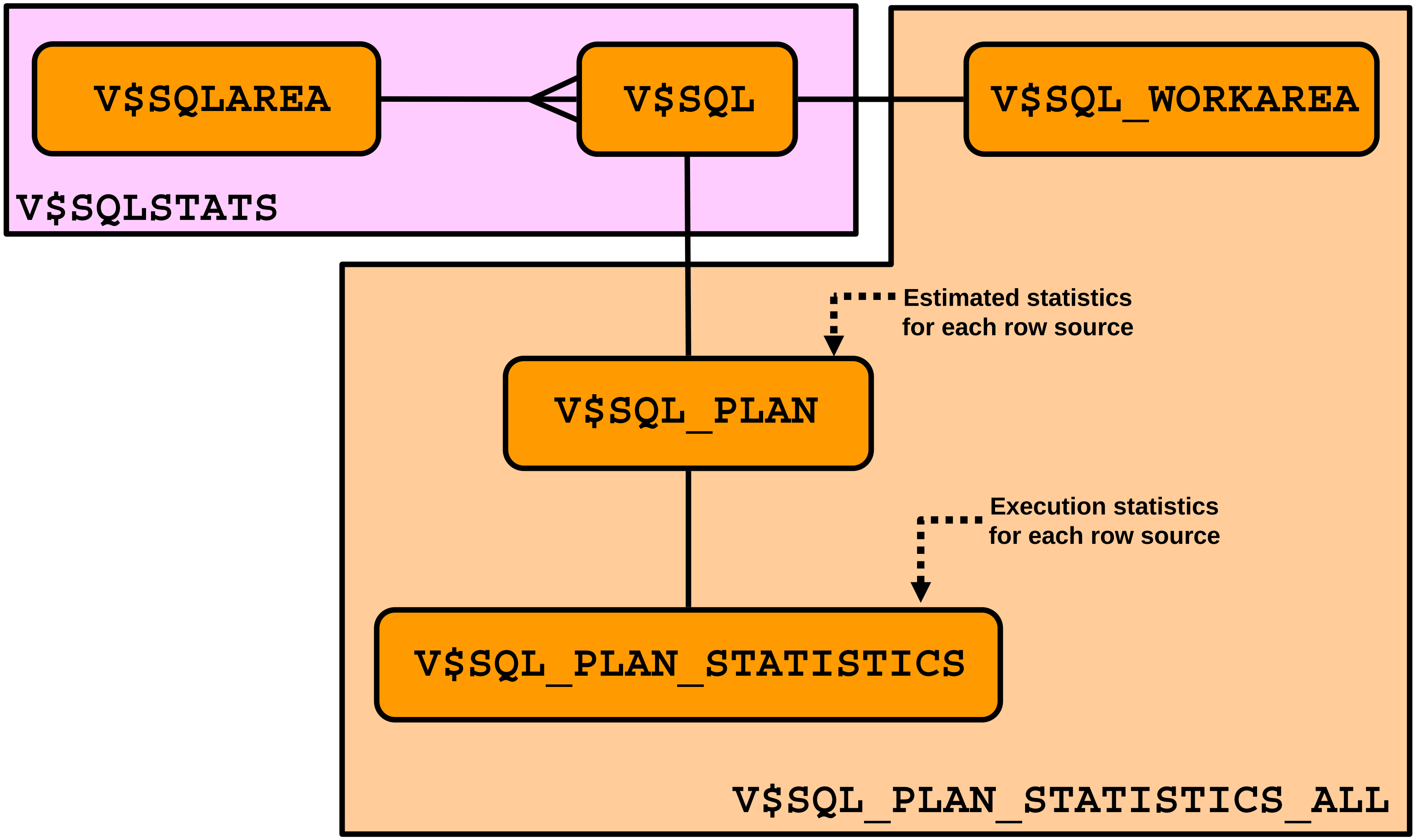
HASH_VALUE	Hash value of the parent statement in the library cache
ADDRESS	Address of the handle to the parent for this cursor
CHILD_NUMBER	Child cursor number using this execution plan
POSITION	Order of processing for all operations that have the same PARENT_ID
PARENT_ID	ID of the next execution step that operates on the output of the current step
ID	Number assigned to each step in the execution plan
PLAN_HASH_VALUE	Numerical representation of the SQL plan for the cursor

Note: This is only a partial listing of the columns.

The V$SQL_PLAN_STATISTICS View

- V$SQL_PLAN_STATISTICS provides actual execution statistics:
  - STATISTICS_LEVEL set to ALL
  - The GATHER_PLAN_STATISTICS hint
- V$SQL_PLAN_STATISTICS_ALL enables side-by-side comparisons of the optimizer estimates with the actual execution statistics.

Links Between Important Dynamic Performance Views


V$SQLAREA
V$SQL
V$SQL_WORKAREA
V$SQLSTATS
Estimated statistics for each row source
V$SQL_PLAN
Execution statistics for each row source
V$SQL_PLAN_STATISTICS
V$SQL_PLAN_STATISTICS_ALL

Querying V$SQL_PLAN

```
SELECT PLAN_TABLE_OUTPUT FROM
TABLE(DBMS_XPLAN.DISPLAY_CURSOR('47ju6102uvq5q'));
```

```
SQL_ID  47ju6102uvq5q, child number 0
-------------------------------------
SELECT e.last_name, d.department_name
FROM hr.employees e, hr.departments d WHERE
e.department_id =d.department_id

Plan hash value: 2933537672
--------------------------------------------------------------------------------
| Id  | Operation                    | Name        | Rows  | Bytes | Cost (%CPU|
--------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |             |       |       |     6 (100|
|   1 |  MERGE JOIN                  |             |   106 |  2862 |     6  (17|
|   2 |   TABLE ACCESS BY INDEX ROWID| DEPARTMENTS |    27 |   432 |     2   (0|
|   3 |    INDEX FULL SCAN           | DEPT_ID_PK  |    27 |       |     1   (0|
|*  4 |   SORT JOIN                  |             |   107 |  1177 |     4  (25|
|   5 |    TABLE ACCESS FULL         | EMPLOYEES   |   107 |  1177 |     3   (0|
--------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------


   4 - access("E"."DEPARTMENT_ID"="D"."DEPARTMENT_ID")
       filter("E"."DEPARTMENT_ID"="D"."DEPARTMENT_ID")

24 rows selected.
```

Automatic Workload Repository (AWR)

- Collects, processes, and maintains performance statistics for problem-detection and self-tuning purposes
- Statistics include:
  - Object statistics
  - Time-model statistics
  - Some system and session statistics
  - Active Session History (ASH) statistics
- Automatically generates snapshots of the performance data

Managing AWR with PL/SQL

- Creating snapshots:

```
SQL> exec DBMS_WORKLOAD_REPOSITORY.CREATE_SNAPSHOT ('ALL');
```

- Dropping snapshots:

```
SQL> exec DBMS_WORKLOAD_REPOSITORY.DROP_SNAPSHOT_RANGE –
    (low_snap_id => 22, high_snap_id => 32, dbid => 3310949047);
```

- Managing snapshot settings:

```
SQL> exec DBMS_WORKLOAD_REPOSITORY.MODIFY_SNAPSHOT_SETTINGS –
     (retention => 43200, interval => 30, dbid => 3310949047);
```

Important AWR Views

- V$ACTIVE_SESSION_HISTORY
- V$ metric views
- DBA_HIST views:
  - DBA_HIST_ACTIVE_SESS_HISTORY
  - DBA_HIST_BASELINE
    DBA_HIST_DATABASE_INSTANCE
  - DBA_HIST_SNAPSHOT
  - DBA_HIST_SQL_PLAN
  - DBA_HIST_WR_CONTROL

Querying the AWR

- Retrieve all execution plans stored for a particular `SQL_ID`.

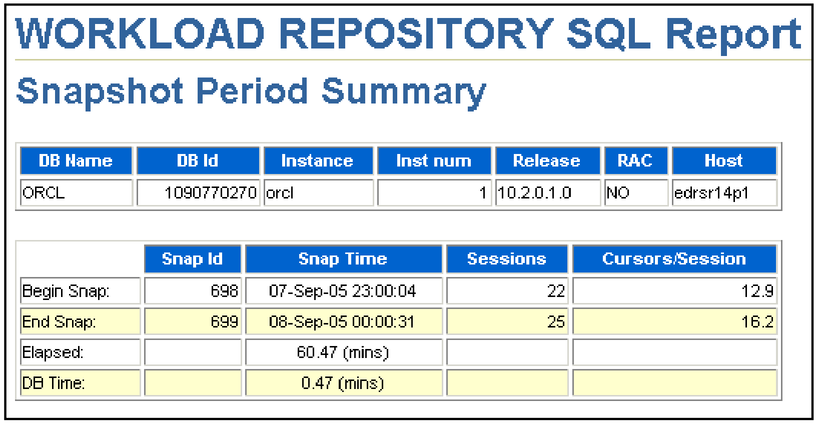
```
SQL> SELECT PLAN_TABLE_OUTPUT FROM TABLE (DBMS_XPLAN.DISPLAY_AWR('454rug2yva18w'));


PLAN_TABLE_OUTPUT
------------------------------------------------------------------------------------
SQL_ID 454rug2yva18w
--------------------
select /* example */ * from hr.employees natural join hr.departments

Plan hash value: 4179021502

----------------------------------------------------------------------------------
| Id  | Operation          | Name        | Rows  | Bytes | Cost (%CPU)| Time     |
----------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |             |       |       |     6 (100)|          |
|   1 |  HASH JOIN         |             |    11 |   968 |     6  (17)| 00:00:01 |
|   2 |   TABLE ACCESS FULL| DEPARTMENTS |    11 |   220 |     2   (0)| 00:00:01 |
|   2 |   TABLE ACCESS FULL| DEPARTMENTS |    11 |   220 |     2   (0)| 00:00:01 |
|   3 |   TABLE ACCESS FULL| EMPLOYEES   |   107 |  7276 |     3   (0)| 00:00:01 |
----------------------------------------------------------------------------------
```

- Display all execution plans of all statements containing “JF.”

```
SELECT tf.* FROM DBA_HIST_SQLTEXT ht, table
    (DBMS_XPLAN.DISPLAY_AWR(ht.sql_id,null, null,  'ALL' )) tf
 WHERE ht.sql_text like '%JF%';
```

Generating SQL Reports from AWR Data

```
SQL> @$ORACLE_HOME/rdbms/admin/awrsqrpt

Specify the Report Type …
Would you like an HTML report, or a plain text report?
Specify the number of days of snapshots to choose from
Specify the Begin and End Snapshot Ids …
Specify the SQL Id …
Enter value for sql_id: 6g1p4s9ra6ag8
Specify the Report Name …
```

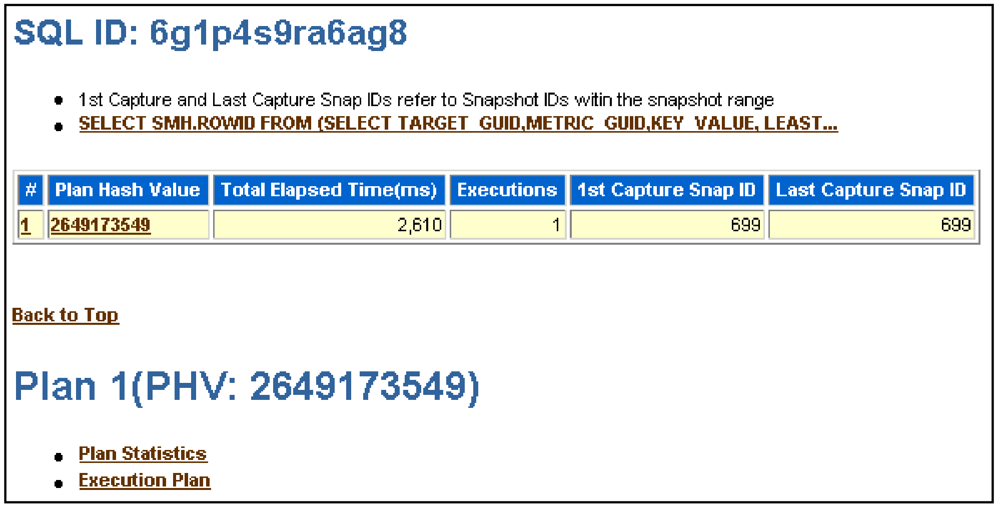


WORKLOAD REPOSITORY SQL Report

Snapshot Period Summary

DB Name	DB Id	Instance	Inst num	Release	RAC	Host
ORCL	1090770270	orcl	1	10.2.0.1.0	NO	edrsr14p1

	Snap Id	Snap Time	Sessions	Cursors/Session
Begin Snap:	698	07-Sep-05 23:00:04	22	12.9
End Snap:	699	08-Sep-05 00:00:31	25	16.2
Elapsed:		60.47 (mins)		
DB Time:		0.47 (mins)		



SQL ID: 6g1p4s9ra6ag8

- 1st Capture and Last Capture Snap IDs refer to Snapshot IDs witin the snapshot range
- SELECT SMH.ROWID FROM (SELECT TARGET_GUID,METRIC_GUID,KEY_VALUE, LEAST...

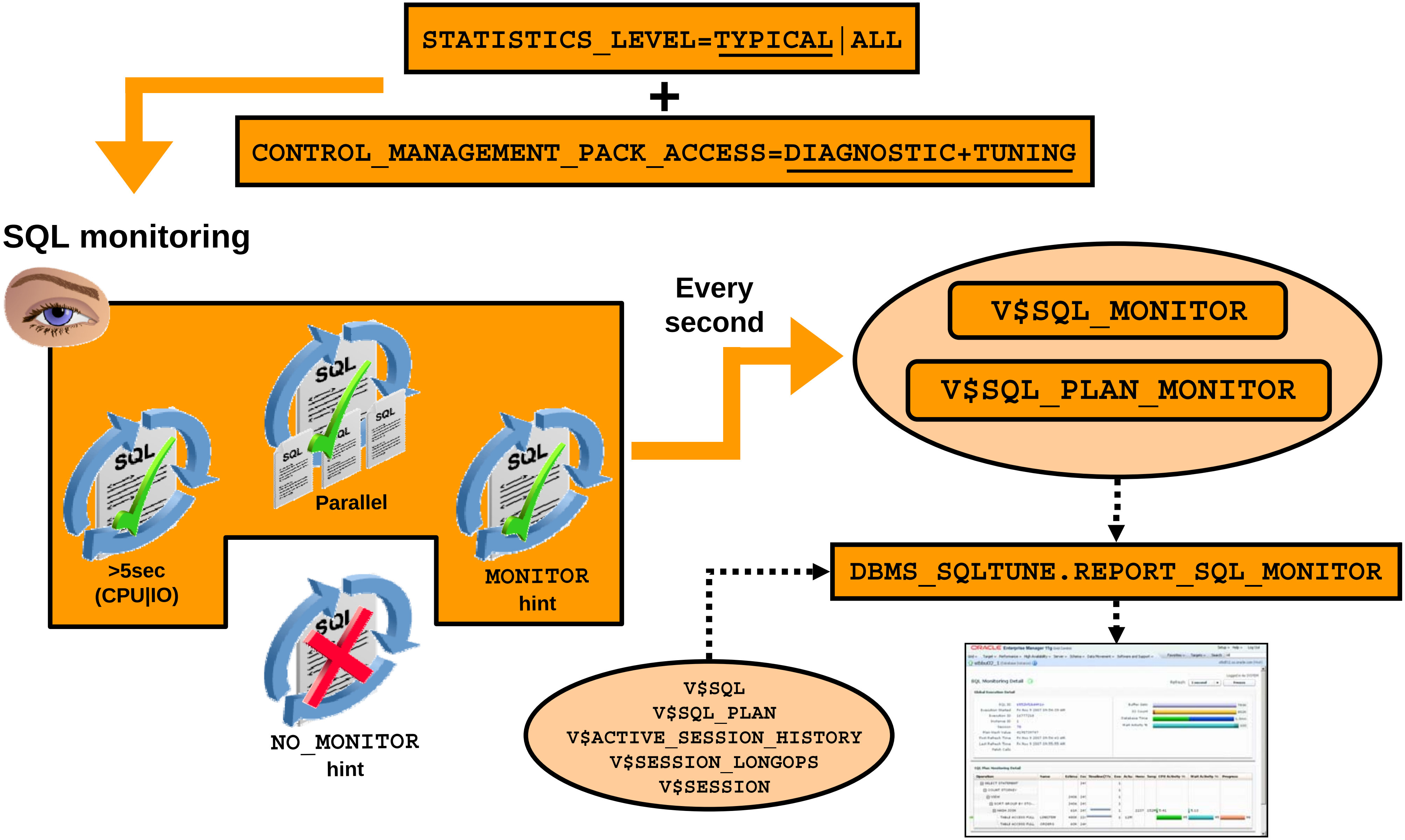
#	Plan Hash Value	Total Elapsed Time(ms)	Executions	1st Capture Snap ID	Last Capture Snap ID
1	2649173549	2,610	1	699	699

Back to Top

Plan 1(PHV: 2649173549)

- Plan Statistics
- Execution Plan

SQL Monitoring: Overview



STATISTICS_LEVEL=TYPICAL|ALL

+

CONTROL_MANAGEMENT_PACK_ACCESS=DIAGNOSTIC+TUNING

SQL monitoring

Parallel

>5sec
(CPU|IO)

MONITOR
hint

NO_MONITOR
hint

Every
second

V$SQL_MONITOR

V$SQL_PLAN_MONITOR

DBMS_SQLTUNE.REPORT_SQL_MONITOR

V$SQL
V$SQL_PLAN
V$ACTIVE_SESSION_HISTORY
V$SESSION_LONGOPS
V$SESSION

SQL Monitoring Report: Example

```
SQL> set long 10000000
SQL> set longchunksize 10000000
SQL> set linesize 200
SQL> select dbms_sqltune.report_sql_monitor from dual;

SQL Monitoring Report

SQL Text
--------------------------
select count(*) from sales

Global Information
 Status               :  EXECUTING
 Instance ID          :  1
 Session ID           :  125
 SQL ID               :  fazrk33ng71km
 SQL Execution ID     :  16777216
 Plan Hash Value      :  1047182207
 Execution Started    :  02/19/2008 21:01:18
 First Refresh Time   :  02/19/2008 21:01:22
 Last Refresh Time    :  02/19/2008 21:01:42
------------------------------------------------------------
| Elapsed |   Cpu   |    IO    |   Other  | Buffer | Reads |
| Time(s) | Time(s) | Waits(s) | Waits(s) |  Gets  |       |
------------------------------------------------------------
|      22 |    3.36 |     0.01 |       19 |   259K |  199K |
------------------------------------------------------------
```

In a different session

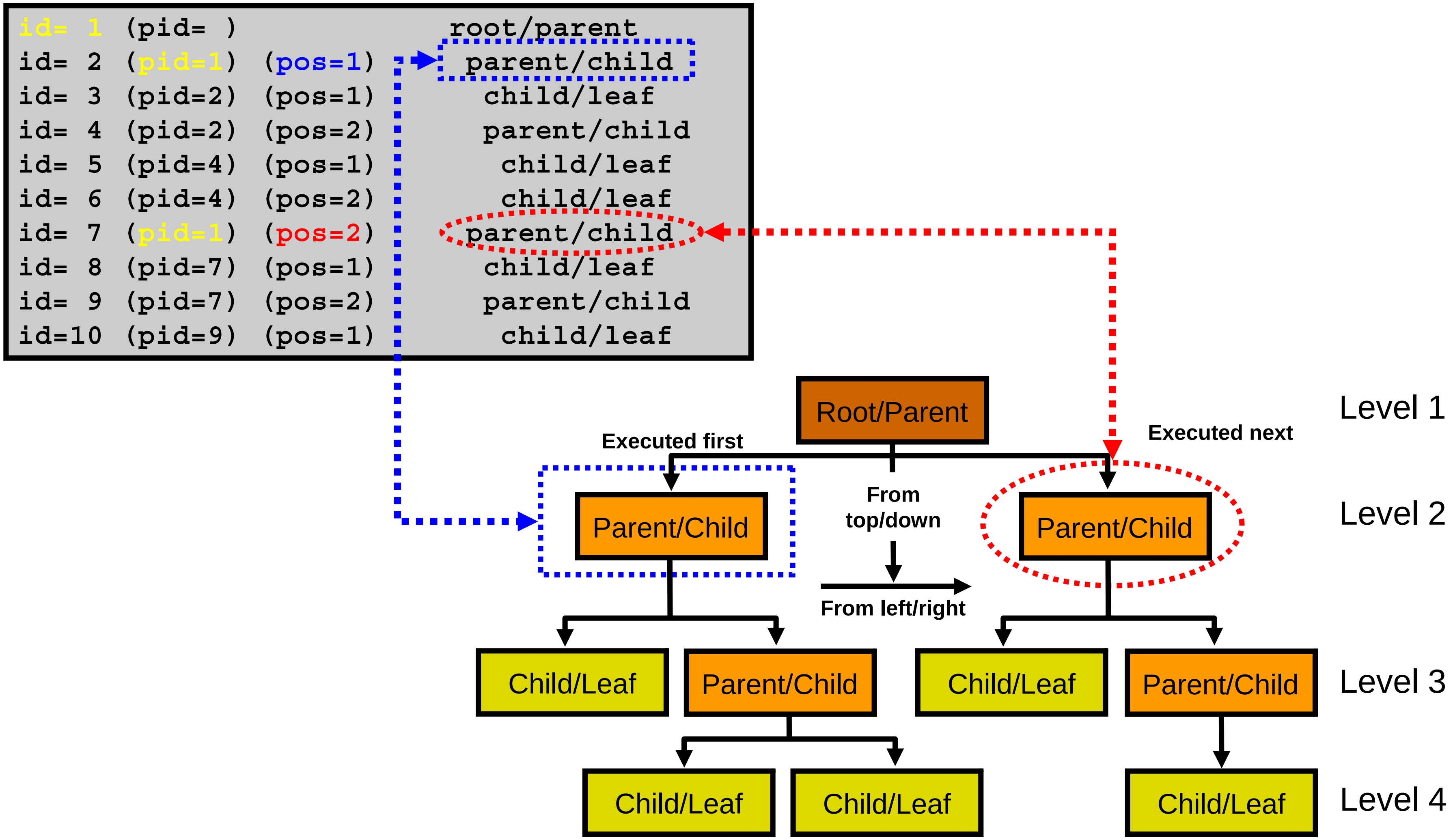
```
SQL> select count(*) from sales;
```

SQL Monitoring Report: Example

```
SQL Plan Monitoring Details
==================================================================================
| Id    |        Operation         | Name  |  Rows   | Cost  |   Time     | Start  |
|       |                          |       | (Estim) |       | Active(s)  | Active |
==================================================================================
|     0 | SELECT STATEMENT         |       |         | 78139 |            |        |
|     1 |   SORT AGGREGATE         |       |       1 |       |            |        |
| ->  2 |    TABLE ACCESS FULL     | SALES |  53984K | 78139 |         23 |     +1 |
|       |                          |       |         |       |            |        |
==================================================================================



================================================================================
Starts |   Rows    | Activity  |             Activity Detail           | Progress  |
       | (Actual)  | (percent) |               (sample #)              |           |
     1 |           |           |                                       |           |
     1 |           |           |                                       |           |
     1 |   42081K  |    100.00 | Cpu (4)                               | 74%       |
================================================================================
```

Interpreting an Execution Plan

Transform it into a tree.

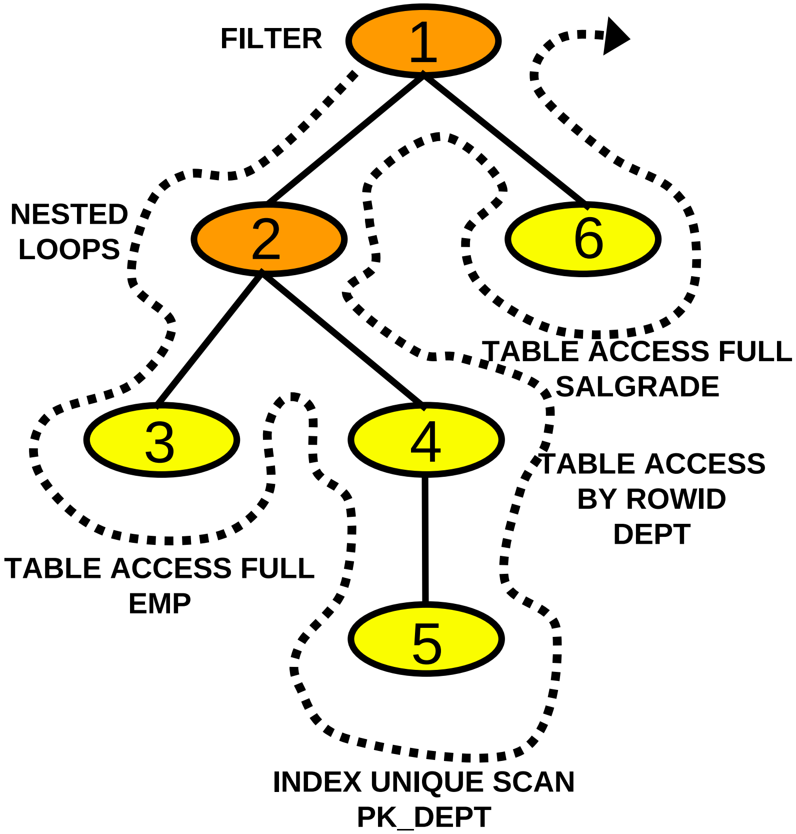


```
id= 1 (pid= )            root/parent
id= 2 (pid=1) (pos=1)     parent/child
id= 3 (pid=2) (pos=1)      child/leaf
id= 4 (pid=2) (pos=2)      parent/child
id= 5 (pid=4) (pos=1)       child/leaf
id= 6 (pid=4) (pos=2)       child/leaf
id= 7 (pid=1) (pos=2)     parent/child
id= 8 (pid=7) (pos=1)      child/leaf
id= 9 (pid=7) (pos=2)      parent/child
id=10 (pid=9) (pos=1)       child/leaf
```

Execution Plan Interpretation: Example 1

```
SELECT /*+ RULE */ ename,job,sal,dname
FROM emp,dept
WHERE dept.deptno=emp.deptno and not exists(SELECT *
                                            FROM salgrade
                                            WHERE emp.sal between losal and hisal);
```

```
--------------------------------------------------------
| Id  | Operation                       | Name        |
--------------------------------------------------------
|   0 | SELECT STATEMENT                |             |
|*  1 |  FILTER                         |             |
|   2 |   NESTED LOOPS                  |             |
|   3 |    TABLE ACCESS FULL            | EMP         |
|   4 |    TABLE ACCESS BY INDEX ROWID| DEPT        |
|*  5 |     INDEX UNIQUE SCAN           | PK_DEPT     |
|*  6 |   TABLE ACCESS FULL             | SALGRADE    |
--------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   1 - filter( NOT EXISTS
     (SELECT 0 FROM "SALGRADE" "SALGRADE" WHERE
      "HISAL">=:B1 AND "LOSAL"<=:B2))
   5 - access("DEPT"."DEPTNO"="EMP"."DEPTNO")
   6 - filter("HISAL">=:B1 AND "LOSAL"<=:B2)
```


FILTER
1
NESTED LOOPS
2
6
TABLE ACCESS FULL SALGRADE
3
4
TABLE ACCESS BY ROWID DEPT
TABLE ACCESS FULL EMP
5
INDEX UNIQUE SCAN PK_DEPT

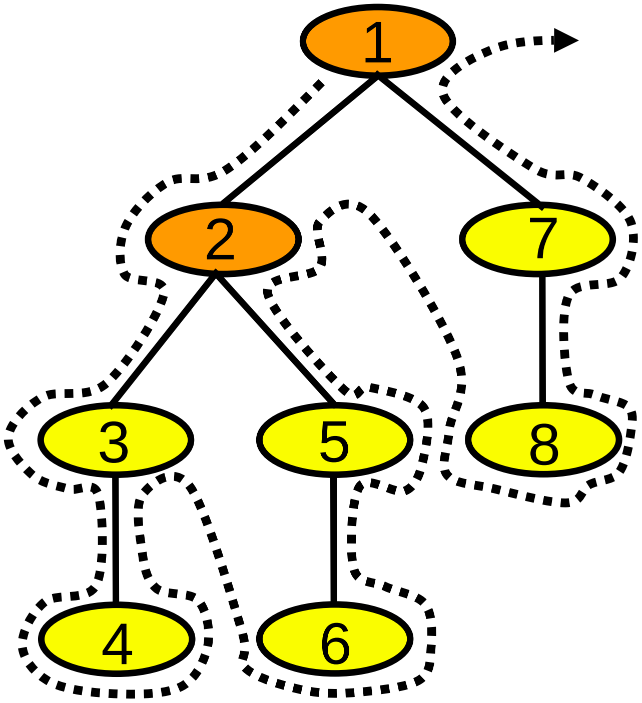
Execution Plan Interpretation: Example 1

```
SQL> alter session set statistics_level=ALL;

Session altered.

SQL> select /*+ RULE to make sure it reproduces 100% */ ename,job,sal,dname
from emp,dept where dept.deptno = emp.deptno and not exists (select * from salgrade
where emp.sal between losal and hisal);

no rows selected

SQL> select * from table(dbms_xplan.display_cursor(null,null,'TYPICAL IOSTATS
LAST'));

SQL_ID  274019myw3vuf, child number 0
-------------------------------------
…
Plan hash value: 1175760222
--------------------------------------------------------------------------------
| Id  | Operation                    | Name     | Starts | A-Rows | Buffers |
--------------------------------------------------------------------------------
|*  1 |  FILTER                      |          |      1 |      0 |      61 |
|   2 |   NESTED LOOPS               |          |      1 |     14 |      25 |
|   3 |    TABLE ACCESS FULL         | EMP      |      1 |     14 |       7 |
|   4 |    TABLE ACCESS BY INDEX ROWID| DEPT     |     14 |     14 |      18 |
|*  5 |     INDEX UNIQUE SCAN        | PK_DEPT  |     14 |     14 |       4 |
|*  6 |   TABLE ACCESS FULL          | SALGRADE |     12 |     12 |      36 |
--------------------------------------------------------------------------------
…
```

Execution Plan Interpretation: Example 2

```
SQL> select /*+ USE_NL(d) use_nl(m) */ m.last_name as dept_manager
  2  ,      d.department_name
  3  ,      l.street_address
  4  from   hr.employees m   join
  5         hr.departments d on (d.manager_id = m.employee_id)
  6         natural join
  7         hr.locations l
  8  where  l.city = 'Seattle';
```

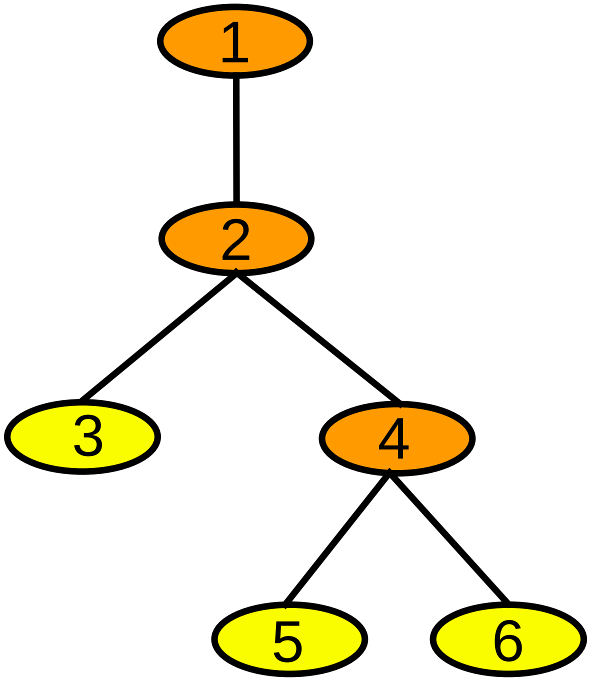
```
0    SELECT STATEMENT
1 0   NESTED LOOPS
2 1    NESTED LOOPS
3 2     TABLE ACCESS BY INDEX ROWID LOCATIONS
4 3      INDEX RANGE SCAN           LOC_CITY_IX
5 2     TABLE ACCESS BY INDEX ROWID DEPARTMENTS
6 5      INDEX RANGE SCAN           DEPT_LOCATION_IX
7 1    TABLE ACCESS BY INDEX ROWID  EMPLOYEES
8 7     INDEX UNIQUE SCAN           EMP_EMP_ID_PK
```

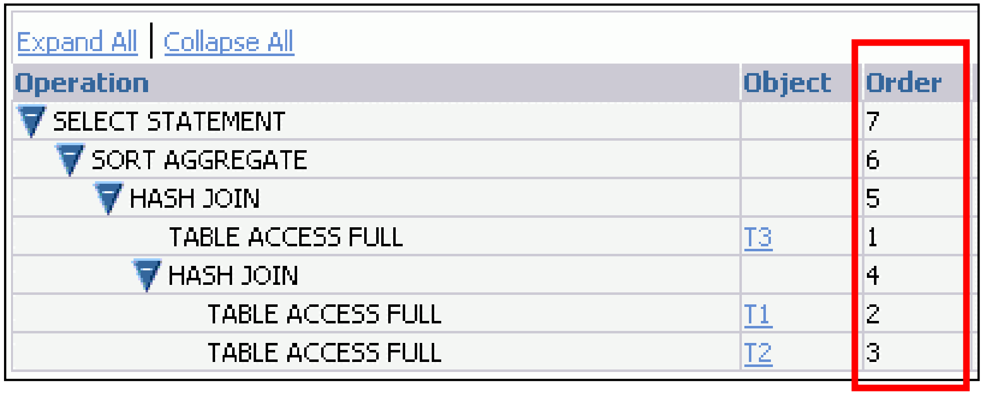


Execution Plan Interpretation: Example 3

```
select /*+ ORDERED USE_HASH(b) SWAP_JOIN_INPUTS(c) */ max(a.i)
from t1 a, t2 b, t3 c
where a.i = b.i and a.i = c.i;
```

```
0    SELECT STATEMENT
1     SORT AGGREGATE
2 1    HASH JOIN
3 2     TABLE ACCESS FULL T3
4 2     HASH JOIN
5 4      TABLE ACCESS FULL T1
6 4      TABLE ACCESS FULL T2
```



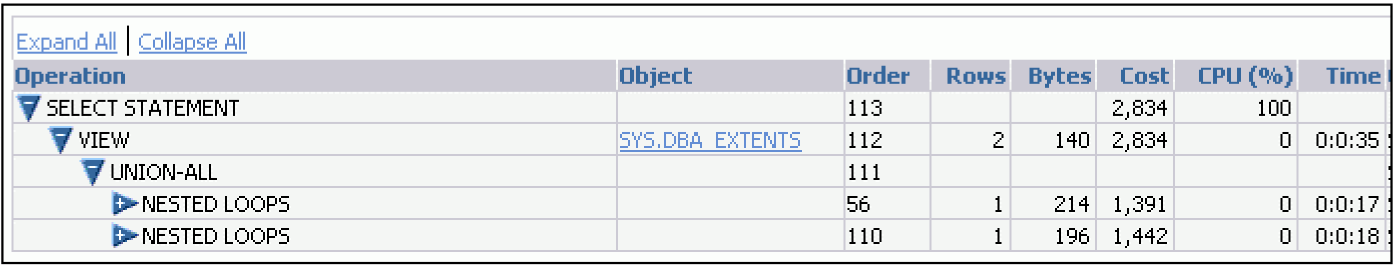


Operation	Object	Order
SELECT STATEMENT		7
SORT AGGREGATE		6
HASH JOIN		5
TABLE ACCESS FULL	T3	1
HASH JOIN		4
TABLE ACCESS FULL	T1	2
TABLE ACCESS FULL	T2	3

Join order is: `T1 - T2 - T3`

Reading More Complex Execution Plans

```
SELECT owner , segment_name , segment_type
FROM dba_extents
WHERE file_id = 1
AND   123213 BETWEEN block_id AND block_id + blocks -1;
```



Operation	Object	Order	Rows	Bytes	Cost	CPU (%)	Time
SELECT STATEMENT		113			2,834	100	
VIEW	SYS.DBA_EXTENTS	112	2	140	2,834	0	0:0:35
UNION-ALL		111					
NESTED LOOPS		56	1	214	1,391	0	0:0:17
NESTED LOOPS		110	1	196	1,442	0	0:0:18

Collapse using indentation
and
focus on operations consuming most resources.

