# Explain plan

Explain Plan command:

***EXPLAIN PLAN***

***[SET STATEMENT\_ID = 'statementid']***

***[INTO table\_name]***

***FOR sql\_statement***

By default**, table\_name = PLAN\_TABLE**

You can create your own plan table using the script **utlxplan.sql** located on **$ORACLE\_HOME/rdbms/admin**

Query plan table:

desc PLAN\_TABLE

**Method 1:**

*SELECT \* FROM PLAN\_TABLE after running EXPLAIN PLAN*

**Method 2**:

Use DBMS\_XPLAN package

SELECT \* FROM **TABLE(DBMS\_XPLAN.DISPLAY());** will not consider the effect of bind variable used in query

SELECT \* FROM **TABLE(DBMS\_XPLAN.DISPLAY\_CURSOR()**); TO SHOW THE ACTUAL EXECUTION PLAN USED BY THE DATABSE

You can identify the query and its SQL\_ID by using multiple dynamic performance view V$SQL and its respective execution plans called from **V$SQL\_PLAN** and **V$SQL\_PLAN\_STATISTICS**

**Check top 10 SQL Statements ordered by elapesed time:**

***SELECT sql\_id, child\_number, sql\_text, elapsed\_time***

***FROM (SELECT sql\_id, child\_number, sql\_text, elapsed\_time,***

***cpu\_time, disk\_reads,***

***RANK() OVER (ORDER bY elapsed\_time DESC) AS elapsed\_rank***

***FROM v$SQL)***

***WHERE elapsed\_rank <= 10;***

After we identify SQL then we can run

DBMS\_XPLAN.DISPLAY\_CURSOR for that SQL as:

***SELECT \* FROM TABLE(DBMS\_XPLAN.DISPLAY\_CURSOR('SQLID', 'CHILD\_NUMBER', 'TYPICAL'));***

# Tracing SQL Execution

Tracing Methods:

1) End-to-End application tracing

2) SQL Tracing

## 1) End-to-End application tracing:

* To enable tracing for client identifier, service, module, action, session, instance or database, execute the appropriate procedure available in DBMS\_MONITOR package.

* Enable tracing for specific diagnosis and workload management by:

- Tracing for Client Identifier

- Tracing for Service, Module, and Action

- Tracing for Session

- Tracing for Entire Instance or Database

* With the criteria, specific trace info is captured in a set of trace files and combined into a single output trace file.

### 1.1) Tracing for Client Identifier:

* The CLIENT\_ID\_TRACE\_ENABLE procedure enables tracing globally for the database for a given client identifier.
* for ex:

***EXECUTE DBMS\_MONITOR.CLIENT\_ID\_TRACE\_ENABLE(client\_id => 'OE.OE', waits=>TRUE, binds=>FALSE);***

-- OE.OE is a client identifier for which SQL tracing is to be enabled.

-- TRUE argument specifies that wait information will be present in the trace.

-- FALSE argument specifies that bind information will not be present in the trace.

* The CLIENT\_ID\_TRACE\_DISABLE procedure disables tracing globally for the databse for a given client identifier.
* for ex:

***EXECUTE DBMS\_MONITOR.CLIENT\_ID\_TRACE\_DISABLE(client\_id => 'OE.OE');***

### 1.2) Tracing for Service, Module, and Action:

* The SERV\_MOD\_ACT\_TRACE\_ENABLE procedure enables SQL Tracing for given combination of service name, module, and action globally for a database unless an instance name is specified in the procedure.

***EXECUTE DBMS\_MONITOR.SERV\_MOD\_ACT\_TRACE\_ENABLE(SERVICE\_NAME => 'ACCTG', MODULE\_NAME => 'PAYROLL', WAITS => TRUE, BINDS => FALSE, INSTANCE\_NAME => 'ORCL');***

-- Service ACCTG is specified.

-- The PAYROLL module. And for all actions.

-- TRUE argument specifies that wait information will be present in the trace.

