Oracle Interview Question and Answer

**1 Different Oracle Versions**

Let’s now examine the various versions that Oracle Corporation has released for the Oracle database:

**1. OracleV2:**This was the first commercially available SQL RDMS launched by Oracle in the market. The launch took place in 1979, and the letter ‘V’ represents the version while ‘2’ signifies the second version. The version was very basic if we are going to compare now. It was only able to run some SQL queries and perform a few join operations.

**2. OracleV3:**This was the second launch in the year 1983. It brought many new features like concurrency control and scalability.

**3. OracleV4:**This was released in the market in the year 1984, and it had all the features of the previous versions. However, one noted addition was that it was compatible with the MS-DOS operating system developed by Microsoft.

**4. OracleV5:**This was released in the year 1985 and it introduced client-server computing and [**distributed database systems**](https://www.educba.com/distributed-database-system/), which means not all storage devices are attached to a standard processor. It may be stored in multiple computers which are located in the same physical location. This release was a very important breakthrough.

**5. OracleV6:**It was released in 1988, and for the first time, row-level locking, online backup, and recovery were introduced in this release.

**6. Oracle7:**This was released in the year 1992, and it was an important release because it was this release that introduced PL/SQL, that is, Procedural SQL stored procedures so that it increases the reusability, triggers if we need to do any operation based on any insertion or updation, shared cursors. It also introduced a parallel server option, which supports simultaneous database access from two or more loosely coupled systems. It helps to achieve high performance

**7. Oracle8:**It was first released in June 1997. Later on, 8i was released, where ‘i’ means the internet. The 8i was released in the year 2000. The important features of this release were Native Internet protocols and Java. It has PLSQL Gateway introduced for deploying applications on the Web. XML parser for Java. The SQL CASE statement, similar to the DECODE() function, introduced it.

**8 Oracle9i:**This version was released in the year 2001. Here also, the’ i’ stands for the internet. It introduced Oracle RAC, which stands for Oracle Real Application Clusters (RAC). Oracle RAC provides software for clustering and high availability in Oracle database environments. It includes clustering by allowing multiple computers to run Oracle RDBMS simultaneously while accessing a single database. It also brought out data mining and data analysis algorithms for analytics in its 9.2.0.8 release in 2007.

**9. Oracle10g Release1:**The Oracle 10g was first released in 2003. Here, the letter ‘g’ indicates the grid. This feature was introduced first in Oracle 10g. Grid computing provides an adaptive software infrastructure that makes use of modular storage and low–risk servers. This helps balance the overall workload and provides capacity on demand. It also had new features like an Automated Database management system, Grid infrastructure, and Oracle ASM.

**10. Oracle 10g Release2: The second release** was released in July 2005. Several more features were added. The features are Real Application Testing, Online Indexing, and Transparent Data Encryption.

**11. Oracle 11g Release1:**The Oracle 11g was released in September 2007. Like in 10g, the ‘g’ here also stands for grid. This release also introduced a lot of features like SQL Tuning. Active data guard feature. This feature is an extension of the RDMS. It provides GUI (Graphical User Interface) and CUI (Command User Interface) for managing data guard features.

**12. Oracle 11g Release2:**This was released in August 2013. The new release added features such as Hybrid Columnar Compression, Cluster File System, and Database Appliance. The Oracle Database Appliance feature consists of a single-box device that contains hardware, networking, software, and storage to build a clustered database server.

**13. Oracle12c Release1:**Oracle 12c was released in July 2013. The ‘c’ here stands for the cloud. This is the first database offered to buy Oracle with cloud computing. It has a multitenant architecture that simplifies consolidation without doing any changes in the application. It also has an In-Memory column store, SQL Pattern Matching, and native JSON.

**14. Oracle 12c Release2:**It was released in September 2016. It retained the earlier features but some features like Exadata Cloud Service and Cloud at Customer.

**15. Oracle 18c:**Like the previous release here, ‘ c’ stands for the cloud. This version was released in the cloud in February 2018. It is the next iteration of Oracle 12c. It contains a lot of improvements. It contains no drastic new features like 12c had over 11g but only incremental improvements. The improvements come by features like Polymorphic Table Functions.

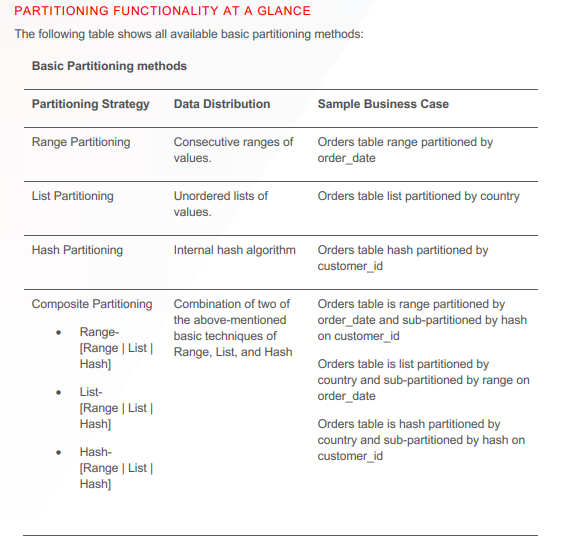
**16. Oracle 19c:**It was released on the cloud in April 2019 for Linux and other platforms and in June 2019. The features included Automated Index Creation, SQL Queries on Object stores, and Real-time statistics maintenance.

**17. Oracle 21c:**An innovation release of the Oracle Database, known as Oracle Database 21c, was accessible in September 2021. It introduces features including enhanced security, native JSON support, automatic indexing, blockchain tables, automated machine learning, and native JSON support. It also improves in-memory processing, multitenant architecture, and native persistent memory support.

**18. Oracle 23c:**Oracle Database 23c is Oracle’s latest database management system version. Its comprehensive features and tools enable secure and efficient management and storage of data. The “c” in 23c denotes Oracle’s emphasis on cloud-based database management solutions. This database version optimizes cloud use with features like automatically tuning database performance, scaling resources, and enhancing security. It also supports new technologies like blockchain, machine learning, and IoT. Overall, Oracle Database 23c is a powerful and versatile tool for managing data in the cloud.

**2 Oracle Partitioning**

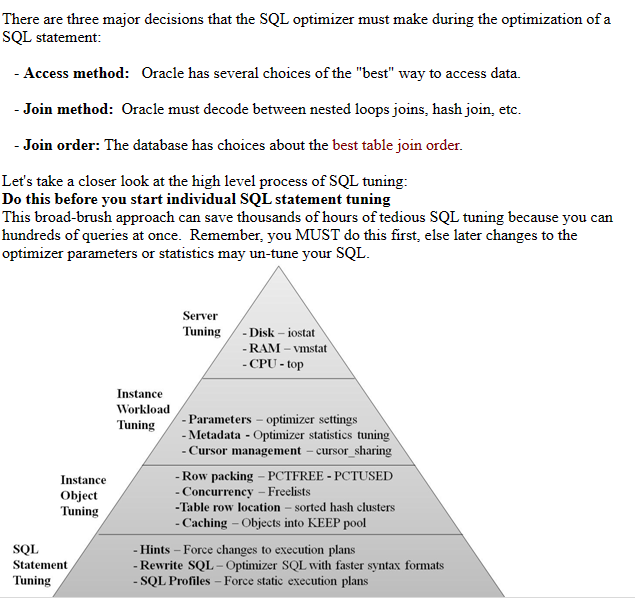
Partitioning enables tables and indexes to be subdivided into individual smaller pieces. Each piece of the database object is called a partition.

****

**A screenshot of a document

Description automatically generated**

**Oracle SQL tuning - Tune individual SQL statements**



# Oracle SQL Performance Tuning Cheatsheet

[Oracle SQL Performance Tuning Cheatsheet (github.com)](https://gist.github.com/kzhangkzhang/2866c2530c5a0ec337c475879eecabc0#file-performancetuning_cheatsheet-md)

F:\WorkingKnowles\Oracle\Utility\12i\PT\PerformanceTuning\_CheatSheet.md

## Reference

* [\*\* How To Fetch Multiple Oracle Execution Plans in One Nice Query](https://blog.jooq.org/2017/03/31/how-to-fetch-multiple-oracle-execution-plans-in-one-nice-query/)
* [Oracle Dev Gym class - Databases for Developers: Performance](https://devgym.oracle.com/pls/apex/f?p=10001:29::::RP:P29_CLASS_ID:8481)
* [White Paper -Oracle 12c Optimizer Explain the Explain Plan](https://www.oracle.com/technetwork/database/bi-datawarehousing/twp-explain-the-explain-plan-052011-393674.pdf)
* [Youtube - How to read an execute plan](https://www.youtube.com/watch?v=8hr3oN_YK0U)
* [Blog - How to create an execution plan](https://blogs.oracle.com/sql/how-to-create-an-execution-plan)
* [Youtube - Interpret Oracle Explain Plan](https://www.youtube.com/watch?v=AhKgGRM_piY)
* [\*\*\* Performance Tuning Series - Expert Oracle](http://expertoracle.com/2017/11/15/db-tuning-basics-1-selectivity-and-cardinality/)

## Common Command

### Flush Shared Pool, Buffer Cache

alter system flush shared\_pool;

alter system flush buffer\_cache;

### Execution Plan Golden step

* do this when tuning any SQL
* **DON'T** forget turn off serveroutput

set serveroutput off

set linesize 400

-- ================================================

-- Speical Instruction for using bind variable

-- ================================================

-- Skip Below if you don't use bind variable

-- if need using bind variable (e.g. APEX query)

variable app\_session number;

variable app\_id number;

BEGIN

:app\_session := 999;

:app\_id := 666;

END;

/

-- end of special instruction for bind variables

SELECT /\*+ gather\_plan\_statistics \*/ ...