Reviewing the Execution Plan

- Drive from the table that has most selective filter.
- Look for the following:
  - Driving table has the best filter
  - Fewest number of rows are returned to the next step
  - The join method is appropriate for the number of rows returned
  - Views are correctly used
  - Unintentional Cartesian products
  - Tables accessed efficiently

Looking Beyond Execution Plans

- An execution plan alone cannot tell you whether a plan is good or not.
- May need additional testing and tuning:
  - SQL Tuning Advisor
  - SQL Access Advisor
  - SQL Performance Analyzer
  - SQL Monitoring
  - Tracing

Summary

In this lesson, you should have learned how to:

- Gather execution plans
- Display execution plans
- Interpret execution plans

Practice 5: Overview

This practice covers the following topics:

- Using different techniques to extract execution plans
- Using SQL monitoring

6

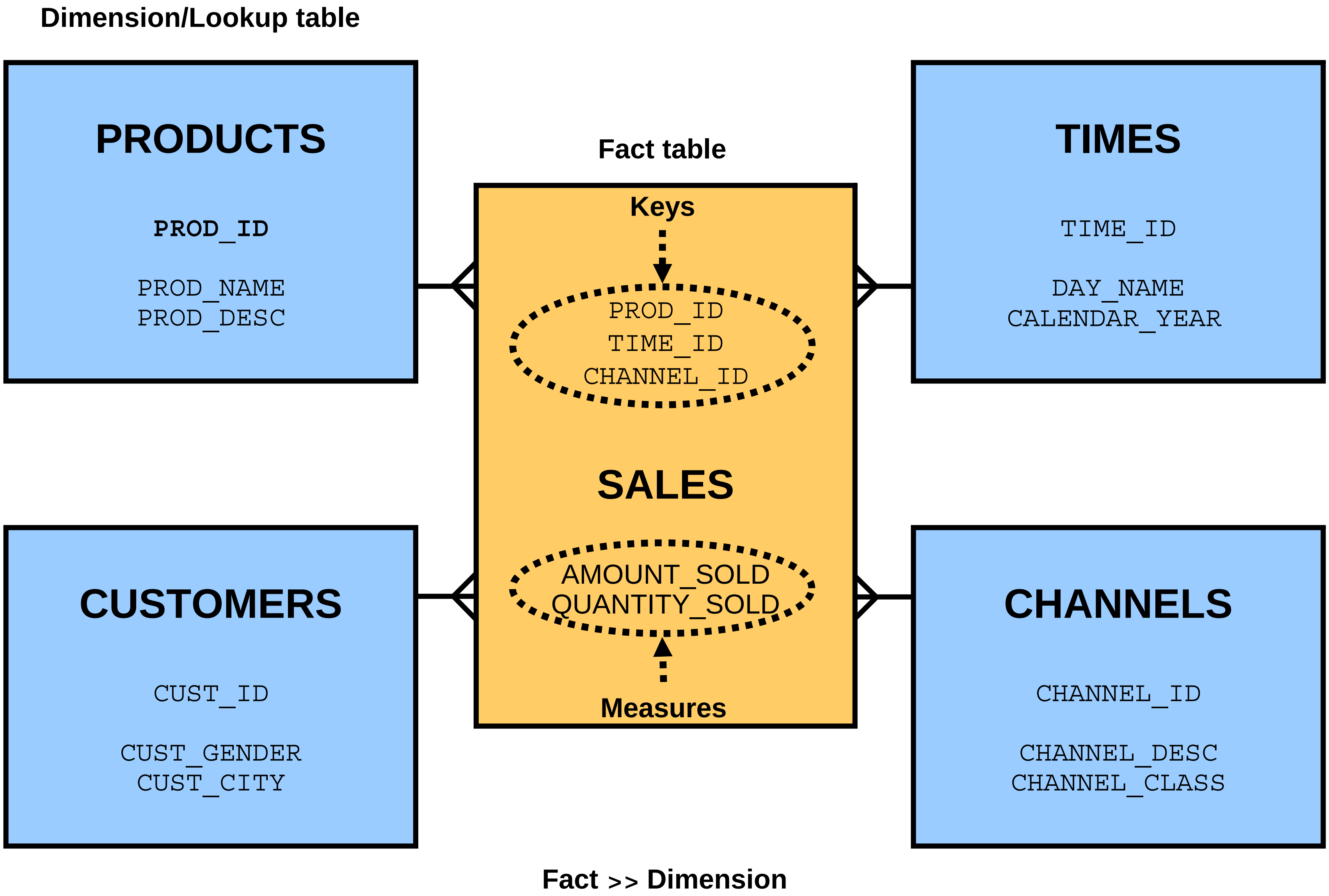
Case Study: Star Transformation

Objectives

After completing this lesson, you should be able to:

- Define a star schema
- Show a star query plan without transformation
- Define the star transformation requirements
- Show a star query plan after transformation

The Star Schema Model



Dimension/Lookup table

PRODUCTS

PROD_ID

PROD_NAME
PROD_DESC

Fact table

Keys

PROD_ID
TIME_ID
CHANNEL_ID

SALES

AMOUNT_SOLD
QUANTITY_SOLD

Measures

TIMES

TIME_ID

DAY_NAME
CALENDAR_YEAR

CUSTOMERS

CUST_ID

CUST_GENDER
CUST_CITY

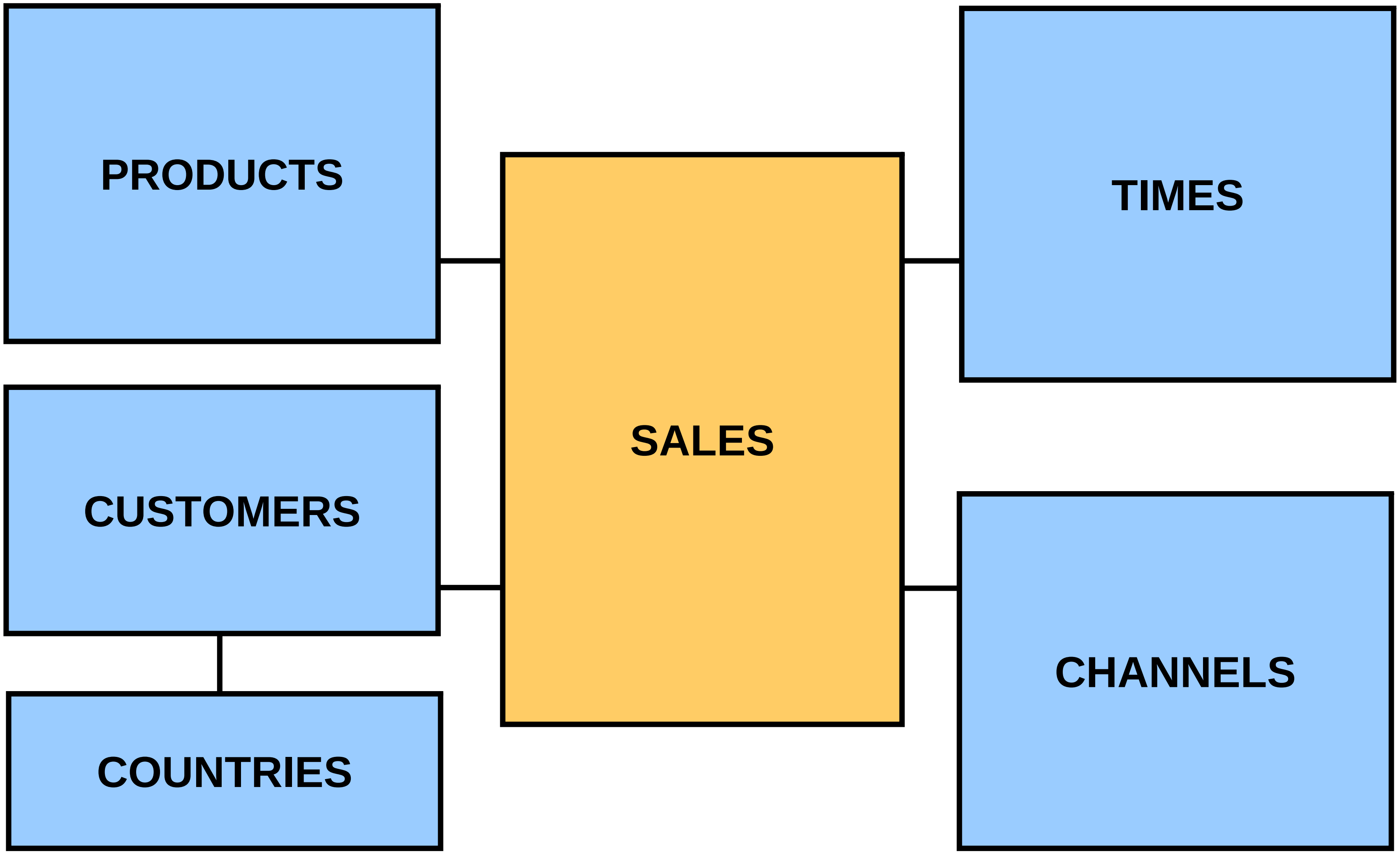
CHANNELS

CHANNEL_ID

CHANNEL_DESC
CHANNEL_CLASS

Fact >> Dimension

The Snowflake Schema Model

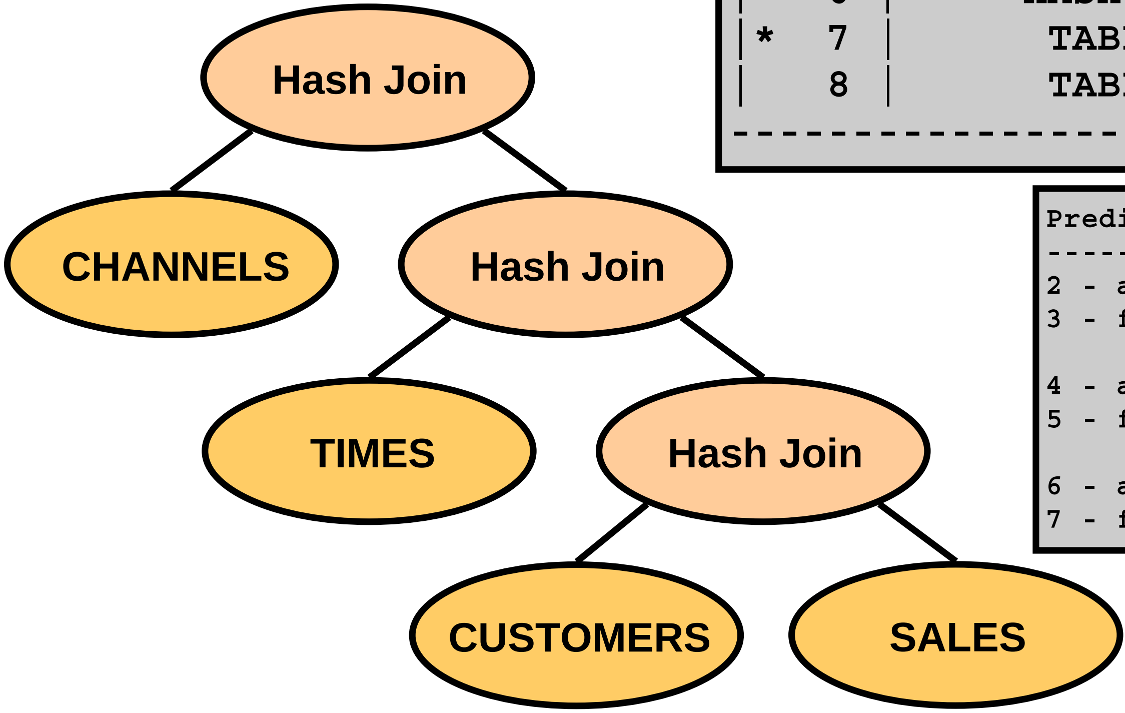


Star Query: Example

```
SELECT ch.channel_class, c.cust_city,
       t.calendar_quarter_desc,
       SUM(s.amount_sold) sales_amount

FROM sales s,times t,customers c,channels ch

WHERE s.time_id = t.time_id AND

      s.cust_id = c.cust_id AND

      s.channel_id = ch.channel_id AND

      c.cust_state_province = 'CA' AND
      ch.channel_desc IN ('Internet','Catalog') AND
      t.calendar_quarter_desc IN ('1999-Q1','1999-Q2')

GROUP BY ch.channel_class, c.cust_city,
         t.calendar_quarter_desc;
```

Execution Plan Without Star Transformation

```
-------------------------------------------
| Id  | Operation             | Name      |
-------------------------------------------
|   0 | SELECT STATEMENT      |           |
|   1 |  HASH GROUP BY        |           |
|*  2 |   HASH JOIN           |           |
|*  3 |    TABLE ACCESS FULL  | CHANNELS  |
|*  4 |    HASH JOIN          |           |
|*  5 |     TABLE ACCESS FULL | TIMES     |
|*  6 |     HASH JOIN         |           |
|*  7 |      TABLE ACCESS FULL| CUSTOMERS |
|   8 |      TABLE ACCESS FULL| SALES     |
-------------------------------------------
```



```
Predicate Information (by operation id):
--------------------------------------------
2 - access("S"."CHANNEL_ID"="CH"."CHANNEL_ID")
3 - filter("CH"."CHANNEL_DESC"='Catalog' OR
           "CH"."CHANNEL_DESC"='Internet')
4 - access("S"."TIME_ID"="T"."TIME_ID")
5 - filter("T"."CALENDAR_QUARTER_DESC"='1999-Q1' OR
           "T"."CALENDAR_QUARTER_DESC"='1999-Q2')
6 - access("S"."CUST_ID"="C"."CUST_ID")
7 - filter("C"."CUST_STATE_PROVINCE"='CA')
```

Star Transformation

- Create bitmap indexes on fact tables foreign keys.
- Set `STAR_TRANSFORMATION_ENABLED` to `TRUE`.
- Requires at least two dimensions and one fact table
- Gather statistics on all corresponding objects.
- Carried out in two phases:
  - First, identify interesting fact rows using bitmap indexes based on dimensional filters.
  - Join them to the dimension tables.

Star Transformation: Considerations

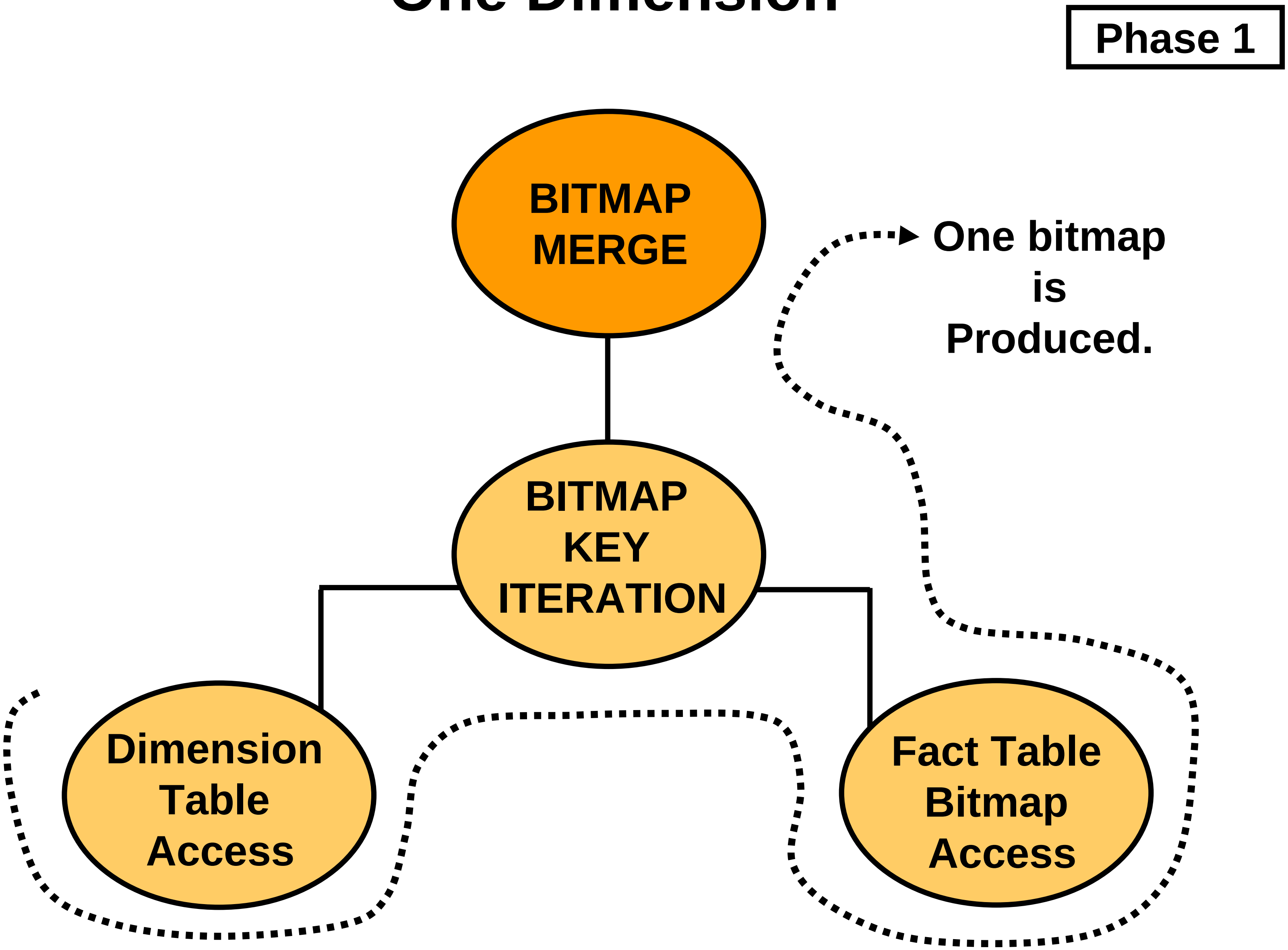
- Queries containing bind variables are not transformed.
- Queries referring to remote fact tables are not transformed.
- Queries containing antijoined tables are not transformed.
- Queries referring to unmerged nonpartitioned views are not transformed.

Star Transformation: Rewrite Example

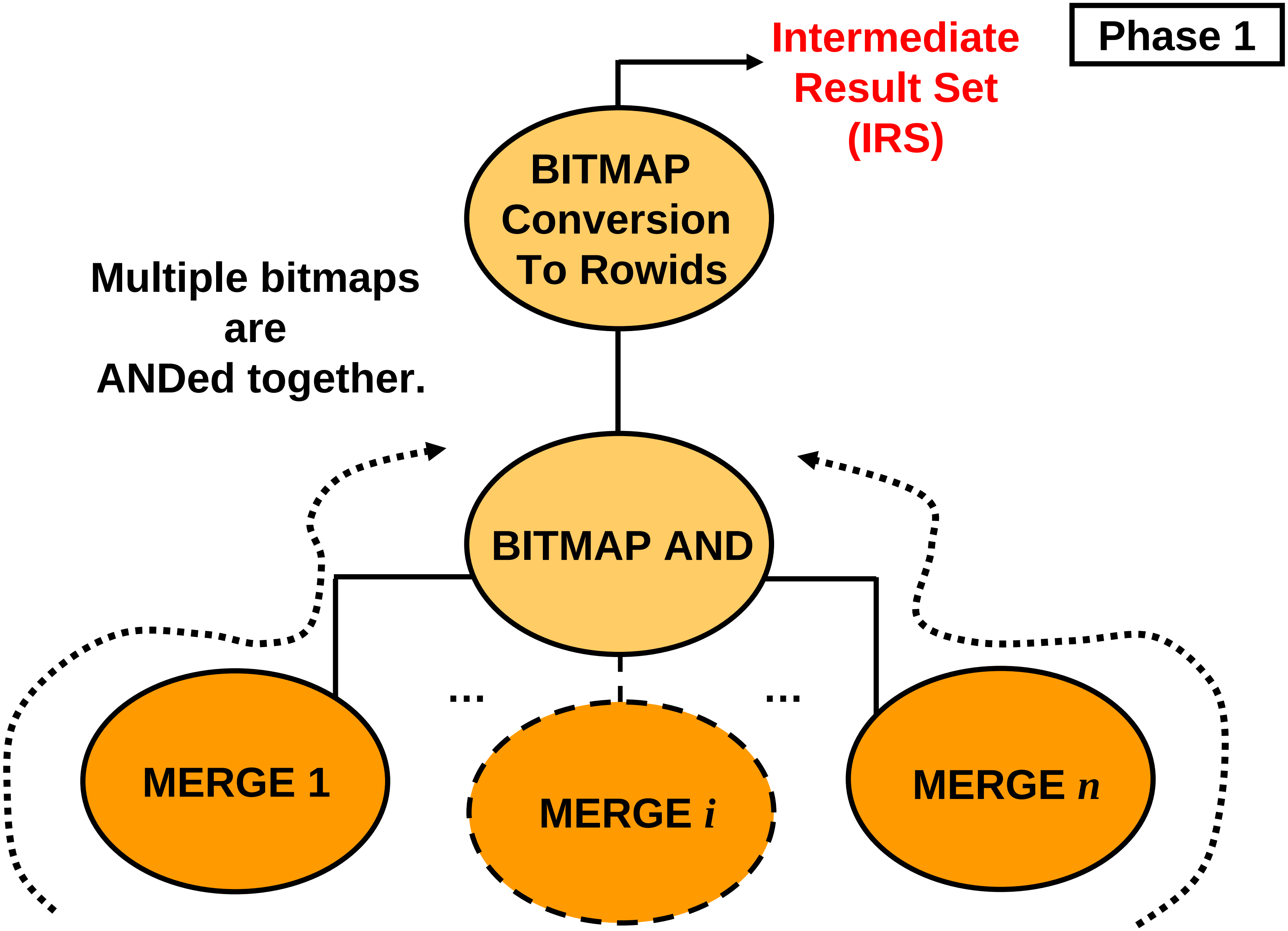
Phase 1

```
SELECT s.amount_sold
FROM sales s

WHERE time_id IN (SELECT time_id
                  FROM times
                  WHERE calendar_quarter_desc
                      IN('1999-Q1','1999-Q2'))

AND   cust_id IN (SELECT cust_id
                  FROM customers
                  WHERE cust_state_province = 'CA')

AND channel_id IN(SELECT channel_id
                  FROM channels
                  WHERE channel_desc IN
                            ('Internet','Catalog'));
```

Retrieving Fact Rows from One Dimension

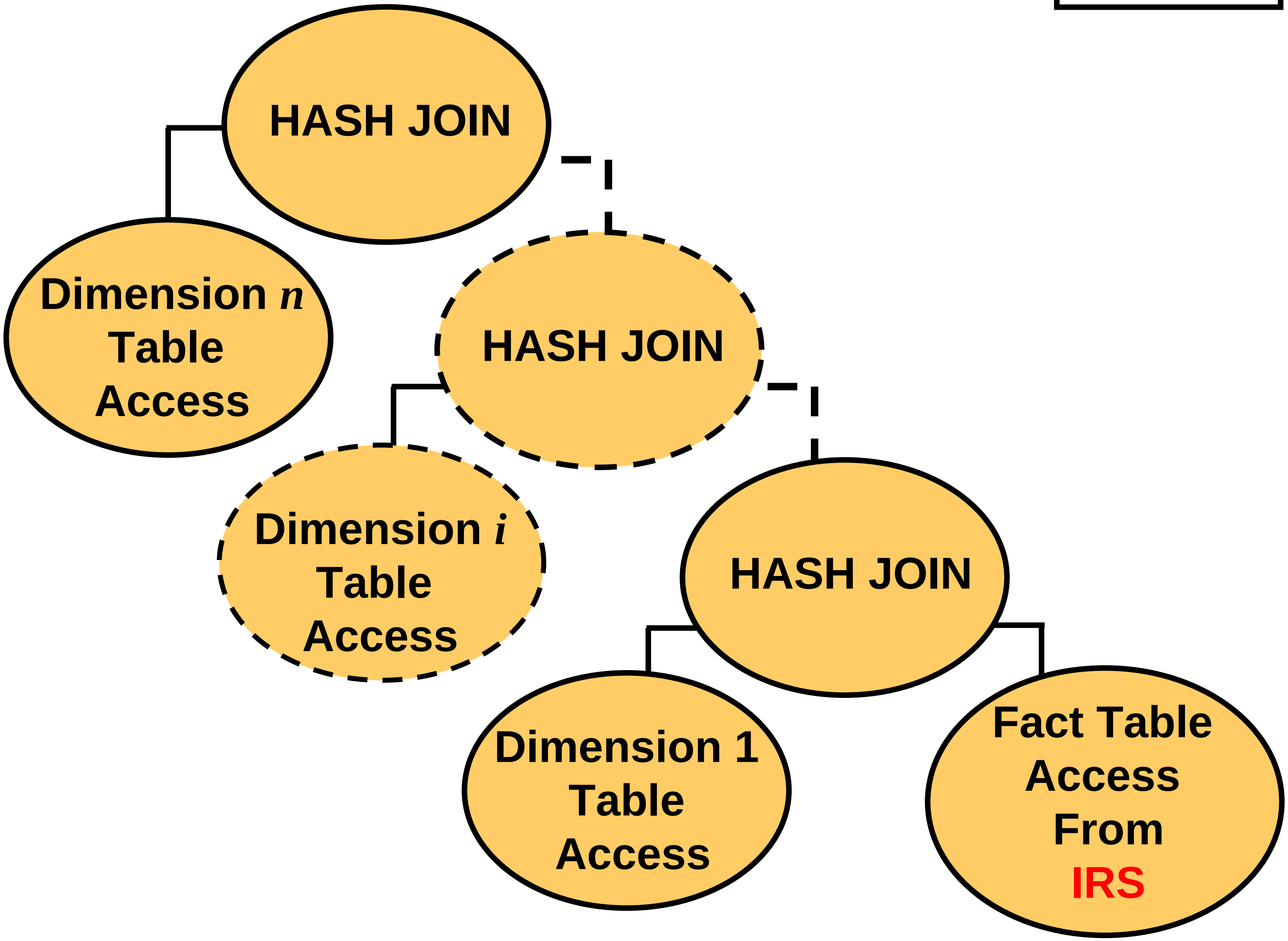


Retrieving Fact Rows from All Dimensions


Phase 1
Intermediate Result Set (IRS)
BITMAP Conversion To Rowids
Multiple bitmaps are ANDed together.
BITMAP AND
...
...
MERGE 1
MERGE *i*
MERGE *n*

Joining the Intermediate Result Set with Dimensions

Phase 2


HASH JOIN
Dimension *n* Table Access
HASH JOIN
Dimension *i* Table Access
HASH JOIN
Dimension 1 Table Access
Fact Table Access From IRS

Star Transformation Plan: Example 1

```
SORT GROUP BY
 HASH JOIN
    HASH JOIN
      TABLE ACCESS BY INDEX ROWID SALES
       BITMAP CONVERSION TO ROWIDS
        BITMAP AND
         BITMAP MERGE
          BITMAP KEY ITERATION
           BUFFER SORT
            TABLE ACCESS FULL CHANNELS
           BITMAP INDEX RANGE SCAN SALES_CHANNELS_BX
         BITMAP MERGE
          BITMAP KEY ITERATION
           BUFFER SORT
            TABLE ACCESS FULL TIMES
           BITMAP INDEX RANGE SCAN SALES_TIMES_BX

         …
      TABLE ACCESS FULL CHANNELS
    TABLE ACCESS FULL TIMES
```

Star Transformation: Further Optimization

- In a star transformation execution plan, dimension tables are accessed twice; once for each phase.
- This might be a performance issue in the case of big dimension tables and low selectivity.
- If the cost is lower, the system might decide to create a temporary table and use it instead of accessing the same dimension table twice.
- Temporary table's creation in the plan:

```
LOAD AS SELECT              SYS_TEMP_0FD9D6720_BEBDC
  TABLE ACCESS FULL         CUSTOMERS
…
  filter("C"."CUST_STATE_PROVINCE"='CA')
```

Using Bitmap Join Indexes

- Volume of data to be joined is reduced
- Can be used to eliminate bitwise operations
- More efficient in storage than MJVs

```
CREATE BITMAP INDEX sales_q_bjx
ON sales(times.calendar_quarter_desc)
FROM sales, times
WHERE sales.time_id = times.time_id
```

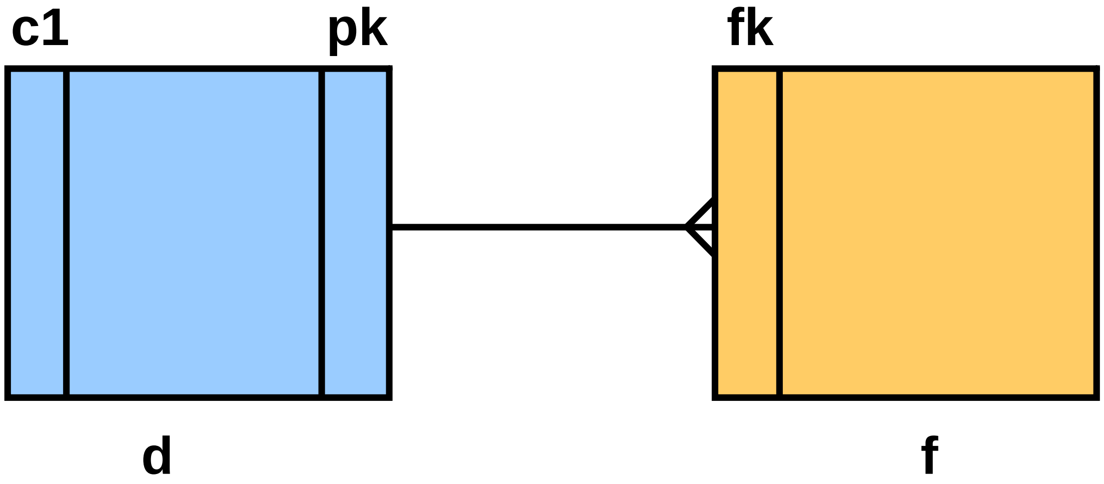
Star Transformation Plan: Example 2

```
SORT GROUP BY
 HASH JOIN
    HASH JOIN
       TABLE ACCESS BY INDEX ROWID SALES
         BITMAP CONVERSION TO ROWIDS
          BITMAP AND
           BITMAP MERGE
            BITMAP KEY ITERATION
             BUFFER SORT
              TABLE ACCESS FULL CHANNELS
             BITMAP INDEX RANGE SCAN SALES_CHANNELS_BX
           BITMAP OR
             BITMAP INDEX SINGLE VALUE SALES_Q_BJX
             BITMAP INDEX SINGLE VALUE SALES_Q_BJX
       TABLE ACCESS FULL CHANNELS
    TABLE ACCESS FULL TIMES
```

Star Transformation Hints

- The `STAR_TRANSFORMATION` hint: Use best plan containing a star transformation, if there is one.
- The `FACT`(<table_name>) hint: The hinted table should be considered as the fact table in the context of a star transformation.
- The `NO_FACT` (<table_name>) hint: The hinted table should not be considered as the fact table in the context of a star transformation.
- The `FACT` and `NO_FACT` hints are useful for star queries containing more than one fact table.

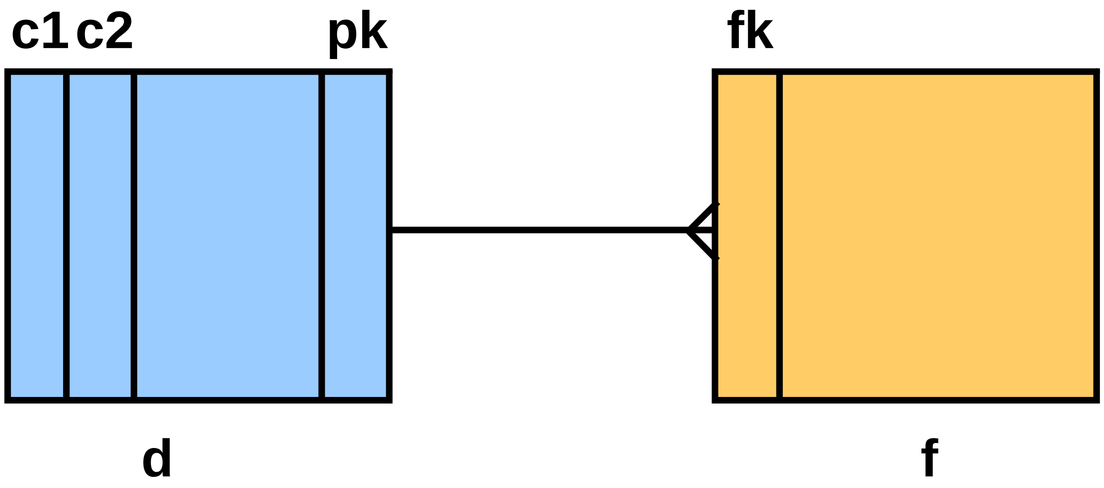
Bitmap Join Indexes: Join Model 1



```
CREATE BITMAP INDEX bji ON f(d.c1)
FROM f, d
WHERE d.pk = f.fk;
```

```
SELECT sum(f.facts)
FROM d, f
WHERE d.pk = f.fk AND d.c1 = 1;
```

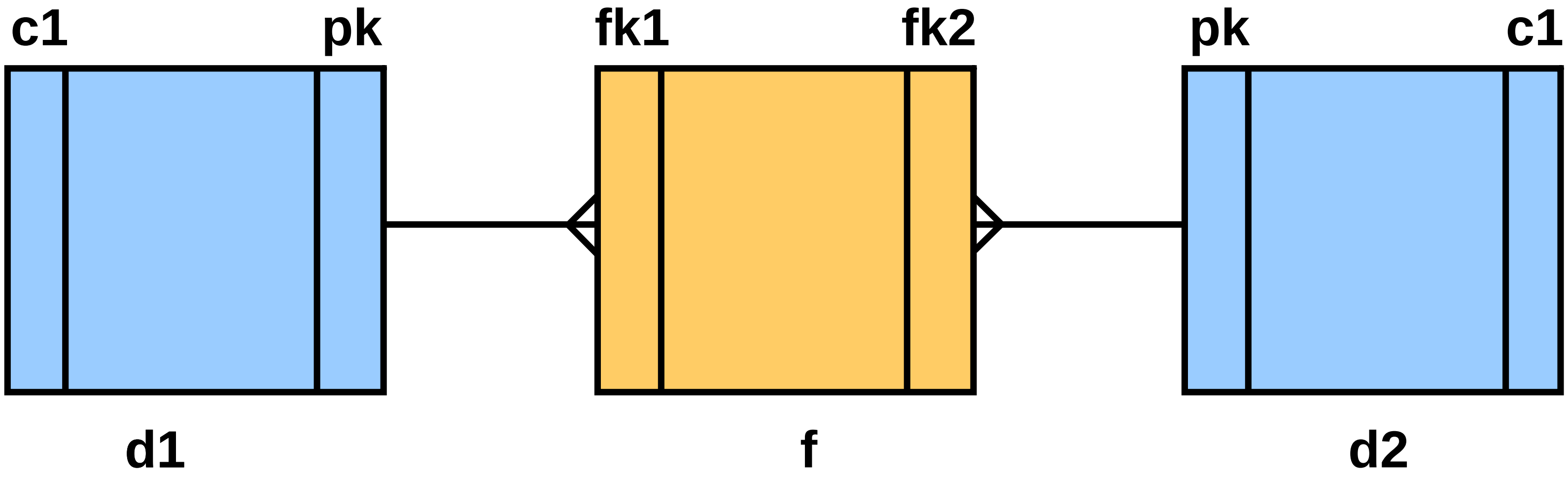
Bitmap Join Indexes: Join Model 2



```
CREATE BITMAP INDEX bjx ON f(d.c1,d.c2)
FROM f, d
WHERE d.pk = f.fk;
```

```
SELECT sum(f.facts)
FROM d, f
WHERE d.pk = f.fk AND d.c1 = 1 AND d.c2 = 1;
```

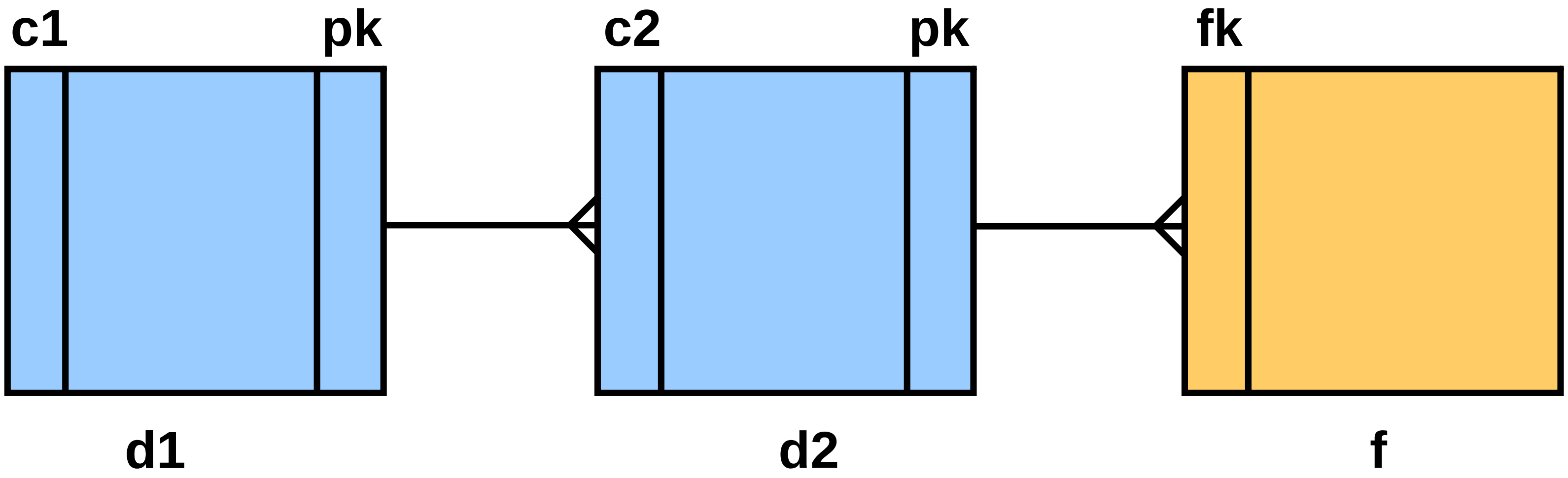
Bitmap Join Indexes: Join Model 3



```
CREATE BITMAP INDEX bjx ON f(d1.c1,d2.c1)
FROM f, d1, d2
WHERE d1.pk = f.fk1 AND d2.pk = f.fk2;
```

```
SELECT sum(f.sales)
FROM d1, f, d2
WHERE d1.pk = f.fk1 AND d2.pk = f.fk2 AND
          d1.c1 = 1 AND d2.c1 = 2;
```

Bitmap Join Indexes: Join Model 4



```
CREATE BITMAP INDEX bjx ON f(d1.c1)
FROM f, d1, d2
WHERE d1.pk = d2.c2 AND d2.pk = f.fk;
```

```
SELECT sum(f.sales)
FROM d1, d2, f
WHERE d1.pk = d2.c2 AND d2.pk = f.fk AND
      d1.c1 = 1;
```

Summary

In this lesson, you should have learned how to:

- Define a star schema
- Show a star query plan without transformation
- Define the star transformation requirements
- Show a star query plan after transformation

Practice 6: Overview

This practice covers using the star transformation technique to optimize your query.

