

**ORACLE\***

**SQLT (SQLTXPLAIN) 11.2.9.6**

**Note: 215187.1**

Carlos Sierra  
GPS ST CoE

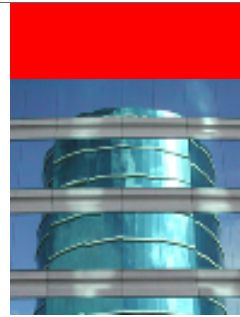
## SQLT (SQLTXPLAIN)

- Overview
- Installing and uninstalling
- Main Methods
- Utilities
- How to
- References



ORACLE

# Overview



ORACLE

## SQLT Description

*SQLT (SQLTXPLAIN) is a tool that inputs one SQL statement and outputs a set of comprehensive diagnostic files for SQL performance analysis and tuning.*

## SQLT Characteristics

- Free download and use
- 9i, 10g and 11g databases
- Unix, Linux and Windows platforms
- Unwrapped source (SQL and PL/SQL)
- Installs under its own schema (self-contained)
- Small footprint (space and CPU)
- Not intrusive (does not expose application data)
- Recognizes RAC environments
- Helps to expedite a SQL performance analysis and tuning

ORACLE

5

## SQLT Input and Output

- One SQL Statement
  - As a text file
  - From memory or AWR
  - Within a script



### SQLT

- Connects as application user
- Gathers diagnostics info



- Reports
  - Main
  - Frames
  - Lite
  - 10053 10046
  - Readme
  - TRCA
  - TKPROF
  - Logs
- Scripts
  - Metadata
  - Set CBO Env
  - TC SQL
  - TC Script
  - TC Builder
  - XPLORE
- Snapshot
  - CBO Stats
  - SQL Text
  - Explain Plan
  - Init Params
  - Tables
  - Indexes
  - Partitions
  - Table Cols
  - Reports
  - Hints

ORACLE

## SQLT Input

### One SQL DML statement

- XPLAIN
  - Stand-alone SQL placed in a flat text file
- XTRACT
  - Memory or AWR resident SQL, identified by its hash\_value or sql\_id
- XECUTE
  - Stand-alone script that contains one SQL, together with the declaration and values of its bind variables
- XTRXEC
  - XTRACT and XECUTE combined

ORACLE

7

**The recommended methods are XECUTE and XTRACT. Try avoiding the XPLAIN method since the Explain Plan generated by it may not be accurate if your SQL contains bind variables.**

## SQLT Output Reports

- Main html report
- Frames html report
- Lite report
- CBO trace 10053 and SQL trace 10046
- Readme text file
- Trace Analyzer and TKPROF
- Log files

ORACLE

8

### **Main html report**

Comprehensive diagnostics information related to one SQL

### **Frames html report**

Ease navigation on main html report

### **Lite report**

DBMS\_XPLAN plus Tables/Indexes and CBO stats

### **CBO trace 10053 and SQL trace 10046**

10053 in all methods and 10046 in XECUTE

### **Readme text file**

### **Trace Analyzer and TKPROF**

Only in XECUTE method. Trace Analyzer is optional.

### **Log files**



## SQLT Output Scripts

- Metadata
- Set CBO Environment
- TC SQL
- TC Script
- TC Builder
- XPLORE

ORACLE

9

### **Metadata**

Schema objects needed for Test Case (TC) creation

### **Set CBO Environment**

Alter session commands for TC

### **TC SQL**

XTRACT: script with the one SQL text for TC XPLAIN

### **TC Script**

XTRACT: script with one SQL and its binds for TC XECUTE

### **TC Builder**

11g XTRACT and XECUTE

Invokes DBMS\_SQLDIAG.EXPORT\_SQL\_TESTCASE

### **XPLORE**

Generated by XECUTE

Discovers potential plans

## SQLT Snapshot

- Used by COMPARE and TC creation
- Snapshot objects

ORACLE

10

### Used by COMPARE and TC creation

#### Snapshot objects

CBO statistics of SQL related schema objects

SQL Text

Plan Table

Initialization parameters and Fix Control

Tables and Indexes

Partitions and Subpartitions

Table Columns

Reports History

Stored Outline and SQL Profile Hints

## What is used where?

	COMPARE	SQLT TC	Reproduce Explain Plan	11g TCB
SQLT Readme	Y	Y	Y	
Metadata		Y		
Set CBO Env		Y	Y	
TC SQL		Y	Y	
TC Script		Y	Y	
TC Builder				Y
Snap: CBO Stats	Y	Y	Y	
Snap: Other Obj	Y			

ORACLE

11

## Installing and uninstalling



ORACLE

12

## SQLT Directories

```
sqlt/  
  doc/  
  install/  
  run/  
    input/  
  utl/
```

ORACLE

13

sqlt/	-- instructions and subdirectories
doc/	-- this pdf and list of changes
install/	-- install and uninstall scripts
run/	-- execution scripts (methods)
input/	-- recommended input directory
utl/	-- tool utilities like compare

## Installing

- Steps
  - Download sqlt.zip from 215187.1
  - Unzip sqlt.zip in server
  - Connect as SYS
  - Execute sqlt/install/sqcreate.sql
- Parameters asked during the installation
  - SQLTXPLAIN password (case sensitive in 11g)
  - Optional Host String (TNS Alias) including “@”
  - Schema name of main application user of SQLT
  - SQLTXPLAIN default tablespace
  - SQLTXPLAIN temporary tablespace