FROM ...;

select \* from table(dbms\_xplan.display\_cursor(format => 'ALLSTATS LAST +COST'));

select \* from table(dbms\_xplan.display\_cursor(format => 'ALLSTATS LAST +COST +BYTES'));

### SQL Developer

-- run with hint to turn on monitoring for short(quick run) SQL

select /\*+MONITOR\*/ \* FROM apps.oe\_order\_lines\_all;

-- turn off monitoring for SQL

select /\*+NO\_MONITOR\*/ \* FROM apps.oe\_order\_lines\_all;

## General Tip

* CARDINALITY \* LAST\_STARTS ~ LAST\_OUTPUT\_ROWS
  + if roughly in the same magnitude, optimizer most likely choose the good plan
* column correlation and data skew can cause optimizer choose wrong plan
  + column correlation ==> creates extended statistics
  + data skew
    - 2 forms: range skew and value skew
    - creates histogram
      * Frequency
      * Height-balanced
      * Hybrid
      * Top-Frequency
* Make sure table stats is collected, especially after you do a truncate/delete and import data for the table
* If there is a corelation among columns, create **EXTENDED STATS** to help optimizer to estimate # of rows
* if data is skew (uneven distribution), create histogram
* Inefficient access
  + creating index
* Repeating Work
  + rewrite the SQL
* Reading Many Rows returning Few
  + creating materialized view (enable query rewrite)
* Index tips
  + keep # indexes small
  + when creating multiple column indexes
    - the leading column of the index should be the one most likely to be used in WHERE clauses and also the most **selective** column of the set
    - **compressible** columns first
* Use hints as temporary solution or last option (if you run out of clue)
  + Some common hints:
    - INDEX, ORDERED, USE\_HASH, PARALLEL, ALL\_ROWS, FIRST\_ROWS, APPEND, CACHE,RESULT\_CACHE
* Trigger on tables can cause performance issue for DML (insert/update/delete)

## General Concept

### What is a good plan for the Optimizer

The optimizer has 2 different goals

* Serial execution: it is all about cost
  + the cheap, the better
* Parallel execution: it is all about performance
  + the faster, the better

| **Term** | **Description** |
| --- | --- |
| Cost | - The optimizer's cost model accounts for the **IO, CPU** and **network resources** that will be used by the query - Estimate of amount of CPU, disk I/O, used to perform an operation - The cost is an **internal unit** and is only displayed to allow for plan comparisons - Used for compare different plans for the same query |
| Performance | - it is fastest possible response time for a query - Goal is to complete the query as quickly as possible - Optimizer does not focus on resources needed to execute the plan |
| AWR | Automatic Workload Repository |
| STS | SQL Tuning Set |
| SPM | SQL Plan Baseline |
| ROWID | Datafile ==> Data block ==> location of row |
| ASSM | Automatic Segment Space Management tablespace type, which is designed to avoid contention by concurrent inserts from different sessions |

## What to Look For in Execution Plan

* Full table scan (FST)
* Join Methods, Join Order
* Index Access Methods
* Filters
* Parallel Operations
* Partition Processing
* Dynamic Statistics
* Cost

## Execution Plan Component

| **Component** | **Description** |
| --- | --- |
| Cardinality | - **Estimate** of the number of rows coming out of each of the operations - Cardinality for a single column eqality predicate = (total # of rows)/(num of distinct values) - It influences everything: access method, join type, join order etc |
| Access method | the way in which the data is being accessed, via either a table or index access |
| Join method | the method (e.g. hash, sort-merge etc) used to join tables with each other |
| Join type | the type of join (e.g. outer, anti, semi etc) |
| Join order | the order in which the tables are joined to each other |
| Partition pruning | are only the necessary partitions being accessed to answer the query? |
| Parallel Execution | In case of parallel execution, is each operation in the plan being conducted in parallel? Is the right data redistribution method being used? |

### Data Access Method

Ref: <https://techiedba.wordpress.com/2011/08/12/data-access-methods-in-oracle/>

* Full Table Scan (FTS)
* Table Access by ROWID
* Index Unique Scan
* Index Range Scan
* Index Full Scan
  + In certain circumstances it is possible for the whole index to be scanned where no constraining
  + it will perform **single** block i/o's and so it may prove to be inefficient
* Index Fast Full Scan
  + Scans all the block in the index Rows are not returned in sorted order
  + Use **multiblock** i/o and can be executed in parallel
* Index Skip Scan
  + The optimizer can perform skip scans to retrieve rowids for values that do not use the prefix of a concatenated index
  + Initiated by probing the index for distinct values of the prefix column. Each of these distinct values is then used as a starting point for a regular index search
* Index Joins
* Hash Access
* Cluster Access
* Bit Map Index

### Join Method

Ref: <https://www.youtube.com/watch?v=pJWCwfv983Q>

* Hash join
* Nested Loop
* (Sort) Merge join
* Cartesian join

Nested loop join is better for joining small result sets where the joni columns are indexed Hash join is better for joining larger result sets where once or more indexes are missing on the join columns (Sort) Merge join is better than nested loop joins for joining larger result sets but typically not as good as hash joins

| **Job Method** | **Description** |
| --- | --- |
| Hash join | - Used for joining **large data sets**. - The optimizer uses the **smaller** of the two tables or data sources to **build a hash table**, based on the join key, **in memory**. -It then scans the larger table, and performs the same hashing algorithm on the join columns(s). It then probes the previously built hash table for each value and if they match., it returns a row |
| Nested Loop | -It is useful when **small subsets** of data are being joined and if there is an efficient way of accessing the 2nd table (for example, an index lookup). -For every row in the first table (the outer table), Oracle access **all the rows** in the second table (the inner table) |
| Sort Merge | - It is useful when the join condition between two tables is an **in-equality** condition such as, <, <=, > or >=. - Sorge merge joins can perform better than the nested loop joins for large data sets. - A sort merge join is more likely to be chosen if there is an index on one of the tables that will eliminate one of the sorts |
| Cartesian join | - The optimizer joins every row from one data source with every row from the other data source. - Typically this is only chosen if the tables involves are small or if one or more of the tables doesn't not have a join conditions to any other table in the statement |

### Nested Loop

* First we return all the rows from row source 1 (Driving Table), typically the smaller of the two row sources
* Then we probe row source 2 (Inner Table) once for each row returned from row source 1
* Good for joining smaller row sources
* Best used within indexed foreign key columns

The optimizer may choose nested loops when:

* Joining small data sets
* Fetching a small fraction of large data sets
* The query of the inner table is efficient (e.g. there is an index on its join columns)
* All the join criteria are inequalities (not equals)

### Hash Join

The steps to do a hash join are:

1. Return all the rows from the smaller data set
2. Build a hash table using the join columns for these rows
3. Read the rows in the larger table
4. Probe the hash table built at step 2 by applying the same hash function to the join columns of the second table
5. If this finds a matching entry in the hash table, the database passes the joined rows to the next step of the plan

* Smallest row source is chosen and used to build a hash table (in memory) and a bitmap
* The second row source is hashed and checked against the hash table looking for joins. The bitmap is used as a quick lookup to check if rows are in the hash table
* Good for joining larger row sources
* Needs PGA memory

There **must** be equality "=" conditions in the join for the optimizer to choose a hash join. When there is, it may choose this join method when joining either:

* Large data sets OR
* Most of the rows in small tables OR
* There are no indexes on the join columns for either table

### Hash Join vs. Nested Loop

This highlights a key difference between hash joins and nested loops. A hash join must read all the rows in the first data set to build the hash table. Then start reading the second table.

Nested loops can read rows in the inner table after reading just one row from the outer table. Provided the lookup of the inner table is fast, this means it can start returning rows faster than a hash join.