7

Optimizer Statistics

Objectives

After completing this lesson, you should be able to do the following:

- Gather optimizer statistics
- Gather system statistics
- Set statistic preferences
- Use dynamic sampling
- Manipulate optimizer statistics

Optimizer Statistics

- Describe the database and the objects in the database
- Information used by the query optimizer to estimate:
  - Selectivity of predicates
  - Cost of each execution plan
  - Access method, join order, and join method
  - CPU and input/output (I/O) costs
- Refreshing optimizer statistics whenever they are stale is as important as gathering them:
  - Automatically gathered by the system
  - Manually gathered by the user with `DBMS_STATS`

Types of Optimizer Statistics

- Table statistics:
  - Number of rows
  - Number of blocks
  - Average row length
- Index Statistics:
  - B*-tree level
  - Distinct keys
  - Number of leaf blocks
  - Clustering factor
- System statistics
  - I/O performance and utilization
  - CPU performance and utilization
- Column statistics
  - Basic: Number of distinct values, number of nulls, average length, min, max
  - Histograms (data distribution when the column data is skewed)
  - Extended statistics

Table Statistics (`DBA_TAB_STATISTICS`)

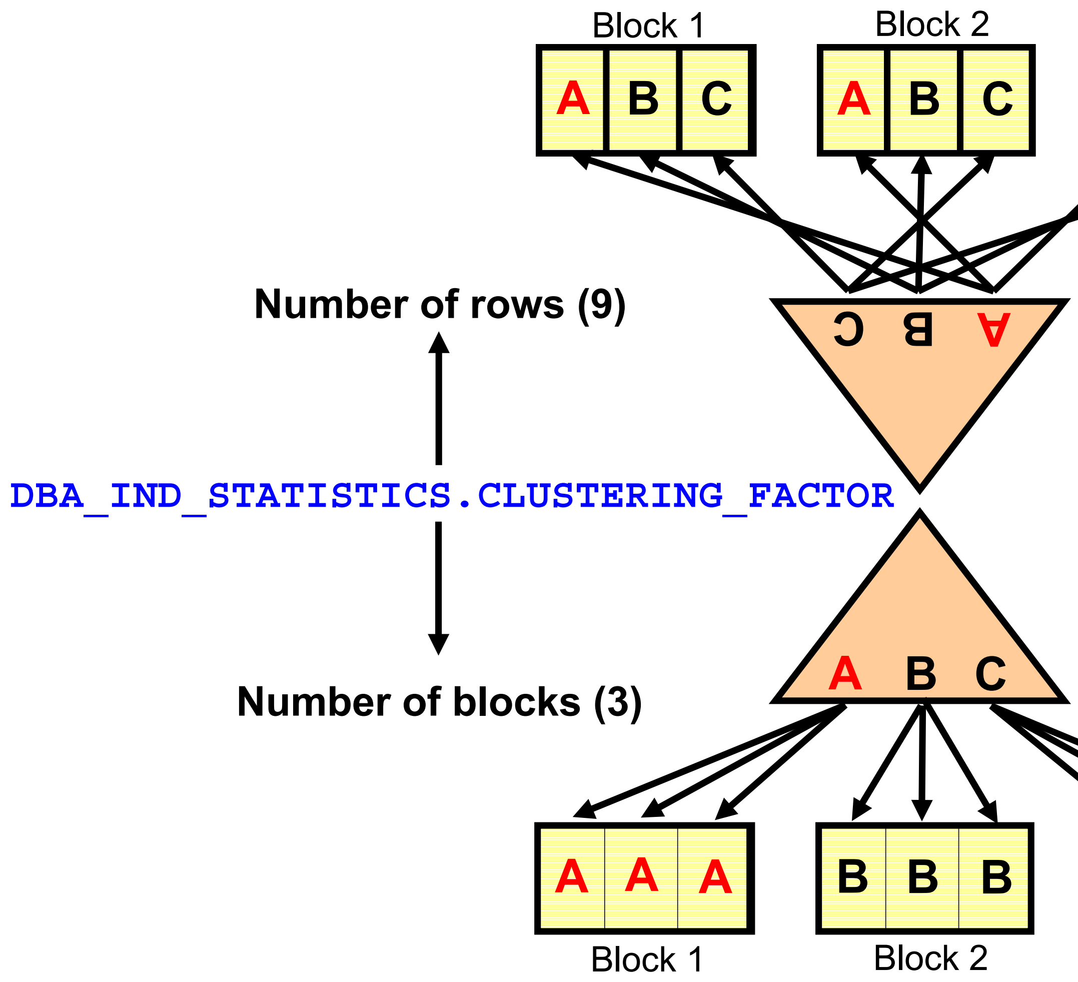
- Used to determine:
  - Table access cost
  - Join cardinality
  - Join order
- Some of the statistics gathered are:
  - Row count (`NUM_ROWS`)
  - Block count (`BLOCKS`) *Exact*
  - Empty blocks (`EMPTY_BLOCKS`) *Exact*
  - Average free space per block (`AVG_SPACE`)
  - Number of chained rows (`CHAIN_CNT`)
  - Average row length (`AVG_ROW_LEN`)
  - Statistics status (`STALE_STATS`)

Index Statistics (DBA_IND_STATISTICS)

- Used to decide:
  - Full table scan versus index scan
- Statistics gathered are:
  - B*-tree level (BLEVEL) *Exact*
  - Leaf block count (LEAF_BLOCKS)
  - Clustering factor (CLUSTERING_FACTOR)
  - Distinct keys (DISTINCT_KEYS)
  - Average number of leaf blocks in which each distinct value in the index appears (AVG_LEAF_BLOCKS_PER_KEY)
  - Average number of data blocks in the table pointed to by a distinct value in the index (AVG_DATA_BLOCKS_PER_KEY)
  - Number of rows in the index (NUM_ROWS)

Index Clustering Factor

Must read all blocks to retrieve all As



Block 1 Block 2 Block 3

A B C A B C A B C

Number of rows (9)

DBA_IND_STATISTICS.CLUSTERING_FACTOR

Number of blocks (3)

A B C

A A A B B B C C C

Block 1 Block 2 Block 3

Big clustering factor:
Favor alternative paths

Small clustering factor:
Favor the
index range scan path

Only need to read one block to retrieve all As

Column Statistics (`DBA_TAB_COL_STATISTICS`)

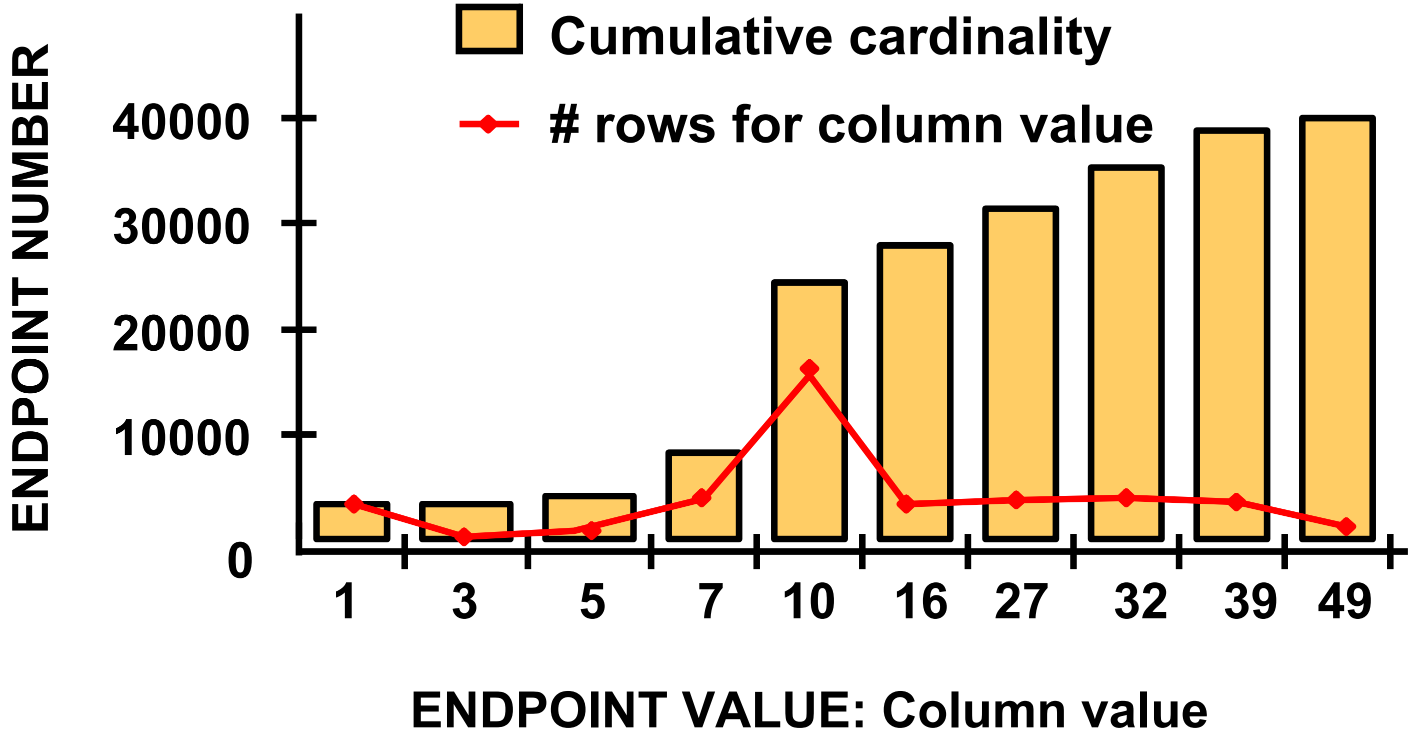
- Count of distinct values of the column (`NUM_DISTINCT`)
- Low value (`LOW_VALUE`) *Exact*
- High value (`HIGH_VALUE`) *Exact*
- Number of nulls (`NUM_NULLS`)
- Selectivity estimate for nonpopular values (`DENSITY`)
- Number of histogram buckets (`NUM_BUCKETS`)
- Type of histogram (`HISTOGRAM`)

Histograms

- The optimizer assumes uniform distributions; this may lead to suboptimal access plans in the case of data skew.
- Histograms:
  - Store additional column distribution information
  - Give better selectivity estimates in the case of nonuniform distributions
- With unlimited resources you could store each different value and the number of rows for that value.
- This becomes unmanageable for a large number of distinct values and a different approach is used:
  - Frequency histogram (#distinct values ≤ #buckets)
  - Height-balanced histogram (#buckets < #distinct values)
- They are stored in `DBA_TAB_HISTOGRAMS`.

Frequency Histograms

10 buckets, 10 distinct values



Distinct values: 1, 3, 5, 7, 10, 16, 27, 32, 39, 49

Number of rows: 40001

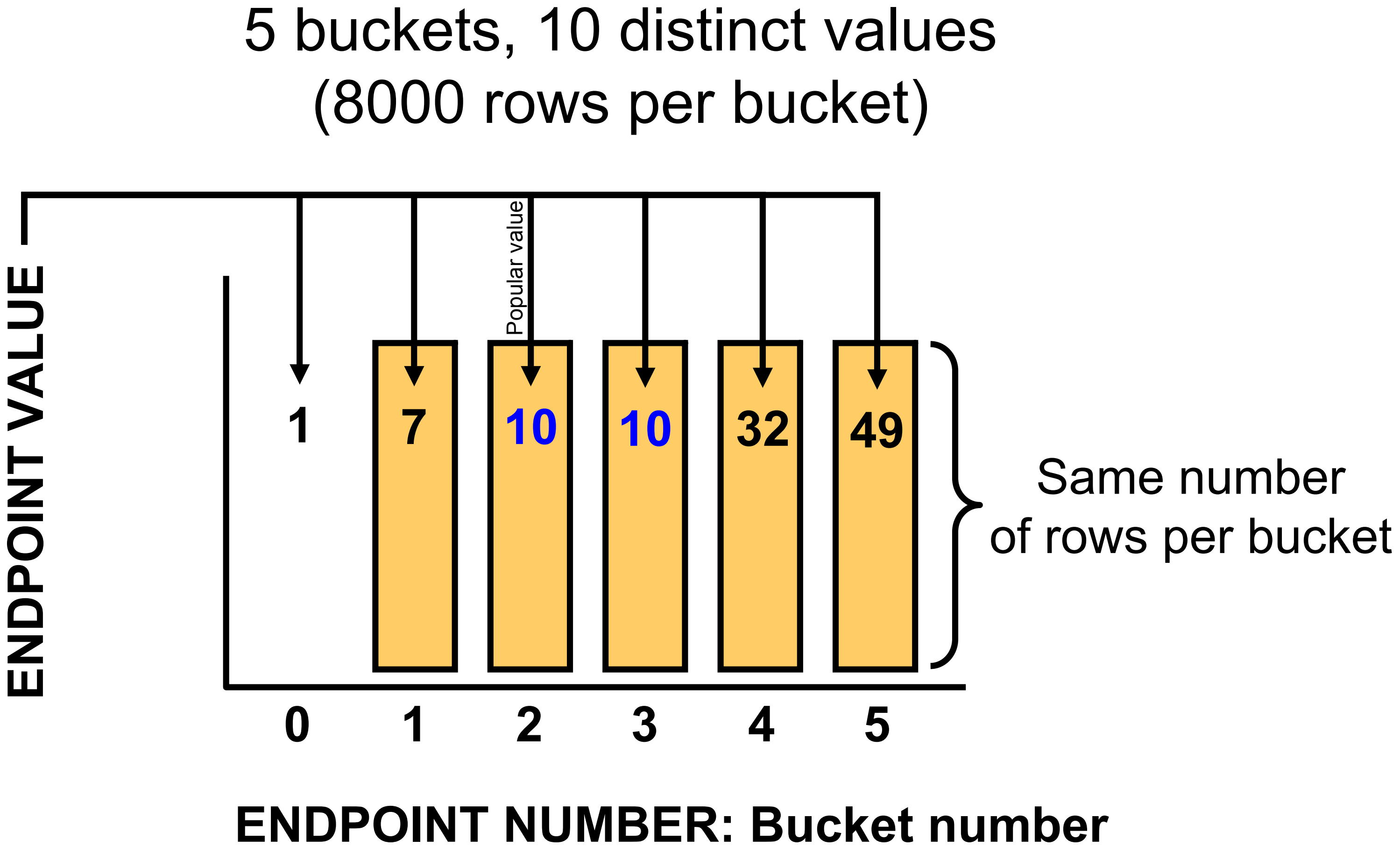
Viewing Frequency Histograms

```
BEGIN
 DBMS_STATS.gather_table_STATS (OWNNAME=>'OE', TABNAME=>'INVENTORIES',
       METHOD_OPT => 'FOR COLUMNS SIZE 20 warehouse_id');
END;
```

```
SELECT column_name, num_distinct, num_buckets, histogram
FROM   USER_TAB_COL_STATISTICS
WHERE  table_name = 'INVENTORIES' AND
       column_name = 'WAREHOUSE_ID';

COLUMN_NAME  NUM_DISTINCT NUM_BUCKETS HISTOGRAM
------------ ------------ ----------- ---------
WAREHOUSE_ID            9           9 FREQUENCY
```

```
SELECT endpoint_number, endpoint_value
FROM   USER_HISTOGRAMS
WHERE  table_name = 'INVENTORIES' and column_name = 'WAREHOUSE_ID'
ORDER BY endpoint_number;

ENDPOINT_NUMBER ENDPOINT_VALUE
--------------- --------------
36                           1
213                          2
261                          3
…
```

Height-Balanced Histograms



Distinct values: 1, 3, 5, 7, 10, 16, 27, 32, 39, 49

Number of rows: 40001

Viewing Height-Balanced Histograms

```
BEGIN
  DBMS_STATS.gather_table_STATS(OWNNAME =>'OE', TABNAME=>'INVENTORIES',
  METHOD_OPT => 'FOR COLUMNS SIZE 10 quantity_on_hand');
END;
```

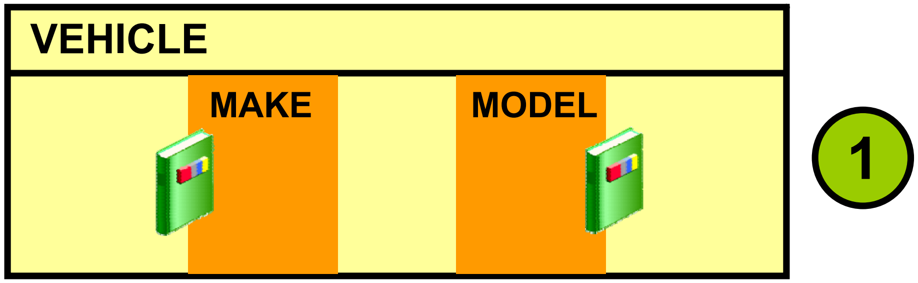
```
SELECT column_name, num_distinct, num_buckets, histogram
  FROM USER_TAB_COL_STATISTICS
 WHERE table_name = 'INVENTORIES' AND column_name = 'QUANTITY_ON_HAND';

COLUMN_NAME                    NUM_DISTINCT NUM_BUCKETS HISTOGRAM
------------------------------ ------------ ----------- ---------------
QUANTITY_ON_HAND                        237          10 HEIGHT BALANCED
```

```
SELECT endpoint_number, endpoint_value
FROM USER_HISTOGRAMS
WHERE table_name = 'INVENTORIES' and column_name = 'QUANTITY_ON_HAND'
ORDER BY endpoint_number;

ENDPOINT_NUMBER ENDPOINT_VALUE
--------------- --------------
              0              0
              1             27
              2             42
              3             57
…
```

Histogram Considerations

- Histograms are useful when you have a high degree of skew in the column distribution.
- Histograms are *not* useful for:
  - Columns which do not appear in the `WHERE` or `JOIN` clauses
  - Columns with uniform distributions
  - Equality predicates with unique columns
- The maximum number of buckets is the least (254,# distinct values).
- Do not use histograms unless they substantially improve performance.

Multicolumn Statistics: Overview



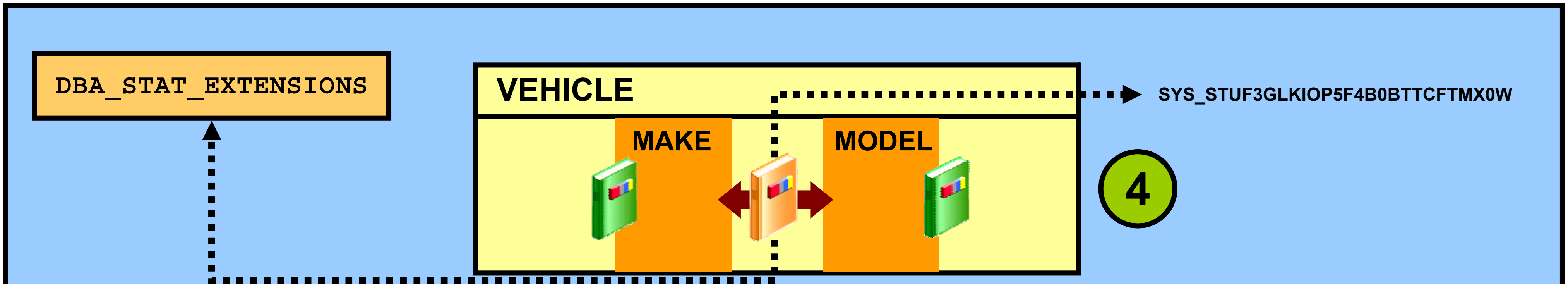
S(MAKE ∧ MODEL)=S(MAKE)xS(MODEL)

```
select
dbms_stats.create_extended_stats('jfv','vehicle','(make,model)')
from dual;
```

2

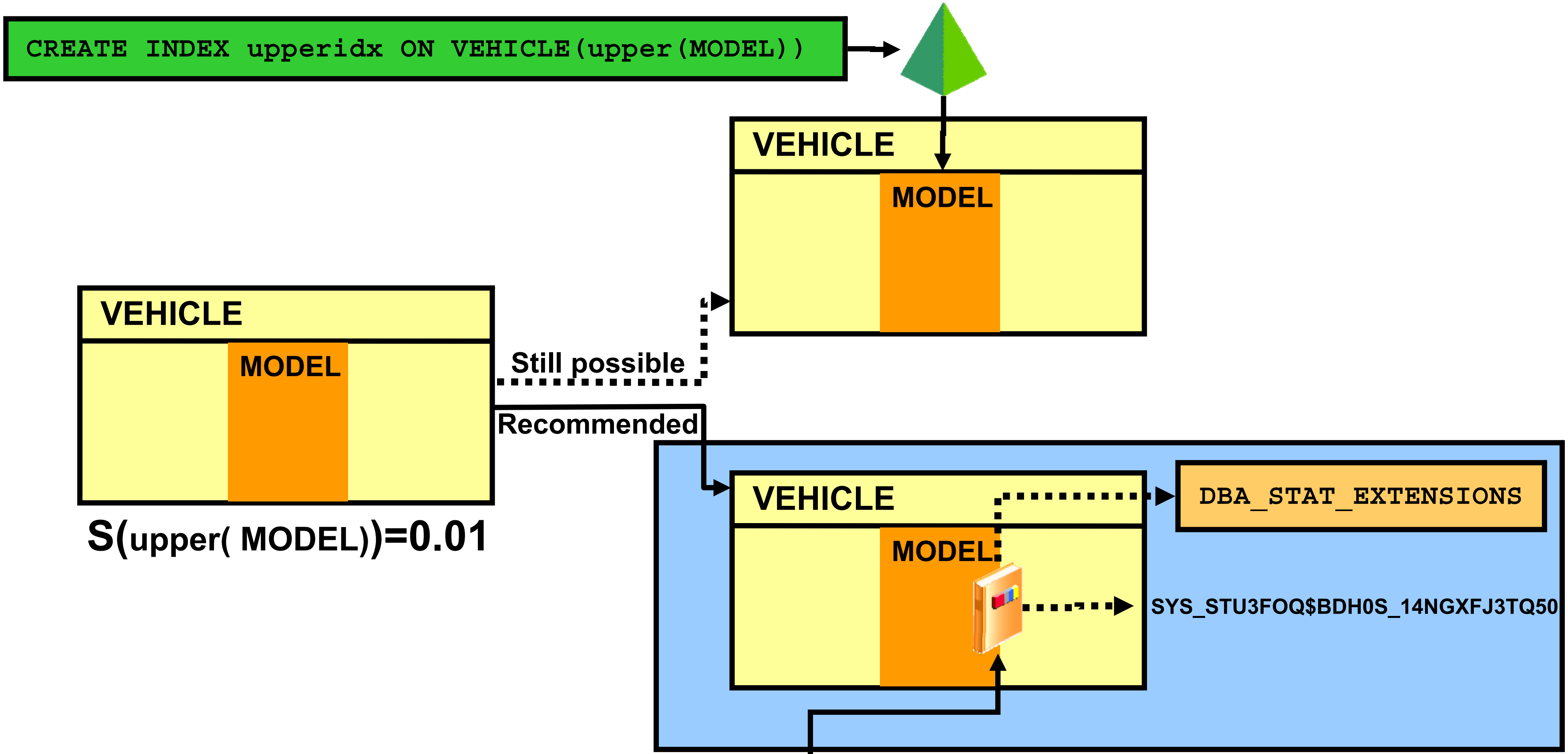
```
exec dbms_stats.gather_table_stats('jfv','vehicle',-
method_opt=>'for all columns size 1 for columns (make,model) size 3');
```

3



S(MAKE ∧ MODEL)=S(MAKE,MODEL)

Expression Statistics: Overview



```
CREATE INDEX upperidx ON VEHICLE(upper(MODEL))
```

```
select dbms_stats.create_extended_stats('jfv','vehicle','(upper(model))') from dual;
```

```
exec dbms_stats.gather_table_stats('jfv','vehicle',-
method_opt=>'for all columns size 1 for columns (upper(model)) size 3');
```

Gathering System Statistics

- System statistics enable the CBO to use CPU and I/O characteristics.
- System statistics must be gathered on a regular basis; this does not invalidate cached plans.
- Gathering system statistics equals analyzing system activity for a specified period of time:
- Procedures:
  - `DBMS_STATS.GATHER_SYSTEM_STATS`
  - `DBMS_STATS.SET_SYSTEM_STATS`
  - `DBMS_STATS.GET_SYSTEM_STATS`
- `GATHERING_MODE`:
  - `NOWORKLOAD|INTERVAL`
  - `START|STOP`

Gathering System Statistics: Example

First day

```
EXECUTE DBMS_STATS.GATHER_SYSTEM_STATS(
interval => 120,
stattab => 'mystats', statid => 'OLTP');
```

First night

```
EXECUTE DBMS_STATS.GATHER_SYSTEM_STATS(
interval => 120,
stattab => 'mystats', statid => 'OLAP');
```

Next days

```
EXECUTE DBMS_STATS.IMPORT_SYSTEM_STATS(
stattab => 'mystats', statid => 'OLTP');
```

Next nights

```
EXECUTE DBMS_STATS.IMPORT_SYSTEM_STATS(
stattab => 'mystats', statid => 'OLAP');
```

Gathering System Statistics: Example

- Start manual system statistics collection in the data dictionary:

```
SQL> EXECUTE DBMS_STATS.GATHER_SYSTEM_STATS( -
  2  gathering_mode => 'START');
```

- Generate the workload.
- End the collection of system statistics:

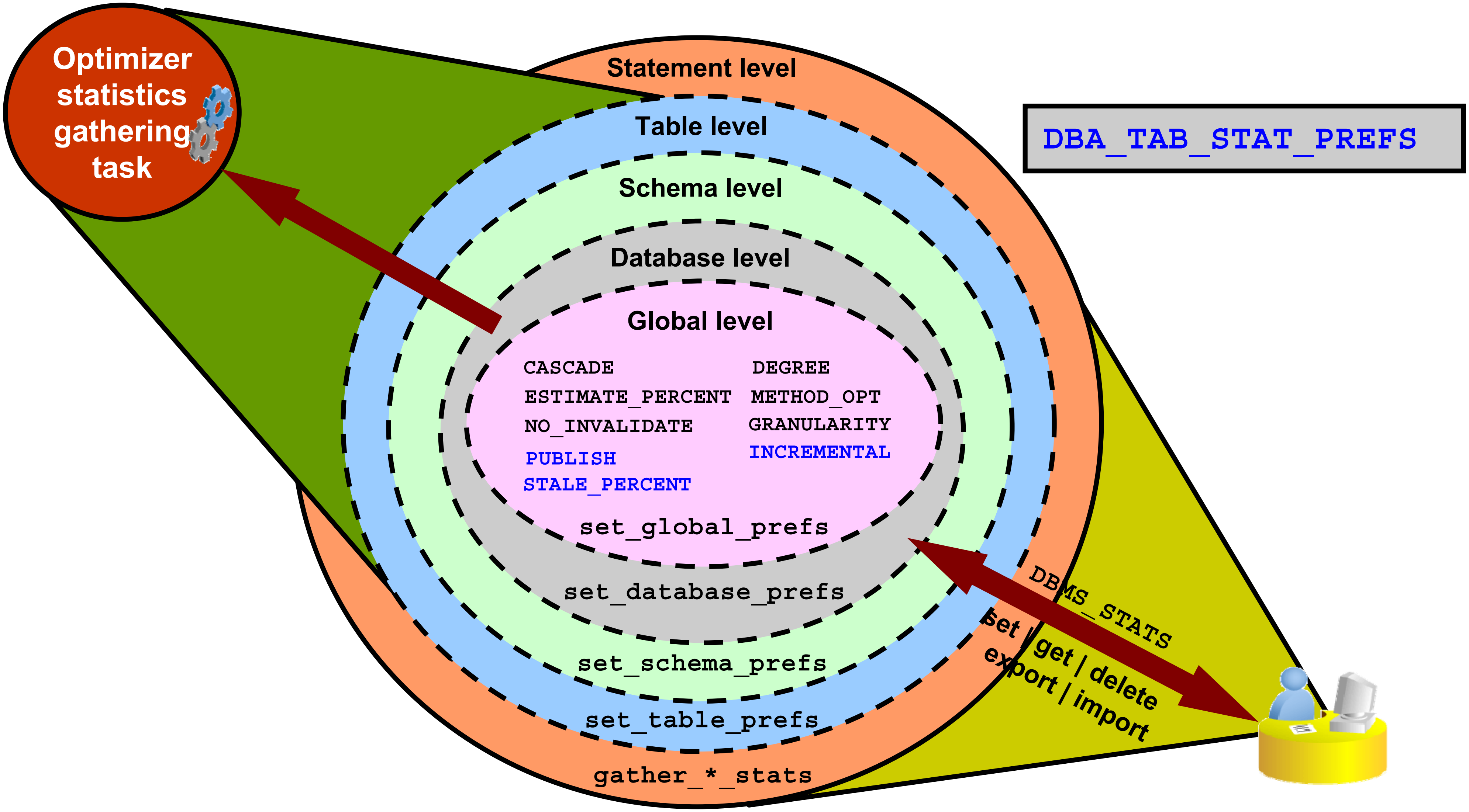
```
SQL> EXECUTE DBMS_STATS.GATHER_SYSTEM_STATS( -
  2  gathering_mode => 'STOP');
```

Mechanisms for Gathering Statistics

- Automatic statistics gathering
  - `gather_stats_prog` automated task
- Manual statistics gathering
  - `DBMS_STATS` package
- Dynamic sampling
- When statistics are missing:

Selectivity:	
Equality	**1%**
Inequality	**5%**
Other predicates	**5%**
Table row length	**20**
# of index leaf blocks	**25**
# of distinct values	**100**
Table cardinality	**100**
Remote table cardinality	**2000**

Statistic Preferences: Overview



```
exec dbms_stats.set_table_prefs('SH','SALES','STALE_PERCENT','13');
```

When to Gather Statistics Manually

- Rely mostly on automatic statistics collection:
  - Change the frequency of automatic statistics collection to meet your needs.
  - Remember that `STATISTICS_LEVEL` should be set to `TYPICAL` or `ALL` for automatic statistics collection to work properly.
- Gather statistics manually for:
  - Objects that are volatile
  - Objects modified in batch operations
  - External tables, system statistics, fixed objects
  - Objects modified in batch operations: Gather statistics as part of the batch operation.
  - New objects: Gather statistics right after object creation.

Manual Statistics Gathering

You can use Enterprise Manager and the `DBMS_STATS` package to:

- Generate and manage statistics for use by the optimizer:
  - Gather/Modify
  - View/Name
  - Export/Import
  - Delete/Lock
- Gather statistics on:
  - Indexes, tables, columns, partitions
  - Object, schema, or database
- Gather statistics either serially or in parallel
- Gather/Set system statistics (currently not possible in EM)

Manual Statistics Collection: Factors

- Monitor objects for DMLs.
- Determine the correct sample sizes.
- Determine the degree of parallelism.
- Determine if histograms should be used.
- Determine the cascading effects on indexes.
- Procedures to use in `DBMS_STATS`:
  - `GATHER_INDEX_STATS`
  - `GATHER_TABLE_STATS`
  - `GATHER_SCHEMA_STATS`
  - `GATHER_DICTIONARY_STATS`
  - `GATHER_DATABASE_STATS`
  - `GATHER_SYSTEM_STATS`

Managing Statistics Collection: Example

```
dbms_stats.gather_table_stats
('sh'                  -- schema
,'customers'           -- table
, null                 -- partition
, 20                   -- sample size(%)
, false                -- block sample?
,'for all columns'     -- column spec
, 4                    -- degree of parallelism
,'default'             -- granularity
, true );              -- cascade to indexes
```

```
dbms_stats.set_param('CASCADE',
                  'DBMS_STATS.AUTO_CASCADE');
dbms_stats.set_param('ESTIMATE_PERCENT','5');
dbms_stats.set_param('DEGREE','NULL');
```

Optimizer Dynamic Sampling: Overview

- Dynamic sampling can be done for tables and indexes:
  - Without statistics
  - Whose statistics cannot be trusted
- Used to determine more accurate statistics when estimating:
  - Table cardinality
  - Predicate selectivity
- Feature controlled by:
  - The `OPTIMIZER_DYNAMIC_SAMPLING` parameter
  - The `OPTIMIZER_FEATURES_ENABLE` parameter
  - The `DYNAMIC_SAMPLING` hint
  - The `DYNAMIC_SAMPLING_EST_CDN` hint

Optimizer Dynamic Sampling at Work

- Sampling is done at compile time.
- If a query benefits from dynamic sampling:
  - A recursive SQL statement is executed to sample data
  - The number of blocks sampled depends on the `OPTIMIZER_DYNAMIC_SAMPLING` initialization parameter
- During dynamic sampling, predicates are applied to the sample to determine selectivity.
- Use dynamic sampling when:
  - Sampling time is a small fraction of the execution time
  - Query is executed many times
  - You believe a better plan can be found

OPTIMIZER_DYNAMIC_SAMPLING

- Dynamic session or system parameter
- Can be set to a value from “0” to “10”
- “0” turns off dynamic sampling
- “1” samples all unanalyzed tables, if an unanalyzed table:
  - Is joined to another table or appears in a subquery or nonmergeable view
  - Has no indexes
  - Has more than 32 blocks
- “2” samples all unanalyzed tables
- The higher the value the more aggressive application of sampling
- Dynamic sampling is repeatable if no update activity

Locking Statistics

- Prevents automatic gathering
- Is mainly used for volatile tables:
  - Lock without statistics implies dynamic sampling.

```
BEGIN
  DBMS_STATS.DELETE_TABLE_STATS('OE','ORDERS');
  DBMS_STATS.LOCK_TABLE_STATS('OE','ORDERS');
END;
```

  - Lock with statistics for representative values.

```
BEGIN
  DBMS_STATS.GATHER_TABLE_STATS('OE','ORDERS');
  DBMS_STATS.LOCK_TABLE_STATS('OE','ORDERS');
END;
```

- The `FORCE` argument overrides statistics locking.

```
SELECT stattype_locked FROM dba_tab_statistics;
```

Summary

In this lesson, you should have learned how to:

- Collect optimizer statistics
- Collect system statistics
- Set statistic preferences
- Use dynamic sampling
- Manipulate optimizer statistics

Practice 7: Overview

This practice covers the following topics:

- Using system statistics
- Using automatic statistics gathering

8

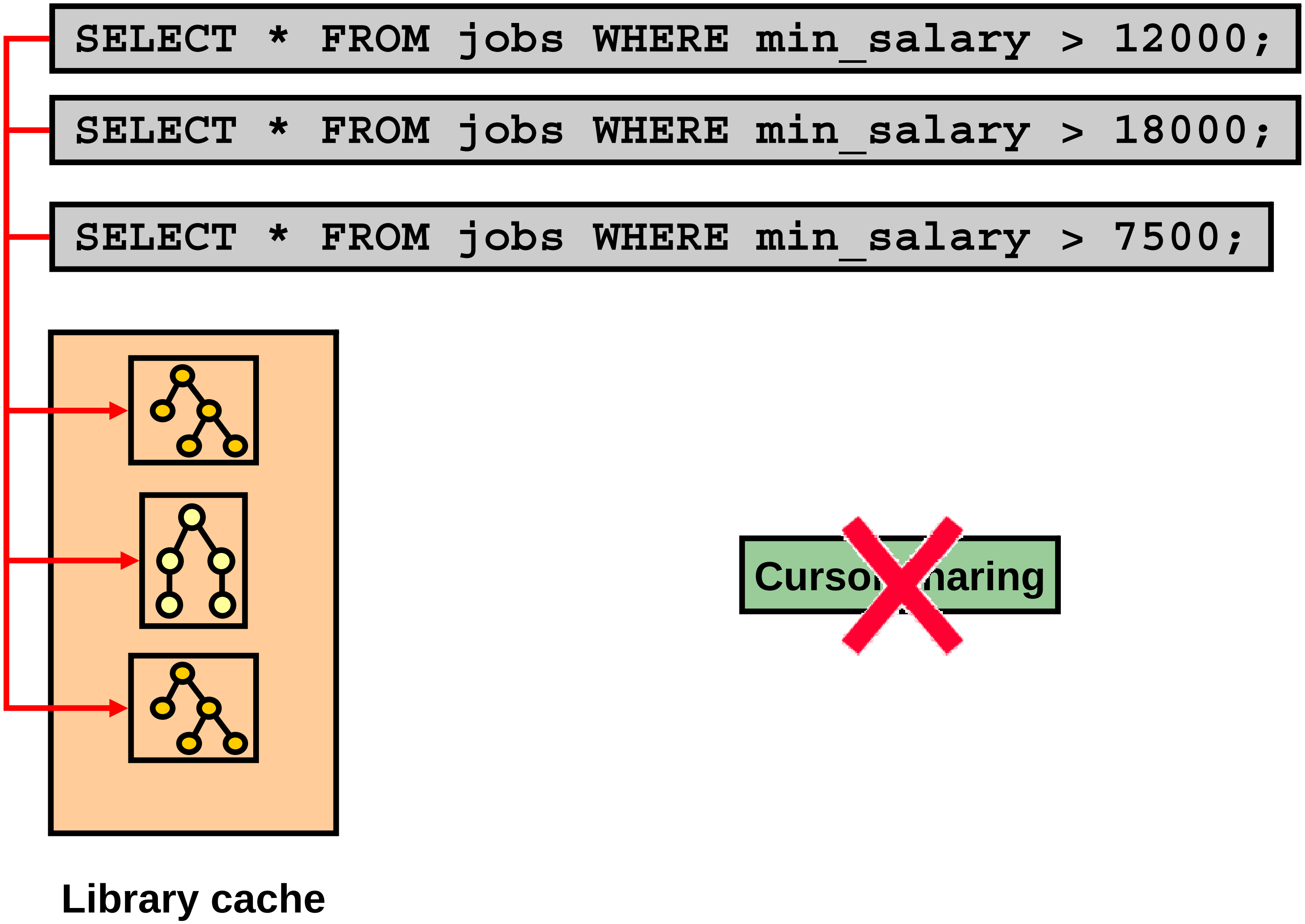
Using Bind Variables

Objectives

After completing this lesson, you should be able to:

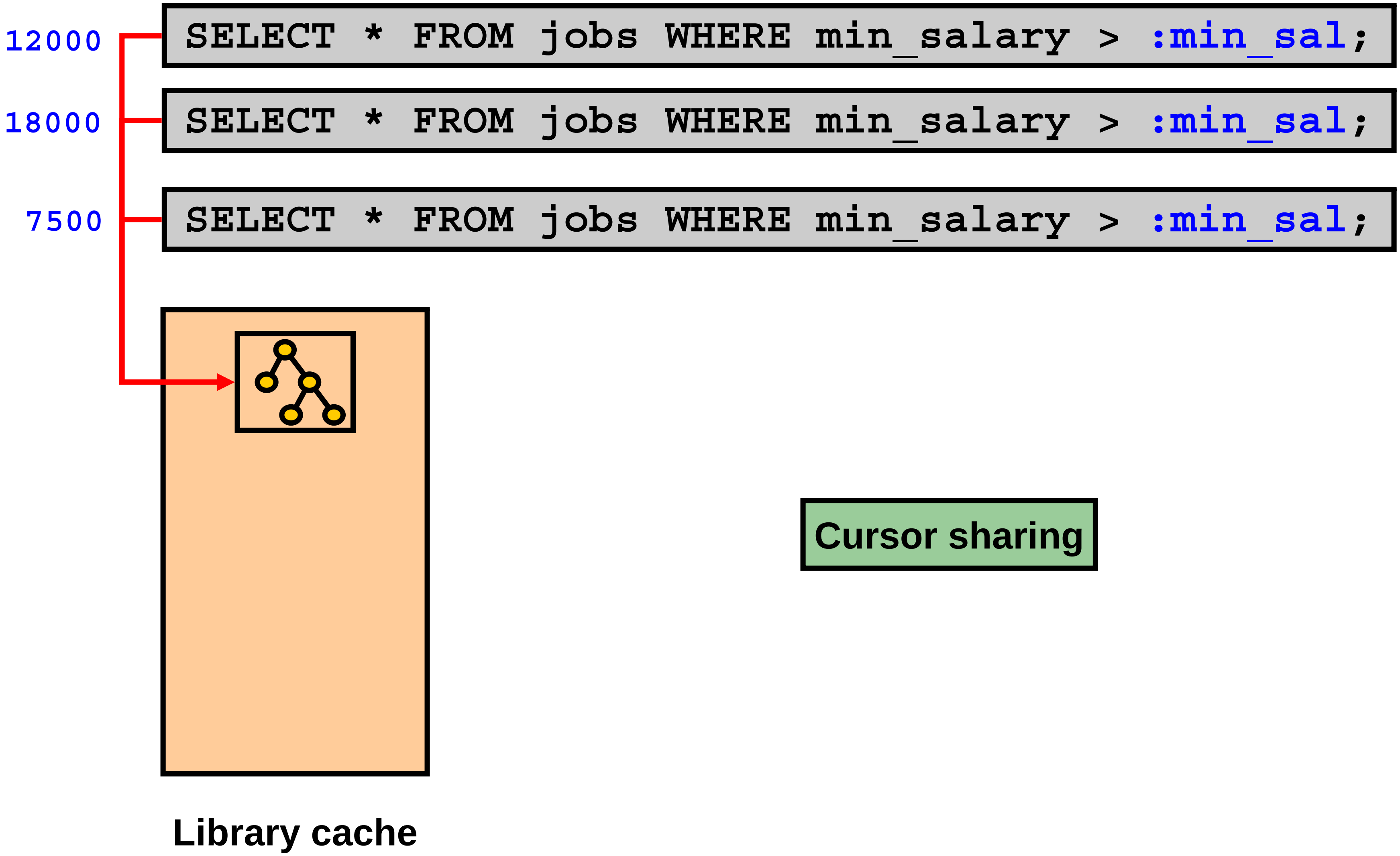
- List the benefits of using bind variables
- Use bind peeking
- Use adaptive cursor sharing

Cursor Sharing and Different Literal Values



```
SELECT * FROM jobs WHERE min_salary > 12000;
SELECT * FROM jobs WHERE min_salary > 18000;
SELECT * FROM jobs WHERE min_salary > 7500;
```

Cursor Sharing and Bind Variables



12000
18000
7500

```
SELECT * FROM jobs WHERE min_salary > :min_sal;
SELECT * FROM jobs WHERE min_salary > :min_sal;
SELECT * FROM jobs WHERE min_salary > :min_sal;
```

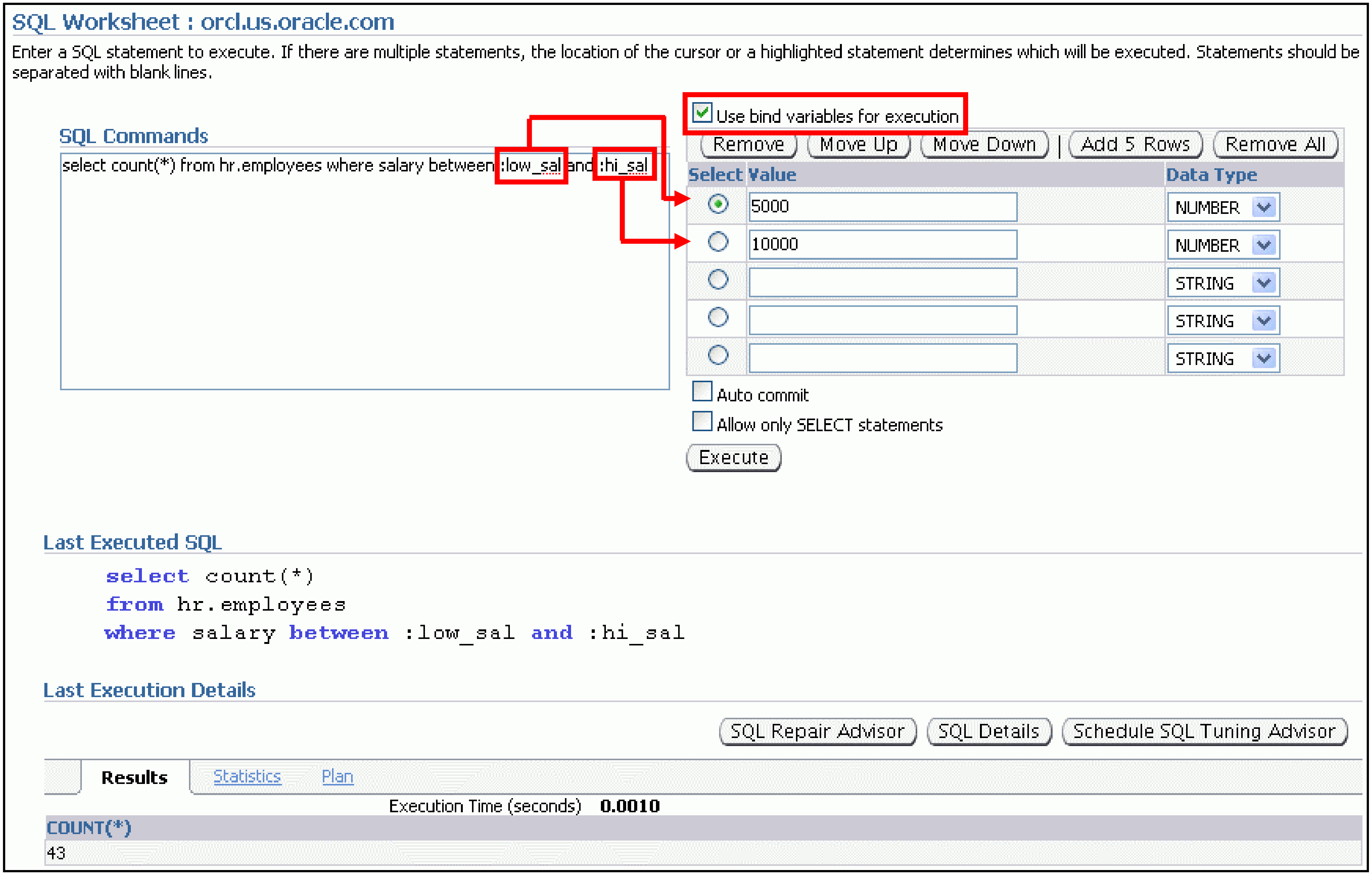
Cursor sharing

Library cache

Bind Variables in SQL*Plus

```
SQL> variable job_id varchar2(10)
SQL> exec :job_id := 'SA_REP';

PL/SQL procedure successfully completed.

SQL> select count(*) from employees where job_id = :job_id;

  COUNT(*)
----------
        30

SQL> exec :job_id := 'AD_VP';

PL/SQL procedure successfully completed.

SQL> select count(*) from employees where job_id = :job_id;

  COUNT(*)
----------
         2
```

Bind Variables in Enterprise Manager



SQL Worksheet : orcl.us.oracle.com

Enter a SQL statement to execute. If there are multiple statements, the location of the cursor or a highlighted statement determines which will be executed. Statements should be separated with blank lines.