ORACLE

14

**Actual tool is provided as a zip file in Note 215187.1.**

**For 9i, 10g or 11g: use sqlt.zip file.**

**For 8i: use sqlt\_old.zip.**

**Read included files: instructions.txt and changes.txt.**

**Installation removes prior version automatically.**

**Several \*.lis files are created during installation.**

## Uninstalling

- Steps
  - Connect as SYS
  - Execute `sqlt/install/sqdrop.sql`

ORACLE

15

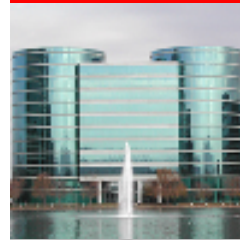
**sqltxplain schema objects and user are dropped**

**Installation removes prior version automatically**

**No parameter values are requested**

**No spool files are created during uninstall**

## Main Methods



ORACLE

16



## Main Methods

- XPLAIN
  - Stand-alone SQL placed in a flat text file
- XTRACT
  - Memory or AWR resident SQL, identified by its hash\_value or sql\_id
- XECUTE
  - Stand-alone script that contains one SQL, together with the declaration and values of its bind variables
- XTRXEC
  - XTRACT and XECUTE combined

ORACLE

17

**The recommended methods are XECUTE and XTRACT. Try avoiding the XPLAIN method since the Explain Plan generated by it may not be accurate if your SQL contains bind variables.**

## **XPLAIN Overview**

- SQL Statement is placed within a text file
- Name of file passed as inline parameter
- No need to replace bind variables with literals
- Uses EXPLAIN PLAN FOR which is blind to bind peeking
- To overcome limitations of EXPLAIN PLAN FOR, manually replace binds with literals of same type
- Be aware of implicit data type conversions
- Date variables may require to adjust SQL Text

ORACLE

18

### **Two entry points**

sqltxplain.sql (comprehensive)

sqltxplite.sql (lite)

## XPLAIN

### sample\_sql1.sql file

```
select comps.component_item_id ,bill.organization_id ,bill.assembly_item_id ,
       comps.operation_seq_num ,subs.substitute_component_id ,
       substitute_item_quantity
from
bom_bill_of_materials bill ,mrp_plan_organizations_v orgs ,mrp_system_items
msi ,bom_inventory_components comps ,bom_substitute_components subs where
((((((((bill.common_bill_sequence_id=decode(orgs.planned_organization,
null ,null ,comps.bill_sequence_id) and bill.organization_id=
orgs.planned_organization) and orgs.compile_designator=:b0) and
orgs.organization_id=:b1) and msi.organization_id=orgs.planned_organization)
and msi.compile_designator=orgs.compile_designator) and
msi.inventory_item_id=comps.component_item_id) and
TRUNC(comps.effectivity_date)<=TRUNC(sysdate )) and
comps.implementation_date is not null ) and NVL(comps.disable_date,
(sysdate +1))<sysdate ) and comps.component_sequence_id=
subs.component_sequence_id) order by bill.organization_id,
comps.component_item_id;
```

ORACLE

19

### Input file with one SQL

SQL must be valid (correct syntax and semantics)

Should end with one semicolon “;” (optional)

Should not have empty lines (required)

It can contain bind variables (like :b1)

Should be the same SQL that performs poorly

The created file must contain one and only one SQL

The file must be placed into the sqlt/run directory

For a sample file with one SQL see file sample\_sql1.sql

## **XPLAIN**

### **Main HTML Report**

- Enhanced Explain Plan
- Observations
- Object Dependencies
- Objects
- Segments
- Tables
- Indexes

ORACLE

20

#### **Enhanced Explain Plan**

PLAN\_TABLE

GV\_\$SQLTEXT\_WITH\_NEWLINES

#### **Observations**

##### **Object Dependencies**

GV\_\$OBJECT\_DEPENDENCY

DBA\_DEPENDENCIES

DBA\_SOURCE

##### **Objects**

DBA\_OBJECTS

##### **Segments**

DBA\_SEGMENTS

##### **Tables**

DBA\_TABLES

DBA\_TAB\_MODIFICATIONS

TAB\$

WRI\$\_OPTSTAT\_TAB\_HISTORY

##### **Indexes**

DBA\_INDEXES

WRI\$\_OPTSTAT\_IND\_HISTORY

## **XPLAIN**

### **Main HTML Report**

- Table Columns
- Index Columns
- Constraints
- Column Histograms
- Table Partitions
- Index Partitions
- Table Partition Columns
- Table Partition Histograms

ORACLE

21

#### **Table Columns**

DBA\_TAB\_COLUMNS

DBA\_TAB\_COLS

#### **Index Columns**

DBA\_IND\_COLUMNS

#### **Constraints**

DBA\_CONSTRAINTS

#### **Column Histograms**

DBA\_TAB\_HISTOGRAMS

#### **Table Partitions**

DBA\_TAB\_PARTITIONS

TABPART\$

#### **Index Partitions**

DBA\_IND\_PARTITIONS

#### **Table Partition Columns**

DBA\_PART\_COL\_STATISTICS

#### **Table Partition Histograms**

DBA\_PART\_HISTOGRAMS

## **XPLAIN**

### **Main HTML Report**

- Table Subpartitions
- Index Subpartitions
- Table Subpartition Columns
- Table Subpartition Histograms
- Metadata
- Tablespaces
- DBMS\_STATS Setup