-- FALSE argument specifies that bind information will not be present in the trace.

-- The ORCL instance is specified to enable tracing only for that specific instance.

* The SERV\_MOD\_ACT\_TRACE\_DISABLE procedure disables the trace at all enabled instances for a given combination of service name, module, and action name globally.

***EXECUTE DBMS\_MONITOR.SERV\_MOD\_ACT\_TRACE\_ENABLE(SERVICE\_NAME => 'ACCTG', MODULE\_NAME => 'PAYROLL', INSTANCE\_NAME => 'ORCL');***

### 1.3) Tracing for Session:

* The SESSION\_TRACE\_ENABLE procedure enables the trace for a given database session identifier(SID), on the local machine.
* To enable tracing for a specific session ID and serial number, determing the values for the session to trace:

***SELECT SID, SERIAL#, USERNAME FROM V$SESSION;***

***EXECUTE DBMS\_MONITOR.SESSION\_TRACE\_ENABLE(session\_id => 1234, serial\_num => 60, waits=>TRUE, binds=>FALSE);***

* The SESSION\_TRACE\_DISABLE procedure disables the trace for a given databse sid and serial number.

**EXECUTE DBMS\_MONITOR.SESSION\_TRACE\_DISABLE(session\_id => 1234, serial\_num => 60);**

* While the DBMS\_MONITOR package can only be invoked by a user with the DBA role, any user can also enable SQL tracing for their own session by using the DBMA\_SESSION package. A user an invoke the SESSION\_TRACE\_ENABLE procedure to enable session level SQL trace for the user's session.

* for ex:

***EXECUTE DBMS\_SESSION.SESSION\_TRACE\_ENABLE(waits => TRUE, binds => FALSE);***

* The SESSION\_TRACE\_DISABLE procedure disables the trace for the invoking session.
* for ex:

***EXECUTE DBMS\_SESSION.SESSION\_TRACE\_DISABLE();***

### 1.4) Tracing for Entire Instance or Database:

* The DATABASE\_TRACE\_ENABLE procedure enables SQL tracing for a given instance or an entire database. Tracing is enabled for all current and future sessions. for ex:

***EXECUTE DBMS\_MONITOR.DATABSE\_TRACE\_ENABLE(waits=>TRUE, binds=>FALSE, instance\_name => 'ORCL');***

-- the ORCL instance is specified to enable tracing for that instance.

* This example results in SQL tracing of all SQL in the ORCL instance.

* The DATABASE\_TRACE\_ENABLE procedure overrides all other session-level traces, but will be complementary to the client identifier, service, module and action traces.

* All new sessions will inherit the wait and bind information specified by this procedure until the DATABASE\_TRACE\_DISABLE procedure is called.
* When this procedure is invoked with the instance\_name parameter specified. then it will reset the session-level SQL trace for the named instance.

* If this procedure is invoked without the instance\_name parameter specified. then it will reset the session-level SQL trace for the entire database.

***EXECUTE DBMS\_MONITOR.DATABASE\_TRACE\_DISABLE(instance\_name => 'ORCL');***

* In this example, all session-level SQL tacing will be disabled for the ORCL instance.

* To disable the session-level SQL tracing for an entire database, invoke the DATABASE\_TRACE\_DISABLE procedure without specifying the instance\_name parameter.

***EXECUTE DBMS\_MONITOR.DATABASE\_TRACE\_DISABLE();***

## 2) SQL Tracing:

* The SQL Trace facility provides performance information on individual SQL Statements. It generates the statistics for each statement.

- Parse, execute and fetch counts

- CPU and elapsed times

- Physical reads and logical reads

- Number of rows processed

- Misses on the library cache

- Username in which each parse occured

- Each commit and rollback

- Wait event data for each SQL Statement and summary for each trace file.

* In case the cursor for the SQL Statement is closed, the SQL Trace also provides row source information that includes:

- Row Operations showing the actual execution plan of each SQL statement.