SQL Commands

```
select count(*) from hr.employees where salary between :low_sal and :hi_sal
```

☑ Use bind variables for execution

Select	Value	Data Type
●	5000	NUMBER
○	10000	NUMBER
○		STRING
○		STRING
○		STRING

□ Auto commit

□ Allow only SELECT statements

Last Executed SQL

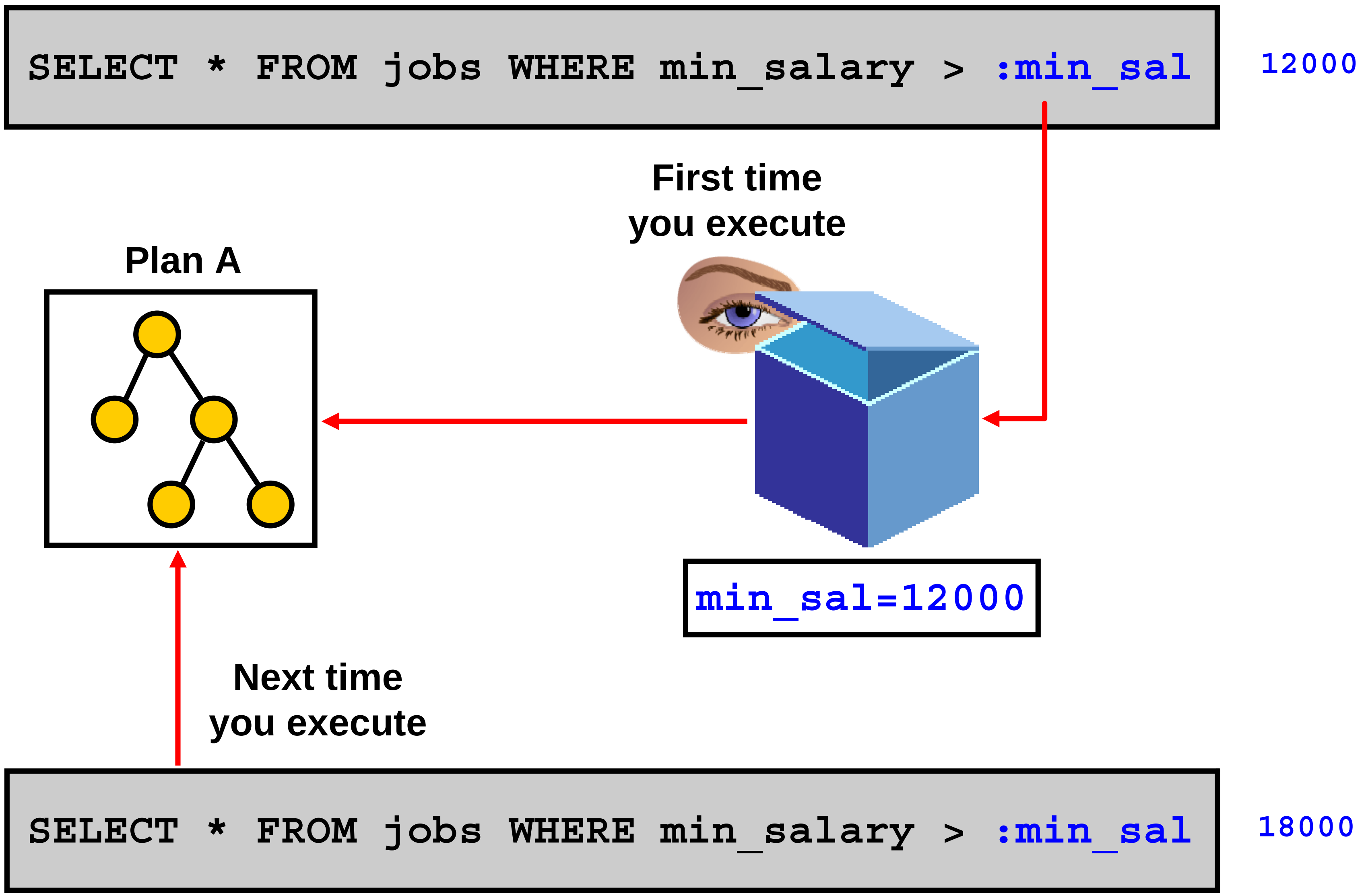
```
select count(*)
from hr.employees
where salary between :low_sal and :hi_sal
```

Last Execution Details

Execution Time (seconds) **0.0010**

COUNT(*)
43

Bind Variable Peeking



```
SELECT * FROM jobs WHERE min_salary > :min_sal
```

12000

First time you execute

Plan A

```
min_sal=12000
```

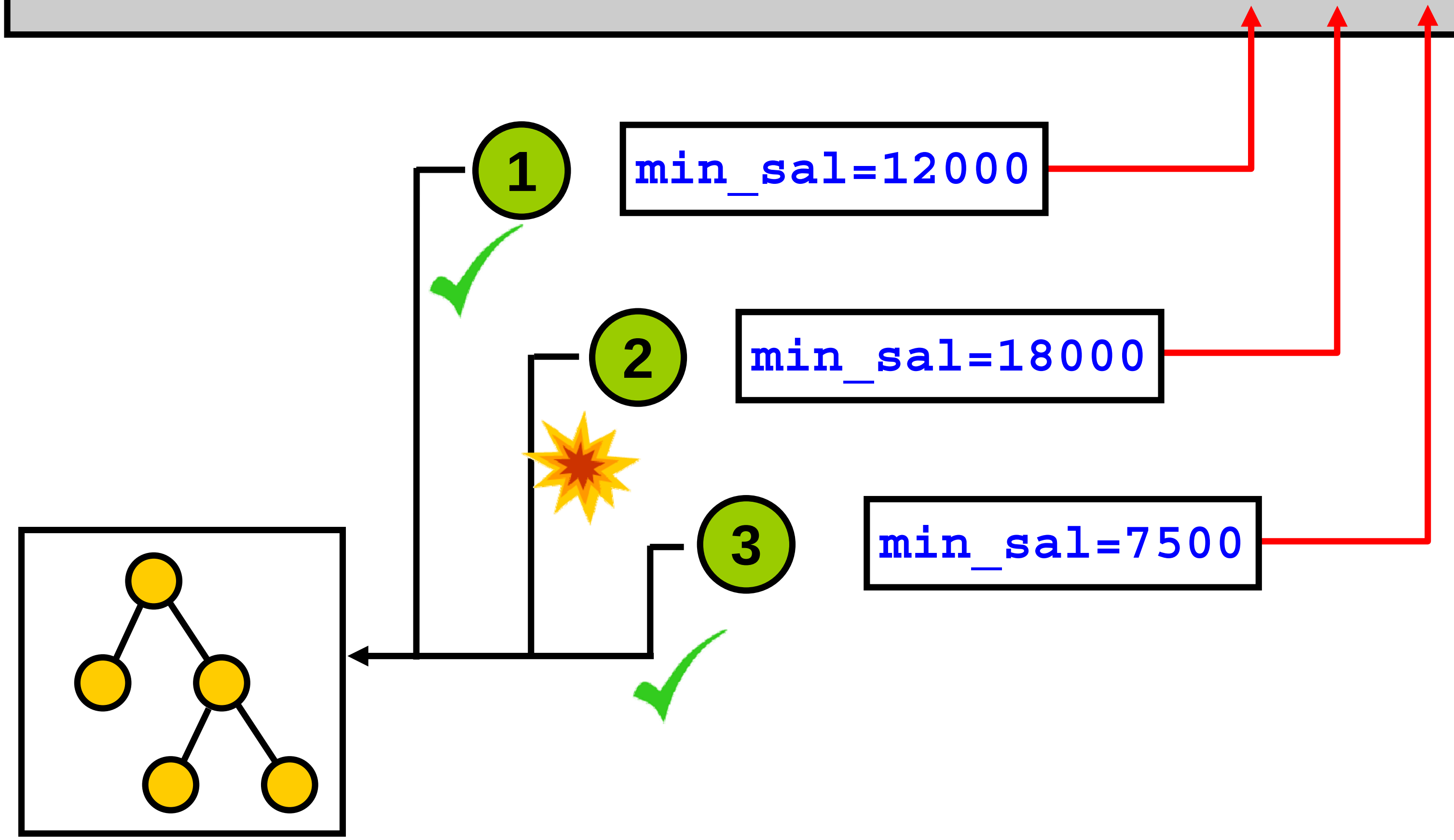
Next time you execute

```
SELECT * FROM jobs WHERE min_salary > :min_sal
```

18000

Bind Variable Peeking

```
SELECT * FROM jobs WHERE min_salary > :min_sal
```



min_sal=12000

min_sal=18000

min_sal=7500

Plan A

One plan not always appropriate for all bind values

Cursor Sharing Enhancements

- Oracle8*i* introduced the possibility of sharing SQL statements that differ only in literal values.
- Oracle9*i* extends this feature by limiting it to similar statements only, instead of forcing it.
- Similar: Regardless of the literal value, same execution plan

```
SQL> SELECT * FROM employees
  2  WHERE employee_id = 153;
```

- Not similar: Possible different execution plans for different literal values

```
SQL> SELECT * FROM employees
  2  WHERE department_id = 50;
```

The CURSOR_SHARING Parameter

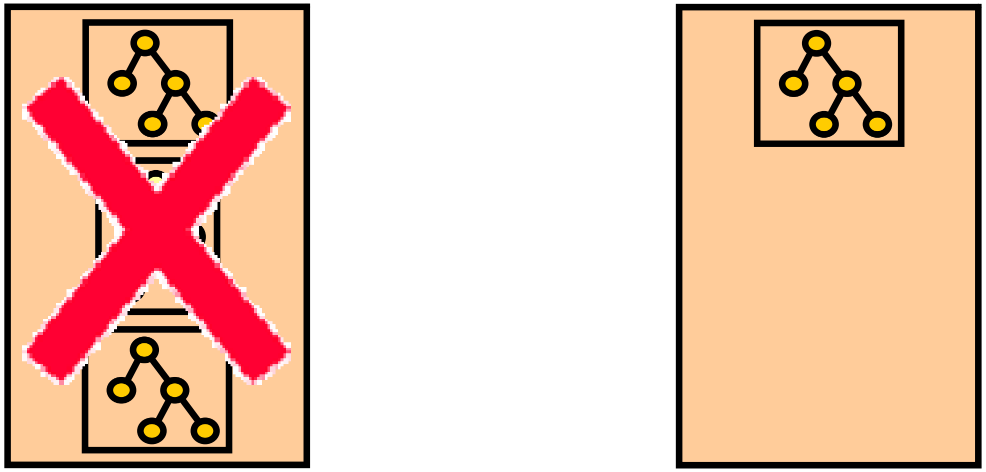
- The CURSOR_SHARING parameter values:
  - FORCE
  - EXACT (default)
  - SIMILAR
- CURSOR_SHARING can be changed using:
  - ALTER SYSTEM
  - ALTER SESSION
  - Initialization parameter files
- The CURSOR_SHARING_EXACT hint

Forcing Cursor Sharing: Example

```
SQL> alter session set cursor_sharing = FORCE;
```

```
SELECT * FROM jobs WHERE min_salary > 12000;
SELECT * FROM jobs WHERE min_salary > 18000;
SELECT * FROM jobs WHERE min_salary > 7500;
```

```
SELECT * FROM jobs WHERE min_salary > :"SYS_B_0"
```



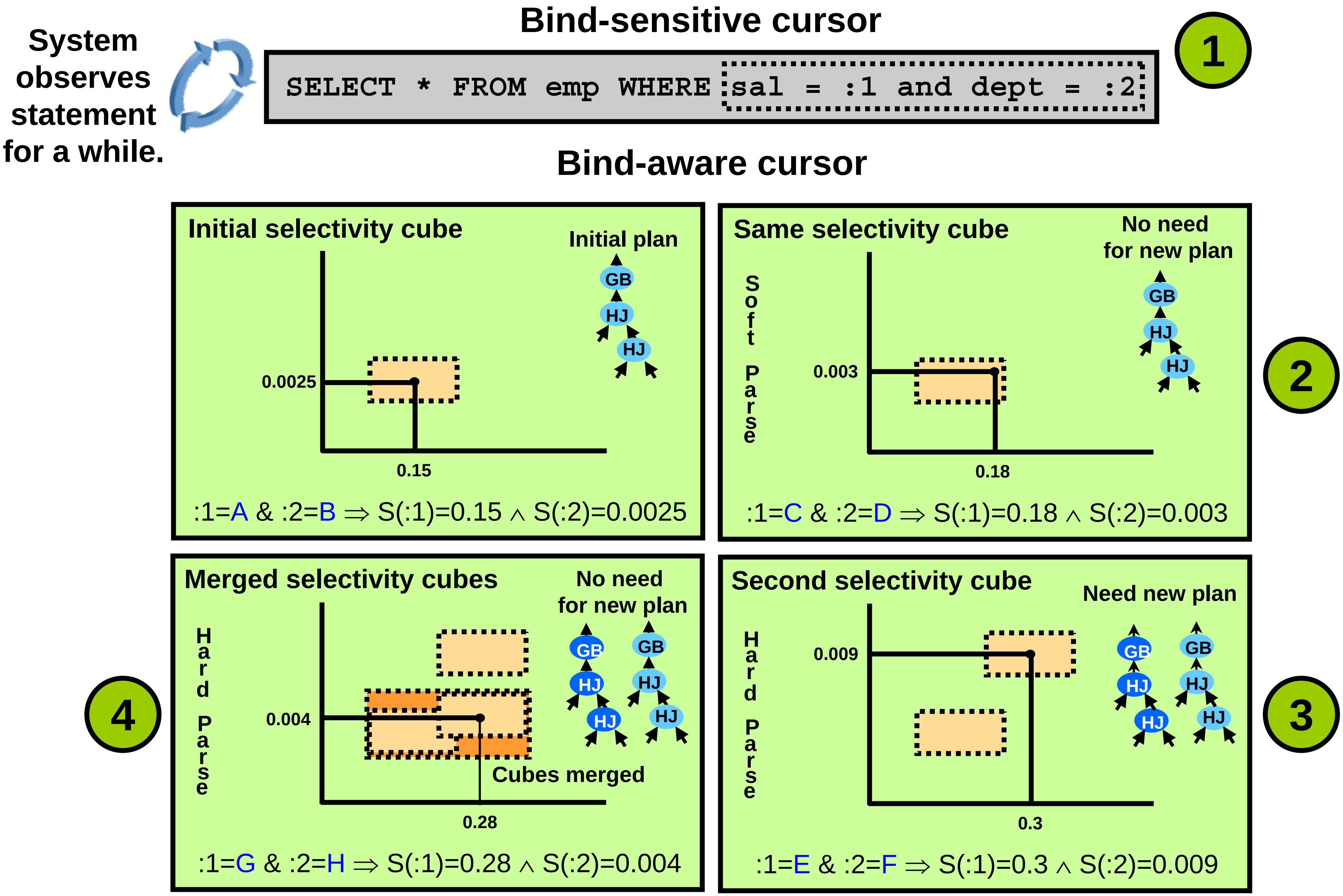
System-generated
bind variable

Adaptive Cursor Sharing: Overview

Adaptive cursor sharing:

- Allows for intelligent cursor sharing only for statements that use bind variables
- Is used to compromise between cursor sharing and optimization
- Has the following benefits:
  - Automatically detects when different executions would benefit from different execution plans
  - Limits the number of generated child cursors to a minimum
  - Automated mechanism that cannot be turned off

Adaptive Cursor Sharing: Architecture



System observes statement for a while.

Bind-sensitive cursor

```
SELECT * FROM emp WHERE sal = :1 and dept = :2
```

Bind-aware cursor

Initial selectivity cube

:1=A & :2=B ⇒ S(:1)=0.15 ∧ S(:2)=0.0025

Same selectivity cube

:1=C & :2=D ⇒ S(:1)=0.18 ∧ S(:2)=0.003

Second selectivity cube

:1=E & :2=F ⇒ S(:1)=0.3 ∧ S(:2)=0.009

Merged selectivity cubes

:1=G & :2=H ⇒ S(:1)=0.28 ∧ S(:2)=0.004

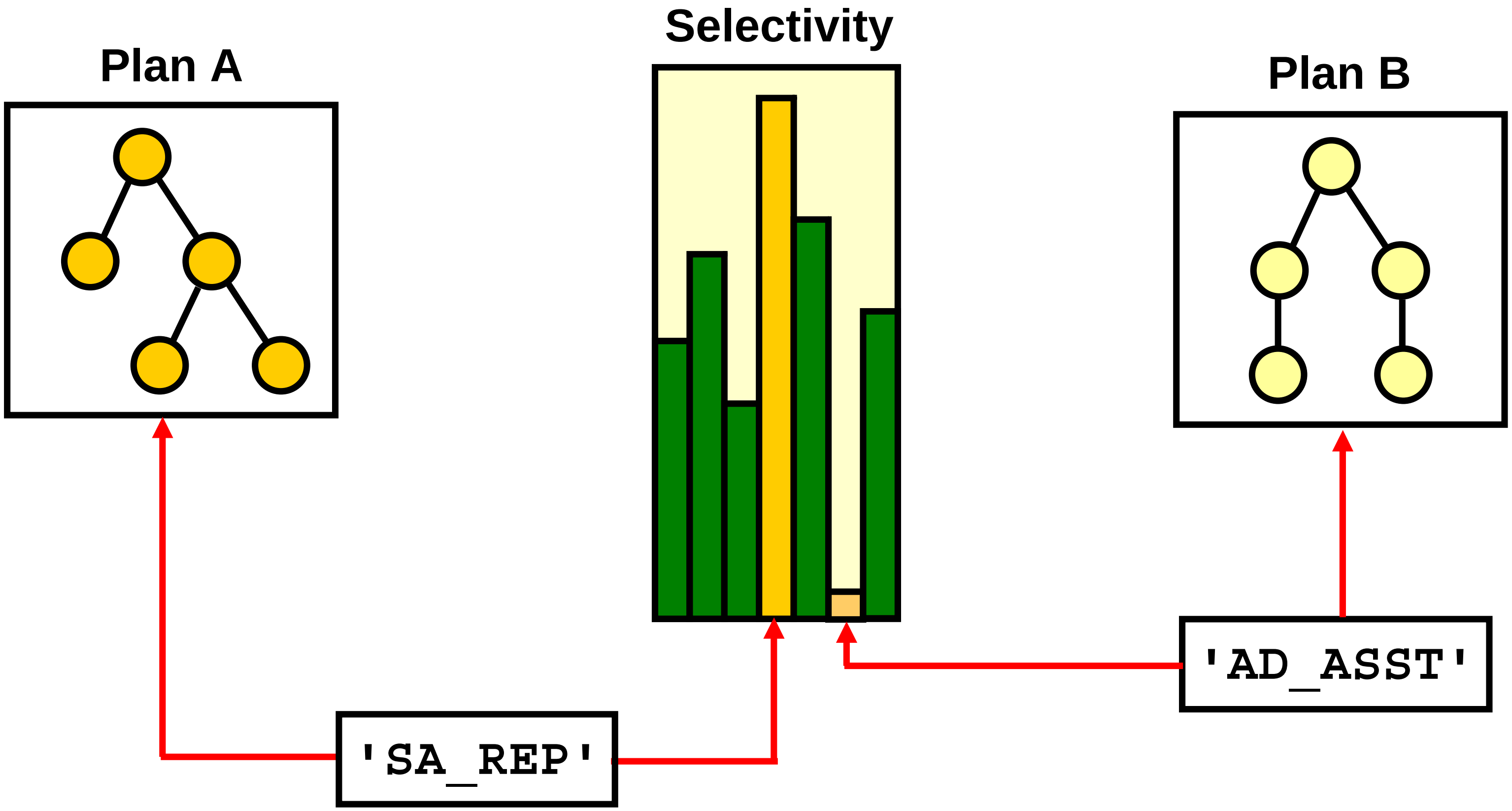
Adaptive Cursor Sharing: Views

The following views provide information about adaptive cursor sharing usage:

`V$SQL`	Two new columns show whether a cursor is bind sensitive or bind aware.
`V$SQL_CS_HISTOGRAM`	Shows the distribution of the execution count across the execution history histogram
`V$SQL_CS_SELECTIVITY`	Shows the selectivity cubes stored for every predicate containing a bind variable and whose selectivity is used in the cursor sharing checks
`V$SQL_CS_STATISTICS`	Shows execution statistics of a cursor using different bind sets

Adaptive Cursor Sharing: Example

```
SQL> variable job varchar2(6)
SQL> exec :job := 'AD_ASST'
SQL> select count(*), max(salary) from emp where job_id=:job;
```



Interacting with Adaptive Cursor Sharing

- `CURSOR_SHARING`:
  - If `CURSOR_SHARING <> EXACT`, statements containing literals may be rewritten using bind variables.
  - If statements are rewritten, adaptive cursor sharing may apply to them.
- SQL Plan Management (SPM):
  - If `OPTIMIZER_CAPTURE_SQL_PLAN_BASELINES` is set to `TRUE`, only the first generated plan is used.
  - As a workaround, set this parameter to `FALSE`, and run your application until all plans are loaded in the cursor cache.
  - Manually load the cursor cache into the corresponding plan baseline.

Summary

In this lesson, you should have learned how to:

- List the benefits of using bind variables
- Use bind peeking
- Use adaptive cursor sharing

Practice 8: Overview

This practice covers the following topics:

- Using adaptive cursor sharing and bind peeking
- Using the `CURSOR_SHARING` initialization parameter

9

Using Optimizer Hints

Objectives

After completing this lesson, you should be able to :

- Use hints when appropriate
- Specify hints for:
  - Optimizer mode
  - Query transformation
  - Access path
  - Join orders
  - Join methods

Optimizer Hints: Overview

Optimizer hints:

- Influence optimizer decisions
- Example:

```
SELECT /*+ INDEX(e empfirstname_idx) skewed col */ *
FROM employees e
WHERE first_name='David'
```

- HINTS SHOULD ONLY BE USED AS A LAST RESORT.
- When you use a hint, it is good practice to also add a comment about that hint.

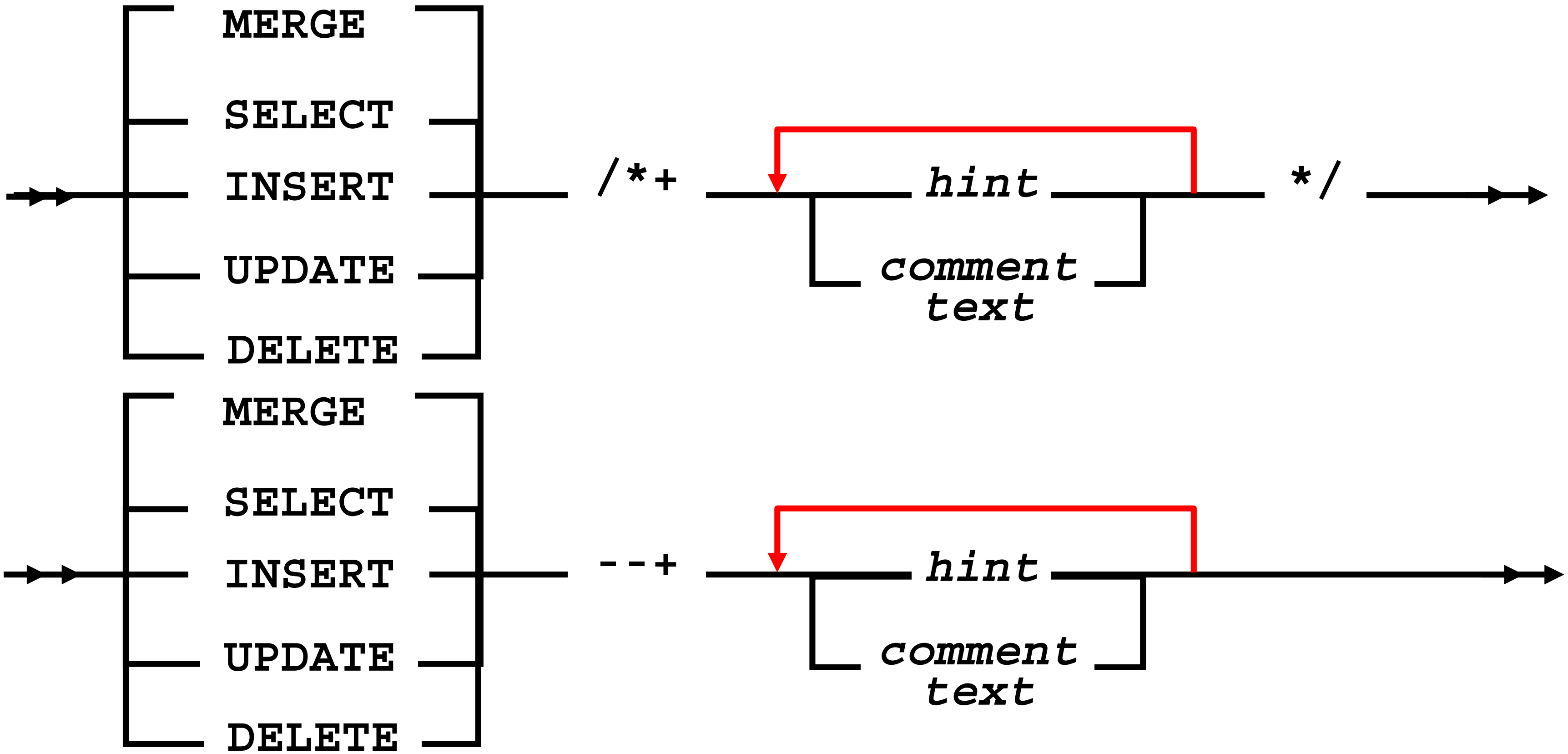
Types of Hints

Single-table hints	Specified on one table or view
Multitable hints	Specify more than one table or view
Query block hints	Operate on a single query block
Statement hints	Apply to the entire SQL statement

Specifying Hints

Hints apply to the optimization of only one statement block:

- A self-contained DML statement against a table
- A top-level DML or a subquery



Rules for Hints

- Place hints immediately after the first SQL keyword of a statement block.
- Each statement block can have only one hint comment, but it can contain multiple hints.
- Hints apply to only the statement block in which they appear.
- If a statement uses aliases, hints must reference the aliases rather than the table names.
- The optimizer ignores hints specified incorrectly without raising errors.

Hint Recommendations

- Use hints carefully because they imply a high-maintenance load.
- Be aware of the performance impact of hard-coded hints when they become less valid.

Optimizer Hint Syntax: Example

```
UPDATE /*+ INDEX(p PRODUCTS_PROD_CAT_IX)*/
products p
SET   p.prod_min_price =
        (SELECT
         (pr.prod_list_price*.95)
         FROM products pr
         WHERE p.prod_id = pr.prod_id)
WHERE p.prod_category = 'Men'
AND   p.prod_status = 'available, on stock'
/
```

Hint Categories

There are hints for:

- Optimization approaches and goals
- Access paths
- Query transformations
- Join orders
- Join operation
- Parallel execution
- Additional hints

Optimization Goals and Approaches

`ALL_ROWS`	Selects a cost-based approach with a goal of best throughput
`FIRST_ROWS(`*`n`*`)`	Instructs the Oracle server to optimize an individual SQL statement for fast response

Note: The `ALTER SESSION... SET OPTIMIZER_MODE` statement does not affect SQL that is run from within PL/SQL.

Hints for Access Paths

FULL	Performs a full table scan
CLUSTER	Accesses table by cluster scan
HASH	Accesses table by hash scan
ROWID	Accesses a table by ROWID
INDEX	Scans an index in the ascending order
INDEX_ASC	Scans an index in the ascending order
INDEX_COMBINE	Explicitly chooses a bitmap access path

Hints for Access Paths

`INDEX_JOIN`	Instructs the optimizer to use an index join as an access path
`INDEX_DESC`	Selects an index scan for the specified table
`INDEX_FFS`	Performs a fast-full index scan
`INDEX_SS`	Performs an index skip scan
`NO_INDEX`	Does not allow using a set of indexes
`AND_EQUAL`	Merges single-column indexes

The INDEX_COMBINE Hint: Example

```
SELECT --+INDEX_COMBINE(CUSTOMERS)
       cust_last_name
FROM  SH.CUSTOMERS
WHERE ( CUST_GENDER= 'F' AND
CUST_MARITAL_STATUS =  'single')
OR     CUST_YEAR_OF_BIRTH BETWEEN '1917'
AND '1920';
```

The INDEX_COMBINE Hint: Example

```
Execution Plan
---------------------------------------------------
|   0 | SELECT STATEMENT                |
|   1 |  TABLE ACCESS BY INDEX ROWID    | CUSTOMERS
|   2 |   BITMAP CONVERSION TO ROWIDS   |
|   3 |    BITMAP OR                    |
|   4 |     BITMAP MERGE                |
|   5 |      BITMAP INDEX RANGE SCAN    | CUST_YOB_BIX
|   6 |     BITMAP AND                  |
|   7 |      BITMAP INDEX SINGLE VALUE  | CUST_MARITAL_BIX
|   8 |      BITMAP INDEX SINGLE VALUE  | CUST_GENDER_BIX
```

Hints for Query Transformation

NO_QUERY_TRANSFORMATION	Skips all query transformation
USE_CONCAT	Rewrites OR into UNION ALL and disables INLIST processing
NO_EXPAND	Prevents OR expansions
REWRITE	Rewrites query in terms of materialized views
NO_REWRITE	Turns off query rewrite
UNNEST	Merges subquery bodies into surrounding query block
NO_UNNEST	Turns off unnesting

Hints for Query Transformation

MERGE	Merges complex views or subqueries with the surrounding query
NO_MERGE	Prevents merging of mergeable views
STAR_TRANSFORMATION	Makes the optimizer use the best plan in which the transformation can be used
FACT	Indicates that the hinted table should be considered as a fact table
NO_FACT	Indicates that the hinted table should not be considered as a fact table

Hints for Join Orders

`ORDERED`	Causes the Oracle server to join tables in the order in which they appear in the `FROM` clause
`LEADING`	Uses the specified tables as the first table in the join order

Hints for Join Operations

`USE_NL`	Joins the specified table using a nested loop join
`NO_USE_NL`	Does not use nested loops to perform the join
`USE_NL_WITH_INDEX`	Similar to `USE_NL`, but must be able to use an index for the join
`USE_MERGE`	Joins the specified table using a sort-merge join
`NO_USE_MERGE`	Does not perform sort-merge operations for the join
`USE_HASH`	Joins the specified table using a hash join
`NO_USE_HASH`	Does not use hash join
`DRIVING_SITE`	Instructs the optimizer to execute the query at a different site than that selected by the database

Additional Hints

APPEND	Enables direct-path `INSERT`
NOAPPEND	Enables regular `INSERT`
ORDERED_PREDICATES	Forces the optimizer to preserve the order of predicate evaluation
CURSOR_SHARING_EXACT	Prevents replacing literals with bind variables
CACHE	Overrides the default caching specification of the table
PUSH_PRED	Pushes join predicate into view
PUSH_SUBQ	Evaluates nonmerged subqueries first
DYNAMIC_SAMPLING	Controls dynamic sampling to improve server performance

Additional Hints

`MONITOR`	Forces real-time query monitoring
`NO_MONITOR`	Disables real-time query monitoring
`RESULT_CACHE`	Caches the result of the query or query fragment
`NO_RESULT_CACHE`	Disables result caching for the query or query fragment
`OPT_PARAM`	Sets initialization parameter for query duration

Hints and Views

- Do not use hints in views.
- Use view-optimization techniques:
  - Statement transformation
  - Results accessed like a table
- Hints can be used on mergeable views and nonmergeable views.

Global Table Hints

- Extended hint syntax enables specifying for tables that appear in views
- References a table name in the hint with a recursive dot notation

```
CREATE view city_view AS
SELECT *
FROM   customers c
WHERE  cust_city like 'S%';
```

```
SELECT /*+ index(v.c cust_credit_limit_idx) */
    v.cust_last_name, v.cust_credit_limit
FROM   city_view v
WHERE  cust_credit_limit > 5000;
```

Specifying a Query Block in a Hint

```
explain plan for
select /*+ FULL(@strange dept) */ ename
from emp e, (select /*+ QB_NAME(strange) */ *
             from dept where deptno=10) d
where e.deptno = d.deptno and d.loc = 'C';
```

```
SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY(NULL, NULL, 'ALL'));

Plan hash value: 615168685
---------------------------------------------------------------
| Id  | Operation          | Name | Rows  | Bytes | Cost(%CPU)|
---------------------------------------------------------------
|   0 | SELECT STATEMENT   |      |     1 |    41 |     7 (15)|
|*  1 |  HASH JOIN         |      |     1 |    41 |     7 (15)|
|*  2 |   TABLE ACCESS FULL| DEPT |     1 |    21 |     3  (0)|
|*  3 |   TABLE ACCESS FULL| EMP  |     3 |    60 |     3  (0)|
---------------------------------------------------------------
Query Block Name / Object Alias (identified by operation id):
-------------------------------------------------------------
   1 - SEL$DB579D14
   2 - SEL$DB579D14 / DEPT@STRANGE
   3 - SEL$DB579D14 / E@SEL$1
```

Specifying a Full Set of Hints

```
SELECT /*+ LEADING(e2 e1) USE_NL(e1)
   INDEX(e1 emp_emp_id_pk) USE_MERGE(j) FULL(j) */
    e1.first_name, e1.last_name, j.job_id,
     sum(e2.salary) total_sal
FROM hr.employees e1, hr.employees e2,
hr.job_history j
WHERE e1.employee_id = e2.manager_id
AND e1.employee_id = j.employee_id
AND e1.hire_date = j.start_date
GROUP BY e1.first_name, e1.last_name, j.job_id
ORDER BY total_sal;
```

Summary

In this lesson, you should have learned how to:

- Set the optimizer mode
- Use optimizer hint syntax
- Determine access-path hints
- Analyze hints and their impact on views

Practice 9: Overview

This practice covers using various hints to influence execution plans.

10

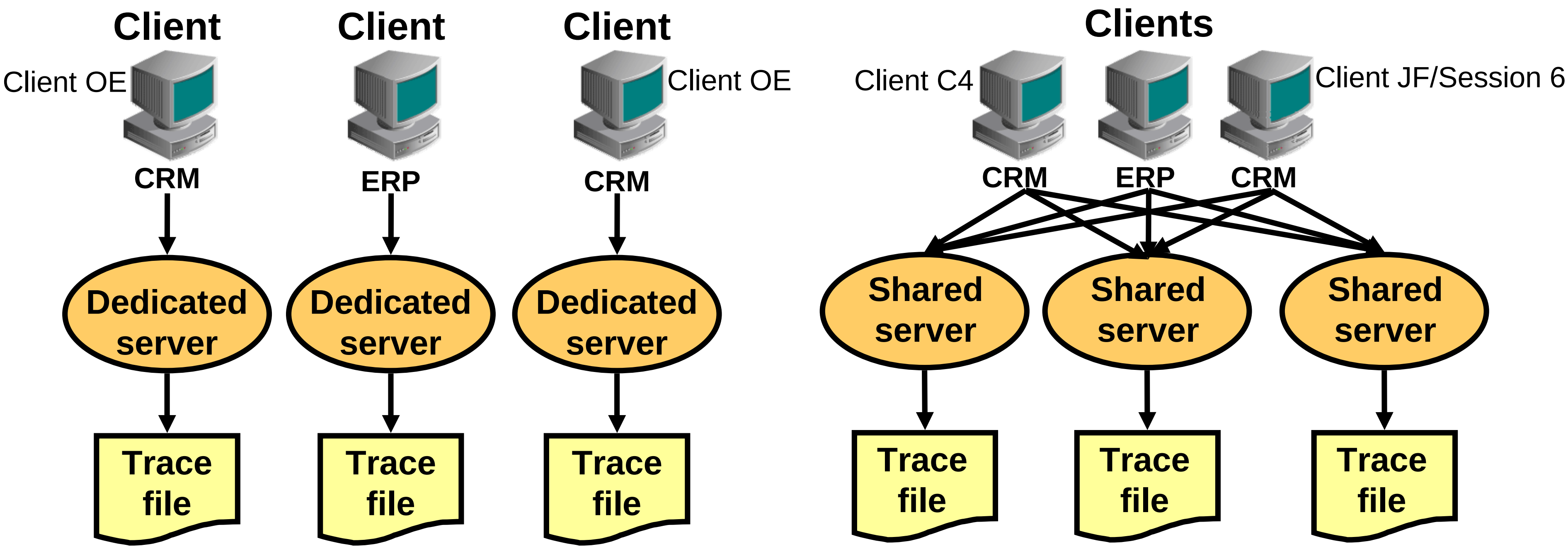
Application Tracing

Objectives

After completing this lesson, you should be able to do the following:

- Configure the SQL Trace facility to collect session statistics
- Use the `TRCSESS` utility to consolidate SQL trace files
- Format trace files using the `tkprof` utility
- Interpret the output of the `tkprof` command

End-to-End Application Tracing Challenge



- I want to retrieve traces from CRM service.
- I want to retrieve traces from client C4.
- I want to retrieve traces from session 6.

