# DBMS\_XPLAN cheat sheet

|  |  |
| --- | --- |
| Method #1: EXPLAIN PLAN explain plan [ set statement\_id = '*xxx*' ] for *SELECT\_or\_DML\_query*;  This computes the plan of the target query (without running it), and inserts the details of the  plan into the PLAN\_TABLE; then:  select \* from table(dbms\_xplan.display('PLAN\_TABLE', '*xxx*', '*display\_fmt*'));  Or simply, if not using a statement identifier:  select \* from table(dbms\_xplan.display(format => '*display\_fmt*')); Requirements  * Permissions to run the target SELECT or DML statement * If using views: READ or SELECT privilege on underlying tables (otherwise ORA-01039 is raised)  Display formats  * Default: typical; but use at least: +alias (= 'typical +alias') * Preferred: all -projection (this adds the “Hint Report” section if DB ≥ 19c) * Append: +outline, or use: advanced -projection to show the outline data; if using the latter, also add: -qbregistry if DB ≥ 19c, to turn off the Query Block Registry section * Append: +projection if you need the Column Projection section * Add: +adaptive to fully show adaptive plans, if the DB is configured to use them * DB ≥ 23c: the new “SQL Analysis Report” section may be turned off if needed, by appending -sql\_analysis\_report  Limitations  * explain plandoes not use *bind peeking*, therefore it may generate plans which differ from actual plans if *bind variables* are being used, and column histograms exist * DBAs beware: the MERGE ANY VIEW privilege (included in the DBA role) causes *secure view merging* to be bypassed, creating opportunities for explain plan to generate plans unavailable to users subjected to *secure view merging* * explain plan always uses the latest (published) statistics—possibly causing it to not reproduce plans created earlier in the cursor cache  Notes  * The (estimated) Bytes, Rows, Cost, and Time columns are *per-execution* of the concerned plan line operation (on the contrary, the columns Starts, A-Rows, A‑Time, Buffers, and Reads, are *cumulative*) * Caution: explain plan *inserts* rows into the PLAN\_TABLE, which means that if you didn’t have an on-going transaction *before* explain plan, you have one *after*. | Method #2: DBMS\_XPLAN.DISPLAY\_CURSOR *SELECT\_or\_DML\_query*;  … then retrieve the *sql\_id* + *child\_number* of the cursor, and then:  select \* from table(dbms\_xplan.display\_cursor('*sql\_id*', *child\_number*,'*display\_fmt*'));  Special case: latest cursor in this session (†) (‡)  select \* from table(dbms\_xplan.display\_cursor(null, null, '*display\_fmt*')); Requirements  * READ / SELECT privilege on V$SQL, V$SQL\_PLAN\_STATISTICS\_ALL   (†) Additional privilege required: READ / SELECT on V$SESSION  (‡) set serveroutput off is needed for this to work as expected Display formats  * Same formats as in dbms\_xplan.display * Use: +peeked\_binds (or advanced) to view the values, if any, of binds peeked at parse time * Add: +iostats | memstats | allstats [ all | last ] to print *actual* execution statistics:   + all | last: show statistics for all executions / for the last execution (default: all) Remark: statistics from *on‑going* executions are not available   + +memstats: adds the 0Mem, 1Mem, Used-Mem (or 0/1/M) columns   + +iostats: adds the Starts, A-Rows, A-Time, Buffers, Reads, and Writes columns   + +allstats = both iostats and memstats   Note: +iostats requires either:   * alter session set statistics\_level = all; caution: high overhead due to timing code, making queries possibly slower—for tests only!   Or:   * the /\*+ gather\_plan\_statistics \*/ hint, which incurs lower timing overhead, but can result in the A‑Time column being (very) inaccurate  Limitations  * The security requirements make this a privileged operation: users with the required permissions may see every plan (and every query) database wide * In some cases, dbms\_xplan.display\_cursor does not correctly render complex predicates, yielding meaningless expressions such as: filter( IS NULL), access("COL"=PRIOR NULL), etc. This is a limitation in V$SQL\_PLAN\_STATISTICS\_ALL, which cannot correctly render such predicates—only explain plan can * +allstats last does not function correctly with parallel queries: execution statistics from PX processes are not added to those of the QC; workaround: use +allstats (all implicitly)  Reminder: always pay attention to the Notes section, if any |
| Plan table columns  | Column Name | Description | | --- | --- | | Id | Identifying number of the operation in the plan; in the case of adaptive plans, use +adaptive to display the “true” plan line identifiers from the full (unresolved) plan | | Operation | Name of the operation, along with its execution options | | Name | Name of the object that is read from / written into | | Starts | Actual number of times that this operation was started; special case: number of times that the memory structure of the 2nd child of a MERGE JOIN was probed | | Rows / E-Rows | Estimated number of rows to be produced (note: *per execution*) | | Bytes / E-Bytes | Estimated number of bytes to be produced (note: *per execution*) | | TempSpc / E-Temp | Estimated amount of temp. storage required for this operation, in bytes (note: *per execution*) | | Cost (%CPU) | Estimated “cost” of the operation & percentage