ORACLE

22

#### **Table Subpartitions**

DBA\_TAB\_SUBPARTITIONS

TABCOMPART\$

#### **Index Subpartitions**

DBA\_IND\_SUBPARTITIONS

#### **Table Subpartition Columns**

DBA\_SUBPART\_COL\_STATISTICS

#### **Table Subpartition Histograms**

DBA\_SUBPART\_HISTOGRAMS

#### **Metadata**

DBMS\_METADATA.GET\_DDL

#### **Tablespaces**

DBA\_TABLESPACES

#### **DBMS\_STATS Setup**

DBA\_SCHEDULER\_JOBS

DBA\_AUTOTASK\_CLIENT

WRI\$\_OPTSTAT\_OPR

## XPLAIN

### Main HTML Report

- System Statistics
- Stored Outlines
- SQL Profiles
- Initialization Parameters
- Tool Configuration
- APPS specific

ORACLE

23

#### System Statistics

AUX\_STAT\$

WRI\$\_OPTSTAT\_AUX\_HISTORY

#### Stored Outlines

OUTLN.OL\$

OUTLN.OL\$HINTS

#### SQL Profiles

SYS.WRI\$\_ADV\_RATIONALE

SYS.WRI\$\_ADV\_TASKS

#### Initialization Parameters

GV\_\$PARAMETER2

GV\_\$SYSTEM\_PARAMETER2

GV\_\$SQL\_OPTIMIZER\_ENV

#### Tool Configuration

Thresholds and flags

Setup commands

#### APPS specific

FND\_HISTOGRAM\_COLS

## XPLAIN Execution

- Comprehensive execution (recommended)
  - # cd sqlt/run
  - # sqlplus [apps user]/[apps pwd]
  - SQL> start **sqltxplain.sql** [path and name of file with SQL]
  - SQL> start sqltxplain.sql input/sample\_sql1.sql <== your file

ORACLE

24

### Lite execution

```
# cd sqlt/run
```

```
# sqlplus [apps user]/[apps pwd]
```

```
SQL> start sqltxplite.sql [path and name of file with SQL]
```

```
SQL> start sqltxplite.sql input/sample_sql1.sql <== your file
```





## **XPLAIN**

### **Pros and Cons**

- Pros
  - SQL is not executed (may take hours otherwise)
  - No need to know the values of bind variables
  - Easiest method to execute
- Cons
  - Bind peeking may cause different plan (and 10053)
  - Cannot display ACTUAL ROWS values
  - Cannot invoke SQL Tuning Advisor
  - Blind to child plans and plan statistics

ORACLE

25

## XTRACT Overview

- SQL is passed by its known hash\_value or sql\_id
- SQL must be memory or AWR resident
- How to find hash\_value in SQL Trace?

```
PARSING IN CURSOR #7 len=2008 ... hv=1250854193 ad='4cf494ec'  
select comps.component_item_id ,bill.organization_id ,bill.as...
```

ORACLE

26

### Three entry points

sqltxtract.sql (comprehensive)

- hash\_value or sql\_id

sqltxtrlite.sql (lite)

- hash\_value or sql\_id

sqltxtract2.sql (comprehensive)

- hash\_value, address, child\_number, sql\_id or plan\_hash\_value

## **XTRACT Main HTML Report**

### **In addition to XPLAIN content**

- SQL Tuning Advisor
- Peaked Binds
- Bind Variables for Latest Execution
- SQL Monitor (11g)
- SQL and Plan Statistics
- Child Plan and Workareas
- AWR Hist SQL Statistics and Plan

ORACLE

27

#### **SQL Tuning Advisor**

DBMS\_SQLTUNE

#### **Bind Variables Latest Execution**

V\$SQL\_BIND\_CAPTURE

#### **SQL Monitor (11g)**

GV\$SQL\_MONITOR and GV\$SQL\_PLAN\_MONITOR

#### **SQL and Plan Statistics**

GV\$SQL and GV\$SQL\_PLAN\_STATISTICS

#### **Child Plan and Workareas**

GV\$SQL\_PLAN and GV\$SQL\_WORAREA

#### **AWR Hist SQL Statistics and Plan**

DBA\_HIST\_SQLSTAT and DBA\_HIST\_SQL\_PLAN

## XTRACT Execution

- Comprehensive execution (recommended)
  - # cd sqlt/run
  - # sqlplus [apps user]/[apps pwd]
  - SQL> start **sqltxtract.sql** [hash\_value or sql\_id for one SQL]
  - SQL> start sqltxtract.sql 2524255098 <== your SQL hash\_value
  - SQL> start sqltxtract.sql 0w6uydn50g8cx <== your SQL sql\_id

ORACLE

28

### Lite execution

```
# cd sqlt/run
```

```
# sqlplus [apps user]/[apps pwd]
```

```
SQL> start sqltxtlite.sql [hash_value or sql_id for one SQL]
```

```
SQL> start sqltxtlite.sql 2524255098 <== your SQL hash_value
```

```
SQL> start sqltxtlite.sql 0w6uydn50g8cx <== your SQL sql_id
```

## XTRACT

### Pros and Cons

- Pros
  - Child plans and plan statistics (if STATISTICS\_LEVEL = ALL)
  - AWR plans and statistics
  - Actual Execution Plan (not an explain plan)
  - Invokes SQL Tuning Advisor
  - SQL is not executed (may take hours otherwise)
  - Easy to execute (if hash\_value or sql\_id are known)
- Cons
  - 10053 is generated based on EXPLAIN PLAN FOR (may not be accurate due to binds peeking)
  - Requires to know, or get before hand, hash\_value or sql\_id

ORACLE

29

## **XECUTE Overview**

- One input parameter
  - The name of a script that contains
    - Bind Variables declaration
    - Bind Variables assignment
    - One well formed SQL DML statement
- Accurate 10053 using bind peeking
- Invokes Trace Analyzer if installed
- Invokes TKPROF