This makes nested loop operations the best way to get a small fraction of large data sets. For example, when doing top-N queries or master-detail joins, such as orders to order items.

### Merge Join

* Sort the rows in the first data set
* Sort the rows in the second data set
* For each row in the first data set, find a starting row in the second data set
* Read the second data set until you find a row with a value greater than the current row in the first data set
* Read the next row in the first data set and repeat

The optimizer may choose a merge join when:

* The join uses range comparisons (<, >=, between, etc.)
* The join has an equality (=) and one of the data sets is already ordered, enabling the database to avoid a sort
* Sorting is slow, so it's rare for the optimizer to use a merge join.

Indexes are ordered data structures. So if there is an index on the join columns for one table, the optimizer can use this to avoid a sort. Oracle Database will always sort the second data set, even if it there are indexes on the second table's join columns.

### Merge Join Cartesian

The optimizer considers a merge join Cartesian if:

* It's joining every row in one table to every row in another (a cross join)
* It expects to get exactly one row from the first table and join it to every row in the second table
* It may generate the Cartesian product of two tiny tables before joining these to a huge table

It's rare to see this join method. And it's often a sign of a problem. For example missing join criteria or out-of-date statistics.

### Adaptive Plan

* It can be tough to choose between nested loops and hash joins. Provided there is an index on the join columns, nested loops are fast when you fetch few rows from the first table.
* But nested loops query the inner table once for each row from the outer table. As you fetch more rows from the first table, you quickly reach a point where it's faster to use a hash join. This fetches all the rows from the second table once.
* To handle this, Oracle Database 12c added adaptive plans. This is one plan which considers two different join methods. You can see this query uses an adaptive plan by looking at the Note section:
* Full dynamic plan displayed when format parameter '+all\_dyn\_plan' is used
* New adaptive optimization section shown when format parameter '+adaptive' is set

Note

-----

- this is an adaptive plan

select \* from table(dbms\_xplan.display\_cursor(format => '+all\_dyn\_plan +adaptive'));

The database can then choose the best join method at runtime. It does this by looking at the actual number of rows the query processes.

To see which joins the optimizer considered, get the plan with the +ADAPTIVE format:

select /\*+ gather\_plan\_statistics \*/\*

from card\_deck d1

join card\_deck d2

on d1.val = d2.val

where d1.notes = 'SQL is awesome!';

select \*

from table(dbms\_xplan.display\_cursor(:LIVESQL\_LAST\_SQL\_ID, format => 'IOSTATS LAST +ADAPTIVE'));

This has some key differences to a standard plan:

* The plan includes both NESTED LOOPS and HASH JOIN operations
* There is a STATISTICS COLLECTOR operation above the first table in the plan
* A minus ('-') appears before some of the steps in the plan

#### Statistics Collector

* This operation watches the number of rows flowing out of the table below it. Provided this number stays under a threshold, the optimizer will use nested loops. But if the number of rows exceeds this threshold, it'll switch to a hash join.

### Join Type

* Inner join
* Left/right outer join
* Full outer join
* Anti join
* Semi join
* Grouped outer join

### Filters

* Restrictions in the WHERE clause
* Optimizer will try to filter rows before performing joins and this is the **goal**

### Parallel Operations

* Some SQL processing can be broken into separate parallel processing steps
* Parallel processing can be hinted
* Some operations that might be seen in the execution plan:
  + PX COORDINATOR
  + PX BLOCK ITERATOR
  + PX SEND
  + PX RECEIVE

PX: Parallel Transaction

#### Parallel Execution Execution Plan

* Instead of using "ALLSTATS LAST", you need use "ALLSTATS ALL" to see info on all parallel server processes
* [What are Cardinality Estimiates in Oracle](https://www.youtube.com/watch?v=OIKm2qILtPk&list=PLR0UXVQ7j2yxBfC7a_As-d756LBLpkZra&index=3)
  + watch 6min55second

SELECT \* FROM table (

DBMS\_XPLAN.DISPLAY\_CURSOR(FORMAT => 'ALLSTATS ALL')

);

### Partition Processing

* Common practice to partition tables that are expected to contain a large volume of rows
* The optimizer is partition-smart, and its plans should reflect this (Partition Pruning)
* Partition access in the execution plan
  + PARTITION LIST **SINGLE**
  + PARTITION LIST

### Dynamic Statistics

* indicates object statistics are missing

## Important View

[Oracle Database Reference 19c](https://docs.oracle.com/en/database/oracle/oracle-database/19/refrn/V-SQL.html)

| **Name** | **Description** |
| --- | --- |
| V$SQL\_PLAN | - A dictionary view introduced in Oracle 9i that shows the execute plan for a SQL statement that has been compiled into a cursor cache - A dynamic performance view that shows the execution plan for a SQL statement that has been compiled into a cursor and stored in the cursor cache Its definition is similar to the PLAN\_TABLE |
| V$SQL\_PLAN\_STATISTICS\_ALL | It contains memory usage statstics for row sources that use SQL memory (sort or hash-join). This view concatenates information in V$SQL\_PLAN with execution statistics from V$SQL\_PLAN\_STATISTICS and V$SQL\_WORKAREA |
| V$SQL | V$SQL lists statistics on shared SQL areas **without** the GROUP BY clause and contains one row for each child of the original SQL text entered. Statistics displayed in V$SQL are normally updated at the end of query execution. However, for long running queries, they are updated every 5 seconds. This makes it easy to see the impact of long running SQL statements while they are still in progress. |
| V$SQLAREA | V$SQLAREA displays statistics on shared SQL areas and contains one row per SQL string. It provides statistics on SQL statements that are in memory, parsed, and ready for execution. |
| V$SQL\_SHARED\_CURSOR | This view explains why a particular child cursor is not shared with existing child cursors which caused more than one child cursor to be created for same Parent cursor. Each column in this view identifies a specific reason why the cursor cannot be shared. The columns describe the various reasons with “Y” or “N” for the value. You should focus on the column which has value as ‘Y’. |
| V$SESSION |  |
| V$SESSTAT |  |
| V$STATNAME |  |
| V$MYSTAT |  |
| ALL\_TAB\_STATS\_HISTORY |  |

### Logical I/Os

* Consistent (read) gets
  + These access blocks as they existed at some point in the past. This is usually the time the query started. This ensures that the query only returns values that existed when the query started. Any changes made after this are **ignored**
* Concurrent mode gets
  + This gets the block as it exists right now. You will see these gets when changing data in INSERT, UPDATE and DELETE statements.
* When reading a block, whether in consistent or concurrent mode, if it is not cached in memory, the data will also do a physical read. This fetches the block from disk.
* you can view the I/O for a statement with the IOSTATS format in DBMS\_XPLAN. This adds a buffers column to the plan. This sums the consistent and concurrent mode gets for the operations
  + The **IOSTATS** format reports the logical I/O in the buffers column for each operation in the plan. These figures are cumulative.

select /\*+ gather\_plan\_statistics \*/count(\*) from bricks;

select \*

from table(dbms\_xplan.display\_cursor(format => 'IOSTATS LAST'));

### Memory

* Sorting and joining operations need to read and write data too. Often these are able to process the rows in memory. You can see how much memory an operation needed by getting the plan with the **MEMSTATS** format.
  + this adds OMem, 1Mem and Used-Mem columns to the plan
  + Unlike the I/O figures (which is cumulative), these values are **specific to each operation** in the plan
  + The OMem and 1Mem figures are estimates. Used-Mem reports how much memory it actually used.
  + If memory is limited or when processin ghubge data sets all the rows may not fit in memory. In this case the db neess to write the data to disk to complete the operation. Oracle db uses the temporary tablespace to do this.
* DBMS\_XPlan only returns metrics which have non-zero values for at least one operation in the plan. If a column is missing (e.g. Used-Mem) this means the database needed no extra memory to process the query!

select /\*+ gather\_plan\_statistics \*/\* from bricks

order by brick\_id

fetch first 1 rows only;

select \*

from table(dbms\_xplan.display\_cursor(format => 'MEMSTATS'));

### Disk Reads and Writes

* When the database is unable to sort rows in memory it writes them to temporary disk. You can see the amount of disk needed in the Used-Tmp column with the MEMSTATS format.
* To view the number of disk reads and writes, check the Reads and Writes columns with the IOSTATS format.
* An easy way to display all this information is with the ALLSTATS format. This is shorthand for MEMSTATS IOSTATS ROWSTATS.

-- The alter session statements in this section limit the amount of memory available for joins and sorts.