- Number of rows, number of consistent reads, number of physical reads, number of physical writes, and time elapsed for each operation on a row.

### 1.1) SQL Tracing:

Enable tracing from within your session.

***ALTER SESSION SET SQL\_TRACE=TRUE***

This creates a basic trace, in which SQL statement execution statistics and execution plans are recorded but not the values of bind variables or the time spent waiting for various events.

To get a more advanced trace, we can use DBMS\_SESSION package:

***DBMS\_SESSION.SESSION\_TRACE\_ENABLE(***

***waits IN BOOLEAN DEFAULT TRUE,***

***binds IN BOOLEAN DEFAULT FALE,***

***plan\_start IN VARCHAR2 DEFAULT NULL --11G ONLY***

***);***

For instance, plan\_start => 'all\_execution'

# Identifying the TRACE file:

* It can be hard to identify individual trace files. One way to make it easier is to specifya trace file identifier for your session.
* This can be done setting the TRACEFILE\_IDENTIFIER parameter from within your session.

***ALTER SESSION SET TRACEFILE\_IDENTIFIER = PT;***

Now, when we look in the trace file directory, the trace file can be identified by the trailing "PT".

## To identify if tracing is active or not for a session, use following query:

***SELECT s.sql\_trace, s.sel\_trace\_waits, s.sql\_trace\_binds, traceid, tracefile***

***FROM V$session s***

***JOIN v$process***

***ON p.addr = s.paddr***

***WHERE audsid = USERENV('SESSIONID');***

## Invoking trace in another session:

***DBMS\_MONITOR.SESSION\_TRACE\_ENABLE(***

***session\_id IN BINARY\_INTEGER DEFAULT NULL,***

***serail\_num IN BINARY\_INTEGER DEFAULT NULL,***

***waits IN BOOLEAN DEFAULT TRUE,***

***binds IN BOOLEAN DEFAULT FALE,***

***plan\_start IN VARCHAR2 DEFAULT NULL --11G ONLY***

***);***

# TKPROF

TKPROF for trace analysis

* TKPROF accepts as input a trace file produced by the SQL trac facility and it produces a formatted output file.
* TKPROF can also be used to generate execution plans.

After the SQL trace facility has generated trace files, you can:

* + Run TKPROF on each individual trace file, producing several formatted output files, one for each session.
  + Run the **trcsess** command line utility to consolidate tracing information from several trace files, the run TKPROF on the result.
  + TKPROF does not report COMMITs and ROLLBACKs that are recorded in the trace file.

Practical:

***> show parameter user\_dump;***

***> ALTER SESSION SET tracefile\_identifier=PT;***

***> ALTER SESSION SET SQL\_TRACE = TRUE;***

***> RUN QUERY : SELECT \* FROM ABC;***

***> ALTER SESSION SET SQL\_TRACE = FALSE;***

***> SELECT s.sql\_trace, s.sel\_trace\_waits, s.sql\_trace\_binds, traceid, tracefile***

***FROM V$session s***

***JOIN v$process***

***ON p.addr = s.paddr***

***WHERE audsid = USERENV('SESSIONID');***

***> EXIT***

***$ cd <location of trace file specified by user\_dump>***

***$ tkprof <filename.trc> <outputfilename>.txt***

***$ ls -ltr***

# AUTOTRACE

Generate execution plan and execution statistics for each SQL statement executed.

The output is not as definitive or extensive as that provided by SQL Trace but provides a good high level view of SQL Performance.

When AUTOTRACE is in effect, an explain plan and/or execution statistics will be printed after every SQL statement execution.

**AUTOTRACE** takes following options:

***SET AUTOTRACE {OFF| ON | TRACE[ONLY]} [EXPLAIN] [STATISTICS]***

-- OFF: Turn off AUTOTRACE output

-- ON: Turn on AUTOTRACE output

-- TRACEONLY: Suppress the output from queries; display the AUTOTRACE output only.

-- EXPLAIN: Generate execution plan only

-- STATISTICS: Generate execution statistics only.