End-to-End Application Tracing

- Simplifies the process of diagnosing performance problems in multitier environments by allowing application workloads to be seen by:
  - Service
  - Module
  - Action
  - Session
  - Client
- End-to-End Application Tracing tools:
  - Enterprise Manager
  - DBMS_APPICATION_INFO, DBMS_SERVICE, DBMS_MONITOR, DBMS_SESSION
  - SQL Trace and TRCSESS utility
  - tkprof

Location for Diagnostic Traces

`DIAGNOSTIC_DEST`

Diagnostic Data	Previous Location	ADR Location
Foreground process traces	`USER_DUMP_DEST`	`$ADR_HOME/trace`
Background process traces	`BACKGROUND_DUMP_DEST`	`$ADR_HOME/trace`
Alert log data	`BACKGROUND_DUMP_DEST`	`$ADR_HOME/alert&trace`
Core dumps	`CORE_DUMP_DEST`	`$ADR_HOME/cdump`
Incident dumps	`USER|BACKGROUND_DUMP_DEST`	`$ADR_HOME/incident/incdir_`*`n`*

`V$DIAG_INFO`

ADR trace = **Oracle Database 10*g* trace – critical error trace**

What Is a Service?

- Is a means of grouping sessions that perform the same kind of work
- Provides a single-system image instead of a multiple-instances image
- Is a part of the regular administration tasks that provide dynamic service-to-instance allocation
- Is the base for High Availability of connections
- Provides a performance-tuning dimension
- Is a handle for capturing trace information

Use Services with Client Applications

```
ERP=(DESCRIPTION=
      (ADDRESS=(PROTOCOL=TCP)(HOST=mynode)(PORT=1521))
    (CONNECT_DATA=(SERVICE_NAME=ERP)))
```

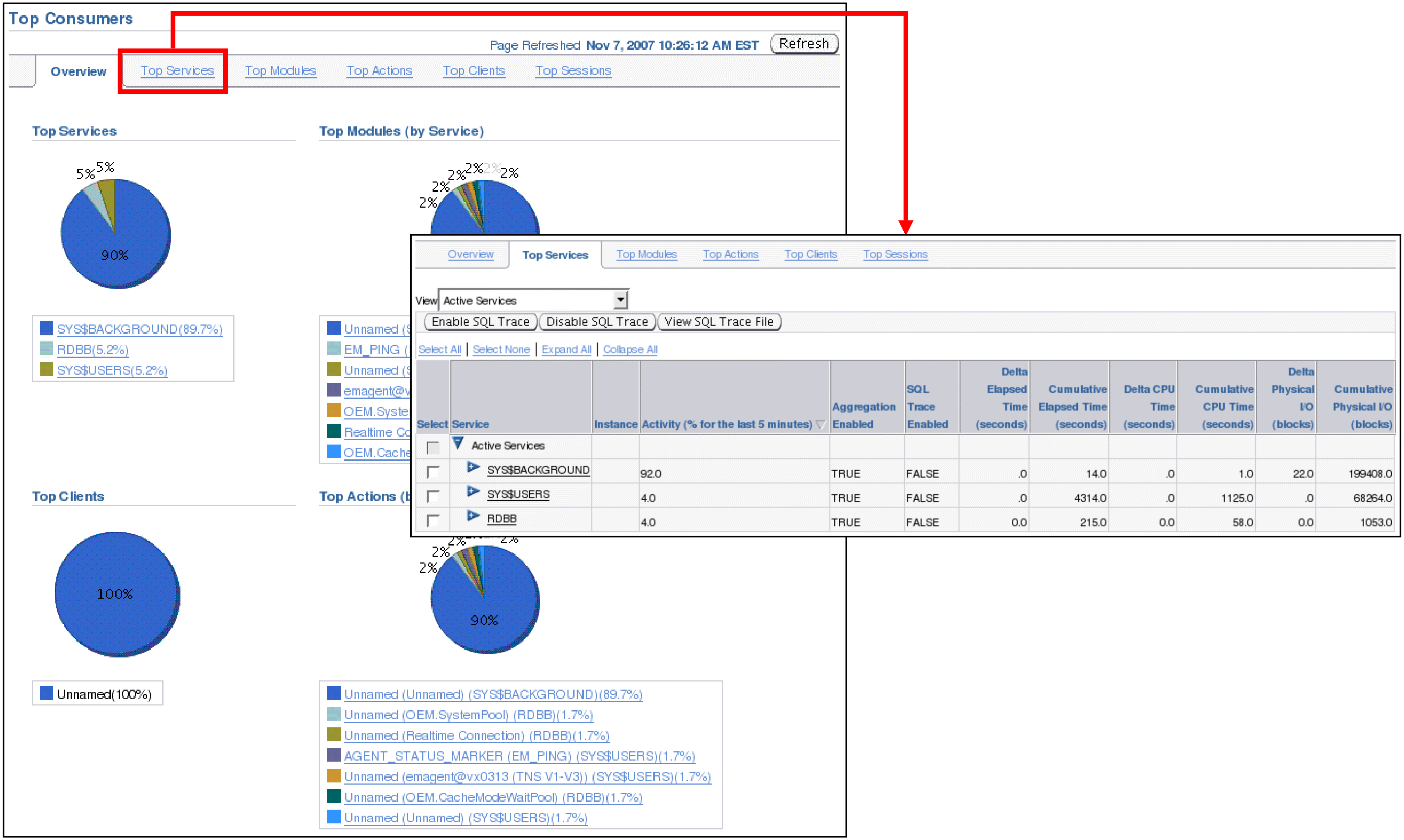
```
url="jdbc:oracle:oci:@ERP"
```

```
url="jdbc:oracle:thin:@(DESCRIPTION=
      (ADDRESS=(PROTOCOL=TCP)(HOST=mynode)(PORT=1521))
    (CONNECT_DATA=(SERVICE_NAME=ERP)))"
```

Tracing Services

- Applications using services can be further qualified by:
  - MODULE
  - ACTION
  - CLIENT_IDENTIFIER
- Set using the following PL/SQL packages:
  - DBMS_APPLICATION_INFO
  - DBMS_SESSION
- Tracing can be done at all levels:
  - CLIENT_IDENTIFIER
  - SESSION_ID
  - SERVICE_NAMES
  - MODULE
  - ACTION
  - Combination of SERVICE_NAME, MODULE, ACTION

Use Enterprise Manager to Trace Services



Top Consumers

Page Refreshed **Nov 7, 2007 10:26:12 AM EST** Refresh

Overview | Top Services | Top Modules | Top Actions | Top Clients | Top Sessions

Top Services

- 🟦 SYS$BACKGROUND(89.7%)
- 🟦 RDBB(5.2%)
- 🟨 SYS$USERS(5.2%)

Top Modules (by Service)

- 🟦 Unnamed (S
- 🟦 EM_PING (
- 🟨 Unnamed (S
- 🟪 emagent@v
- 🟧 OEM.Syste
- 🟩 Realtime Co
- 🟦 OEM.Cache

Top Clients

- 🟦 Unnamed(100%)

Top Actions (b

- 🟦 Unnamed (Unnamed) (SYS$BACKGROUND)(89.7%)
- 🟦 Unnamed (OEM.SystemPool) (RDBB)(1.7%)
- 🟨 Unnamed (Realtime Connection) (RDBB)(1.7%)
- 🟪 AGENT_STATUS_MARKER (EM_PING) (SYS$USERS)(1.7%)
- 🟧 Unnamed (emagent@vx0313 (TNS V1-V3)) (SYS$USERS)(1.7%)
- 🟩 Unnamed (OEM.CacheModeWaitPool) (RDBB)(1.7%)
- 🟦 Unnamed (Unnamed) (SYS$USERS)(1.7%)

Overview | **Top Services** | Top Modules | Top Actions | Top Clients | Top Sessions

View: Active Services

Enable SQL Trace | Disable SQL Trace | View SQL Trace File

Select All | Select None | Expand All | Collapse All

Select	Service	Instance	Activity (% for the last 5 minutes)	Aggregation Enabled	SQL Trace Enabled	Delta Elapsed Time (seconds)	Cumulative Elapsed Time (seconds)	Delta CPU Time (seconds)	Cumulative CPU Time (seconds)	Delta Physical I/O (blocks)	Cumulative Physical I/O (blocks)
☐	▼ Active Services										
☐	▶ SYS$BACKGROUND		92.0	TRUE	FALSE	.0	14.0	.0	1.0	22.0	199408.0
☐	▶ SYS$USERS		4.0	TRUE	FALSE	.0	4314.0	.0	1125.0	.0	68264.0
☐	▶ RDBB		4.0	TRUE	FALSE	0.0	215.0	0.0	58.0	0.0	1053.0

Service Tracing: Example

- Trace on service, module, and action:

```
exec DBMS_MONITOR.SERV_MOD_ACT_TRACE_ENABLE('AP');
```

```
exec DBMS_MONITOR.SERV_MOD_ACT_TRACE_ENABLE(-
     'AP', 'PAYMENTS', 'QUERY_DELINQUENT');
```

- Trace a particular client identifier:

```
exec DBMS_MONITOR.CLIENT_ID_TRACE_ENABLE
   (client_id=>'C4', waits => TRUE, binds => FALSE);
```

Session Level Tracing: Example

- For all sessions in the database:

```
EXEC dbms_monitor.DATABASE_TRACE_ENABLE(TRUE,TRUE);
```

```
EXEC dbms_monitor.DATABASE_TRACE_DISABLE();
```

- For a particular session:

```
EXEC dbms_monitor.SESSION_TRACE_ENABLE(session_id=>
27, serial_num=>60, waits=>TRUE, binds=>FALSE);
```

```
EXEC dbms_monitor.SESSION_TRACE_DISABLE(session_id
=>27, serial_num=>60);
```

Trace Your Own Session

- Enabling trace:

```
EXEC DBMS_SESSION.SESSION_TRACE_ENABLE(waits =>
TRUE, binds => FALSE);
```

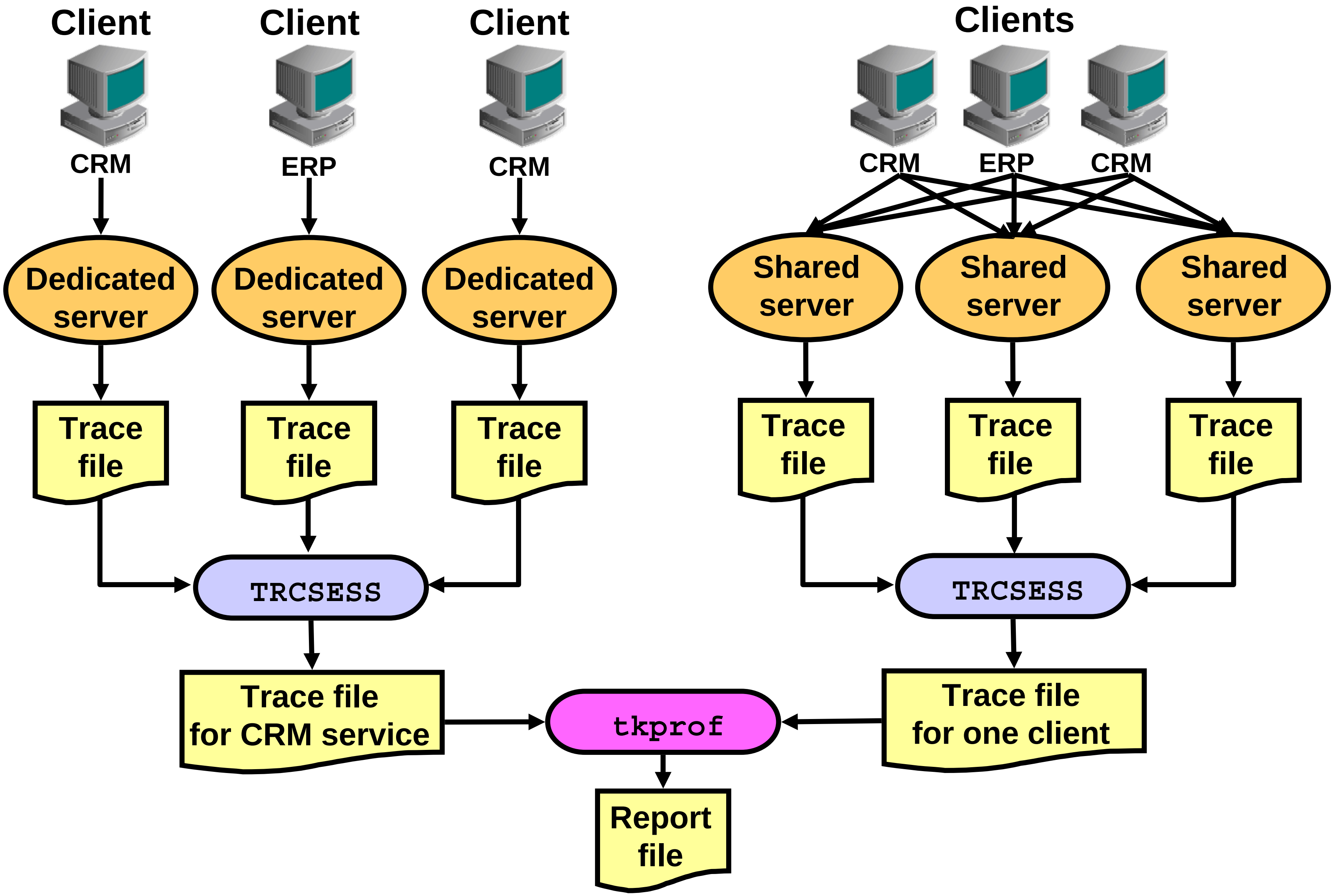
- Disabling trace:

```
EXEC DBMS_SESSION.SESSION_TRACE_DISABLE();
```

- Easily identifying your trace files:

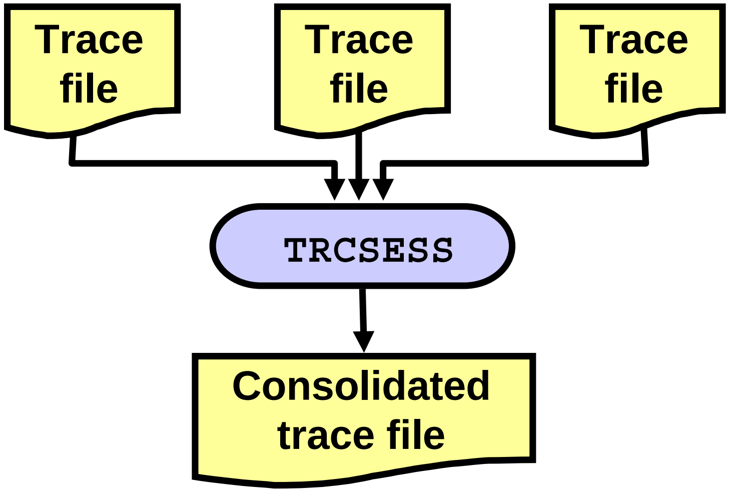
```
alter session set
tracefile_identifier='mytraceid';
```

The `trcsess` Utility



Invoking the `trcsess` Utility

```
trcsess  [output=output_file_name]
         [session=session_id]
         [clientid=client_identifier]
         [service=service_name]
         [action=action_name]
         [module=module_name]
         [<trace file names>]
```



The trcsess Utility: Example

```
exec dbms_session.set_identifier('HR session');
```

First session

Second session

```
exec dbms_session.set_identifier('HR session');
```

```
exec DBMS_MONITOR.CLIENT_ID_TRACE_ENABLE( -
client_id=>'HR session', waits => FALSE, binds => FALSE);
```

Third session

```
select * from employees;
```

...

```
select * from departments;
```

...

```
exec DBMS_MONITOR.CLIENT_ID_TRACE_DISABLE( -
client_id => 'HR session');
```

```
trcsess output=mytrace.trc clientid='HR session'
$ORACLE_BASE/diag/rdbms/orcl/orcl/trace/*.trc
```

SQL Trace File Contents

- Parse, execute, and fetch counts
- CPU and elapsed times
- Physical reads and logical reads
- Number of rows processed
- Misses on the library cache
- Username under which each parse occurred
- Each commit and rollback
- Wait event and bind data for each SQL statement
- Row operations showing the actual execution plan of each SQL statement
- Number of consistent reads, physical reads, physical writes, and time elapsed for each operation on a row

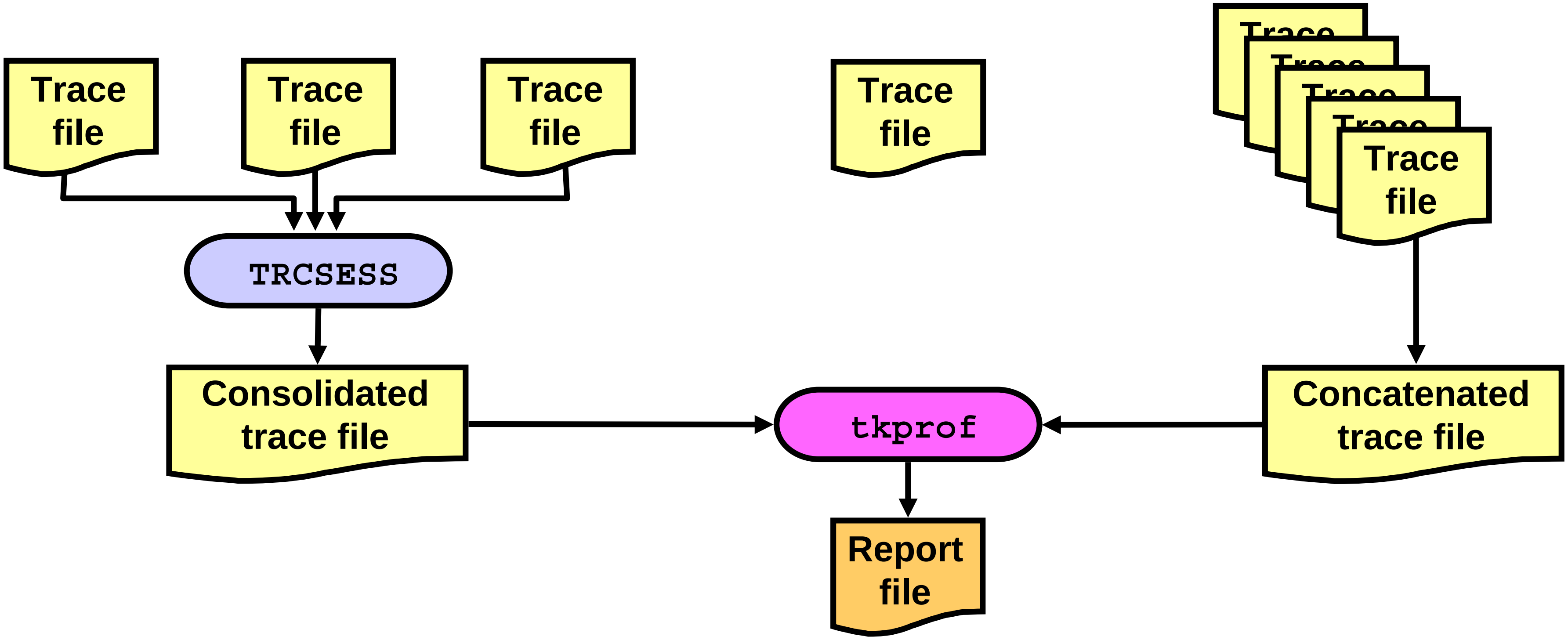
SQL Trace File Contents: Example

```
*** [ Unix process pid: 19687 ]
*** 2008-02-25 15:49:19.820
*** 2008-02-25 15:49:19.820
*** 2008-02-25 15:49:19.820
*** 2008-02-25 15:49:19.820
…
=====================
PARSING IN CURSOR #4 len=23 dep=0 uid=82 oct=3 lid=82 tim=1203929332521849
hv=4069246757 ad='34b6f730' sqlid='f34thrbt8rjt5'
select * from employees
END OF STMT
PARSE #4:c=49993,e=67123,p=28,cr=403,cu=0,mis=1,r=0,dep=0,og=1,tim=1203929332521845
EXEC #4:c=0,e=16,p=0,cr=0,cu=0,mis=0,r=0,dep=0,og=1,tim=1203929332521911
FETCH #4:c=1000,e=581,p=6,cr=6,cu=0,mis=0,r=1,dep=0,og=1,tim=1203929332522553
FETCH #4:c=0,e=45,p=0,cr=1,cu=0,mis=0,r=15,dep=0,og=1,tim=1203929332522936
…
FETCH #4:c=0,e=49,p=0,cr=1,cu=0,mis=0,r=1,dep=0,og=1,tim=1203929333649241
STAT #4 id=1 cnt=107 pid=0 pos=1 obj=70272 op='TABLE ACCESS FULL EMPLOYEES (cr=15
pr=6 pw=6 time=0 us cost=3 size=7276 card=107)'
*** [ Unix process pid: 19687 ]
*** 2008-02-25 15:49:19.820
*** 2008-02-25 15:49:19.820
*** 2008-02-25 15:49:19.820
*** 2008-02-25 15:49:19.820
…
```

Formatting SQL Trace Files: Overview

Use the `tkprof` utility to format your SQL trace files:

- Sort raw trace file to exhibit top SQL statements
- Filter dictionary statements



Invoking the `tkprof` Utility

```
tkprof inputfile outputfile [waits=yes|no]
                            [sort=option]
                            [print=n]
                            [aggregate=yes|no]
                            [insert=sqlscritfile]
                            [sys=yes|no]
                            [table=schema.table]
                            [explain=user/password]
                            [record=statementfile]
                            [width=n]
```

`tkprof` Sorting Options

Sort Option	Description
`prscnt`	Number of times parse was called
`prscpu`	CPU time parsing
`prsela`	Elapsed time parsing
`prsdsk`	Number of disk reads during parse
`prsqry`	Number of buffers for consistent read during parse
`prscu`	Number of buffers for current read during parse
`prsmis`	Number of misses in the library cache during parse
`execnt`	Number of executes that were called
`execpu`	CPU time spent executing
`exeela`	Elapsed time executing
`exedsk`	Number of disk reads during execute
`exeqry`	Number of buffers for consistent read during execute
`execu`	Number of buffers for current read during execute

`tkprof` Sorting Options

Sort Option	Description
`exerow`	Number of rows processed during execute
`exemis`	Number of library cache misses during execute
`fchcnt`	Number of times fetch was called
`fchcpu`	CPU time spent fetching
`fchela`	Elapsed time fetching
`fchdsk`	Number of disk reads during fetch
`fchqry`	Number of buffers for consistent read during fetch
`fchcu`	Number of buffers for current read during fetch
`fchrow`	Number of rows fetched
`userid`	User ID of user that parsed the cursor

Output of the `tkprof` Command

- Text of the SQL statement
- Trace statistics (for statement and recursive calls) separated into three SQL processing steps:

PARSE	Translates the SQL statement into an execution plan
EXECUTE	Executes the statement (This step modifies the data for the `INSERT`, `UPDATE`, and `DELETE` statements.)
FETCH	Retrieves the rows returned by a query (Fetches are performed only for the `SELECT` statements.)

Output of the `tkprof` Command

There are seven categories of trace statistics:

`Count`	Number of times the procedure was executed
`CPU`	Number of seconds to process
`Elapsed`	Total number of seconds to execute
`Disk`	Number of physical blocks read
`Query`	Number of logical buffers read for consistent read
`Current`	Number of logical buffers read in current mode
`Rows`	Number of rows processed by the fetch or execute

Output of the `tkprof` Command

The `tkprof` output also includes the following:

- Recursive SQL statements
- Library cache misses
- Parsing user ID
- Execution plan
- Optimizer mode or hint
- Row source operation

```
...
Misses in library cache during parse: 1
Optimizer mode: ALL_ROWS
Parsing user id: 61

Rows     Row Source Operation
-------  ---------------------------------------------------
     24  TABLE ACCESS BY INDEX ROWID EMPLOYEES (cr=9 pr=0 pw=0 time=129 us)
     24   INDEX RANGE SCAN SAL_IDX (cr=3 pr=0 pw=0 time=1554 us)(object id …
```

tkprof Output with No Index: Example

```
...
select max(cust_credit_limit)
from customers
where cust_city ='Paris'


call     count       cpu    elapsed       disk      query    current      rows
------- ------  -------- ---------- ---------- ---------- ----------  -------
Parse        1      0.02       0.02          0          0          0        0
Execute      1      0.00       0.00          0          0          0        0
Fetch        2      0.10       0.09       1408       1459          0        1
------- ------  -------- ---------- ---------- ---------- ----------  -------
total        4      0.12       0.11       1408       1459          0        1

Misses in library cache during parse: 1
Optimizer mode: ALL_ROWS
Parsing user id: 61

Rows     Row Source Operation
-------  ---------------------------------------------------
      1  SORT AGGREGATE (cr=1459 pr=1408 pw=0 time=93463 us)
     77   TABLE ACCESS FULL CUSTOMERS (cr=1459 pr=1408 pw=0 time=31483 us)
```

tkprof Output with Index: Example

```
...
select max(cust_credit_limit) from customers
where cust_city ='Paris'

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ---------
Parse        1      0.01       0.00          0          0          0          0
Execute      1      0.00       0.00          0          0          0          0
Fetch        2      0.00       0.00          0         77          0          1
------- ------  -------- ---------- ---------- ---------- ----------  ---------
total        4      0.01       0.00          0         77          0          1

Misses in library cache during parse: 1
Optimizer mode: ALL_ROWS
Parsing user id: 61

Rows     Row Source Operation
-------  ---------------------------------------------------
      1  SORT AGGREGATE (cr=77 pr=0 pw=0 time=732 us)
     77   TABLE ACCESS BY INDEX ROWID CUSTOMERS (cr=77 pr=0 pw=0 time=1760 us)
     77    INDEX RANGE SCAN CUST_CUST_CITY_IDX (cr=2 pr=0 pw=0 time=100
                                                            us)(object id 55097)
```

Summary

In this lesson, you should have learned how to:

- Configure the SQL Trace facility to collect session statistics
- Use the `TRCSESS` utility to consolidate SQL trace files
- Format trace files using the `tkprof` utility
- Interpret the output of the `tkprof` command

Practice 10: Overview

This practice covers the following topics:

- Creating a service
- Tracing your application using services
- Interpreting trace information using `trcsess` and `tkprof`

11

Automating SQL Tuning

Objectives

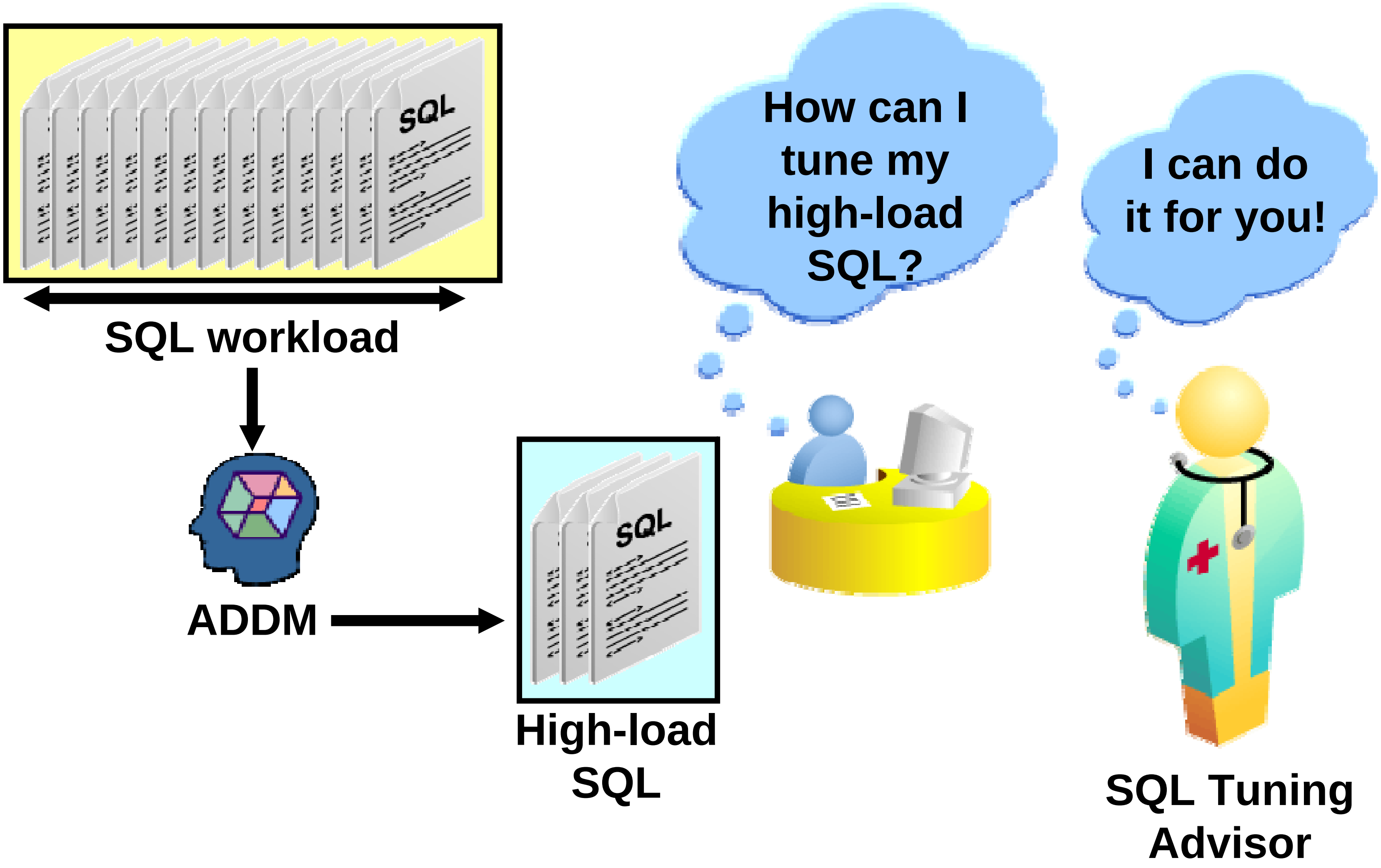
After completing this lesson, you should be able to do the following:

- Describe statement profiling
- Use SQL Tuning Advisor
- Use SQL Access Advisor
- Use Automatic SQL Tuning

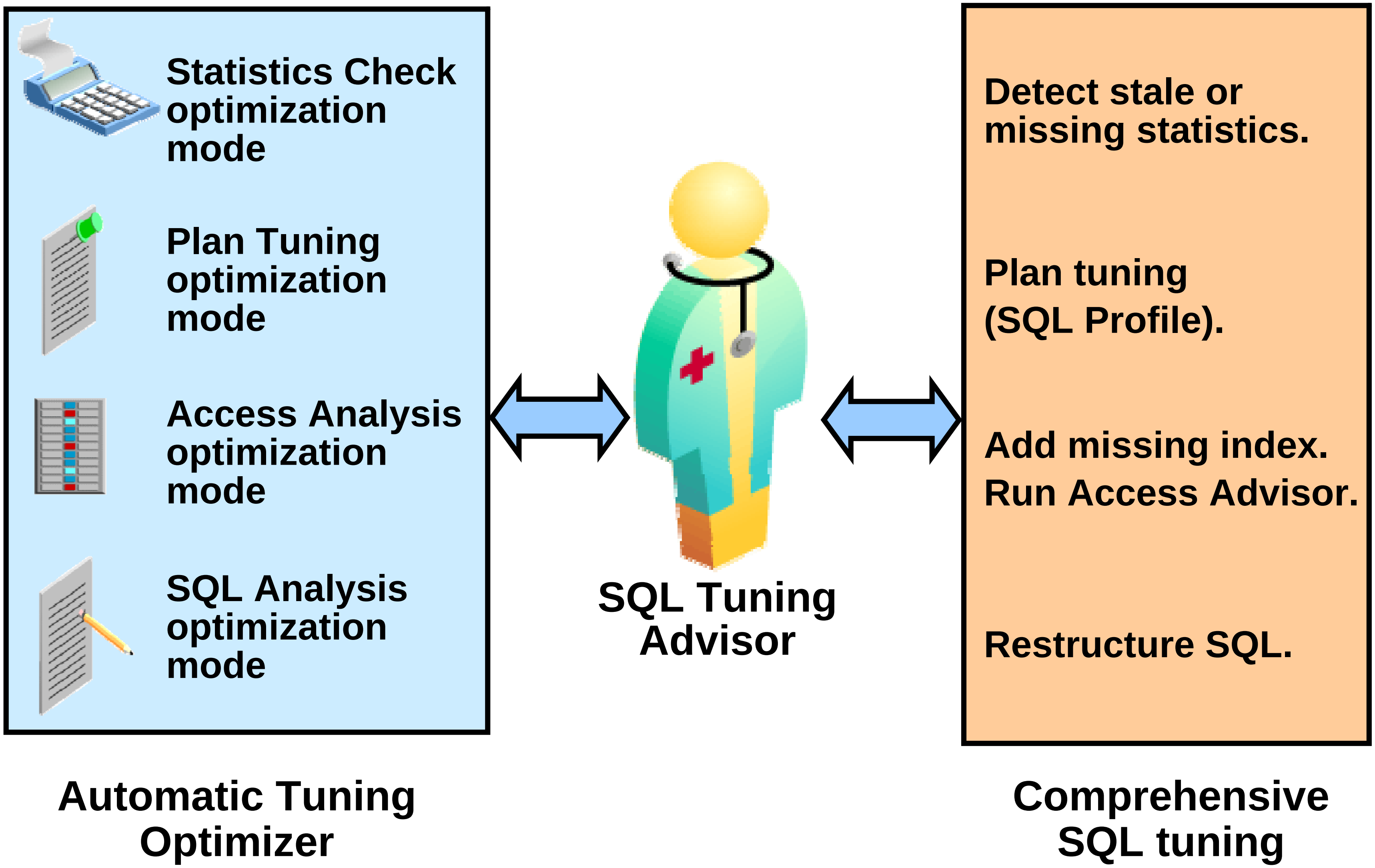
Tuning SQL Statements Automatically

- Tuning SQL statements automatically eases the entire SQL tuning process and replaces manual SQL tuning.
- Optimizer modes:
  - Normal mode
  - Tuning mode or Automatic Tuning Optimizer (ATO)
- SQL Tuning Advisor is used to access tuning mode.
- You should use tuning mode only for high-load SQL statements.

Application Tuning Challenges



SQL Tuning Advisor: Overview



Statistics Check optimization mode

Plan Tuning optimization mode

Access Analysis optimization mode

SQL Analysis optimization mode

Automatic Tuning Optimizer

SQL Tuning Advisor

Detect stale or missing statistics.

Plan tuning (SQL Profile).

Add missing index.
Run Access Advisor.

Restructure SQL.