-- So these operations will need to write to disk

alter session set workarea\_size\_policy = manual;

alter session set sort\_area\_size = 25000;

select /\*+ gather\_plan\_statistics \*/ row\_count from complex\_query;

select \*

from table(dbms\_xplan.display\_cursor(format => 'ALLSTATS LAST'));

Below statistics in execution plan are **cumulative**

* buffer/consistent gets
* disk reads
* disk writes
* time

Below statistics in execution plan are specific for each operation (NOT cumulative)

* Rows
* Memory Used

### Clustering Factor

* By default, tables are heap organized in Oracle DB. This means that the database is free to place rows wherever it wants
* Index are ordered data structures
* The closer the physical order of rows matches the logical order of an index, the more effective that index will be
* The smallest unit of I/O in Oracle db is a block. The more consecutive index entries there are that point to the same block,the more rows you can fetch in one I/O
* When determing how effective an index is, it is the number of I/O operations that matter. Not how many rows it accesses
* The clustering factor is a measure of how closely the logical index order matches the physical table order for rows. The database calculates this when gathering stats.
  + Is the row for the current index entry in the same block as the previous entry or a different one?
  + Each time consecutive index entries are in different blocks, the optimizer increments a counter.
  + The clustering factor's value ranging from the **number of blocks in the table** to the **number of rows**
  + The default calculation for the clustering factor is pessimistic. This means that the optimizer is unlikely to use indexes on mostly clustered values, even when they are faster than a full table scan.
  + The lower this value, the better clustered the rows are and the more likely the database is to use the index
* You can view the clustering factor by querying the \*\_INDEXES views
* Increasing the value TABLE\_CACHED\_BLOCKS preference helps the optimizer spot mostly clustered values.
* the TABLE\_CACHED\_BLOCKS preference tells the optimizer to keep track of how recently it saw a value.
  + default value is 1
  + Use below to change it
  + select dbms\_stats.get\_prefs ( 'table\_cached\_blocks', null, 'bricks' )
  + from dual;
  + begin
  + dbms\_stats.set\_table\_prefs (
  + null, 'bricks', 'table\_cached\_blocks', 16
  + );
  + end;

/

* + - You need to regather stats for th is change (TABLE\_CACHED\_BLOCKS) to take effect

select index\_name, clustering\_factor

from user\_indexes

where table\_name = 'BRICKS';

exec dbms\_stats.gather\_table\_stats ( null, 'bricks' ) ;

select index\_name, clustering\_factor

from user\_indexes

where table\_name = 'BRICKS';

* + It's likely many tables in your database will benefit from increasing TABLE\_CACHED\_BLOCKS. You can set a default for all tables in a schema or database with:
  + dbms\_stats.set\_schema\_prefs ( '<schema name>', 'table\_cached\_blocks', <value> );

dbms\_stats.set\_database\_prefs ( 'table\_cached\_blocks', <value> );

* + Values you set at the schema level override database settings. And table level settings take priority over both. So you can set the default for the current schema to 16 with:

exec dbms\_stats.set\_schema\_prefs ( user, 'table\_cached\_blocks', 16 );

* + And increase or decrease this for specific tables as needed with DBMS\_stats.set\_table\_prefs.
  + Avoid set TABLE\_CACHED\_BLOCKS too high. Sometimes you have to physically re-order rows in the table

### Change Physical Data Model

Re-ordering rows in table is impractical in most applications. Luckily Oracle DB has several data structures you can use to force physical order on data

* Index-organized tables
* Partitioning
* Table clusters

These all have various pros and cons. And require a rebuild to add to existing tables. Making it impractical to change most existing tables. There is another, more lightweight technique available from 12.2:

* Attribute Clustering
  + Attribute clustering defines a physical order for rows. Using a linear order sorts the values in the same way as ORDER BY; it works through the columns left-to-right, only using the next column to resolve ties.

## Important DB parameters

>show parameters optimizer

| **Name** | **Description** |
| --- | --- |
| DB\_FILE\_MULTIBLOCK\_READ\_COUNT | It specifies the maximum number of blocks read in one I/O operation during a sequential scan. It is one of the parameters you can use to minimize I/O during table scans. The total number of I/Os needed to perform a full table scan depends on such factors as the size of the table, the multiblock read count, and whether parallel execution is being utilized for the operation. |
| TABLE\_CACHED\_BLOCKS | DB 12.1+, the average number of blocks cached in the buffer cache for any table we can assume when gathering the index clustering factor |
| OPTIMIZER\_INDEX\_CACHING |  |
| OPTIMIZER\_COST\_ADJ |  |
| OPTIMIZER\_MODE |  |

## Related package

| **Procedure** | **Arguments** |
| --- | --- |
| dbms\_xplan.display() | - plan table name (default 'PLAN\_TABLE') - statement\_id (default null means the last statement inserted into the plan table) - format: control the amount of information displayed (default is 'TYPICAL') |
| dbms\_xplan.display\_cursor() | - sql\_id (default null, means the last SQL statement executed in this session) - child\_number (default 0) - format: controls the amount of information displayed (default 'TYPICAL') |
| dbms\_xplan - Formatting the execution plan | There are three pre-defined formats available: - **BASIC** The plan includes only the ID, operation, and the object name columns. - **TYPICAL** Includes the information shown in BASIC plus additional optimizer-related internal information such as cost, cardinality estimates, etc. This information is shown for every operation in the plan and represents what the optimizer thinks is the operation cost, the number of rows produced, etc. It also shows the predicates evaluated by each operation. There are two types of predicates: **ACCESS** and **FILTER**. The ACCESS predicates for an index are used to fetch the relevant blocks by applying search criteria to the appropriate columns. The FILTER predicates are evaluated after the blocks have been fetched. - **ALL** Includes the information shown in TYPICAL plus the lists of expressions (columns) produced by every operation, the hint alias and query block names where the operation belongs (the outline information). The last two pieces of information can be used as arguments to add hints to the statement |
| DBMS\_ROWID | The DBMS\_ROWID package lets you create ROWIDs and obtain information about ROWIDs from PL/SQL programs and SQL statements. You can find the data block number, the object number, and other ROWID components without writing code to interpret the base-64 character external ROWID. - dbms\_rowid.rowid\_block\_number() |

* DBMS\_XPLAN.DISPLAY takes 3 parameters
  + plan table name (default 'PLAN\_TABLE')
  + statement id (default null)
  + format (default 'TYPICAL')
* DBMS\_XPLAN.DISPLAY\_CURSOR takes 3 parameters
  + SQL\_ID (default last statement executed in this session)
  + Child number (default 0)
  + format (default 'TYPICAL')
* Format is highly customizable - Basic, Typical, All

## How to generate an execution plan

1. EXPLAIN PLAN command
   * Displays an execution plan for a SQL statement without actually executing the statement
2. V$SQL\_PLAN
   * A dictionary view introduced in Oracle 9i that shows the execution plan for a SLQ statement that has been compiled into a cursor in the cursor cache

Under certain conditions that plan shown with EXPLAIN PLAN can be different from the plan from using V$SQL\_PLAN

* binding value
  + EXPLAIN PLAN doesn't do bind peek and assume even distribution of value
  + V$SQL\_PLAN will do bin peek

## Explain Plan

* Using EXPLAIN PLAN command & dbmx\_xplan.display function

EXPLAIN PLAN FOR

SELECT

\*

FROM

apps.ke\_hr\_employee\_info\_v

WHERE

site = 'KEI';

-- use default

select \* from table(dbms\_xplan.display); -- same as below

select \* from table(dbms\_xplan.display(format => 'typical'));

-- customize

SELECT \* FROM dbms\_xplan.display(format => 'basic +cost +rows');

select \* from table(dbms\_xplan.display('plan\_table',null, 'basic'));

select \* from table(dbms\_xplan.display('plan\_table',null, 'all'));

## Execution Plan

Step A

-- ====================================================================

-- EITHER

-- ====================================================================

--set statistics\_level to all in the session

alter session set statistics\_level = all; -- equivalent to using gather\_plan\_statistics hint (in below)

alter session set statistics\_level = typical;

alter session set statistics\_level = basic;

>show parameter statistics\_level

SELECT

\*

FROM

apps.ke\_hr\_employee\_info\_v

WHERE

site = 'KEI';

-- ====================================================================

-- OR

-- ====================================================================

-- you can add below hint to your query

-- Moral of the story: Turn off serveroutput before using GATHER\_PLAN\_STATISTICS in SQL Plus.

-- https://connor-mcdonald.com/2016/01/29/common-gather\_plan\_statistic-confusion/

SELECT /\*+ gather\_plan\_statistics \*/

\*

FROM

apps.ke\_hr\_employee\_info\_v

WHERE

site = 'KEI';

-- The GATHER\_PLAN\_STATISTICS hint allows for the collection of extra metrics during the execution of the query.

-- Specifically, it shows us the Optimizer's estimated number of rows (E-Rows) and the actual number of rows (A-Rows) for each row source.