ORACLE

30

### **One entry point**

sqltxecute.sql (comprehensive)

### **Notes:**

1. SQL statement must end with semicolon “;”
2. XECUTE method generates 10046 then executes Trace Analyzer on it

## sqlt/run/input/sample\_script0.sql

```
REM
REM Sample SCRIPT to be used as input to sqltxecute.sql
REM
REM Bind variables (optional)
REM ~~~~~
REM
VAR b1 CHAR(1);
EXEC :b1 := '1';

REM
REM ONE DML SQL statement to be executed (required)
REM ~~~~~
REM
select v.*, (orders_total - credit_limit) over_limit
from customer v v^unique_id */
where orders_total > credit_limit
and customer_type = :b1
order by over_limit desc;

REM
REM Notes:
REM 1. To force a hard-parse, always add a space in any valid place
REM    of your SQL before using XECUTE (required)
REM 2. To precisely identify your SQL, add optional token ^unique_id
REM    within a comment (not recommended)
REM 3. Your SQL must end with a semi-colon ";" in order to be
REM    complete and executable (required)
REM 4. Your SCRIPT must be able to execute stand-alone:
REM    SQL>start sample_script0.sql
REM
```

ORACLE

31

### Stand-alone input script must contain

- Session environment setup (optional)
- Declaration of bind variables (if SQL has binds)
- Value assignment of bind variables (if SQL has binds)
- One well-formed SQL (correct syntax and semantics)

## **XECUTE Main HTML Report Similar to XTRACT content**

- Includes performance data from dynamic views
- Create XPLORE script
- May exclude (if using SQL was modified)
  - Child Plan and Workareas
  - AWR Hist SQL Statistics and Plan

ORACLE

32

### **Performance data from dynamic views**

GV\$SEGMENT\_STATISTICS

V\$SESSTAT

V\$SESSION\_EVENT

**Child Plan and AWR History are excluded if the SQL was modified since it would have a different SQL\_ID. For example if ^^unique\_id was used.**



## XECUTE Execution

- Comprehensive execution
  - # cd sqlt/run
  - # sqlplus [apps user]/[apps pwd]
  - SQL> start **sqltxecute.sql** [path and name of script with SQL]
  - SQL> start sqltxecute.sql input/sample\_script1.sql <== your file

ORACLE

33

## **XECUTE**

### **Pros and Cons**

- Pros
  - Accurate 10053 (considers bind peeking)
  - Plan execution statistics (sets STATISTICS\_LEVEL = ALL)
  - Actual Execution Plan (may be different than explain plan)
  - Invokes SQL Tuning Advisor
  - No need to know, or get before hand, hash\_value or sql\_id
  - 10046 and Trace Analyzer with statistics by segment
  - TKPROF
- Cons
  - SQL is executed (may take hours)
  - Need to know the values of bind variables

ORACLE

34

## **XTRXEC Overview**

- SQL is passed by its known hash\_value or sql\_id
- SQL must be memory or AWR resident
- XTRACT phase generates TC Script
- XECUTE phase inputs TC Script
- If used on 9i and the SQL contains binds, it may fail on XECUTE phase.

ORACLE

35

### **One entry point**

sqltxtrxec.sql

## XTRXEC Execution

- Comprehensive execution
  - # cd sqlt/run
  - # sqlplus [apps user]/[apps pwd]
  - SQL> start **sqltxtrxec.sql** [hash\_value or sql\_id for one SQL]
  - SQL> start sqltxtrxec.sql 2524255098 <== your SQL hash\_value
  - SQL> start sqltxtrxec.sql 0w6uydn50g8cx <== your SQL sql\_id

ORACLE

36

## **XTRXEC**

### **Pros and Cons**

- Pros
  - Accurate 10053 (considers bind peeking)
  - Plans from memory and AWR
  - Actual Execution Plan (may be different than explain plan)
  - Invokes SQL Tuning Advisor
  - 10046 and Trace Analyzer with statistics by segment
  - TKPROF
- Cons
  - SQL is executed (may take hours)
  - Requires to know, or get before hand, hash\_value or sql\_id

## Differences

	XPLAIN	XTRACT	XECUTE	XTRXEC
Explain Plan	Y			
Actual Plan		Y	Y	Y
Memory plan		Y	Y	Y
Child plans		Y		Y
AWR plans		Y		Y
SQL Monitor		Y		Y
Accurate 10053			Y	Y
Estimated Rows	Y	Y	Y	Y
Actual Rows		Y	Y	Y
Executes SQL			Y	Y
Uses bind peeking		Y	Y	Y
SQL Tuning Advisor		Y	Y	Y
11g TC Builder		Y	Y	Y
Generates 10046			Y	Y
TRCA and TKPROF			Y	Y
Seg/Sess stats			Y	Y

ORACLE

38

The recommended methods are **XECUTE** and **XTRACT**. Try avoiding the **XPLAIN** method since the Explain Plan generated by it may not be accurate if your SQL contains bind variables.