Comprehensive SQL tuning

Stale or Missing Object Statistics

- Object statistics are key inputs to the optimizer.
- ATO verifies object statistics for each query object.
- ATO uses dynamic sampling and generates:
  - Auxiliary object statistics to compensate for missing or stale object statistics
  - Recommendations to gather object statistics where appropriate:

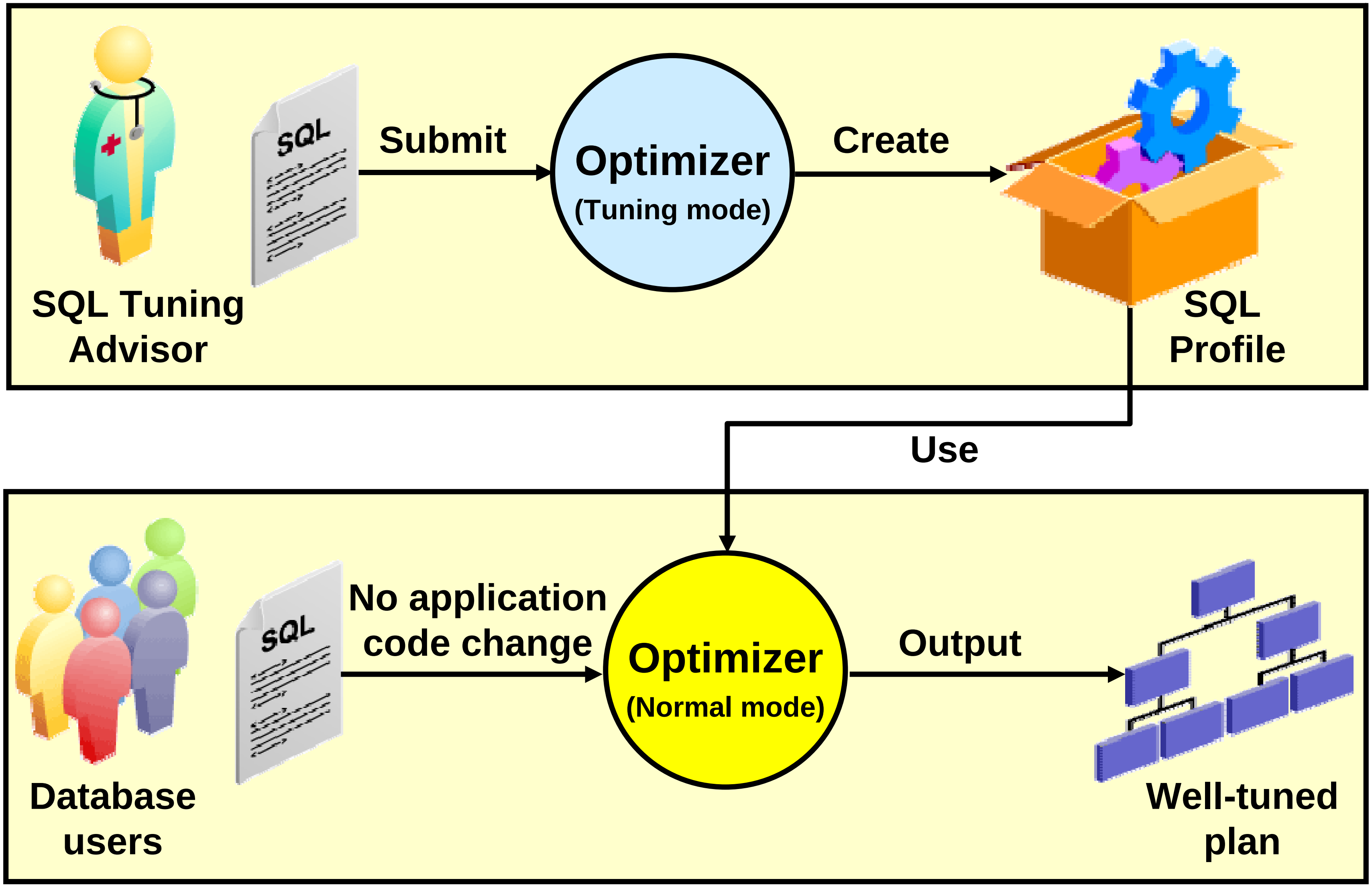
```
DBMS_STATS.GATHER_TABLE_STATS(
ownname=>'SH', tabname=>'CUSTOMERS',
estimate_percent=>DBMS_STATS.AUTO_SAMPLE_SIZE);
```

SQL Statement Profiling

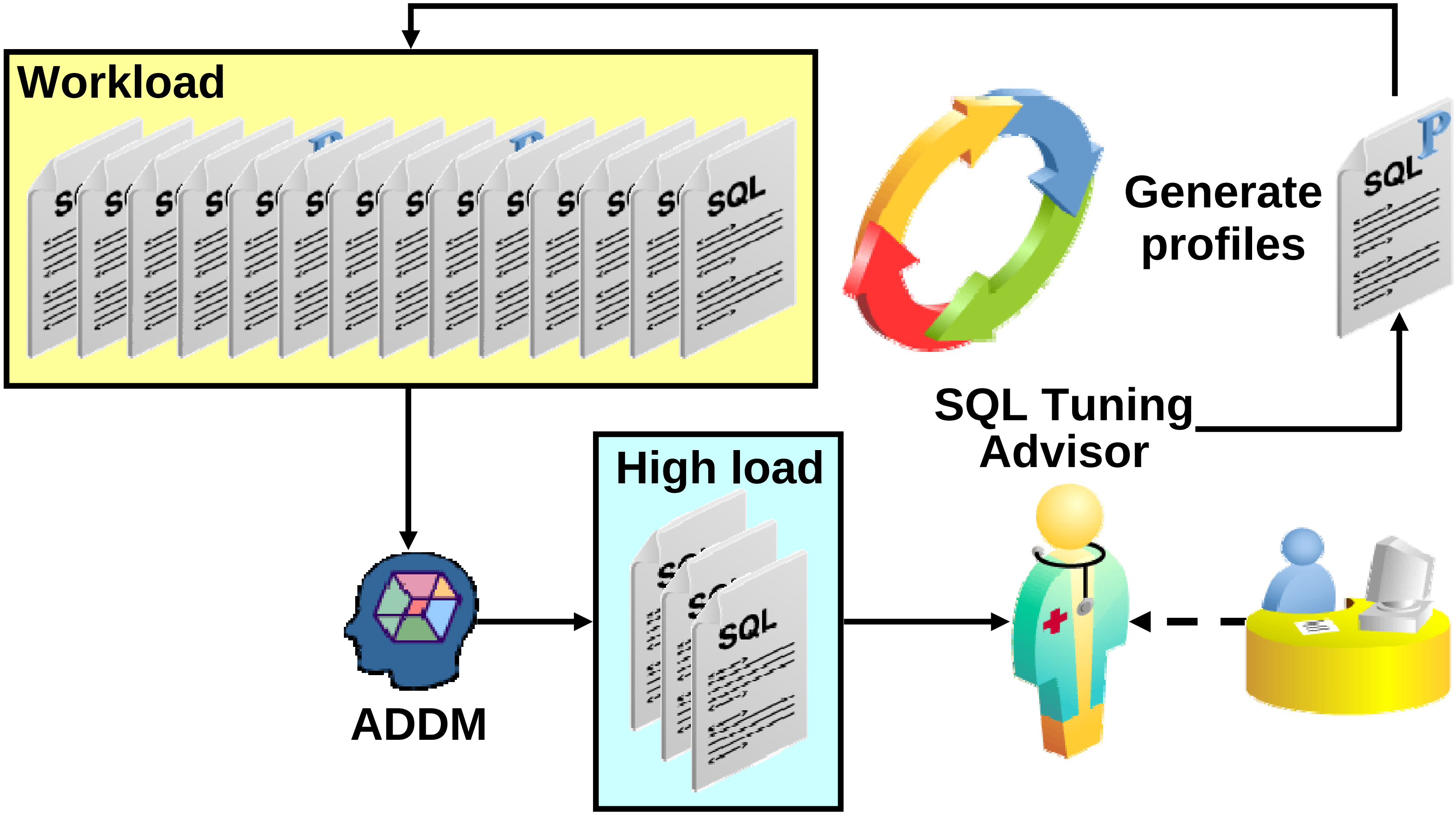
- Statement statistics are key inputs to the optimizer.
- ATO verifies statement statistics such as:
  - Predicate selectivity
  - Optimizer settings (`FIRST_ROWS` versus `ALL_ROWS`)
- Automatic Tuning Optimizer uses:
  - Dynamic sampling
  - Partial execution of the statement
  - Past execution history statistics of the statement
- ATO builds a profile if statistics were generated:

```
exec :profile_name :=                -
dbms_sqltune.accept_sql_profile( -
task_name =>'my_sql_tuning_task');
```

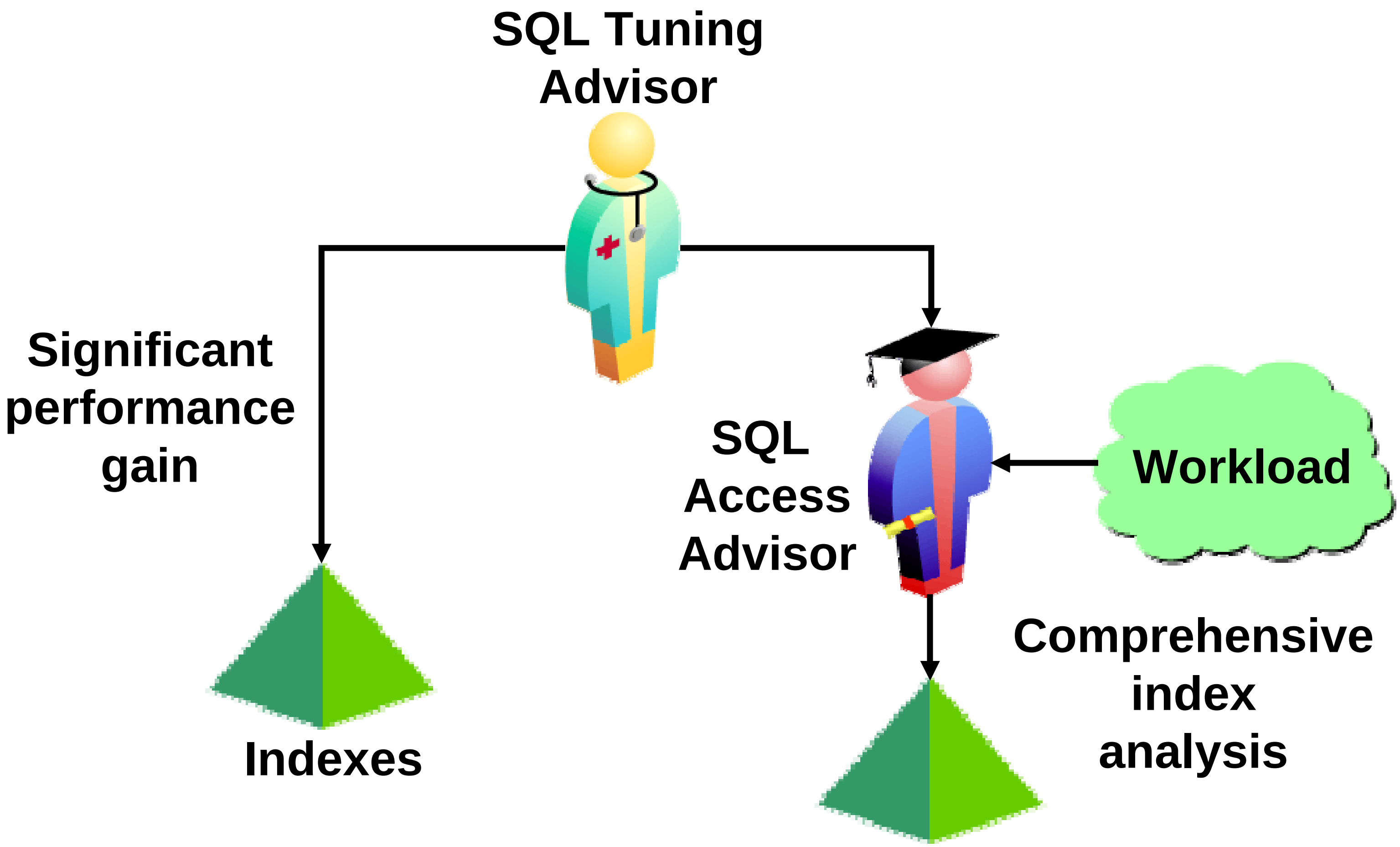
Plan Tuning Flow and SQL Profile Creation



SQL Tuning Loop

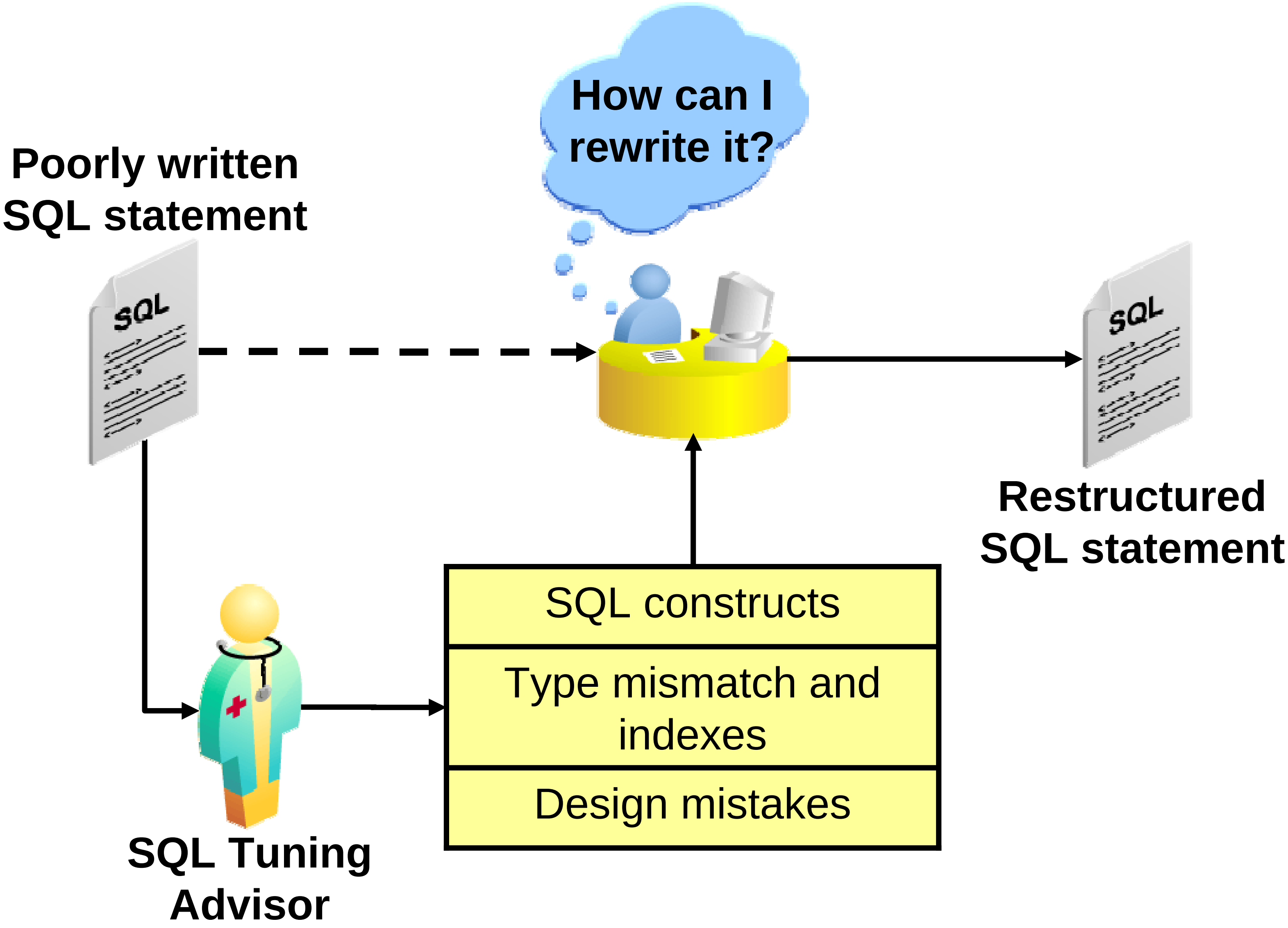


Access Path Analysis

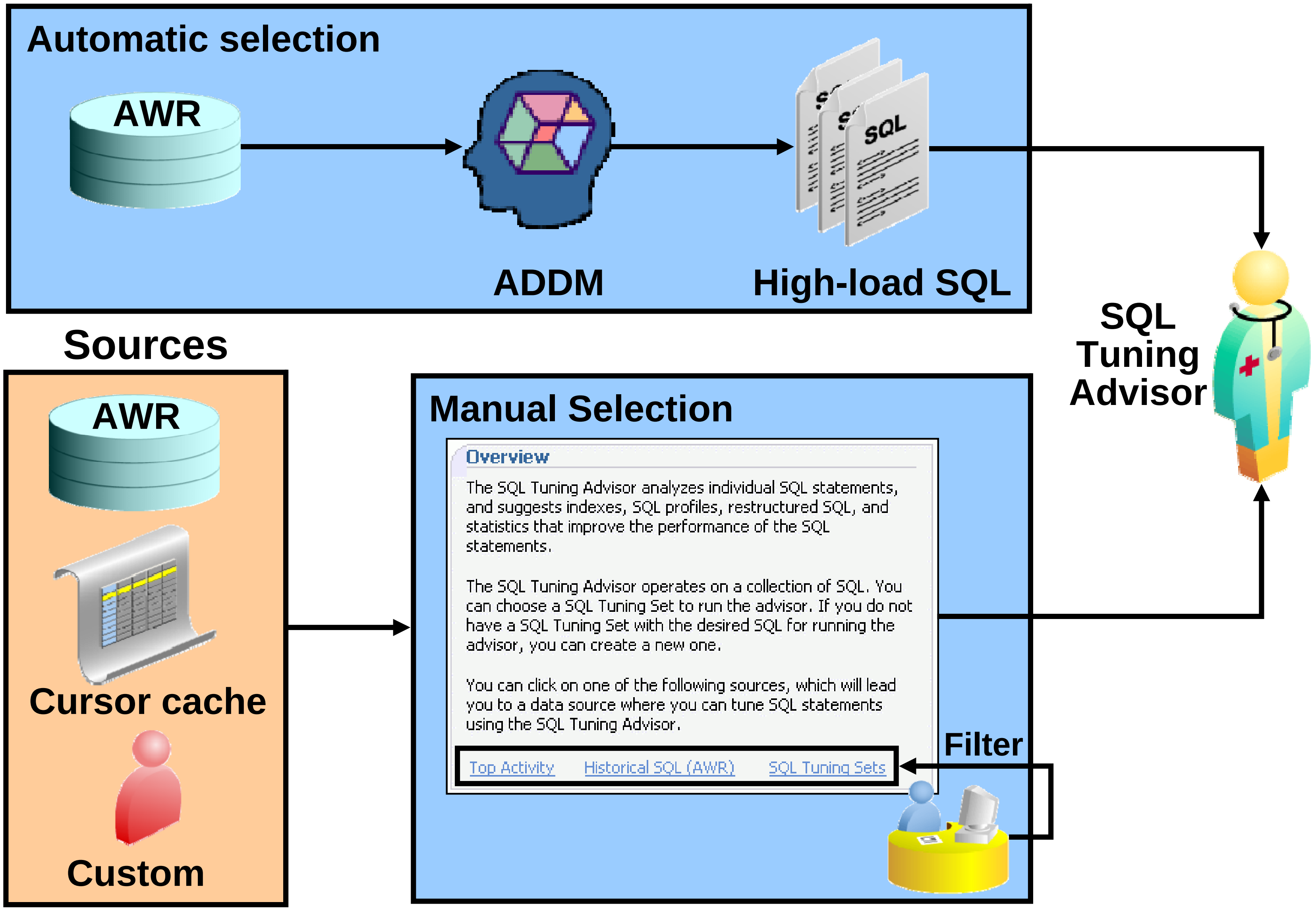


```
CREATE INDEX JFV.IDX$_00002 on JFV.TEST("C");
```

SQL Structure Analysis


How can I rewrite it?
Poorly written SQL statement
SQL
SQL
Restructured SQL statement
SQL constructs
Type mismatch and indexes
Design mistakes
SQL Tuning Advisor

SQL Tuning Advisor: Usage Model



Overview

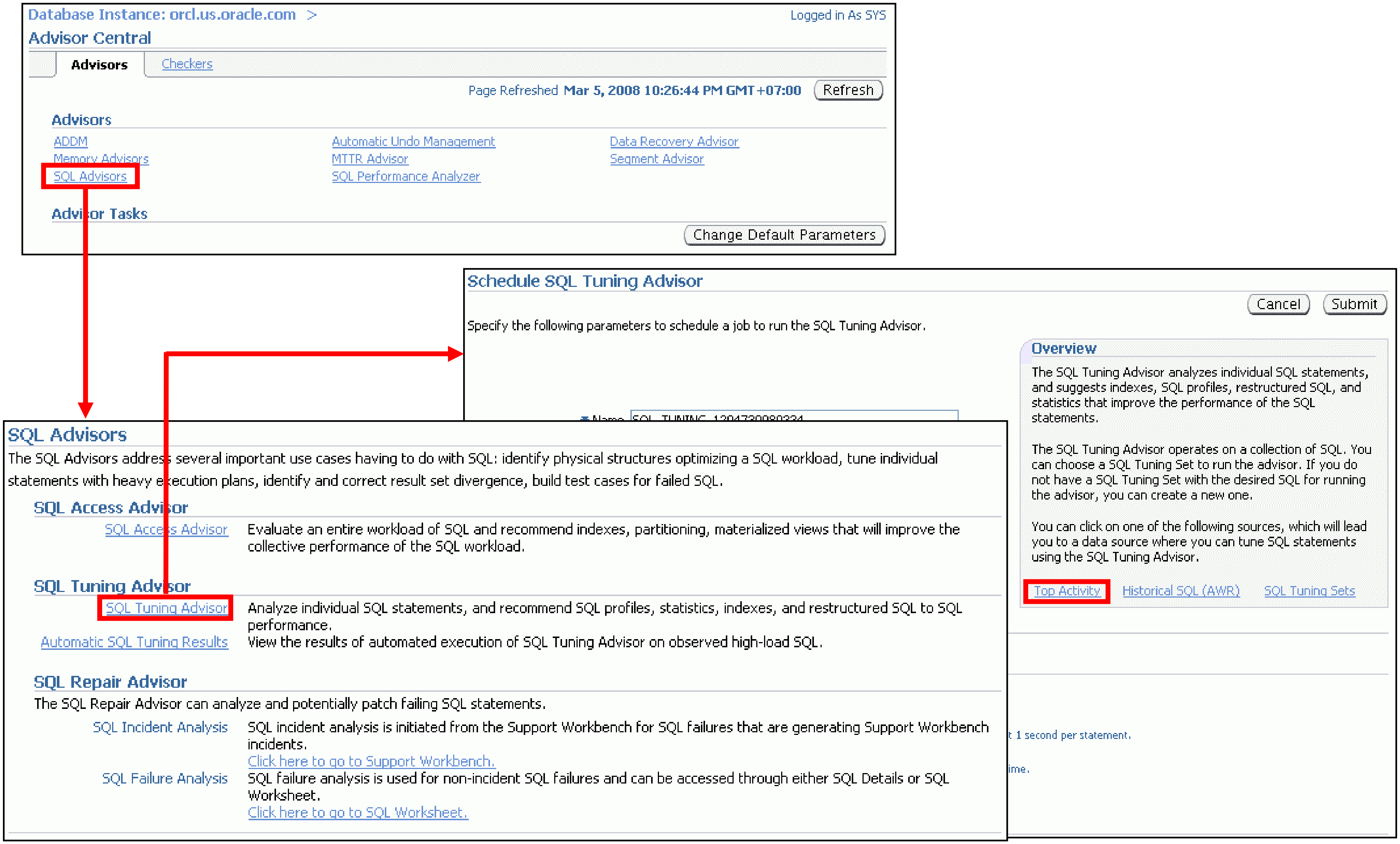
The SQL Tuning Advisor analyzes individual SQL statements, and suggests indexes, SQL profiles, restructured SQL, and statistics that improve the performance of the SQL statements.

The SQL Tuning Advisor operates on a collection of SQL. You can choose a SQL Tuning Set to run the advisor. If you do not have a SQL Tuning Set with the desired SQL for running the advisor, you can create a new one.

You can click on one of the following sources, which will lead you to a data source where you can tune SQL statements using the SQL Tuning Advisor.

Top Activity   Historical SQL (AWR)   SQL Tuning Sets

Database Control and SQL Tuning Advisor



Page Refreshed **Mar 5, 2008 10:26:44 PM GMT+07:00**

Schedule SQL Tuning Advisor

Specify the following parameters to schedule a job to run the SQL Tuning Advisor.

* Name SQL_TUNING_1204720080334

Overview

The SQL Tuning Advisor analyzes individual SQL statements, and suggests indexes, SQL profiles, restructured SQL, and statistics that improve the performance of the SQL statements.

The SQL Tuning Advisor operates on a collection of SQL. You can choose a SQL Tuning Set to run the advisor. If you do not have a SQL Tuning Set with the desired SQL for running the advisor, you can create a new one.

You can click on one of the following sources, which will lead you to a data source where you can tune SQL statements using the SQL Tuning Advisor.

t 1 second per statement.

ime.

SQL Advisors

The SQL Advisors address several important use cases having to do with SQL: identify physical structures optimizing a SQL workload, tune individual statements with heavy execution plans, identify and correct result set divergence, build test cases for failed SQL.

SQL Access Advisor

SQL Access Advisor — Evaluate an entire workload of SQL and recommend indexes, partitioning, materialized views that will improve the collective performance of the SQL workload.

SQL Tuning Advisor

SQL Tuning Advisor — Analyze individual SQL statements, and recommend SQL profiles, statistics, indexes, and restructured SQL to SQL performance.

Automatic SQL Tuning Results — View the results of automated execution of SQL Tuning Advisor on observed high-load SQL.

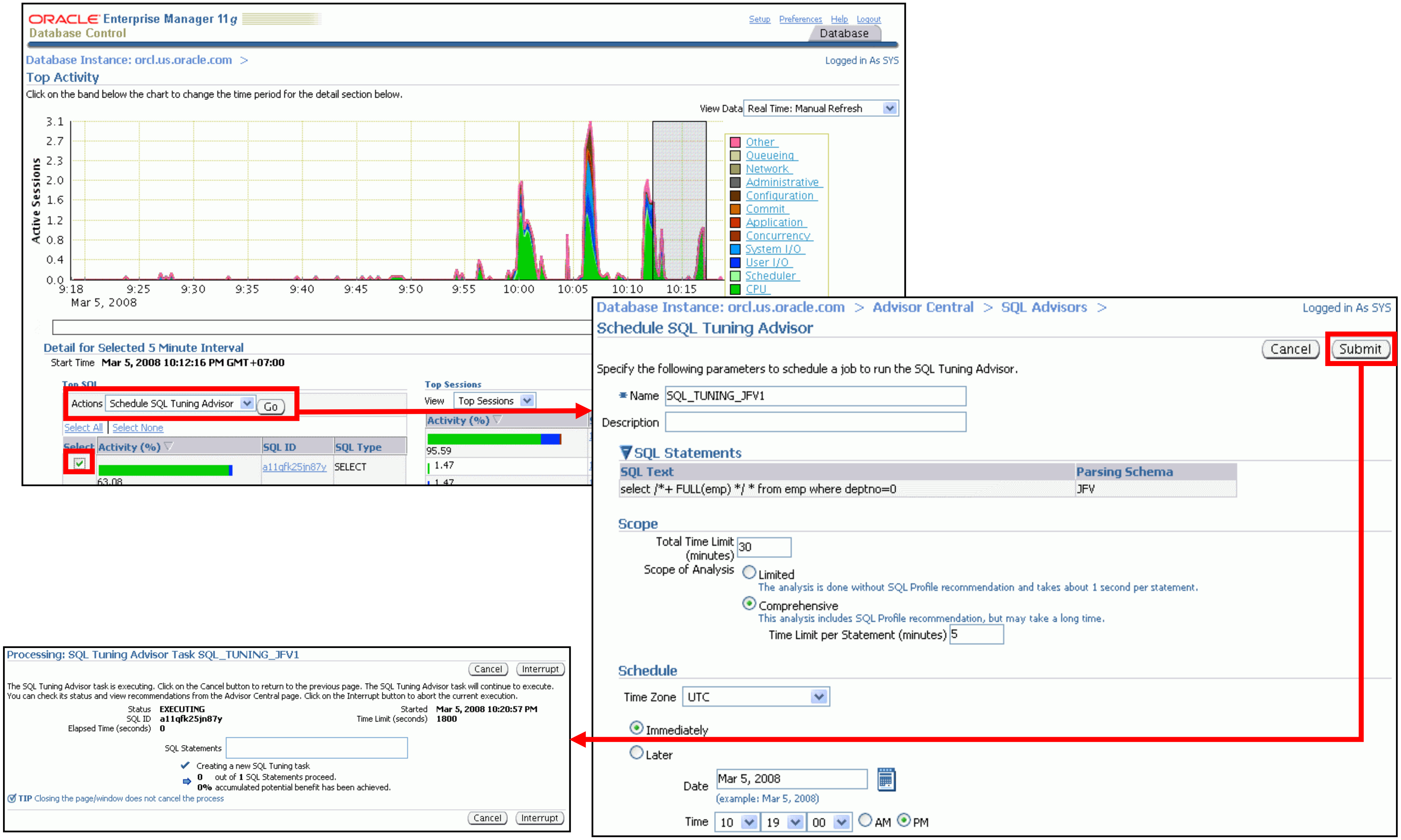
SQL Repair Advisor

The SQL Repair Advisor can analyze and potentially patch failing SQL statements.

SQL Incident Analysis — SQL incident analysis is initiated from the Support Workbench for SQL failures that are generating Support Workbench incidents.
Click here to go to Support Workbench.

SQL Failure Analysis — SQL failure analysis is used for non-incident SQL failures and can be accessed through either SQL Details or SQL Worksheet.
Click here to go to SQL Worksheet.

Running SQL Tuning Advisor: Example



Database Instance: orcl.us.oracle.com >

Logged in As SYS

Top Activity

Click on the band below the chart to change the time period for the detail section below.

View Data Real Time: Manual Refresh

Detail for Selected 5 Minute Interval

Start Time **Mar 5, 2008 10:12:16 PM GMT+07:00**

Top SQL

Actions Schedule SQL Tuning Advisor

Select	Activity (%)	SQL ID	SQL Type
☑	63.08	a11qfk25jn87y	SELECT

Top Sessions

View Top Sessions

Activity (%)
95.59
1.47
1.47

Database Instance: orcl.us.oracle.com > Advisor Central > SQL Advisors >

Logged in As SYS

Schedule SQL Tuning Advisor

Specify the following parameters to schedule a job to run the SQL Tuning Advisor.

* Name SQL_TUNING_JFV1

Description

SQL Statements

SQL Text	Parsing Schema
select /*+ FULL(emp) */ * from emp where deptno=0	JFV

Scope

Total Time Limit (minutes) 30

Scope of Analysis ○ Limited

The analysis is done without SQL Profile recommendation and takes about 1 second per statement.

● Comprehensive

This analysis includes SQL Profile recommendation, but may take a long time.

Time Limit per Statement (minutes) 5

Schedule

Time Zone UTC

● Immediately

○ Later

Date Mar 5, 2008 (example: Mar 5, 2008)

Time 10 19 00 ○ AM ● PM

Processing: SQL Tuning Advisor Task SQL_TUNING_JFV1

The SQL Tuning Advisor task is executing. Click on the Cancel button to return to the previous page. The SQL Tuning Advisor task will continue to execute. You can check its status and view recommendations from the Advisor Central page. Click on the Interrupt button to abort the current execution.

Status **EXECUTING**
SQL ID **a11qfk25jn87y**
Elapsed Time (seconds) **0**
Started **Mar 5, 2008 10:20:57 PM**
Time Limit (seconds) **1800**

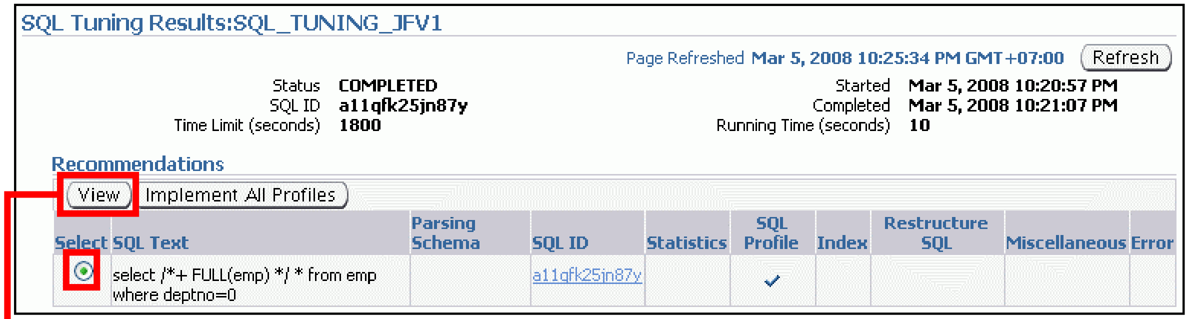
SQL Statements

✓ Creating a new SQL Tuning task

0 out of **1** SQL Statements proceed.
0% accumulated potential benefit has been achieved.

TIP Closing the page/window does not cancel the process

Implementing Recommendations



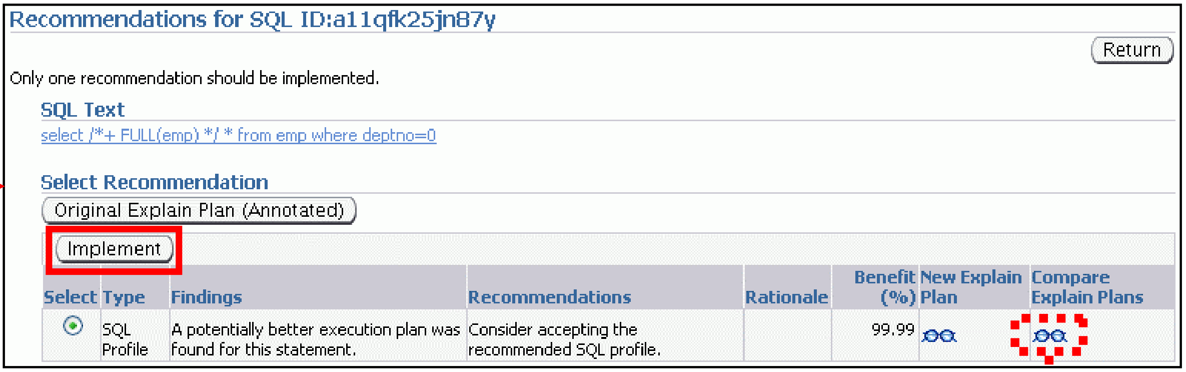
SQL Tuning Results:SQL_TUNING_JFV1

Page Refreshed **Mar 5, 2008 10:25:34 PM GMT+07:00**

Status	**COMPLETED**	Started	**Mar 5, 2008 10:20:57 PM**
SQL ID	**a11qfk25jn87y**	Completed	**Mar 5, 2008 10:21:07 PM**
Time Limit (seconds)	**1800**	Running Time (seconds)	**10**

Recommendations

Select	SQL Text	Parsing Schema	SQL ID	Statistics	SQL Profile	Index	Restructure SQL	Miscellaneous	Error
●	select /*+ FULL(emp) */ * from emp where deptno=0		a11qfk25jn87y		✓				



Recommendations for SQL ID:a11qfk25jn87y

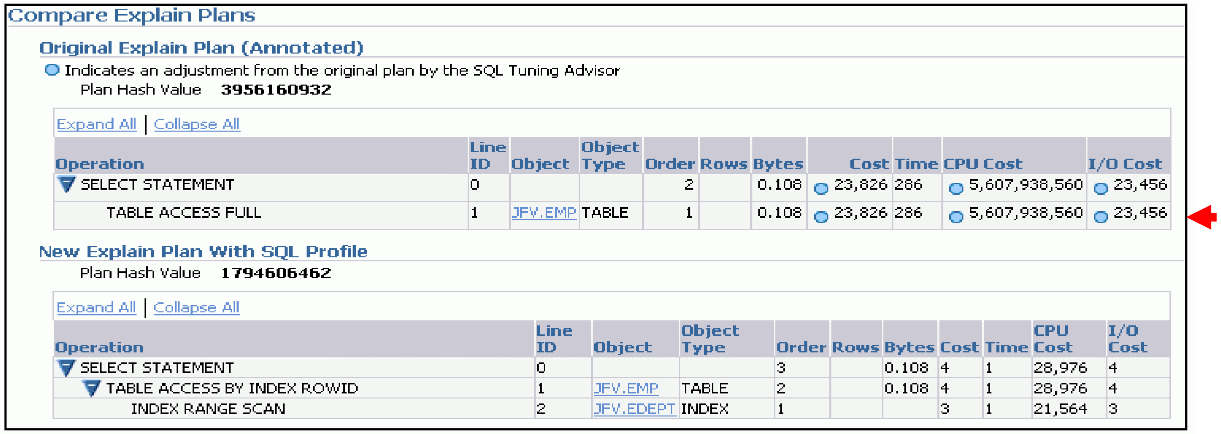
Only one recommendation should be implemented.

SQL Text

select /*+ FULL(emp) */ * from emp where deptno=0

Select Recommendation

Select	Type	Findings	Recommendations	Rationale	Benefit (%)	New Explain Plan	Compare Explain Plans
●	SQL Profile	A potentially better execution plan was found for this statement.	Consider accepting the recommended SQL profile.		99.99	👓	👓



Compare Explain Plans

Original Explain Plan (Annotated)

○ Indicates an adjustment from the original plan by the SQL Tuning Advisor

Plan Hash Value **3956160932**

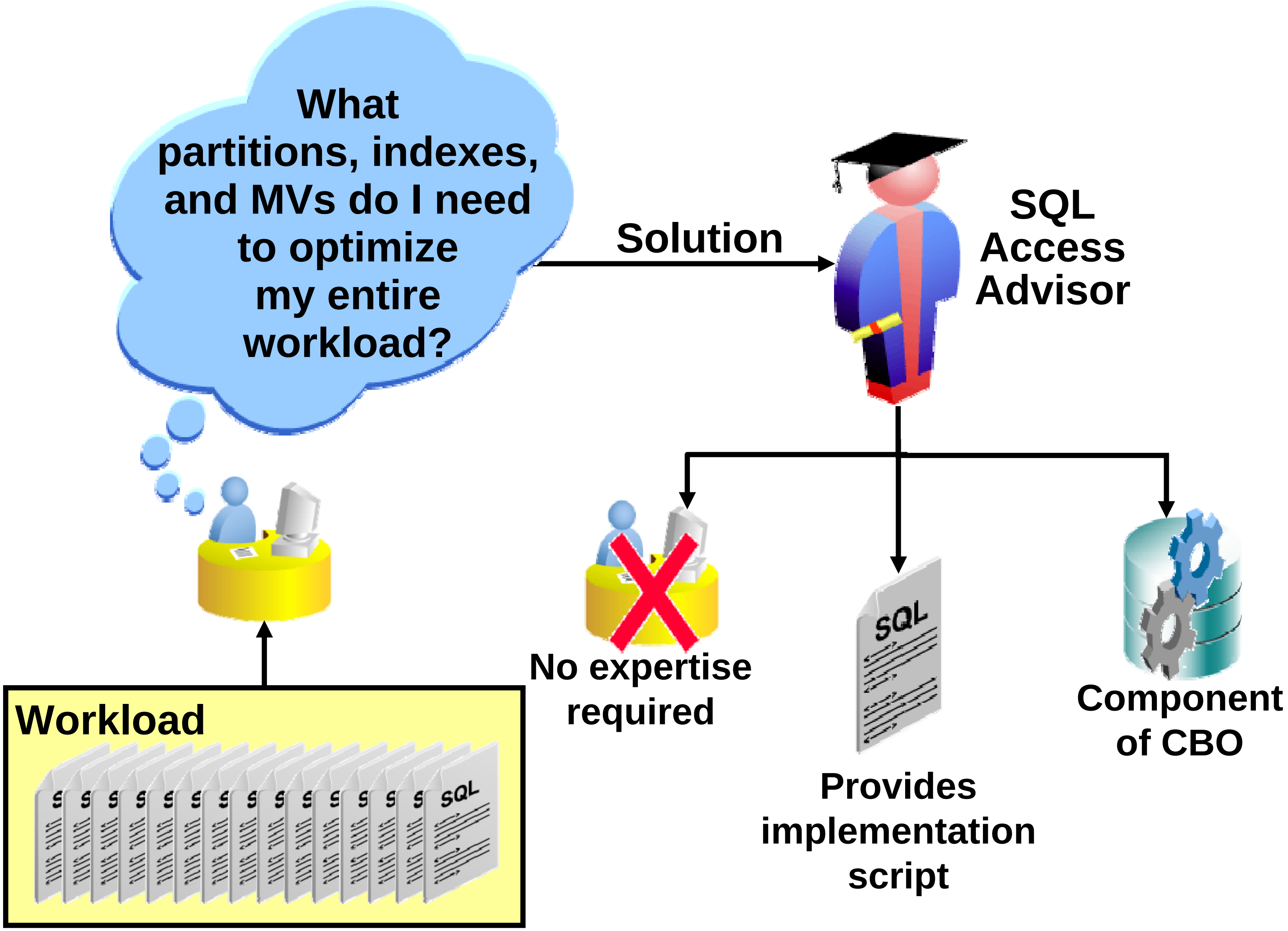
Operation	Line ID	Object	Object Type	Order	Rows	Bytes	Cost	Time	CPU Cost	I/O Cost
▼ SELECT STATEMENT	0			2		0.108	○ 23,826	286	○ 5,607,938,560	○ 23,456
TABLE ACCESS FULL	1	JFV.EMP	TABLE	1		0.108	○ 23,826	286	○ 5,607,938,560	○ 23,456

New Explain Plan With SQL Profile

Plan Hash Value **1794606462**

Operation	Line ID	Object	Object Type	Order	Rows	Bytes	Cost	Time	CPU Cost	I/O Cost
▼ SELECT STATEMENT	0			3		0.108	4	1	28,976	4
▼ TABLE ACCESS BY INDEX ROWID	1	JFV.EMP	TABLE	2		0.108	4	1	28,976	4
INDEX RANGE SCAN	2	JFV.EDEPT	INDEX	1			3	1	21,564	3

SQL Access Advisor: Overview



What partitions, indexes, and MVs do I need to optimize my entire workload?