-- If the estimates are vastly different from the actual, one probably needs to investigate why.

Step B:

-- MAKE SURE 'set serveroutput off' before run below

SET SERVEROUTPUT OFF

select \* from TABLE(dbms\_xplan.display\_cursor());

SELECT \* FROM TABLE(dbms\_xplan.display\_cursor(NULL,NULL, 'cost allstats last'));

/\*

- ALLSTATS - Shorthand for IOSTATS MEMSTATS ROWSTATS. This includes details such as how many rows and consistent gets each step used.

- LAST - Only display the stats for the last execution. Otherwise this default to giving you the information for every execution.

- PARTITION - Include the Pstart and Pstop columns. This tells you which partitions Oracle accessed. Only needed if you're querying partitioned tables!

- PARALLEL - Again, only relevant for parallel statements!

- NOTE - includes the note section. This provides extra information, such as whether Oracle used features like dynamic statistics, adaptive plans, etc.

\*/

-- outline: https://oracle-base.com/articles/misc/outlines

-- the "leading" hint in outline result tells you the JOIN ORDER

SELECT \*

FROM TABLE(dbms\_xplan.display\_cursor(FORMAT=>'allstats last all +outline'));

SELECT \*

FROM TABLE(dbms\_xplan.display\_cursor(sql\_id => '5gr39rg66857y', FORMAT=>'allstats last all +outline'));

SELECT \*

FROM TABLE(dbms\_xplan.display\_cursor(FORMAT=>'allstats last all +cost +bytes'));

SELECT \*

FROM TABLE(dbms\_xplan.display\_cursor(FORMAT=>'allstats last all +peeked\_binds +cost +bytes'));

-- also possible to use the low level arguments to exclude information from the plan

-- customized TYPICAL plan with suppressed options COST and BYTES

SELECT \*

FROM TABLE(dbms\_xplan.display\_cursor(FORMAT=>'typical -cost -bytes'));

-- You can mix exclude(-) and include(+)

-- hint\_report: 19c feature only

SELECT \*

FROM TABLE(dbms\_xplan.display\_cursor(FORMAT=>'typical -predicate +hint\_report'));

## Find SQL ID

SELECT /\* KEVIN\_EXAMPLE \*/

\*

FROM

apps.ke\_hr\_employee\_info\_v

WHERE

site = 'KEI';

SELECT

sql\_id,

sql\_text,

sql\_fulltext,

hash\_value,

plan\_hash\_value

FROM

v$SQL

WHERE

sql\_text LIKE '%KEVIN\_EXAMPLE%'

AND sql\_text NOT LIKE '%v$sql%';

## Gather Schema Stats

BEGIN

dbms\_stats.gather\_schema\_stats(

ownname => 'REPOSITORYAPEX',

cascade => TRUE,

options => 'GATHER AUTO'

);

END;

## Gather Table Stats

BEGIN

dbms\_stats.gather\_table\_stats(

ownname => 'repositoryapex',

tabname => 'ke\_cart\_header');

END;

/\*

Gathering stats on a table may change the optimizer's row estimates for queries against it. If you have lots of queries accessing a table, updating stats may cause the database to have to create a new plan for them all. This adds significant workload to your database!

So optimizer decides when to invalidate the cursors. This means you may not see plan changes immediately after gathering stats on a table.

To force the optimizer to check queries for new plans immediately, set NO\_INVALIDATE to FALSE when calling gather stats:

\*/

BEGIN

dbms\_stats.gather\_table\_stats(

ownname => 'repositoryapex',

tabname => 'ke\_cart\_header',

no\_invalidate => false

);

END;

/\*

When getting plans in these tutorials you may find DBMS\_XPlan reporting the old, "wrong" plan. In these cases making small changes to the query (e.g. adding column aliases) forces the optimizer to reparse the query.

Remember: invalidating cursors when gathering stats this will cause the optimizer to reparse all queries on this table. Use with caution in production environments!

\*/

-- manually set numrows stat for table

BEGIN

dbms\_stats.set\_table\_stats(

ownname => 'repositoryapex',

tabname => 'ke\_cart\_header',

numrows => 100

);

END;

-- manually set stats for table column

BEGIN

dbms\_stats.set\_column\_stats(

ownname => 'repositoryapex',

tabname => 'ke\_cart\_header',

colname => 'capital\_type',

distcnt => 200

);

END;

-- set table stats preference

BEGIN

dbms\_stats.set\_table\_prefs(

ownname => 'repositoryapex',

tabname => 'ke\_cart\_header',

pname => 'stale\_percent',

pvalue => 5 -- 5%

);

END;

-- get table stats preference

SELECT

dbms\_stats.get\_prefs(

ownname => 'repositoryapex',

tabname => 'ke\_cart\_header',

pname => 'stale\_percent'

)

FROM dual;

## Histogram

<http://expertoracle.com/2017/11/28/db-tuning-basics-8-histograms/>

### Type of Histogram

* Frequency
* Height-balanced
* Hybrid
* Top-frequency

### KEY DATABASE PARAMETERS RELATED TO HISTOGRAMS

* METHOD\_OPT

This is the parameter which tells Oracle Engine to create histogram while collecting the statistics.

The default value of METHOD\_OPT from 10g onwards is ‘FOR ALL COLUMNS SIZE AUTO’. Below are possible values for the METHOD

* + AUTO : Oracle determines the columns to collect histograms based on data distribution and the workload of the columns.
  + REPEAT : Collects histograms only on the columns that already have histograms.
  + SKEWONLY : Oracle determines the columns to collect histograms based on the data distribution of the columns.
  + INTEGER VALUE : Number of histogram buckets. Any value between 1 -254. Note that setting 1 will disable the Histogram collection

This basically implies that Oracle will automatically create histograms on those columns which have skewed data distribution and there are SQL statements referencing those columns.

### When to create histogram

* Create Histograms in below situation:
  1. Create for the column which are included in the WHERE clause
  2. AND also when the column is highly skewed.

==> Note that Histograms are not just for indexed column but also for the non-indexed columns

* Do not create Histograms in below situation:
  1. Data in column is evenly distributed.
  2. Column is not used in a where clause.
  3. Do not create on every column of the table as it will increase the stats collection time, parsing time and also can cause optimize to generate bad plans.
  4. Do not create Histograms on primary key.
  5. Do not create histogram when the column is unique and is used only with equality predicates

### Tips for Histograms

* Frequency histograms are more precise, than Height Balanced histograms.
* Histograms are stored in the dictionary and maintenance and space cost is there. So create histograms only when required.
* Histograms are not gathered on columns for which there is no predicate information in the col\_usage$ table.
* col\_usage$ is only populated when columns referenced in predicates and hard parsing occurred.

-- verify your current setting

select dbms\_stats.get\_prefs ('METHOD\_OPT') from dual;

-- to change your global setting to "REPEAT"

exec dbms\_stats.set\_global\_prefs ('METHOD\_OPT','FOR ALL COLUMNS SIZE REPEAT');

-- disable histogram collection globally

exec dbms\_stats.set\_global\_prefs ('METHOD\_OPT','FOR ALL COLUMNS SIZE 1');

-- delete histogram for a column

BEGIN

dbms\_stats.delete\_column\_stats(

ownname => 'repositoryapex',

tabname => 'ke\_spt\_project',

colname => 'project\_type',

col\_stat\_type => 'HISTOGRAM'

)

END;

-- disable histogram for a column

BEGIN

dbms\_stats.set\_table\_prefs(

ownname => 'repositoryapex',

tabname => 'ke\_spt\_project',

method\_opt => 'for columns size 1 project\_type'

);

END;

-- enable histogram for a column

BEGIN

dbms\_stats.set\_table\_prefs(

ownname => 'repositoryapex',

tabname => 'ke\_spt\_project',

method\_opt => 'for columns size auto project\_type'

);

END;

-- enable histogram for multiple columns

BEGIN

dbms\_stats.set\_table\_prefs(

ownname => 'repositoryapex',

tabname => 'ke\_spt\_project',

method\_opt => 'for columns project\_type, project\_status'

);

END;

-- disable histogram for all columns for a table

BEGIN

dbms\_stats.set\_table\_prefs(

ownname => 'repositoryapex',

tabname => 'ke\_spt\_project',

method\_opt => 'for all columns size 1'

);

END;

-- delete/drop histogram for a column

BEGIN

dbms\_stats.delete\_column\_stats(

ownname => 'repositoryapex',

tabname => 'ke\_spt\_project',

colname => 'project\_status',

col\_stat\_type => 'HISTOGRAM'

);

END;