## Utilities



ORACLE

39

## Utilities

- CDIRS Creates Staging Directory
- CONFIG Config Params description and value
- IMP Imports CBO Stats
- IMPFO Imports Fixed Objects CBO Stats
- COMPARE Compares two executions of SQLT
- HISTPURGE Purges SQLT Repository
- HISTFILE Extracts historic report from repository
- PROFILE Generates script to set a known plan
- PROFILE EXT Stand alone coe\_xfr\_sql\_profile.sql
- XPLORE Discovers potential plans

ORACLE

40



**How to**



ORACLE

41

## How to

- Compare Execution Plans from two systems
- Migrate CBO Stats related to one SQL
- Reproduce an Execution Plan from one system into another
- Create TC user and Schema Objects in similar system
- Create a SQLT Test Case
- Create a stand-alone TC based on a SQLT TC
- Gather a CBO Statistics Baseline

ORACLE

42

**Use the readme.txt file generated by SQLT**

## How to

- Transfer a Stored Outline
- Transfer a SQL Profile
- Create a Stored Outline
- Extract a plan from memory or AWR and pin it to a SQL in same or different system
- Discover potential plans

ORACLE

43

**Use the readme.txt file generated by SQLT**

## Compare Execution Plans from two systems

- Prerequisites
  - SOURCE1, SOURCE2 and COMPARE systems could be all same or all different
  - Execute SQLT in SOURCE1 and SOURCE2 using any method
- Steps
  - Import into COMPARE system 5 SQLT tables from SOURCE1 and SOURCE2 (only if different systems)
  - Execute COMPARE utility  
SQL> START sqlt/run/sqltcompare.sql

ORACLE

44

### Import into COMPARE all SQLT tables from SOURCE1 and SOURCE2

```
# imp sqltxplain/<pwd> tables='sqlt$_stattab' file=sqlt_s9120.dmp ignore=y
# imp sqltxplain/<pwd> tables='sqlt$_statement' file=sqlt_s9120.dmp
ignore=y
# imp sqltxplain/<pwd> tables='sqlt$_plan_table' file=sqlt_s9120.dmp
ignore=y
# imp sqltxplain/<pwd> tables='sqlt$_parameter2' file=sqlt_s9120.dmp
ignore=y
# imp sqltxplain/<pwd> tables='sqlt$_indexes' file=sqlt_s9120.dmp
ignore=y
```

## Migrate CBO Stats related to one SQL

- Prerequisites
  - Install SQLT in SOURCE and TARGET
  - Execute SQLT on SOURCE (any method)
- Steps
  - Export CBO Stats from SOURCE (done by SQLT)  
# exp sqltxplain/<pwd> tables=sqlt% file=sqlt\_s9120.dmp  
statistics=none log=sqlt\_s9120\_exp.log
  - Import CBO Stats into TARGET  
# imp sqltxplain/<pwd> tables='sqlt\$\_statab'  
file=sqlt\_s9120.dmp ignore=y
  - Restore CBO Stats into Data Dictionary in TARGET  
SQL> START sqlt/utl/sqltimp.sql s9120\_kepler\_kepler null

ORACLE

45

**Methods XPLAIN, XTRACT and XECUTE automatically generate an export file that includes CBO statistics for all objects related to SQL.**

**Perform step 1 in SOURCE only in case you don't have file sqlt\_s3259.dmp already.**

**1. Export SQLT repository from SOURCE which includes CBO Statistics, connecting as SQLTXPLAIN.**

```
# exp sqltxplain/<pwd> tables=sqlt% file=sqlt_s3259.dmp  
statistics=none log=sqlt_s3259_exp.log
```

**2. Import CBO Stats generated in SOURCE into staging table in TARGET, connecting as SQLTXPLAIN.**

```
# imp sqltxplain/<pwd> tables='sqlt$_statab' file=sqlt_s3259.dmp  
ignore=y
```

**3. Restore the CBO Stats from staging table into the data dictionary, connecting as SQLTXPLAIN, SYSTEM, SYSDBA or the application user.**

```
SQL> START sqlt/utl/sqltimp.sql s3259_v1024_coesrv20 null
```

## Reproduce an Execution Plan from one system into another

- Prerequisites
  - SOURCE and TARGET are similar (release and patching level)
  - SOURCE and TARGET contain same schema objects
  - Execute SQLT in SOURCE (any method)
- Steps
  - Migrate CBO Stats related to one SQL
  - Set CBO environment in TARGET using SETENV script created in SOURCE
  - Use SQLT XPLAIN or XECUTE on TARGET

ORACLE

46

**1. Import into staging table in TARGET the CBO Stats generated by SQLT in SOURCE, connecting as SQLTXPLAIN.**

```
# imp sqltxplain/<pwd> tables='sqlt$_stattab' file=sqlt_s3259.dmp  
ignore=y
```

**2. Restore the CBO Stats from staging table into the data dictionary, connecting as SQLTXPLAIN, SYSTEM, SYSDBA or the application user.**

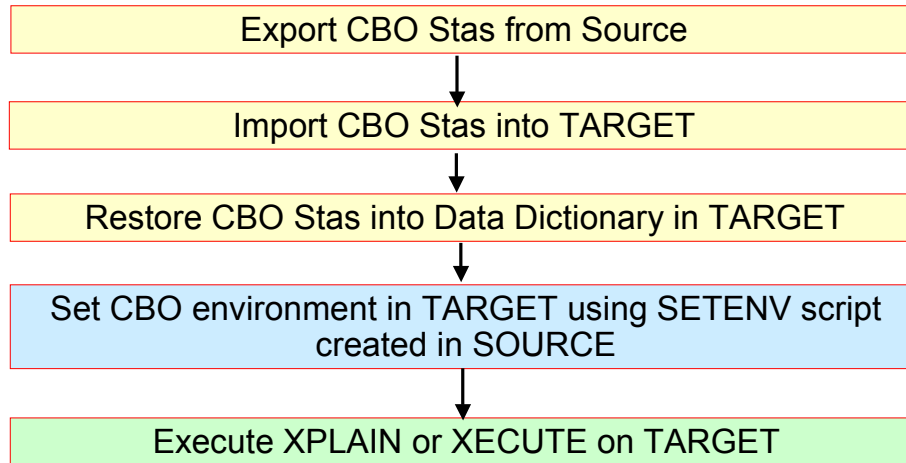
```
SQL> START sqlt/utl/sqltimp.sql s3259_v1024_coesrv20 null
```