Solution

SQL Access Advisor

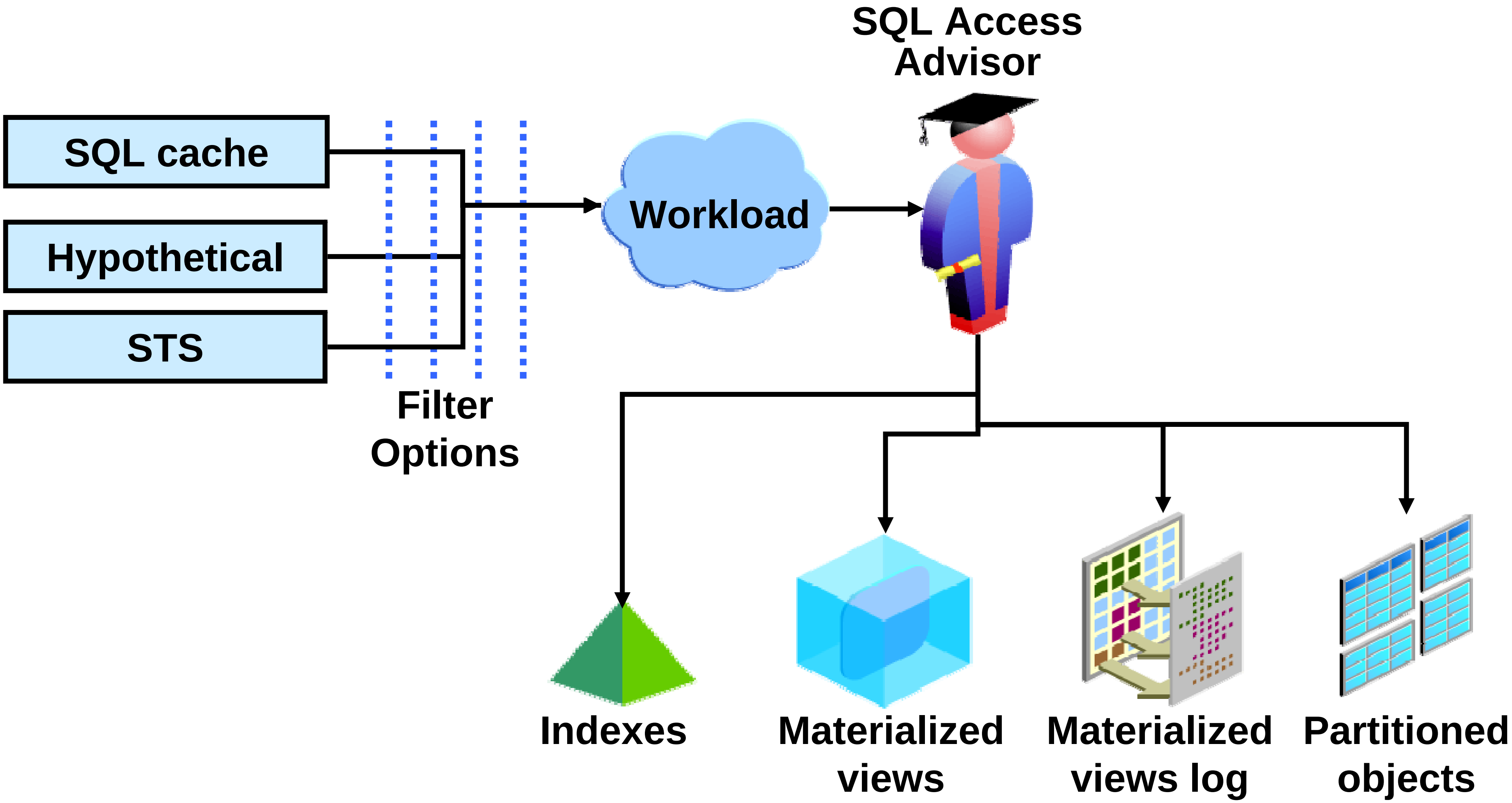
No expertise required

Provides implementation script

Component of CBO

Workload

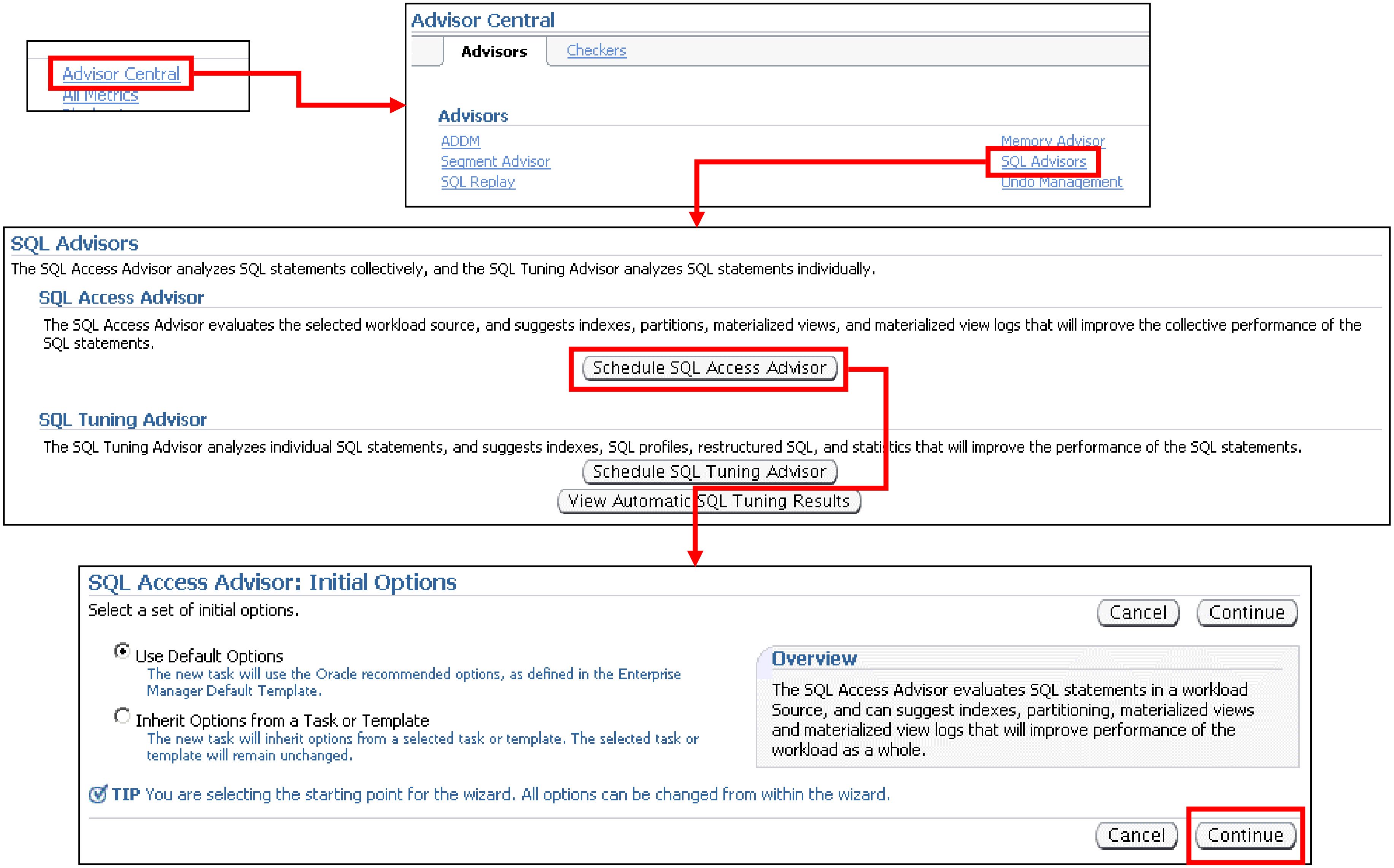
SQL Access Advisor: Usage Model



Possible Recommendations

Recommendation	Comprehensive	Limited
Add new (partitioned) index on table or materialized view.	YES	YES
Drop an unused index.	YES	NO
Modify an existing index by changing the index type.	YES	NO
Modify an existing index by adding columns at the end.	YES	YES
Add a new (partitioned) materialized view.	YES	YES
Drop an unused materialized view (log).	YES	NO
Add a new materialized view log.	YES	YES
Modify an existing materialized view log to add new columns or clauses.	YES	YES
Partition an existing unpartitioned table or index.	YES	YES

SQL Access Advisor Session: Initial Options



Advisor Central
All Metrics

Advisor Central

Advisors | Checkers

Advisors

- ADDM
- Segment Advisor
- SQL Replay
- Memory Advisor
- SQL Advisors
- Undo Management

SQL Advisors

The SQL Access Advisor analyzes SQL statements collectively, and the SQL Tuning Advisor analyzes SQL statements individually.

SQL Access Advisor

The SQL Access Advisor evaluates the selected workload source, and suggests indexes, partitions, materialized views, and materialized view logs that will improve the collective performance of the SQL statements.

SQL Tuning Advisor

The SQL Tuning Advisor analyzes individual SQL statements, and suggests indexes, SQL profiles, restructured SQL, and statistics that will improve the performance of the SQL statements.

Schedule SQL Tuning Advisor
View Automatic SQL Tuning Results

SQL Access Advisor: Initial Options

Select a set of initial options.

● Use Default Options
The new task will use the Oracle recommended options, as defined in the Enterprise Manager Default Template.

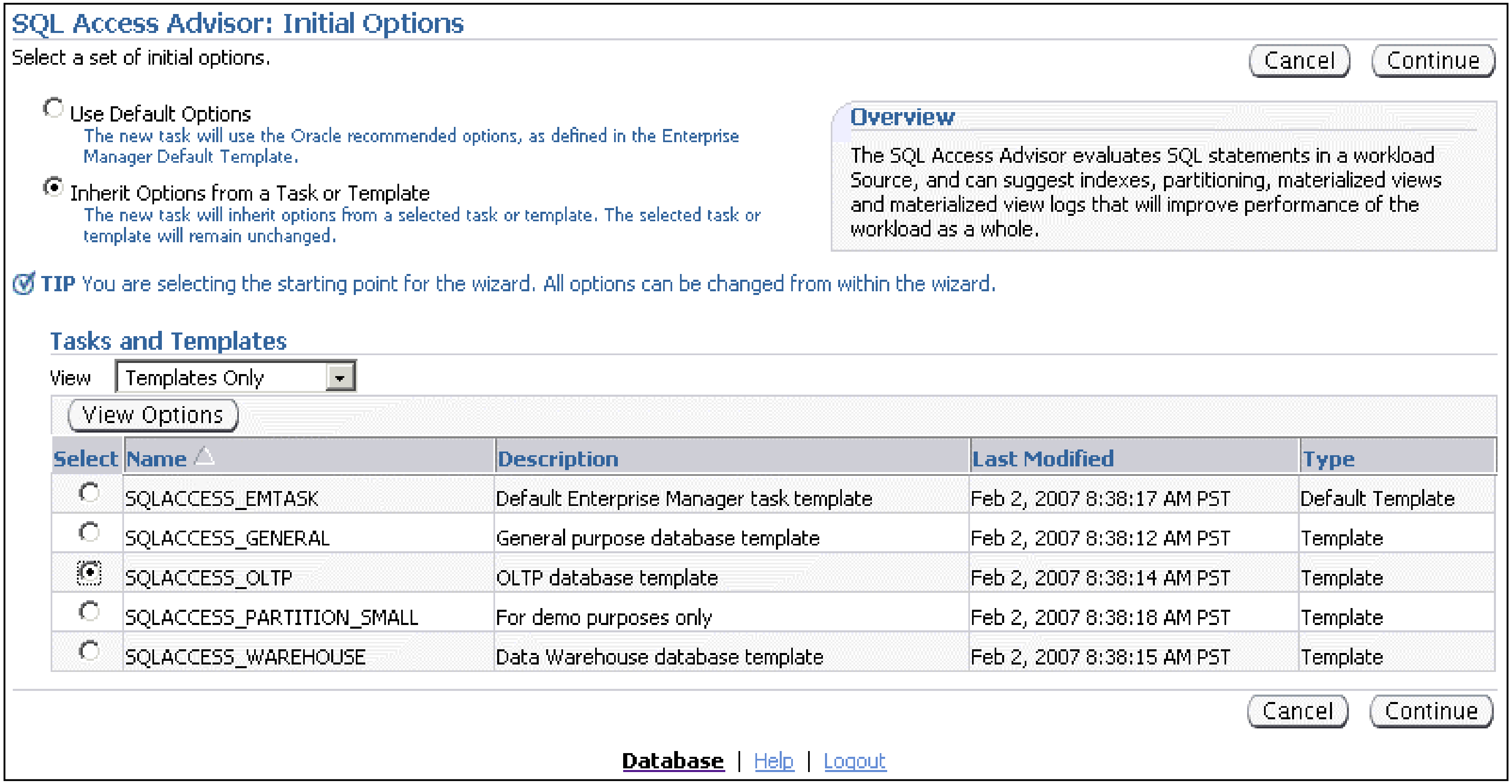
○ Inherit Options from a Task or Template
The new task will inherit options from a selected task or template. The selected task or template will remain unchanged.

Overview

The SQL Access Advisor evaluates SQL statements in a workload Source, and can suggest indexes, partitioning, materialized views and materialized view logs that will improve performance of the workload as a whole.

✓ **TIP** You are selecting the starting point for the wizard. All options can be changed from within the wizard.

SQL Access Advisor Session: Initial Options



SQL Access Advisor: Initial Options

Select a set of initial options.

○ Use Default Options
The new task will use the Oracle recommended options, as defined in the Enterprise Manager Default Template.

● Inherit Options from a Task or Template
The new task will inherit options from a selected task or template. The selected task or template will remain unchanged.

Overview

The SQL Access Advisor evaluates SQL statements in a workload Source, and can suggest indexes, partitioning, materialized views and materialized view logs that will improve performance of the workload as a whole.

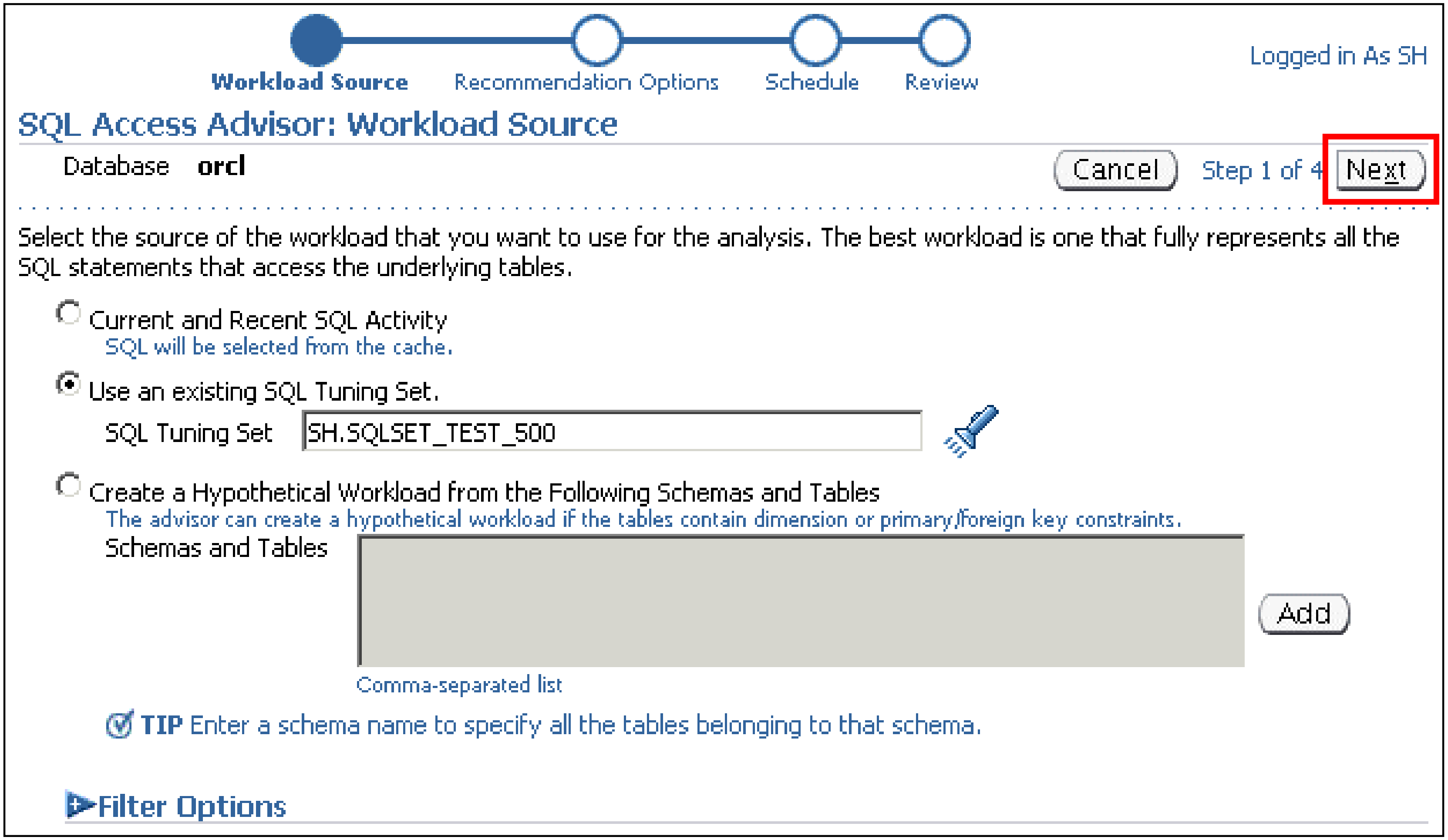
TIP You are selecting the starting point for the wizard. All options can be changed from within the wizard.

Tasks and Templates

View: Templates Only

Select	Name	Description	Last Modified	Type
○	SQLACCESS_EMTASK	Default Enterprise Manager task template	Feb 2, 2007 8:38:17 AM PST	Default Template
○	SQLACCESS_GENERAL	General purpose database template	Feb 2, 2007 8:38:12 AM PST	Template
●	SQLACCESS_OLTP	OLTP database template	Feb 2, 2007 8:38:14 AM PST	Template
○	SQLACCESS_PARTITION_SMALL	For demo purposes only	Feb 2, 2007 8:38:18 AM PST	Template
○	SQLACCESS_WAREHOUSE	Data Warehouse database template	Feb 2, 2007 8:38:15 AM PST	Template

SQL Access Advisor: Workload Source



SQL Access Advisor: Workload Source

Database **orcl**

Select the source of the workload that you want to use for the analysis. The best workload is one that fully represents all the SQL statements that access the underlying tables.

○ Current and Recent SQL Activity
SQL will be selected from the cache.

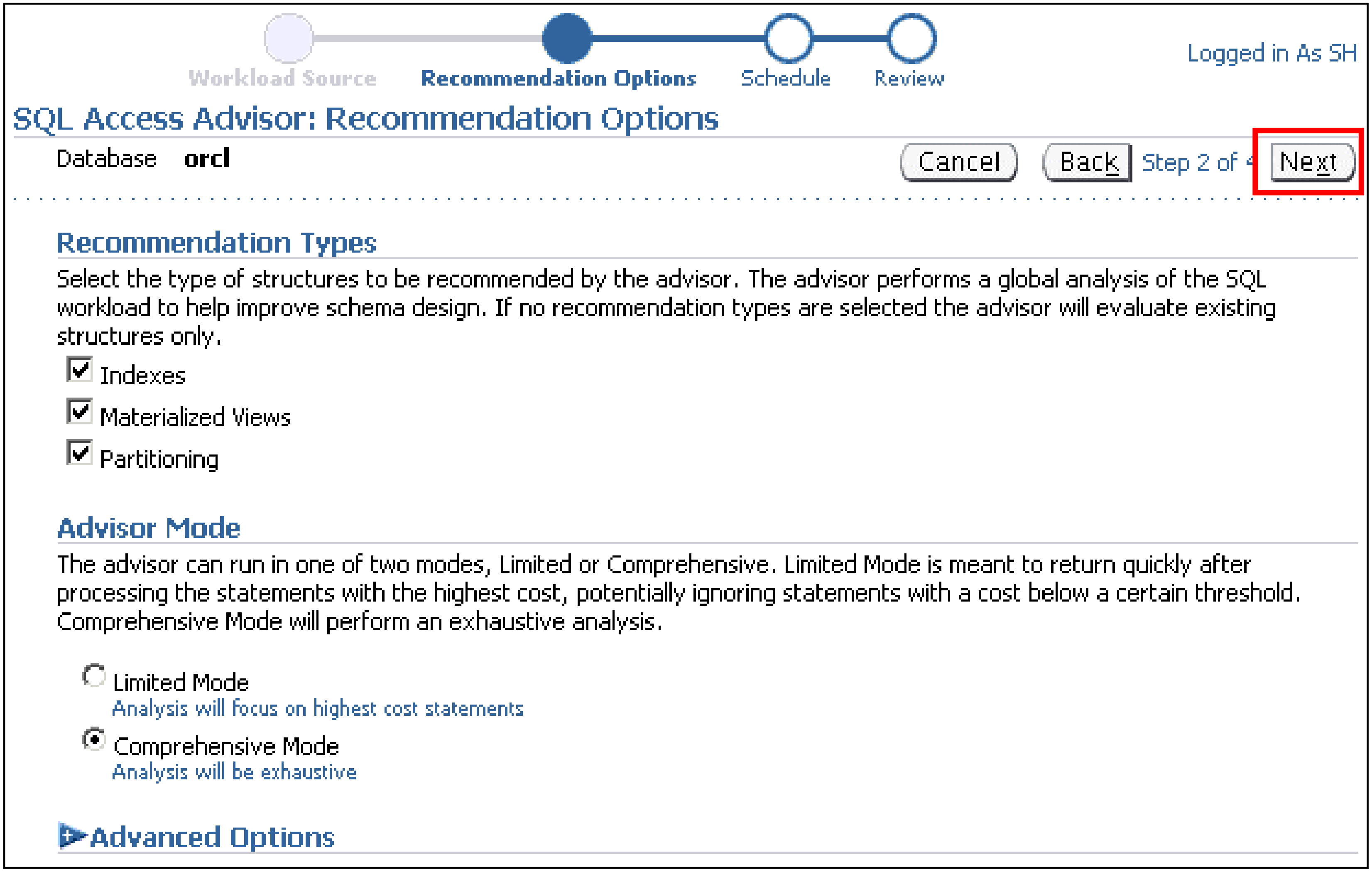
● Use an existing SQL Tuning Set.
SQL Tuning Set SH.SQLSET_TEST_500

○ Create a Hypothetical Workload from the Following Schemas and Tables
The advisor can create a hypothetical workload if the tables contain dimension or primary/foreign key constraints.
Schemas and Tables
Comma-separated list

TIP Enter a schema name to specify all the tables belonging to that schema.

▶Filter Options

SQL Access Advisor: Recommendation Options



SQL Access Advisor: Recommendation Options

Database **orcl**

Recommendation Types

Select the type of structures to be recommended by the advisor. The advisor performs a global analysis of the SQL workload to help improve schema design. If no recommendation types are selected the advisor will evaluate existing structures only.

☑ Indexes

☑ Materialized Views

☑ Partitioning

Advisor Mode

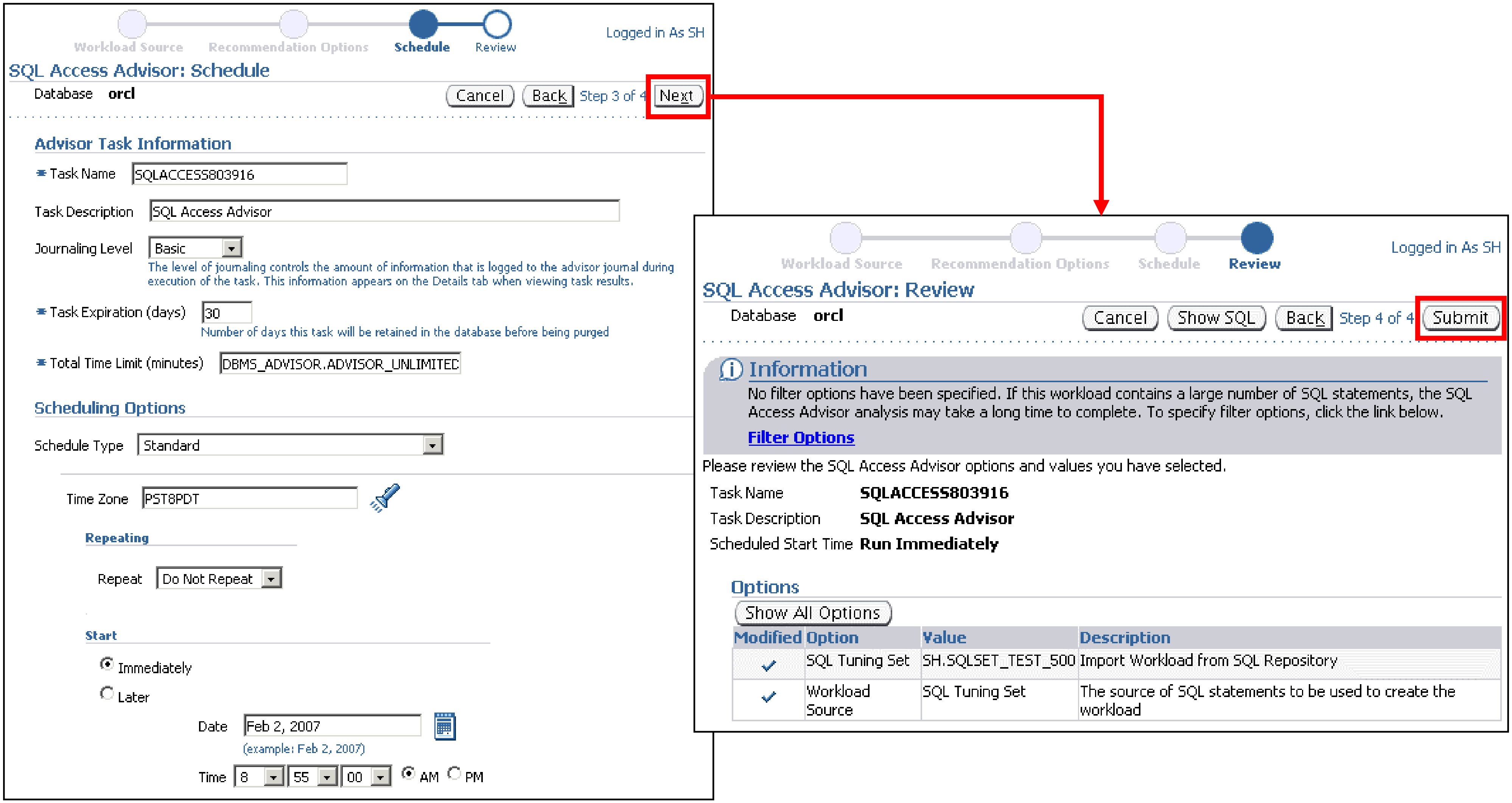
The advisor can run in one of two modes, Limited or Comprehensive. Limited Mode is meant to return quickly after processing the statements with the highest cost, potentially ignoring statements with a cost below a certain threshold. Comprehensive Mode will perform an exhaustive analysis.

○ Limited Mode
Analysis will focus on highest cost statements

● Comprehensive Mode
Analysis will be exhaustive

▶Advanced Options

SQL Access Advisor: Schedule and Review



SQL Access Advisor: Schedule

Database **orcl**

Advisor Task Information

* Task Name	SQLACCESS803916
Task Description	SQL Access Advisor
Journaling Level	Basic

The level of journaling controls the amount of information that is logged to the advisor journal during execution of the task. This information appears on the Details tab when viewing task results.

* Task Expiration (days)	30

Number of days this task will be retained in the database before being purged

* Total Time Limit (minutes)	DBMS_ADVISOR.ADVISOR_UNLIMITED

Scheduling Options

Schedule Type	Standard
Time Zone	PST8PDT

Repeating

Repeat	Do Not Repeat

Start

● Immediately

○ Later

Date	Feb 2, 2007

(example: Feb 2, 2007)

Time	8 : 55 : 00 ● AM ○ PM

SQL Access Advisor: Review

Database **orcl**

Information

No filter options have been specified. If this workload contains a large number of SQL statements, the SQL Access Advisor analysis may take a long time to complete. To specify filter options, click the link below.

Filter Options

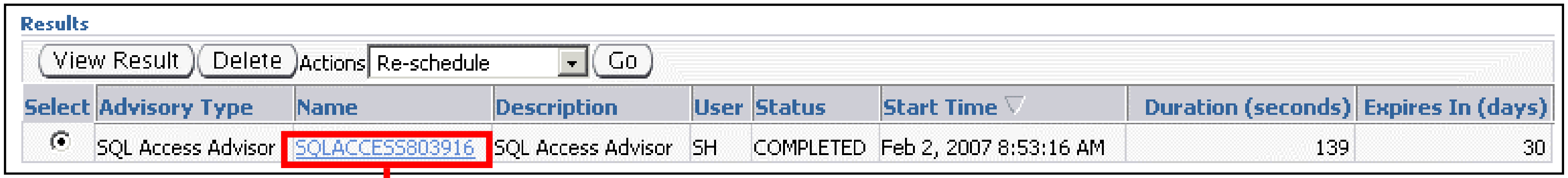
Please review the SQL Access Advisor options and values you have selected.

Task Name	**SQLACCESS803916**
Task Description	**SQL Access Advisor**
Scheduled Start Time	**Run Immediately**

Options

Modified	Option	Value	Description
✓	SQL Tuning Set	SH.SQLSET_TEST_500	Import Workload from SQL Repository
✓	Workload Source	SQL Tuning Set	The source of SQL statements to be used to create the workload

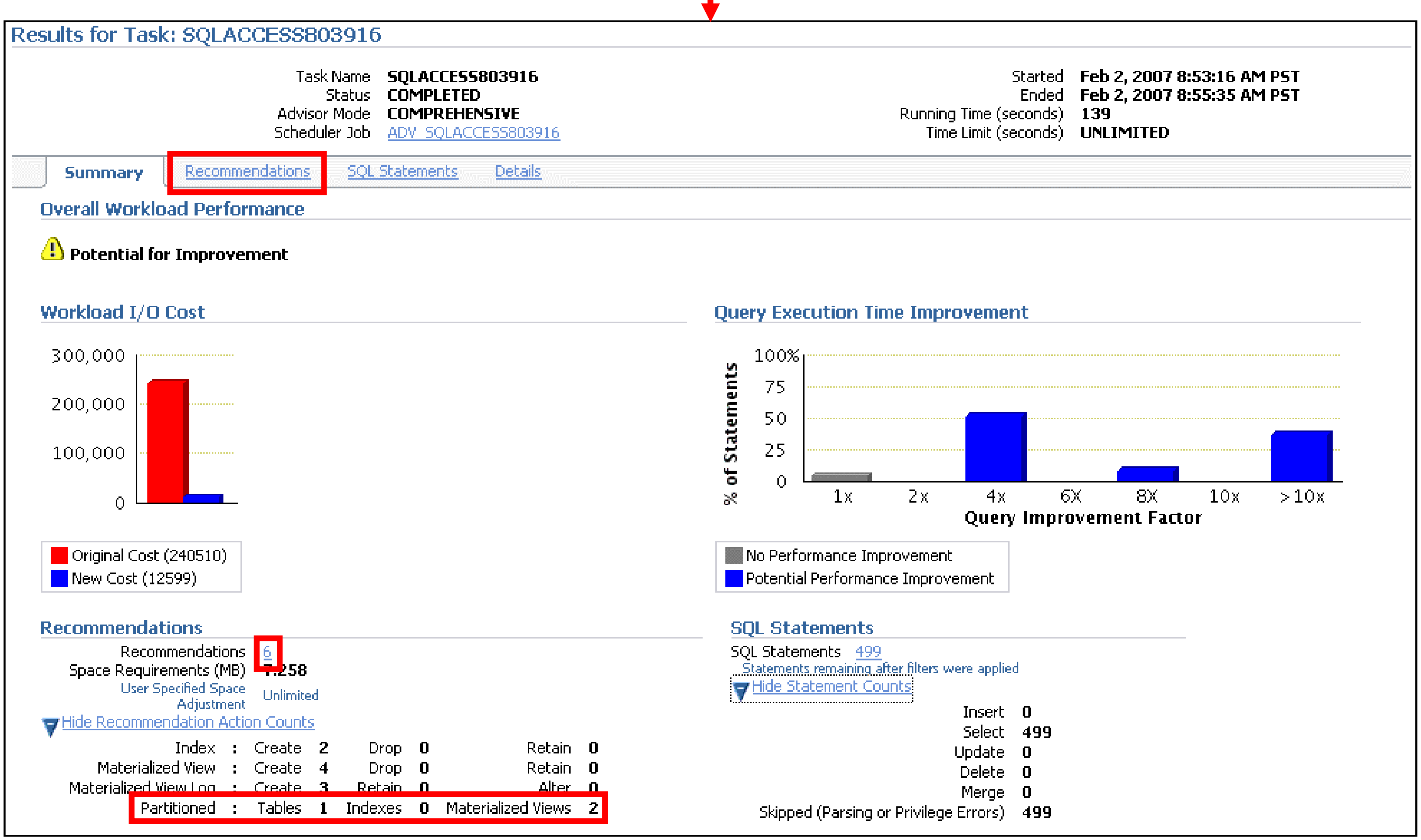
SQL Access Advisor: Results



Results

View Result | Delete | Actions Re-schedule | Go

Select	Advisory Type	Name	Description	User	Status	Start Time ▽	Duration (seconds)	Expires In (days)
●	SQL Access Advisor	SQLACCESS803916	SQL Access Advisor	SH	COMPLETED	Feb 2, 2007 8:53:16 AM	139	30



Results for Task: SQLACCESS803916

Task Name	**SQLACCESS803916**	Started	**Feb 2, 2007 8:53:16 AM PST**
Status	**COMPLETED**	Ended	**Feb 2, 2007 8:55:35 AM PST**
Advisor Mode	**COMPREHENSIVE**	Running Time (seconds)	**139**
Scheduler Job	ADV_SQLACCESS803916	Time Limit (seconds)	**UNLIMITED**

Summary | Recommendations | SQL Statements | Details

Overall Workload Performance

⚠ **Potential for Improvement**

Workload I/O Cost

🟥 Original Cost (240510)
🟦 New Cost (12599)

Query Execution Time Improvement

▨ No Performance Improvement
🟦 Potential Performance Improvement

Recommendations

Recommendations	6
Space Requirements (MB)	1.258
User Specified Space Adjustment	Unlimited

▼ Hide Recommendation Action Counts

Index	:	Create 2		Drop 0		Retain 0		
Materialized View	:	Create 4		Drop 0		Retain 0		
Materialized View Log	:	Create 3		Retain 0		Alter 0		
Partitioned	:	Tables 1		Indexes 0		Materialized Views 2		

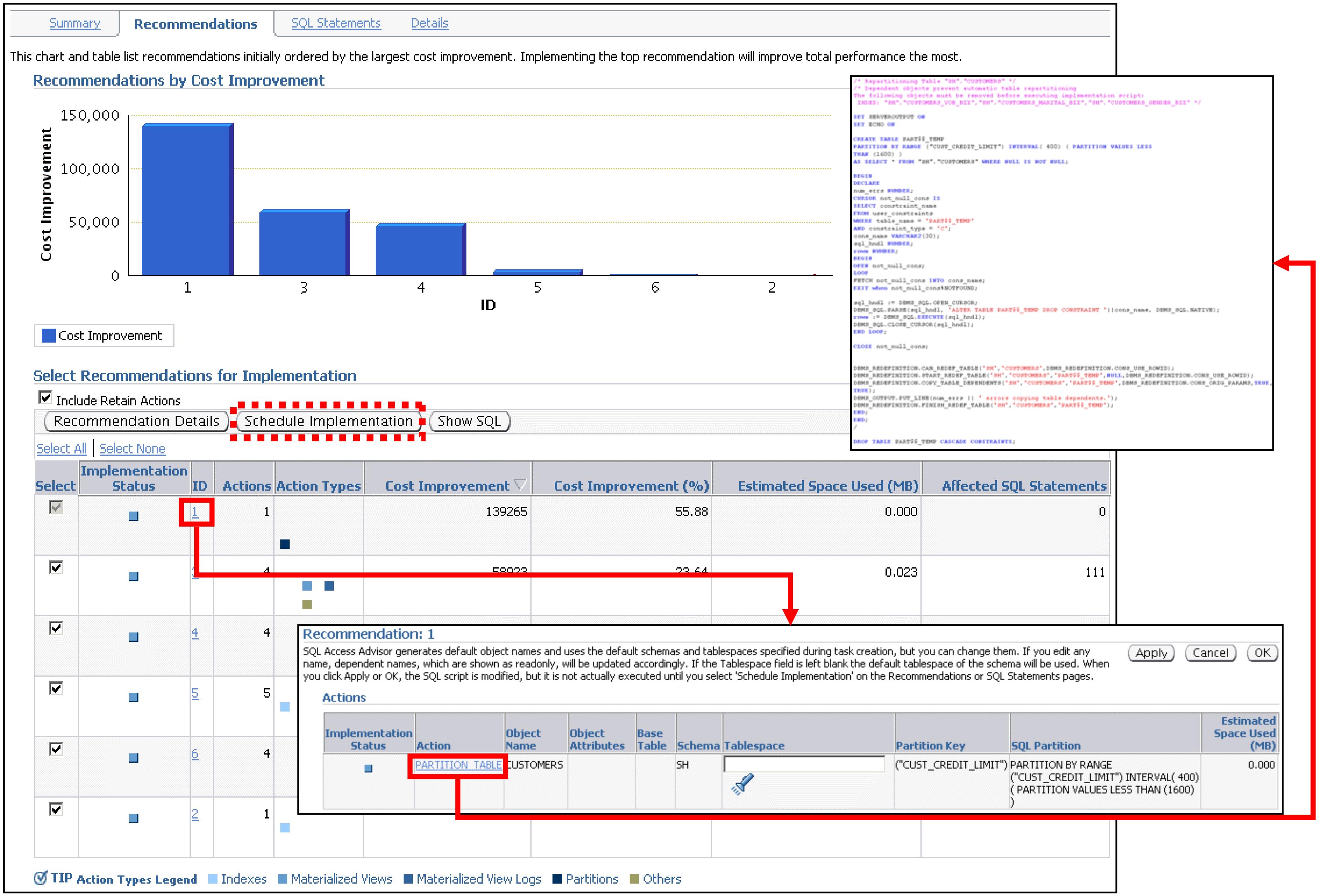
SQL Statements

SQL Statements 499
Statements remaining after filters were applied

▼ Hide Statement Counts

Insert	0
Select	499
Update	0
Delete	0
Merge	0
Skipped (Parsing or Privilege Errors)	499

SQL Access Advisor: Results and Implementation



This chart and table list recommendations initially ordered by the largest cost improvement. Implementing the top recommendation will improve total performance the most.

Recommendations by Cost Improvement

Select Recommendations for Implementation

☑ Include Retain Actions

Select	Implementation Status	ID	Actions	Action Types	Cost Improvement ▽	Cost Improvement (%)	Estimated Space Used (MB)	Affected SQL Statements
☑	■	1	1	■	139265	55.88	0.000	0
☑	■	3	4	■ ■ ■	58923	23.64	0.023	111
☑	■	4	4					
☑	■	5	5	■				
☑	■	6	4					
☑	■	2	1	■				

TIP **Action Types Legend** ■ Indexes ■ Materialized Views ■ Materialized View Logs ■ Partitions ■ Others

```
/* Repartitioning Table "SH"."CUSTOMERS" */
/* Dependent objects prevent automatic table repartitioning
The following objects must be removed before executing implementation script:
 INDEX: "SH"."CUSTOMERS_YOB_BIX","SH"."CUSTOMERS_MARITAL_BIX","SH"."CUSTOMERS_GENDER_BIX" */

SET SERVEROUTPUT ON
SET ECHO ON

CREATE TABLE PART$$_TEMP
PARTITION BY RANGE ("CUST_CREDIT_LIMIT") INTERVAL( 400) ( PARTITION VALUES LESS
THAN (1600) )
AS SELECT * FROM "SH"."CUSTOMERS" WHERE NULL IS NOT NULL;

BEGIN
DECLARE
num_errs NUMBER;
CURSOR not_null_cons IS
SELECT constraint_name
FROM user_constraints
WHERE table_name = 'PART$$_TEMP'
AND constraint_type = 'C';
cons_name VARCHAR2(30);
sql_hndl NUMBER;
rows NUMBER;
BEGIN
OPEN not_null_cons;
LOOP
FETCH not_null_cons INTO cons_name;
EXIT when not_null_cons%NOTFOUND;

sql_hndl := DBMS_SQL.OPEN_CURSOR;
DBMS_SQL.PARSE(sql_hndl, 'ALTER TABLE PART$$_TEMP DROP CONSTRAINT '||cons_name, DBMS_SQL.NATIVE);
rows := DBMS_SQL.EXECUTE(sql_hndl);
DBMS_SQL.CLOSE_CURSOR(sql_hndl);
END LOOP;

CLOSE not_null_cons;


DBMS_REDEFINITION.CAN_REDEF_TABLE('SH','CUSTOMERS',DBMS_REDEFINITION.CONS_USE_ROWID);
DBMS_REDEFINITION.START_REDEF_TABLE('SH','CUSTOMERS','PART$$_TEMP',NULL,DBMS_REDEFINITION.CONS_USE_ROWID);
DBMS_REDEFINITION.COPY_TABLE_DEPENDENTS('SH','CUSTOMERS','PART$$_TEMP',DBMS_REDEFINITION.CONS_ORIG_PARAMS,TRUE,
TRUE);
DBMS_OUTPUT.PUT_LINE(num_errs || ' errors copying table dependents.');
DBMS_REDEFINITION.FINISH_REDEF_TABLE('SH','CUSTOMERS','PART$$_TEMP');
END;
END;
/

DROP TABLE PART$$_TEMP CASCADE CONSTRAINTS;
```

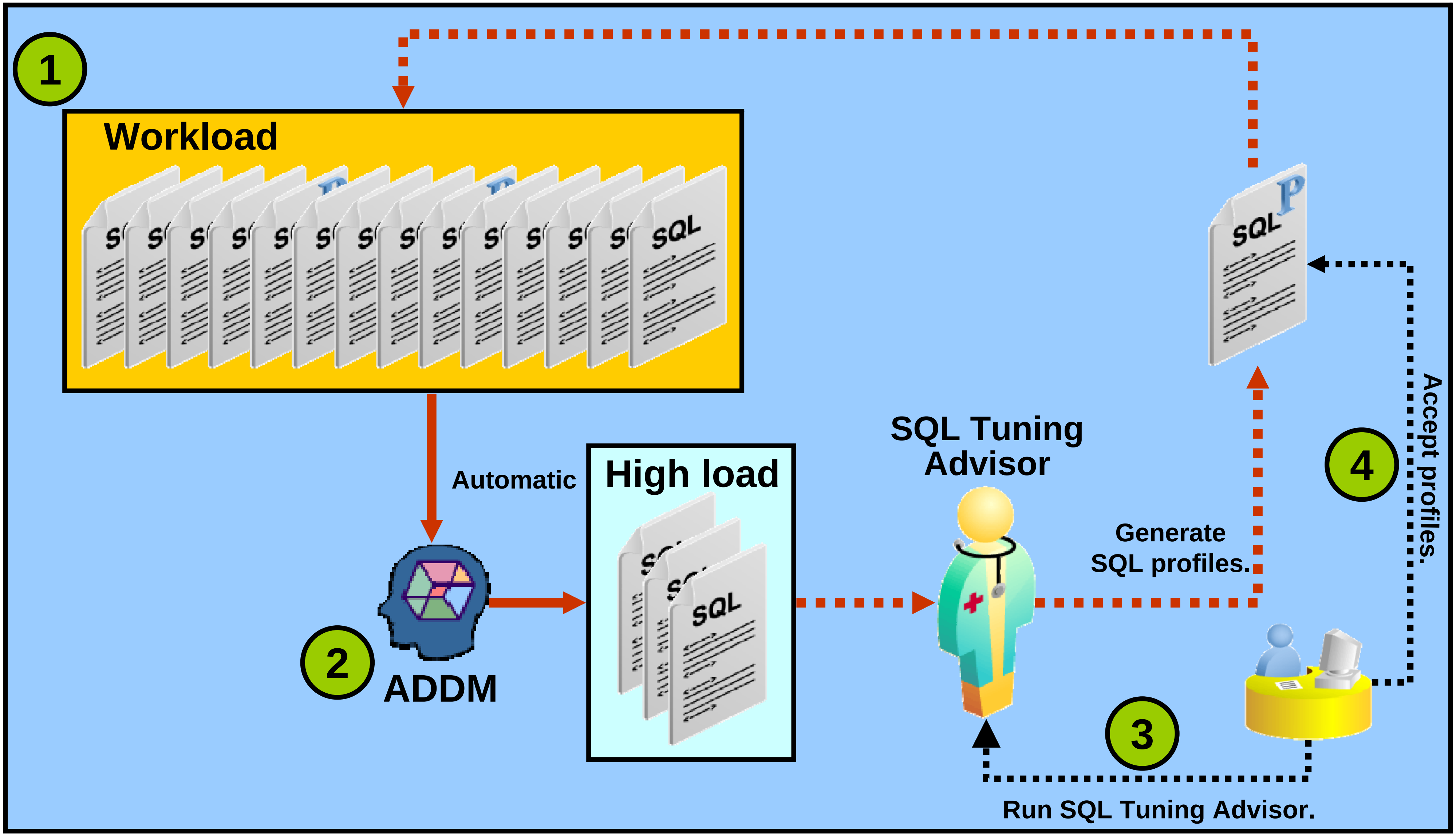
Recommendation: 1

SQL Access Advisor generates default object names and uses the default schemas and tablespaces specified during task creation, but you can change them. If you edit any name, dependent names, which are shown as readonly, will be updated accordingly. If the Tablespace field is left blank the default tablespace of the schema will be used. When you click Apply or OK, the SQL script is modified, but it is not actually executed until you select 'Schedule Implementation' on the Recommendations or SQL Statements pages.