-- continue collecting histograms for all columns except project\_type column

BEGIN

dbms\_stats.set\_table\_prefs(

ownname => 'repositoryapex',

tabname => 'ke\_spt\_project',

method\_opt => 'for all columns size auto, for columns size 1 project\_type'

);

END;

BEGIN

dbms\_stats.gather\_table\_stats(

ownname => 'repositoryapex',

tabname => 'ke\_spt\_project',

method\_opt => 'for columns (division, project\_type)'

);

END;

BEGIN

dbms\_stats.gather\_table\_stats(

ownname => 'repositoryapex',

tabname => 'ke\_spt\_project',

method\_opt => 'for all columns size skewonly for columns (division, project\_type)'

);

END;

## Extended statistics

-- for example, some division is always creating certain project type

SELECT

dbms\_stats.create\_extended\_stats(

ownname => 'repositoryapex',

tabname => 'ke\_spt\_project',

extension => '(division, project\_type)'

)

FROM dual;

-- create extended stats for functional column

select count(\*) from bricks where upper(colour) = 'RED';

BEGIN

dbms\_stats.gather\_table\_stats(

ownname => 'xxx',

tabname => 'bricks',

method\_opt => 'for columns (upper(colour))'

);

END;

--

select

dbms\_stats.create\_extended\_stats ( null, 'bricks', '(weight + 10)' )

from dual;

select /\*+ gather\_plan\_statistics \*/count (\*) from bricks

where weight + 2 = 3;

-- DB 12.1+ : change TABLE\_CACHED\_BLOCKS value

BEGIN

dbms\_stats.set\_table\_prefs(

ownname => 'xxx',

tabname => 'brick',

pname => 'TABLE\_CACHED\_BLOCKS',

pvalue => 10

);

END;

## Autotrace

* [Oracle manual](https://docs.oracle.com/cd/E18283_01/server.112/e16604/ch_eight.htm)

| **Autotrace Setting** | **Result** |
| --- | --- |
| SET AUTOTRACE OFF | No AUTOTRACE report is generated. This is the default |
| SET AUTOTRACE ON EXPLAIN | The AUTOTRACE report shows only the optimizer execution path |
| SET AUTOTRACE ON STATISTICS | The AUTOTRACE REPORT shows only the SQL statement execution statistics |
| SET AUTOTRACE ON | The AUTOTRACE report includes both the optimizer execution path and the SQL steatement execution statistics |
| SET AUTOTRACE TRACEONLY | Like SET AUTOTRACE ON, but suppressed the printing of the user's query output, if any. If STATISTICS is enabled, query data is still fetched, but not printed |

SET autotrace traceonly explain

-- SET autotrace traceonly stat

-- SET autotrace on explain

SELECT

\*

FROM

apps.ke\_hr\_employee\_info\_v

WHERE

site = 'KEI';

SET autotrace OFF

## SQL Monitor

Youtube - [Get the Plan in SQL Developer](https://www.youtube.com/watch?v=-IWxdI9-Z-U&list=PLh0Za9IFK05Zo_zWrAd-TJOH86CBawC8-&index=29&t=0s)

* Not all queries appear by default. Oracle only captures those lasting longer than five seconds (5s) or those running in parallel
  + to get around of this: one way to do whit is by adding the monitor hint:

SELECT /\*+ monitor \*/ ... FROM ..

* You need to have Diagnostics and Tuning packs licenses to use it

-- generate text output for SQL Monitor of a given sql\_id

SELECT

dbms\_sqltune.report\_sql\_monitor(

sql\_id => '&your\_sql\_id',

type => 'TEXT',

report\_level=> 'ALL'

) as sql\_monitor\_output

FROM

DUAL;

-- in 19c and above, you can also use sql\_monitor package

SELECT

dbms\_monitor.report\_sql\_monitor(

sql\_id => '&your\_sql\_id',

type => 'TEXT',

report\_level=> 'ALL'

) as sql\_monitor\_output

FROM

DUAL;

## TKPROF

* TKPROF is a command line utility that analyzes trace files and turns them into readable form.
* It gives you the execution stats for all the SQL in the file
* Tracing adds overhead. So avoid enabling it for the whole database. And remember to disable it when you've finished!

### 1. Generate session trace

#### Method A: SQL Trace

-- A SQL trace captures run time statistics for individual SQL statements.

-- It doesn't give a breakdown of PL/SQL execution times.

-- turn on session SQL tracing

alter session set sql\_trace = true;

-- turn off session SQL tracing

alter session set sql\_trace = false;

-- In 11g and higher, sql\_trace is also an event and can be set with event syntax. Example:

alter session set events 'sql\_trace bind=true';

alter session set events 'sql\_trace bind=true, wait=true';

-- 10046 is special EVENT code. The 10046 trace is the equivalent of setting SQL\_TRACE=TRUE.

-- The advantage of using the event is that extra details may be output to the trace file depending on the level specified with the event.

ALTER SESSION SET EVENTS='10046 trace name context forever, level 1';

...your statement here...

ALTER SESSION SET EVENTS '10046 trace name context off';

-- In 11g and higher, trace can be restricted to a specific SQL ID

alter system set events 'sql\_trace [sql: sql\_id=4k1jlmn567cr7] bind=true, wait=true';

-- multiple SQL IDs can be specified using the pipe symbol as a separator

ALTER SYSTEM SET EVENTS 'sql\_trace [sql: sql\_id=5t6ygtsa3d356|6fa43fgg0rrtp] bind=true, wait=true';

alter session set events '10046 trace name context forever'; -- LEVEL 1 BASIC TRACE

alter session set events '10046 trace name context forever, level 8'; -- LEVEL 8 (WAITS) trace

alter session set events '10046 trace name context forever,level 12'; -- LEVEL 12 (BINDS & WAITS) trace

alter session set events '10046 trace name context off'; -- SWITCH OFF THE TRACE

#### Method B: Tracing using DBMS\_MONITOR/DBMS\_SESSION

-- ref: <https://docs.oracle.com/database/121/ARPLS/d_monitor.htm#ARPLS67178>

##### DBMS\_MONITOR

alter session set tracefile\_identifier = KZHANG';

BEGIN

dbms\_monitor.session\_trace\_enable(

session\_id => 7,

serial\_num => 4634,

waits => TRUE,

binds => TRUE

);

END;

/

BEGIN

dbms\_monitor.session\_trace\_disable(

session\_id => 7,

serial\_num => 4634

);

END;

/

-- quick version to trace your own session

exec DBMS\_monitor.session\_trace\_enable ( null, null, true, true );

\*\*\*your code here\*\*\*

exec DBMS\_monitor.session\_trace\_disable;

##### DBMS\_SESSION

* DBMS\_SESSION can also be used to trace any database session easily.
* Whereas the DBMS\_MONITOR package can only be invoked by a user with the DBA role
* users can also enable SQL tracing for their own session by invoking the:

sql DBMS\_SESSION.SESSION\_TRACE\_ENABLE procedure

1. Log on to the database SQL\*Plus as system or sys user.
2. Find out the serial#, session id from v$session table for the user and run below command to start tracing

sql execute dbms\_session.set\_sql\_trace\_in\_session(sid, seria#, TRUE);

1. Let user perform required actions which need to be traced.
2. Disable the trace using below command:

sql execute the command dbms\_session.set\_sql\_trace\_in\_session(sid, seria#, FALSE);

1. This will generate a .trc file in the database’s trace directory.

### 2. Get the session trace file

Trace file lives on database server. 3 Steps to get it:

* 2a: figure out the database server host name you are connecting
* 2b: figure out which folder/directory the trace file is stored
* 2c: figure out which file

#### 2a: figure out the database server host name

-- either query will give you the answer

SELECT host\_name from v$instance;

SELECT sys\_context('USERENV', 'SERVER\_HOST') from dual;

#### 2b: figure out folder/directory

SELECT

name,

value

FROM

v$diag\_info

#### 2c: figure out which file

* The trace file name is normally: sql<instance>\_ora\_<ospid>\_<identifier>.trc
* To make it easier to identify your trace file (optional step, but recommended), you can do below and this identifier will be appended to the actual trace file name (e.g. ora\_1234\_KZHANG.trc)

alter session set tracefile\_identifier = KZHANG';

-- this only gives the trace file name for \*\*your own\*\* current session

select

name,

value

from

v$diag\_info

where

name = 'Default Trace File';

-- to get trace file only for another session

SELECT

p.tracefile

FROM

v$session s,

v$process p

WHERE

s.paddr = p.addr

AND s.sid = 1234;

* Then you need ask DBA to fetch the trace file for you from DB server
* Starting 12.2, you can spool the output of the trace file from v$diag\_trace\_file\_contents so that you can get a local copy of a trace file for a session

select \* from v$diag\_trace\_file\_contents

where trace\_filename = '&p\_your\_trace\_filename';