**3. Review and set the optimizer environment environment, connected as the application user.**

```
SQL> START sqlt_s3259_v1024_coesrv20_setenv.sql
```

**4. Use SQLT XECUTE or XPLAIN methods, connected as the application user.**

## Reproduce an Execution Plan from one system into another



ORACLE

47

### Steps

- Migrate CBO Stats related to one SQL
- Set CBO environment in TARGET using SETENV script created in SOURCE
- Use SQLT XPLAIN or XECUTE on TARGET

## Create TC user and Schema Objects in similar system

- Prerequisites
  - Install SQLT in SOURCE and TARGET
  - Execute SQLT in SOURCE (any method)
- Steps
  - Customize metadata script from SOURCE
  - Execute customized metadata script in TARGET

ORACLE

48

### Steps:

1. Review "metadata" script generated in SOURCE and execute it into TARGET, connected as SYSTEM or SYSDBA.

```
SQL> START sqlt_s3259_v1024_coesrv20_metadata.sql
```

In most cases you want to consolidate all schema objects into one test case user, (for example TC3259).



## Customize metadata script from SOURCE before execution in TARGET

```
-- Restore System Statistics
EXEC DBMS_STATS.DELETE_SYSTEM_STATS;
EXEC DBMS_STATS.SET_SYSTEM_STATS('CPUSPEEDNW', 2104.202);
EXEC DBMS_STATS.SET_SYSTEM_STATS('IOSEEKTIM', 6.622);
EXEC DBMS_STATS.SET_SYSTEM_STATS('IOTFRSPEED', 13010.645);

-- Create "test case user"
DEF TC_USER = TC1234;
GRANT DBA TO &&TC_USER. IDENTIFIED BY &&TC_USER.;

-- Use DEF command(s) below if you want to consolidate objects into one test case user TC1234
DEF SCHEMA_APPS = &&TC_USER.;
DEF SCHEMA_BOM = &&TC_USER.;
DEF SCHEMA_MRP = &&TC_USER.;

-- Un-comment DEF command(s) below ONLY if you want to create objects into original owner(s).
-- DEF SCHEMA_APPS = APPS;
-- DEF SCHEMA_BOM = BOM;
-- DEF SCHEMA_MRP = MRP;
```

ORACLE

49

**Consolidate all schema objects into one TC\_USER (default)**

## Create a SQLT Test Case

- Prerequisites
  - SOURCE and TARGET are similar (release and patching level)
  - Execute SQLT in SOURCE (any method)
- Steps
  - Create TC user and Schema Objects into TARGET
  - Migrate CBO Stats related to one SQL
  - Set CBO environment in TARGET using SETENV script created in SOURCE
  - Execute XPLAIN or XECUTE on TARGET

ORACLE

50

**1. Review "metadata" script generated in SOURCE and execute it into TARGET**

```
SQL> START sqlt_s3259_v1024_coesrv20_metadata.sql
```

**2. Import CBO Stats generated in SOURCE into staging table in TARGET**

```
# imp sqltxplain/<pwd> tables='sqlt$_stattab' file=sqlt_s3259.dmp  
ignore=y
```

**3. Restore the CBO Stats from staging table into the data dictionary**

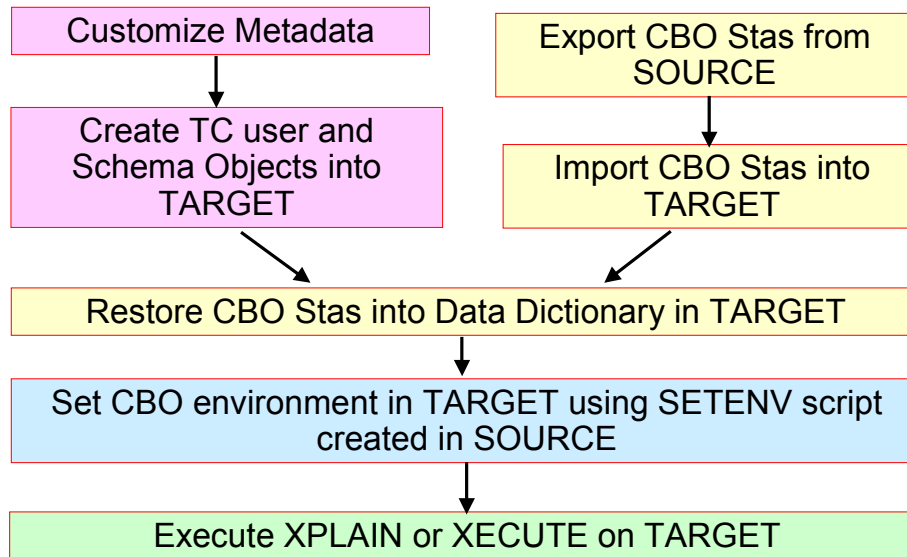
```
SQL> START sqlt/utl/sqltimp.sql s3259_v1024_coesrv20 TC3259
```

**4. Review and set the optimizer environment environment**

```
SQL> START sqlt_s3259_v1024_coesrv20_setenv.sql
```

**5. Use SQLT XECUTE or XPLAIN methods to reproduce desired plan**

## Create a SQLT Test Case



ORACLE

51

### Steps

- Create TC user and Schema Objects into TARGET
- Migrate CBO Stats related to one SQL
- Set CBO environment in TARGET using SETENV script created in SOURCE
- Execute XPLAIN or XECUTE on TARGET

## Create a stand-alone TC based on a SQLT TC

- Prerequisites
  - Create a SQLT Test Case
- Steps
  - Export into STATTAB.dmp the CBO Stats preserved in CBO\_STAT\_TAB\_4TC
  - Write a readme.txt file (sample provided)
  - Create a TC directory with STATTAB.dmp, readme.txt, metadata.sql, setenv.sql and file with one SQL
  - Test stand-alone TC following own readme.txt
  - Create TC.zip out of TC directory with 5 files on it.