of CPU time thereof, if available (note: *per execution*) | | Time / E-Time | Estimated duration of the operation, in hours:minutes:seconds  (note: *per execution*); remark: values are usually inaccurate if system statistics have not been properly configured | | Pstart / Pstop | Partition number of the 1st / last partition iterated upon in this operation, if pre-determined; otherwise the method of determination to be used at runtime, e.g.: KEY, KEY(I), KEY(SQ), :BFnnnn, etc. | | TQ | In parallel queries, identifier of the DFO (*Data Flow Operation*) that this operation belongs to; it is also the name of the “Table Queue” that PX servers running this operation write into | | IN-OUT | In parallel queries, relationship between this operation and related operations | | PQ Distrib | In parallel queries, method of distributing rows in inter-group communications | | A-Rows | Actual number of rows produced by this operation (*cumulative*) Note: for operations on bitmap indexes, this is *not* the count of table rows! | | A-Time | Actual time spent running this operation and its child operations (*cumulative*) Keep in mind that statistics\_level = all causes distortion due to time-measurement code overhead, whereas the alternative, the /\*+ gather\_plan\_statistics \*/ hint, may return false durations | | Buffers | Actual count of logical blocks read by this operation, either from the buffer cache, or from datafiles if doing direct path reads (*cumulative*) | | Reads | Actual count of blocks physically read, if any, by this operation (*cumulative*) | | Writes | Actual count of blocks written, if any, by this operation (*cumulative*) | | 0Mem | Work area size required for running this operation fully in memory; unit: bytes | | 1Mem | Work area size required for running this operation using a one-pass (memory + temp. space) algorithm; unit: bytes | | Used-Mem | Actual amount of memory used in the last execution of this op.; unit: bytes | | 0/1/M | Number of times that this operation was run, respectively, fully in memory, using a one-pass algorithm, or using a multi-pass algorithm (*cumulative*) | | Used-Tmp | Amount of temporary storage used when performing this operation; unit: bytes (*cumulative*); note: the unit is *kilobytes* in DB versions ≤ 12.2 | | Max-Tmp | Maximum amount of temp. storage used when performing this operation; unit: bytes (*cumulative*); note: the unit is *kilobytes* in DB versions ≤ 12.2 | | Plan table columns (continued)  | Column Name | Description | | --- | --- | | Inst | In distributed queries, name of the DB link used by this operation |  Parallel QueriesInput / Output  | IN-OUT | Description | | --- | --- | | P->S | Parallel to Serial: this parallel operation sends rows to the QC (*query coordinator*) of its DFO tree | | S->P | Serial to Parallel: this serial operation sends rows to a parallel operation (remark: this could be inefficient, due to this serial operation possibly acting as a bottleneck) | | P->P | Parallel to Parallel: this parallel operation sends rows to another parallel operation | | PCWC | Parallel combined with Child: each PX process in this operation receives rows from its child operation in the same process (no inter-process communication happens) | | PCWP | Parallel combined with Parent: each PX process in this operation sends rows to its parent operation in the same process (no inter-process communication happens) | | SCWC | Serial combined with Child: this serial operation runs in the same process as its child operation, which it receives rows from (no inter-process communication happens) | | SCWP | Serial combined with Parent: this serial operation runs in the same process as its parent operation, which it returns rows to (no inter-process communication happens) |  Distribution Methods  | PQ Distrib | Description | | --- | --- | | QC (RAND) | The QC (query coordinator) is being sent rows with no ordering | | QC (ORDER) | The QC (query coordinator) consumes rows in order, beginning with rows from the first PX process, and ending with rows from the last PX process | | BROADCAST | Every producer sends all its rows to every consumer | | BROADCAST LOCAL | RAC-only: same as Broadcast, but producers send to consumers in the local instance | | RANGE | Every producer sends each row to a *single* consumer, based on a range condition: rows in the same range are sent to the same consumer | | RND-ROBIN | Each producer sends each row to one consumer in turn (resulting in rows being evenly distributed among consumers) | | HASH | Every producer sends each row to a *single* consumer, based on a hash function; rows with the same hash value are sent to the same consumer | | HYBRID HASH | Producers send rows to consumers using either Broadcast or Hash, the choice between the two being determined at runtime (DB ≥ 12.1) | | PART (ROWID) | Rows are sent to consumers based on the partitioning of a table/index, using the rowid of the row to update/delete | | PART (KEY) | Rows are sent to consumers based on the partitioning of a table/index, using the partition key | | RANDOM LOCAL | A variation of PART (KEY) in which *sets* of consumer processes are maintaining *sets* of partitions | | 1 SLAVE | All rows are sent to a single process in the consumer group (DB ≥ 12.1) | |