### Format trace file

* Once you are having trace file (\*.trc), you need parse it with tkprof so that you can read it
* TKPROF also includes an explain otpion. This will show you explain, not execution plans. Use this with caution
* It includes all the SQL statements run between you starting and stopping tracing. This includes recursive SQL i.e. statements inside triggers, functions, etc.
* It breaks down the execution time into parse, execution and fetch times

tkprof <trace-file-name> <output-file-name>

* The first benefit is great if you have a slow transaction that includes several statements. You can sort them in the output file from slowest to fastest. This helps you spot which is takes the longest. To do this, use the sort option

tkprof <trace\_file\_name> <output\_file\_name> sort=prsela,exeela,fchela

## Materialized View

### Query Rewrite

* To allow existing query to leverage MV, you need to enable query rewrite for the MV
* alter materialized view brick\_colours\_mv

enable query rewrite;

* + "MAT\_VIEW REWRITE ACCESS FULL" operation in the execution plan will tell you if query rewrite is happening
* To do query rewrite, by default there must be an **exact** match between the rows in the table and the data in the MV. If there are **any** changes to the base table, there is a mismatch. And the optimizer will no longer do a rewrite.
* To check if an MV is stable, you can query \*\_MVIEWS
* select mview\_name, staleness

from user\_mviews

### Using Stale Data

* There may be cases where you still want to use a stale MV. By tweaking session setting, you can allow the optimizer to use stale MVs. It's possible to let the optimizer use stale materialized views for query rewrite. Do this by changing the QUERY\_REWRITE\_INTEGRITY parameter. This supports these values:
  + ENFORCED - there must be an exact match between table and MV data for a rewrite to happen. This is the **default**.
  + TRUSTED - the database can do rewrites based on declared constraints that are disabled. This setting rarely affects MV rewrites.
  + STALE\_TOLERATED - the optimizer can do a query rewrite using stale MVs

alter session set query\_rewrite\_integrity = stale\_tolerated;

### Keeping MVs In Sync with the Table

exec dbms\_mview.refresh('brick\_colours\_mv', 'C');

### Incremental Materialized View Refresh

Instead of re-running the whole query in an MV, you can apply the changes since the last refresh. This is called a fast refresh. To do this, first you must create a materialized view log on the table

create materialized view log

on bricks

with primary key, rowid, sequence, commit scn (

colour, shape, weight, insert\_date

)

including new values;

With the log in place, the database can bring an MV up-to-date by applying the changes in the log to the MV itself.

When defining an MV the refresh methods are:

* COMPLETE - refresh the MV by running its query
* FAST- apply changes in the MV log to bring it up-to-date
* FORCE - use FAST refresh if possible and COMPLETE if not

The default refresh clause is:

REFRESH FORCE ON DEMAND

- This means you have to refresh the MV manually. When you do so, the database will pick fast refresh if it can. To ensure the database always applies changes from the MV log, set the MV to REFRESH FAST.

- You can get the database to do this automatically at the end of every transaction. Do this by setting the refresh to ON COMMIT. When you commit, the database will apply any changes in the log to the MV. To set this property the log must be empty, so first do a final complete refresh

```sql

exec dbms\_mview.refresh ( 'brick\_colours\_mv', 'C' );

alter materialized view brick\_colours\_mv

refresh fast on commit;

```

When you create **ENABLE QUERY REWRITE REFRESH FAST ON COMMIT** MVs, the optimizer automatically uses them where possible. So these can give huge performance gains with no code changes!

* Sadly there are many restrictions on the types of query you can use in a fast refresh on commit MV. For example, you can't use COUNT ( DISTINCT )
* A FAST REFRESH ON COMMIT MV also adds some overhead to every transaction using the tables in the MV. In write heavy applications this can put too much stress on the database.

### Real-Time Materailized View (DB V12.2+ )

A real-time MV allows a query to use its stale data. At query run time, the database applies any changes in the MV logs to get current results. To do this, the MV must be:

* FAST REFRESH ON DEMAND
* Use the ENABLE ON QUERY COMPUTATION clause

alter materialized view brick\_colours\_mv

refresh fast on demand;

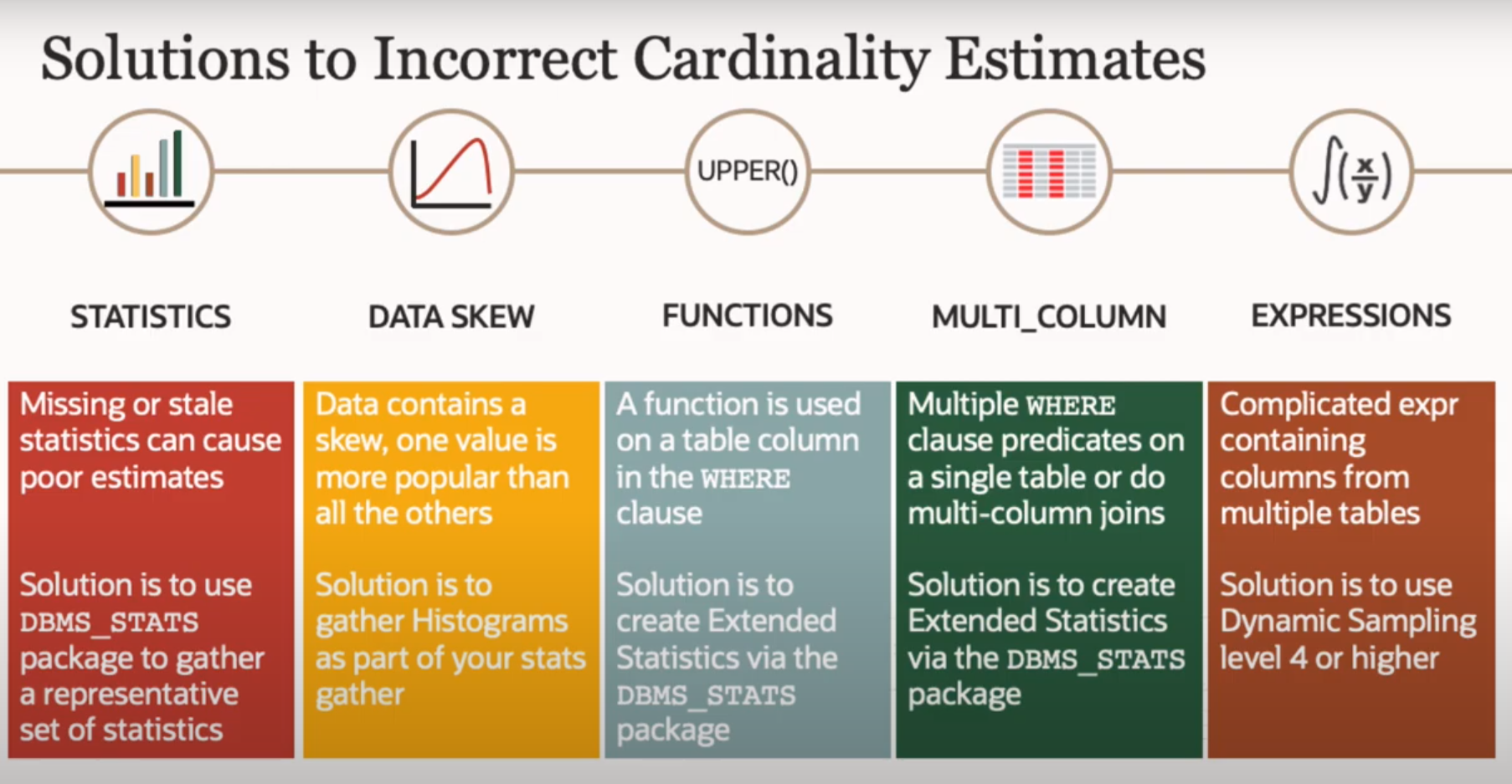
alter materialized view brick\_colours\_mv

enable on query computation;

[](https://gist.github.com/kzhangkzhang)

Author

### [kzhangkzhang](https://gist.github.com/kzhangkzhang)commented [on Aug 17, 2021](https://gist.github.com/kzhangkzhang/2866c2530c5a0ec337c475879eecabc0?permalink_comment_id=3863790#gistcomment-3863790)

[](https://user-images.githubusercontent.com/17183068/129754687-dc2fca8c-861f-451b-90b6-51d496f5f7fd.png)

<https://www.youtube.com/watch?v=yee0RH-qU2o&list=PLR0UXVQ7j2yxBfC7a_As-d756LBLpkZra&index=4>