ORACLE

52

### 1. Export CBO Stats captured automatically during step 5 of SQLT Test Case

```
# exp TC3259/TC3259 tables=CBO_STAT_TAB_4TC  
file=STATTAB.dmp statistics=none log=STATTAB.log
```

### 2. Write stand-alone Test Case instructions into a readme.txt file.

### 3. Create a TC3259 directory and place into it the following files:

- o CBO Stats dump file from step 1: STATTAB.dmp
- o Instructions file from step 2: readme.txt
- o Metadata script used in SQLT TC: sqlt\_s3259\_metadata.sql
- o CBO Set Env script from SQLT TC: sqlt\_s3259\_setenv.sql
- o Script with one SQL from SQLT TC: sample\_script0.sql

### 4. Fully test your stand-alone TC following your own readme.txt created in step 2.

### 5. ZIP stand-alone TC directory as TC3259.zip.

## Sample readme.txt for stand-alone TC based on a SQLT TC

```
-- create test case user TC1234 and schema objects:
# sqlplus / as sysdba;
SQL> START sqlt_s1234_metadata.sql;

-- import and restore cbo stats:
# imp TC1234/TC1234 tables=CBO_STAT_TAB_4TC file=STATTAB.dmp ignore=y
# sqlplus TC1234/TC1234
SQL> EXEC DBMS_STATS.IMPORT_SCHEMA_STATS('TC1234', 'CBO_STAT_TAB_4TC');

-- set cbo environment and generate 10053
# sqlplus TC1234/TC1234
SQL> START sqlt_s1234_setenv.sql;
SQL> ALTER SESSION SET TRACEFILE_IDENTIFIER = "TC1234_10053";
SQL> ALTER SESSION SET EVENTS '10053 TRACE NAME CONTEXT FOREVER, LEVEL 1';
SQL> DEF unique_id = TC1234
SQL> START file_with_sql;
```

ORACLE

53

**-- create test case user TC3259 and schema objects:**

**# sqlplus / as sysdba;**

**SQL> START sqlt\_s3259\_v1024\_coesrv20\_metadata.sql;**

**-- import and restore cbo stats:**

**# imp TC3259/TC3259 tables=CBO\_STAT\_TAB\_4TC file=STATTAB.dmp  
ignore=y**

**# sqlplus TC3259/TC3259**

**SQL> EXEC DBMS\_STATS.IMPORT\_SCHEMA\_STATS('TC3259',  
'CBO\_STAT\_TAB\_4TC');**

**-- set cbo environment and generate 10053**

**# sqlplus TC3259/TC3259**

**SQL> START sqlt\_s3259\_v1024\_coesrv20\_setenv.sql;**

**SQL> ALTER SESSION SET TRACEFILE\_IDENTIFIER =  
"TC3259\_10053";**

**SQL> ALTER SESSION SET EVENTS '10053 TRACE NAME CONTEXT  
FOREVER, LEVEL 1';**

**SQL> DEF unique\_id = TC3259**

**SQL> START sample\_script0.sql;**

## Gather a CBO Statistics Baseline

- Baseline without Histograms:

```
SQL> EXEC  
DBMS_STATS.GATHER_TABLE_STATS(ownname =>  
'QTUNE', tabname => 'CUSTOMER', estimate_percent => 100,  
method_opt => 'FOR ALL COLUMNS SIZE 1', cascade =>  
TRUE, no_invalidate => FALSE);
```

- Baseline with Histograms:

```
SQL> EXEC  
DBMS_STATS.GATHER_TABLE_STATS(ownname =>  
'QTUNE', tabname => 'CUSTOMER', estimate_percent => 100,  
method_opt => 'FOR ALL COLUMNS SIZE 254', cascade =>  
TRUE, no_invalidate => FALSE);
```

ORACLE

54

If suspecting poor schema object CBO statistics, use commands below to generate a new baseline. Then, test SQLT (any method) on your SQL.

Connect as SYSTEM, SYSDBA, or the application user.

### Baseline without Histograms:

```
SQL> EXEC DBMS_STATS.GATHER_TABLE_STATS(ownname =>  
'''QTUNE''', tabname => '''CUSTOMER''', estimate_percent => 100,  
method_opt => 'FOR ALL COLUMNS SIZE 1', cascade => TRUE,  
no_invalidate => FALSE);
```

### Baseline with Histograms:

```
SQL> EXEC DBMS_STATS.GATHER_TABLE_STATS(ownname =>  
'''QTUNE''', tabname => '''CUSTOMER''', estimate_percent => 100,  
method_opt => 'FOR ALL COLUMNS SIZE 254', cascade => TRUE,  
no_invalidate => FALSE);
```

## Transfer a Stored Outline

- Prerequisites
  - Remove from TARGET a prior Stored Outline  
SQL> DROP OUTLINE <stored\_outline\_name>;
  - Create an Stored Outline in SOURCE
- Steps
  - Export Stored Outline from SOURCE
  - Import Stored Outline into TARGET

ORACLE

55

If your SQL uses an Stored Outline, you can export the SO from SOURCE and import it into TARGET.