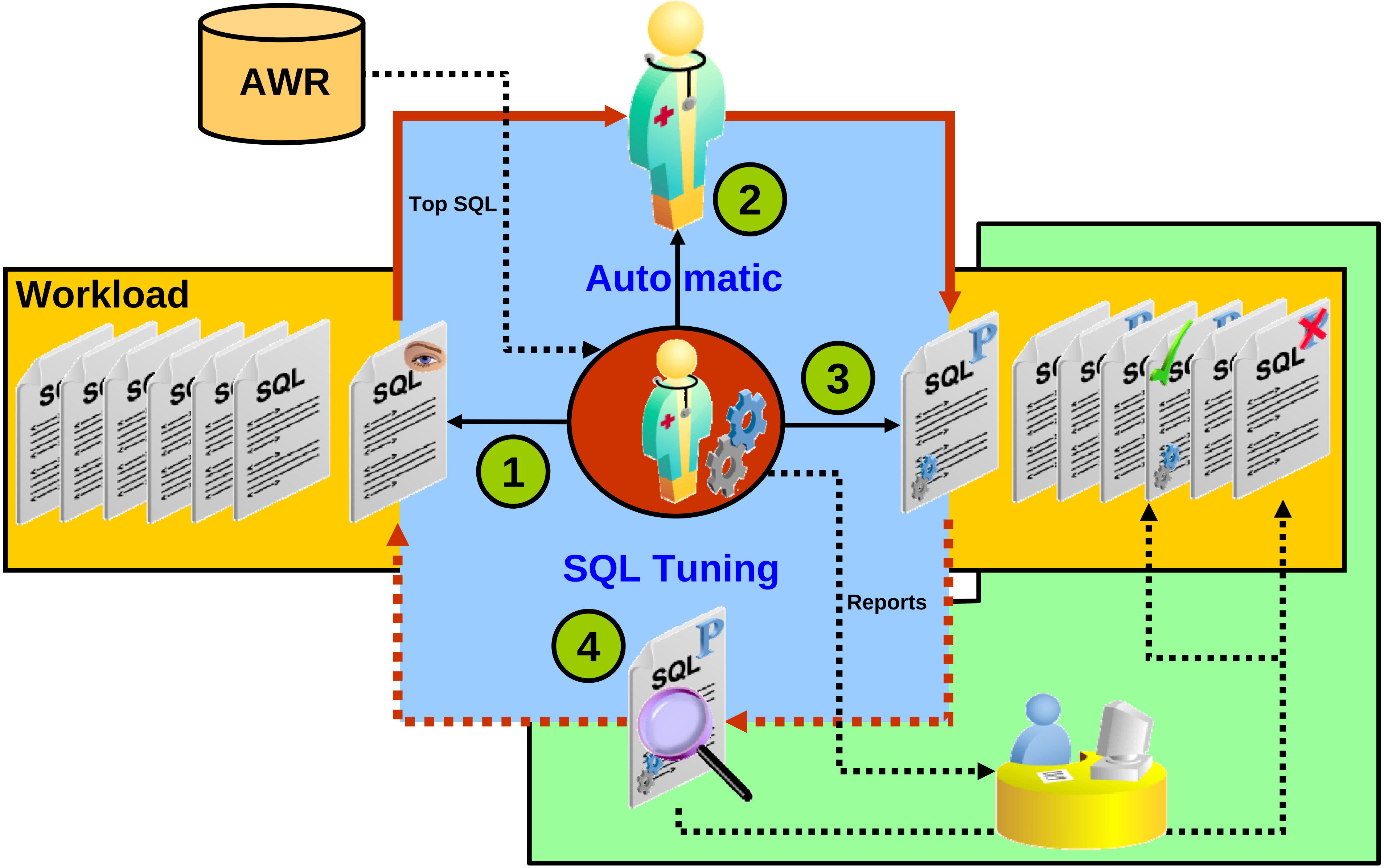
Actions

Implementation Status	Action	Object Name	Object Attributes	Base Table	Schema	Tablespace	Partition Key	SQL Partition	Estimated Space Used (MB)
■	PARTITION TABLE	CUSTOMERS			SH		("CUST_CREDIT_LIMIT")	PARTITION BY RANGE ("CUST_CREDIT_LIMIT") INTERVAL(400) (PARTITION VALUES LESS THAN (1600))	0.000

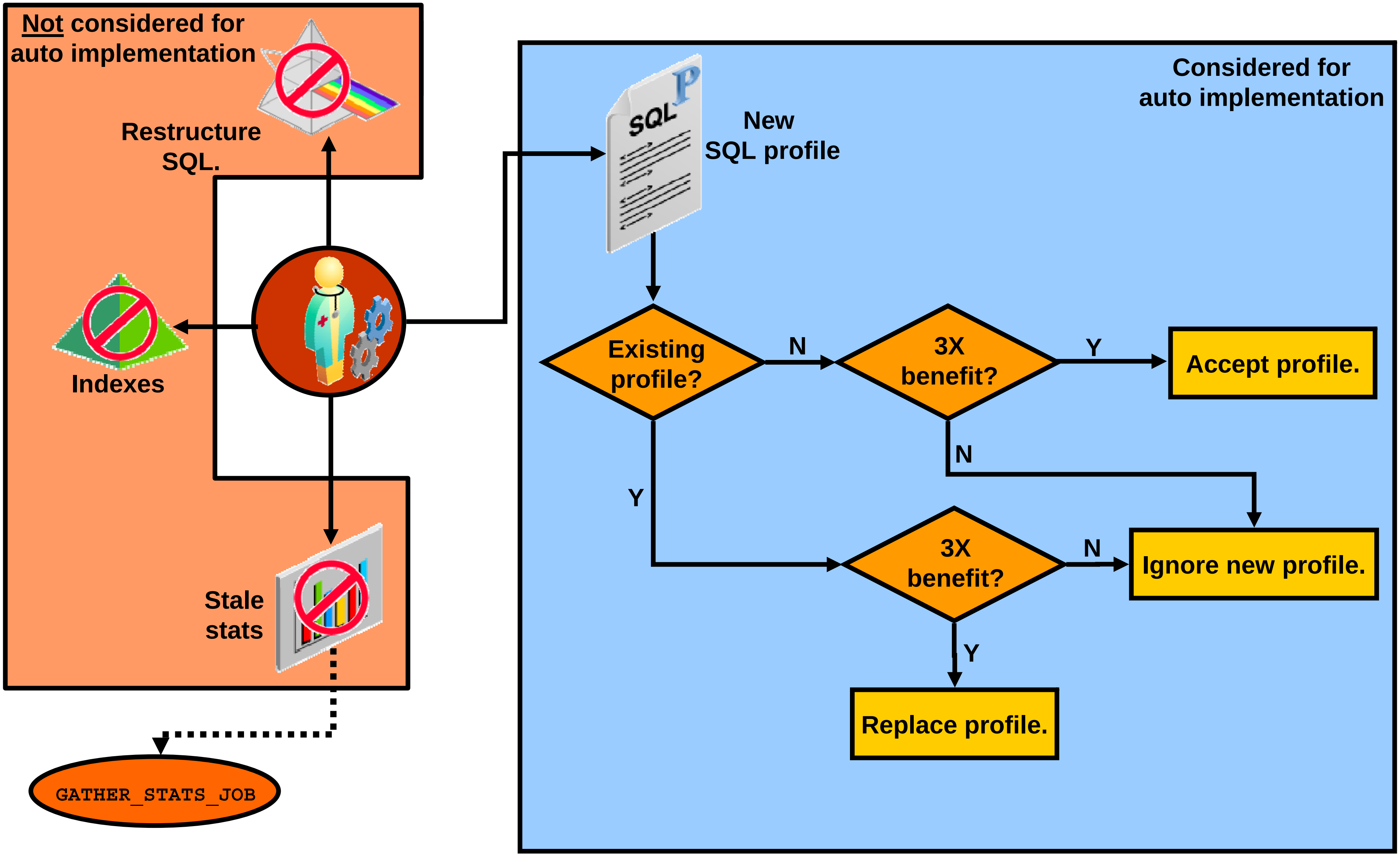
SQL Tuning Loop



Automatic SQL Tuning



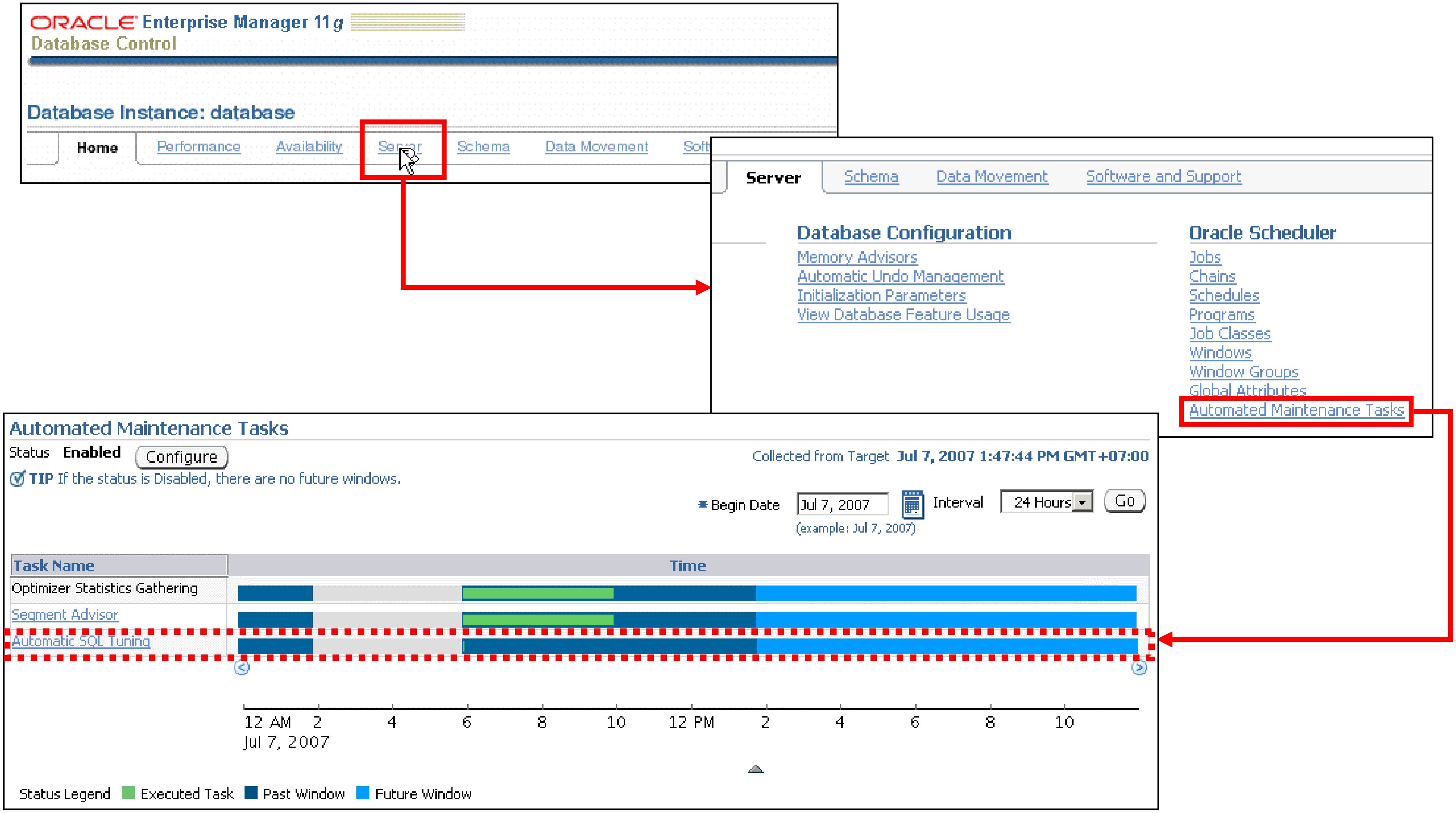
Automatic Tuning Process



Automatic SQL Tuning Controls

- Autotask configuration:
  - On/off switch
  - Maintenance windows running tuning task
  - CPU resource consumption of tuning task
- Task parameters:
  - SQL profile implementation automatic/manual switch
  - Global time limit for tuning task
  - Per-SQL time limit for tuning task
  - Test-execute mode disabled to save time
  - Maximum number of SQL profiles automatically implemented per execution as well as overall
  - Task execution expiration period

Automatic SQL Tuning Task



Automated Maintenance Tasks

Status **Enabled**

Collected from Target **Jul 7, 2007 1:47:44 PM GMT+07:00**

TIP If the status is Disabled, there are no future windows.

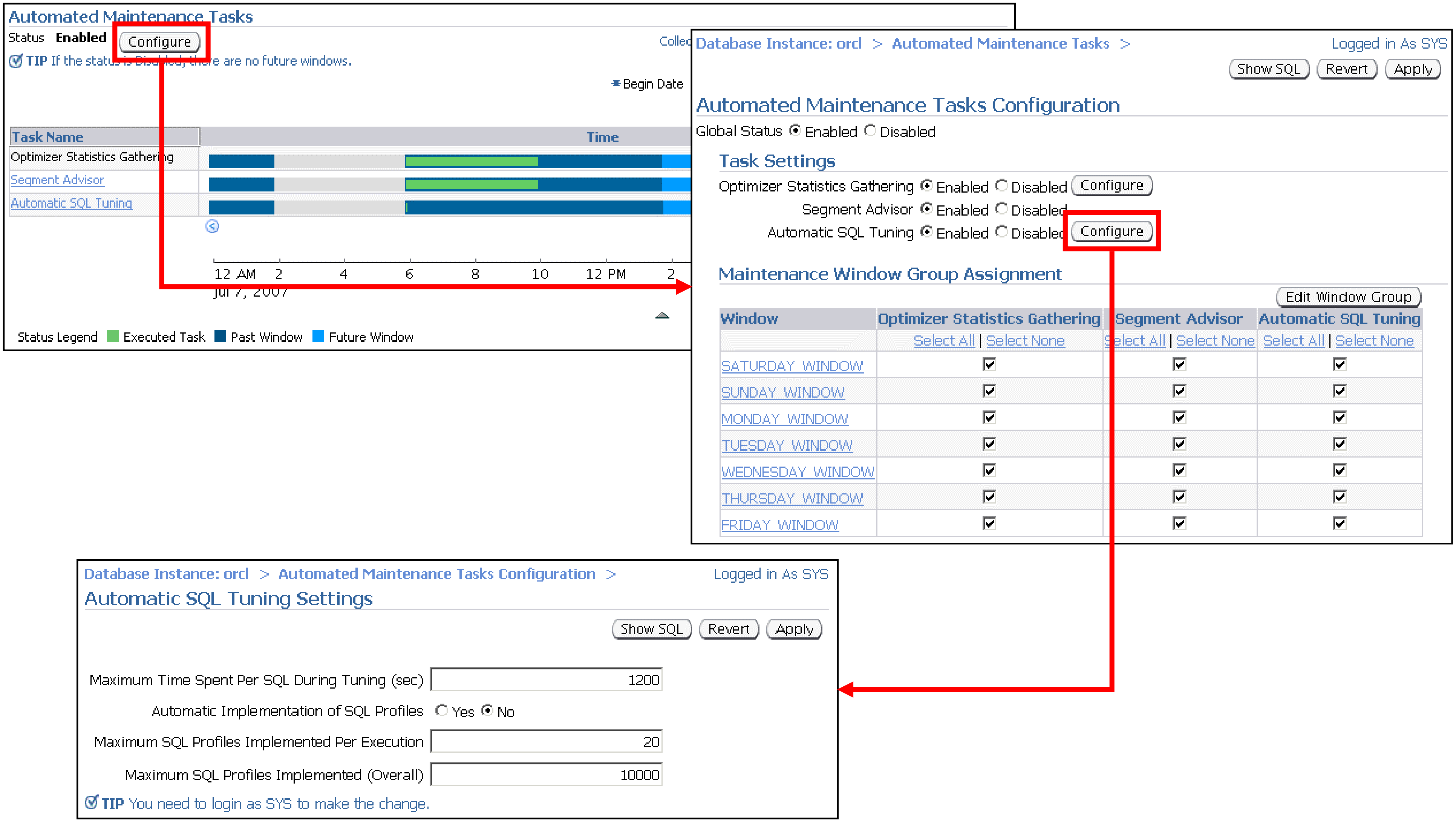
* Begin Date Jul 7, 2007 Interval 24 Hours

(example: Jul 7, 2007)

Task Name	Time
Optimizer Statistics Gathering	
Segment Advisor	
Automatic SQL Tuning	

Status Legend 🟩 Executed Task ■ Past Window ■ Future Window

Configuring Automatic SQL Tuning



Automated Maintenance Tasks

Status **Enabled**

TIP If the status is Disabled, there are no future windows.

Colle

Begin Date

Task Name	Time
Optimizer Statistics Gathering	
Segment Advisor	
Automatic SQL Tuning	

12 AM 2 4 6 8 10 12 PM 2
Jul 7, 2007

Status Legend 🟩 Executed Task ■ Past Window ■ Future Window

Database Instance: orcl > Automated Maintenance Tasks > Logged in As SYS

Automated Maintenance Tasks Configuration

Global Status ● Enabled ○ Disabled

Task Settings

Optimizer Statistics Gathering ● Enabled ○ Disabled

Segment Advisor ● Enabled ○ Disabled

Automatic SQL Tuning ● Enabled ○ Disabled

Maintenance Window Group Assignment

Window	Optimizer Statistics Gathering	Segment Advisor	Automatic SQL Tuning
	Select All | Select None	Select All | Select None	Select All | Select None
SATURDAY_WINDOW	☑	☑	☑
SUNDAY_WINDOW	☑	☑	☑
MONDAY_WINDOW	☑	☑	☑
TUESDAY_WINDOW	☑	☑	☑
WEDNESDAY_WINDOW	☑	☑	☑
THURSDAY_WINDOW	☑	☑	☑
FRIDAY_WINDOW	☑	☑	☑

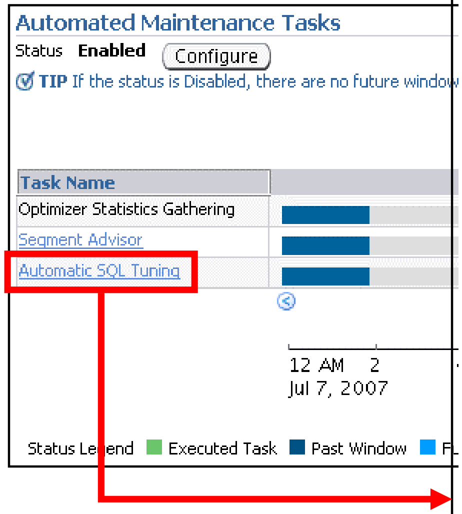
Database Instance: orcl > Automated Maintenance Tasks Configuration > Logged in As SYS

Automatic SQL Tuning Settings

Maximum Time Spent Per SQL During Tuning (sec)	1200
Automatic Implementation of SQL Profiles	○ Yes ● No
Maximum SQL Profiles Implemented Per Execution	20
Maximum SQL Profiles Implemented (Overall)	10000

TIP You need to login as SYS to make the change.

Automatic SQL Tuning: Result Summary



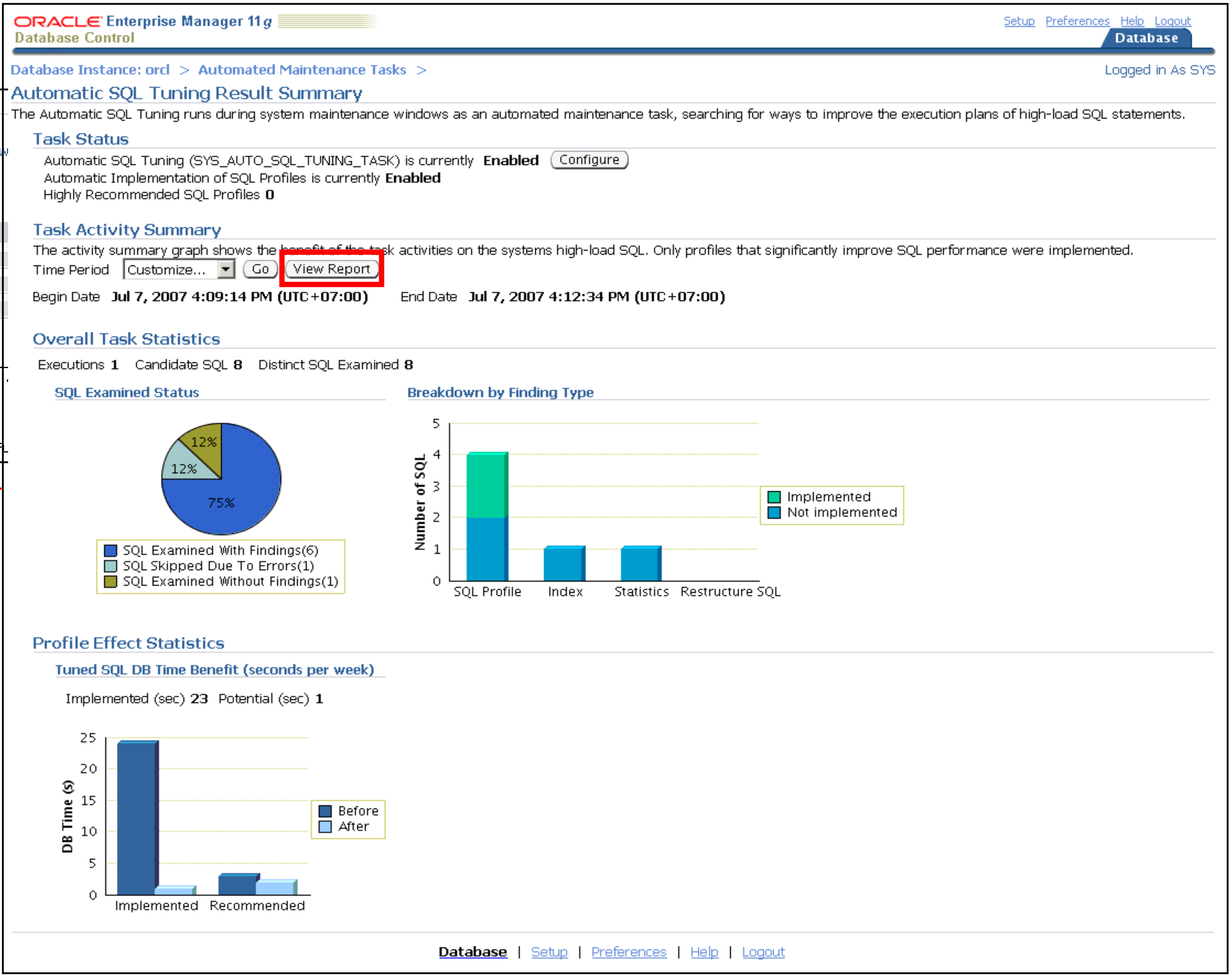
Automated Maintenance Tasks

Status **Enabled** Configure

TIP If the status is Disabled, there are no future window

Task Name	
Optimizer Statistics Gathering	
Segment Advisor	
Automatic SQL Tuning	

Status Legend 🟩 Executed Task ■ Past Window ■ Fu



ORACLE Enterprise Manager 11*g*
Database Control

Database Instance: orcl > Automated Maintenance Tasks >

Logged in As SYS

Automatic SQL Tuning Result Summary

The Automatic SQL Tuning runs during system maintenance windows as an automated maintenance task, searching for ways to improve the execution plans of high-load SQL statements.

Task Status

Automatic SQL Tuning (SYS_AUTO_SQL_TUNING_TASK) is currently **Enabled** Configure

Automatic Implementation of SQL Profiles is currently **Enabled**

Highly Recommended SQL Profiles **0**

Task Activity Summary

The activity summary graph shows the benefit of the task activities on the systems high-load SQL. Only profiles that significantly improve SQL performance were implemented.

Time Period Customize... Go View Report

Begin Date **Jul 7, 2007 4:09:14 PM (UTC+07:00)** End Date **Jul 7, 2007 4:12:34 PM (UTC+07:00)**

Overall Task Statistics

Executions **1** Candidate SQL **8** Distinct SQL Examined **8**

SQL Examined Status

■ SQL Examined With Findings(6)
■ SQL Skipped Due To Errors(1)
■ SQL Examined Without Findings(1)

Breakdown by Finding Type

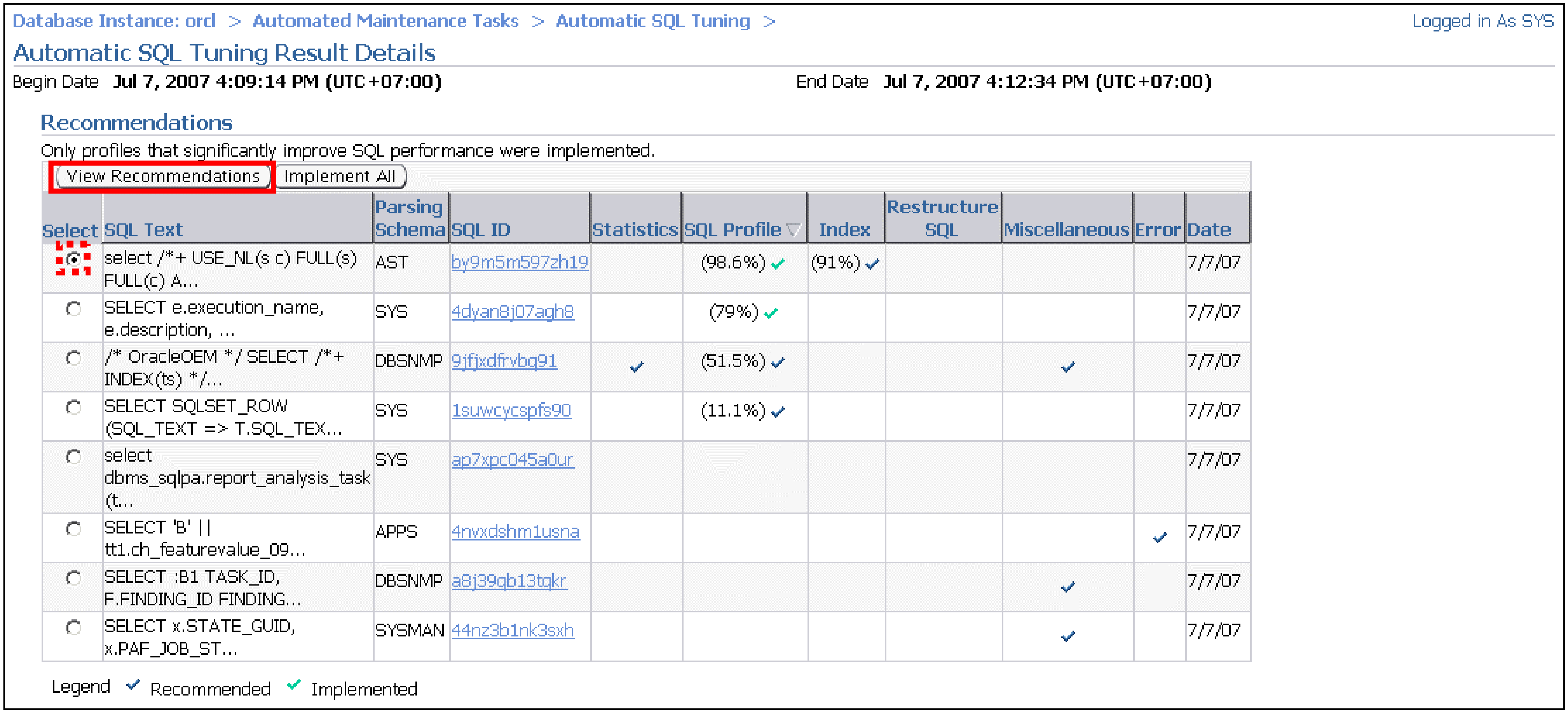
Profile Effect Statistics

Tuned SQL DB Time Benefit (seconds per week)

Implemented (sec) **23** Potential (sec) **1**

Database | Setup | Preferences | Help | Logout

Automatic SQL Tuning: Result Details



Database Instance: orcl > Automated Maintenance Tasks > Automatic SQL Tuning > Logged in As SYS

Automatic SQL Tuning Result Details

Begin Date **Jul 7, 2007 4:09:14 PM (UTC+07:00)** End Date **Jul 7, 2007 4:12:34 PM (UTC+07:00)**

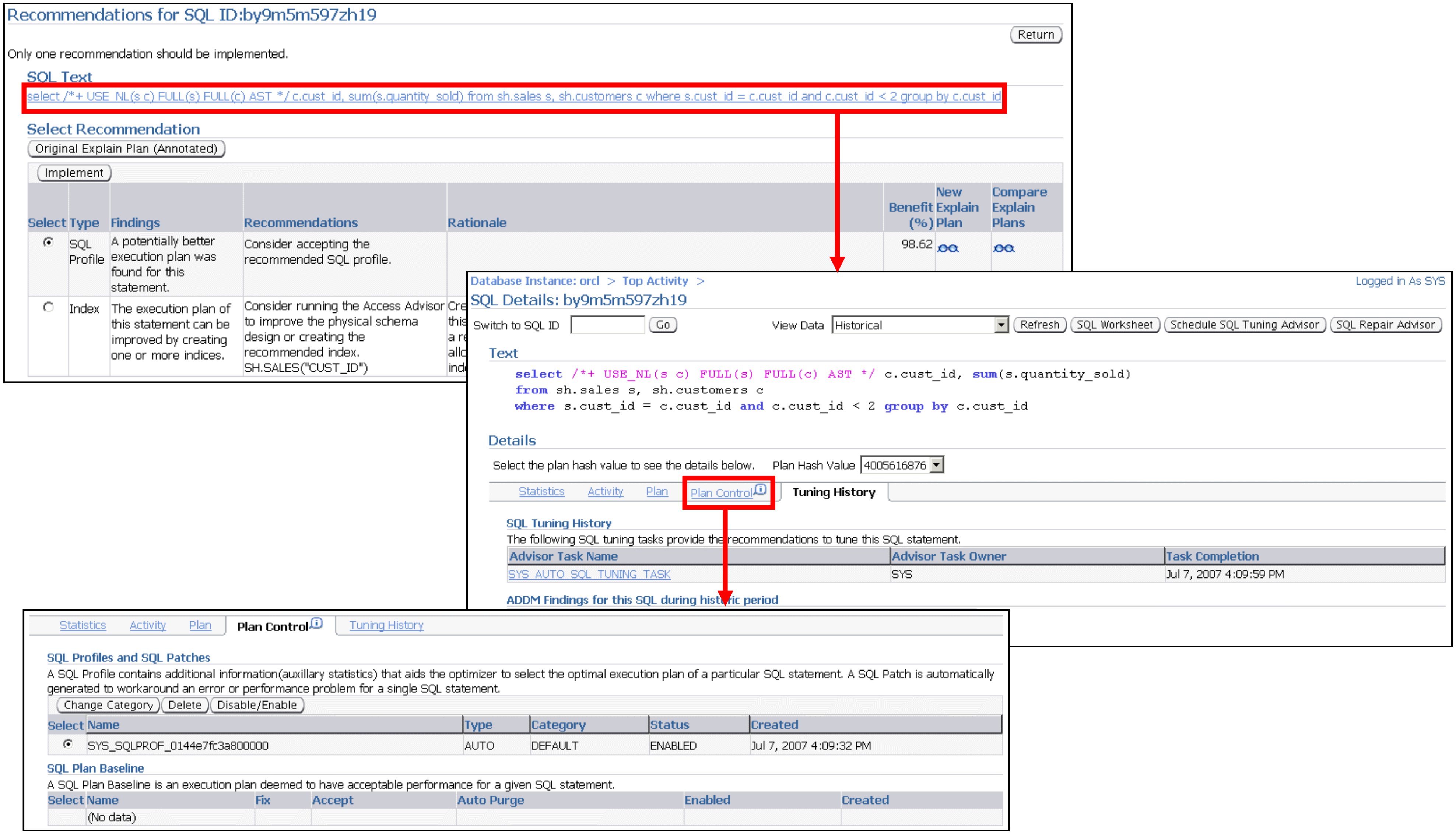
Recommendations

Only profiles that significantly improve SQL performance were implemented.

Select	SQL Text	Parsing Schema	SQL ID	Statistics	SQL Profile ▽	Index	Restructure SQL	Miscellaneous	Error	Date
●	select /*+ USE_NL(s c) FULL(s) FULL(c) A...	AST	by9m5m597zh19		(98.6%) ✓	(91%) ✓				7/7/07
○	SELECT e.execution_name, e.description, ...	SYS	4dyan8j07agh8		(79%) ✓					7/7/07
○	/* OracleOEM */ SELECT /*+ INDEX(ts) */...	DBSNMP	9jfjxdfrvbq91	✓	(51.5%) ✓			✓		7/7/07
○	SELECT SQLSET_ROW (SQL_TEXT => T.SQL_TEX...	SYS	1suwcycspfs90		(11.1%) ✓					7/7/07
○	select dbms_sqlpa.report_analysis_task (t...	SYS	ap7xpc045a0ur							7/7/07
○	SELECT 'B' || tt1.ch_featurevalue_09...	APPS	4nvxdshm1usna						✓	7/7/07
○	SELECT :B1 TASK_ID, F.FINDING_ID FINDING...	DBSNMP	a8j39qb13tqkr					✓		7/7/07
○	SELECT x.STATE_GUID, x.PAF_JOB_ST...	SYSMAN	44nz3b1nk3sxh					✓		7/7/07

Legend ✓ Recommended ✓ Implemented

Automatic SQL Tuning Result Details: Drilldown



Recommendations for SQL ID:by9m5m597zh19

Only one recommendation should be implemented.

SQL Text

select /*+ USE_NL(s c) FULL(s) FULL(c) AST */ c.cust_id, sum(s.quantity_sold) from sh.sales s, sh.customers c where s.cust_id = c.cust_id and c.cust_id < 2 group by c.cust_id

Select Recommendation

Select	Type	Findings	Recommendations	Rationale	Benefit (%)	New Explain Plan	Compare Explain Plans
●	SQL Profile	A potentially better execution plan was found for this statement.	Consider accepting the recommended SQL profile.		98.62	👓	👓
○	Index	The execution plan of this statement can be improved by creating one or more indices.	Consider running the Access Advisor to improve the physical schema design or creating the recommended index. SH.SALES("CUST_ID")	Cre this a re allo inde			

Database Instance: orcl > Top Activity >

Logged in As SYS

SQL Details: by9m5m597zh19

Switch to SQL ID ______ Go   View Data Historical

Text

```
select /*+ USE_NL(s c) FULL(s) FULL(c) AST */ c.cust_id, sum(s.quantity_sold)
from sh.sales s, sh.customers c
where s.cust_id = c.cust_id and c.cust_id < 2 group by c.cust_id
```

Details

Select the plan hash value to see the details below.   Plan Hash Value 4005616876

SQL Tuning History

The following SQL tuning tasks provide the recommendations to tune this SQL statement.

Advisor Task Name	Advisor Task Owner	Task Completion
SYS_AUTO_SQL_TUNING_TASK	SYS	Jul 7, 2007 4:09:59 PM

ADDM Findings for this SQL during historic period

SQL Profiles and SQL Patches

A SQL Profile contains additional information(auxillary statistics) that aids the optimizer to select the optimal execution plan of a particular SQL statement. A SQL Patch is automatically generated to workaround an error or performance problem for a single SQL statement.

Select	Name	Type	Category	Status	Created
●	SYS_SQLPROF_0144e7fc3a800000	AUTO	DEFAULT	ENABLED	Jul 7, 2007 4:09:32 PM

SQL Plan Baseline

A SQL Plan Baseline is an execution plan deemed to have acceptable performance for a given SQL statement.

Select	Name	Fix	Accept	Auto Purge	Enabled	Created
	(No data)					

Automatic SQL Tuning Considerations

- SQL not considered for Automatic SQL Tuning:
  - Ad hoc or rarely repeated SQL
  - Parallel queries
  - Long-running queries after profiling
  - Recursive SQL statements
  - DML and DDL
- These statements can still be manually tuned by using SQL Tuning Advisor.

Summary

In this lesson, you should have learned the following:

- Statement profiling
- SQL Tuning Advisor
- SQL Access Advisor
- Automatic SQL Tuning

Practice 11: Overview

This practice covers the following topics:

- Using ADDM and SQL Tuning Advisor to tune your SQL statements
- Using SQL Access Advisor to change your schema
- Using Automatic SQL Tuning to tune your statements