### Steps:

#### 1. Export Stored Outline from SOURCE

connecting as OUTLN, SYSTEM or SYSDBA.

```
# exp system/<pwd> tables=outln.ol% file=sqlt_s2398_outln.dmp
statistics=none query=\"WHERE ol_name = '<stored_outline_name>'\"
log=sqlt_exp_outln.log
```

#### 2. Import Stored Outline into TARGET

connecting as OUTLN, SYSTEM or SYSDBA.

```
# imp system/<pwd> file=sqlt_s2398_outln.dmp fromuser=outln
touser=outln ignore=y
```

### Notes:

1. If TARGET already contains a Stored Outline for your SQL, find its name and drop it before import step.  
Connect as OUTLN, SYSTEM or SYSDBA to drop an outline.  
SQL> DROP OUTLINE <stored\_outline\_name>;

## Transfer a SQL Profile

- Prerequisites
  - Create a SQL Profile on SOURCE
  - User must have CREATE ANY SQL PROFILE privilege and the SELECT privilege on staging table.
- Steps
  - Create staging table in SOURCE
  - Pack SQL Profile into staging table in SOURCE
  - Export staging table from SOURCE
  - Import staging table into TARGET
  - Unpack SQL Profile from staging table in TARGET

ORACLE

56

### 1. Create staging table in SOURCE

```
SQL> EXEC DBMS_SQLTUNE.CREATE_STGTAB_SQLPROF  
(table_name => 'STGTAB_SQLPROF', schema_name => USER);
```

### 2. Pack SQL Profile into staging table in SOURCE

```
SQL> EXEC DBMS_SQLTUNE.PACK_STGTAB_SQLPROF  
(profile_name => '<sql_profile_name>', profile_category => 'DEFAULT',  
staging_table_name => 'STGTAB_SQLPROF', staging_schema_owner  
=> USER);
```

### 3. Export staging table from SOURCE

```
# exp <usr>/<pwd> tables=stgtab_sqlprof file=sqlprof.dmp  
statistics=none log=sqlprof_exp.log
```

### 4. Import staging table into TARGET

```
# imp <usr>/<pwd> tables=stgtab_sqlprof file=sqlprof.dmp ignore=y  
log=sqlprof_imp.log
```

### 5. Unpack SQL Profile from staging table in TARGET

```
SQL> EXEC DBMS_SQLTUNE.UNPACK_STGTAB_SQLPROF  
(profile_name => '<sql_profile_name>', profile_category => 'DEFAULT',  
replace => TRUE, staging_table_name => 'STGTAB_SQLPROF',  
staging_schema_owner => USER);
```



## Create a Stored Outline

- Prerequisites
  - User must have CREATE ANY OUTLINE grant or DBA role.
  - Set your optimizer environment first
- Steps
  - SQL> ALTER SESSION SET create\_stored\_outlines = TRUE;
  - SQL> EXEC DBMS\_OUTLN.CREATE\_OUTLINE (hash\_value => <hash\_value>, child\_number => <child>);
  - SQL> ALTER SESSION SET create\_stored\_outlines = FALSE;
  - SQL> SELECT \* FROM DBA\_OUTLINES WHERE signature = '<stored\_outline\_signature>';

ORACLE

57

If you want to create an Stored Outline for your SQL, execute these commands connected as the application user:

```
SQL> ALTER SESSION SET create_stored_outlines = TRUE;
SQL> EXEC DBMS_OUTLN.CREATE_OUTLINE (hash_value =>
3750939470, child_number => 0);
SQL> ALTER SESSION SET create_stored_outlines = FALSE;
SQL> SELECT * FROM DBA_OUTLINES WHERE signature =
'109E470DBB49E51B0A3C6838E5BB7A56';
```

### Notes:

1. User must have CREATE ANY OUTLINE grant or DBA role.
2. Set your optimizer environment first (you may want to use the setenv script).

## **Extract a plan from memory or AWR and pin it to a SQL in same or different system**

- Prerequisites
  - 10.2.0.4 or later
  - Having executed SQLT XTRACT or XECUTE in SOURCE
- Steps
  - Execute sqlt/utl/sqltprofile.sql which generates another script
  - Execute generated script on same SOURCE or similar TARGET

ORACLE

58

**Plan is extracted from plan\_table.other\_xml from memory or AWR**

**Generated script calls DBMS\_SQLTUNE.IMPORT\_SQL\_PROFILE**

**If SQLT is not installed, use sqlt/utl/coe\_xfr\_sql\_profile.sql instead**

## Discover potential plans

- Toggles parameters and bug fix control
- Generates over 500 tests on 11g (less on 10g or 9i)
- Lists discovered plans and tests producing them
- Helps to narrow causes for a plan to go bad
- Use with no data or small data set (brute force)
- SOURCE → SQLT TC (XECUTE) → XPLORE

ORACLE

59

## Summary

- *SQLT (SQLTXPLAIN) is a tool that inputs one SQL statement and outputs a set of comprehensive diagnostic files for SQL performance analysis and tuning.*
- Use SQLT to expedite turnaround of SQL performance analysis and tuning issues
- If moving into process analysis and tuning, consider using Trace Analyzer (TRCA)

## References



ORACLE

61



## References

- SQLT (SQLTXPLAIN) – Note 215187.1
- Trace Analyzer (TRCA) – Note 224270.1