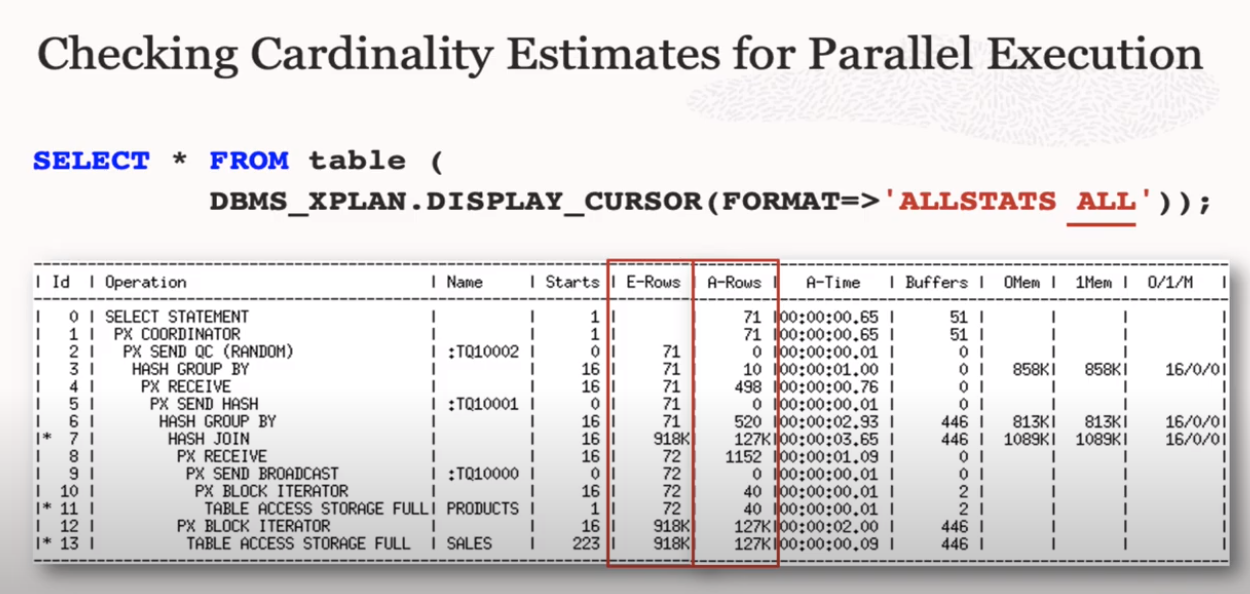
Top of Form

Bottom of Form

[](https://gist.github.com/kzhangkzhang)

Author

### [kzhangkzhang](https://gist.github.com/kzhangkzhang)commented [on Aug 17, 2021](https://gist.github.com/kzhangkzhang/2866c2530c5a0ec337c475879eecabc0?permalink_comment_id=3863791#gistcomment-3863791)

[](https://user-images.githubusercontent.com/17183068/129755219-3c8fcc7e-c9fa-4f9e-bc6a-e33190e9a030.png)

<https://www.youtube.com/watch?v=OIKm2qILtPk&list=PLR0UXVQ7j2yxBfC7a_As-d756LBLpkZra&index=3>

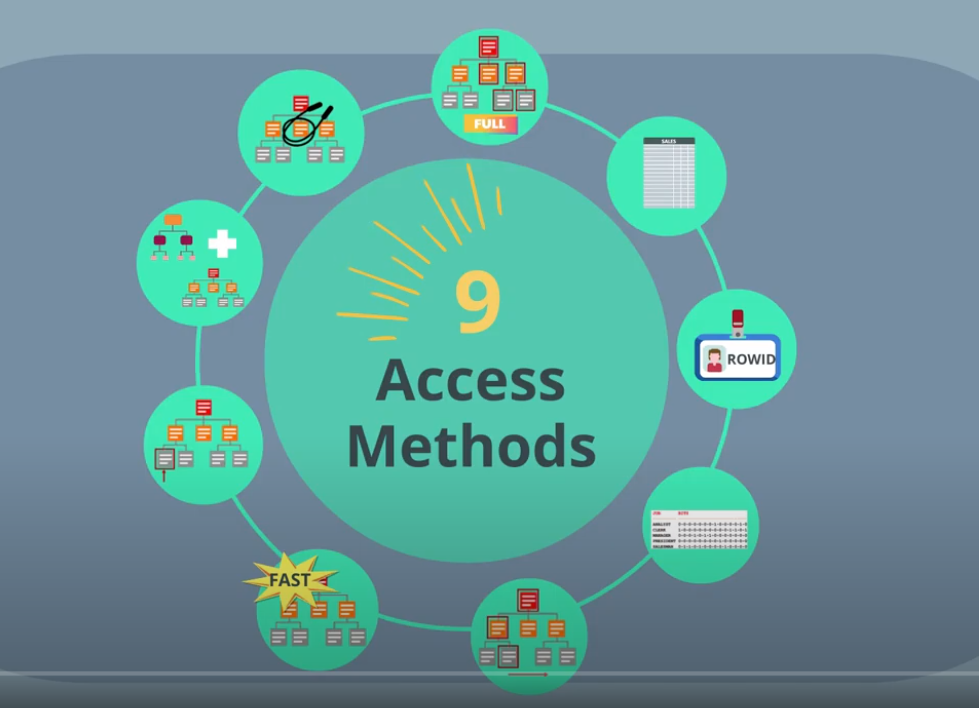
Top of Form

Bottom of Form

[](https://gist.github.com/kzhangkzhang)

Author

### [kzhangkzhang](https://gist.github.com/kzhangkzhang)commented [on Aug 17, 2021](https://gist.github.com/kzhangkzhang/2866c2530c5a0ec337c475879eecabc0?permalink_comment_id=3863805#gistcomment-3863805)

[](https://user-images.githubusercontent.com/17183068/129755562-058a0735-0673-4697-95cc-901bfcade6df.png)

<https://www.youtube.com/watch?v=jYIjzmYCc0U&list=PLR0UXVQ7j2yxBfC7a_As-d756LBLpkZra&index=5>

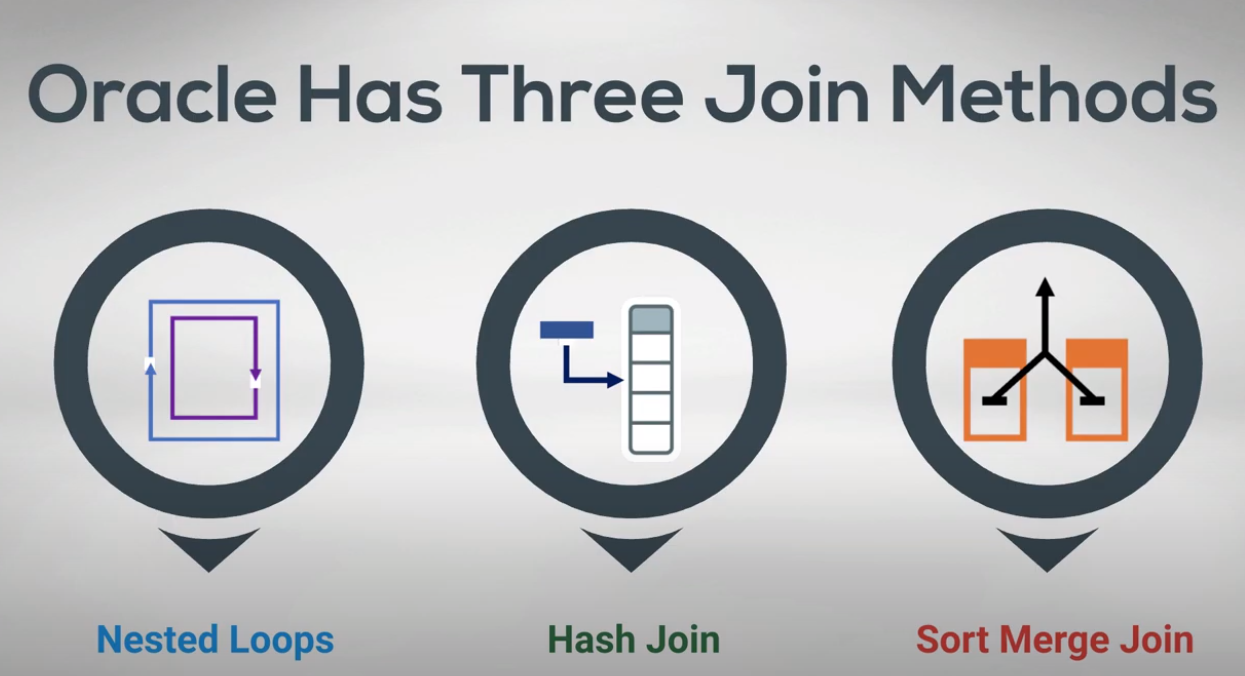
Top of Form

Bottom of Form

[](https://gist.github.com/kzhangkzhang)

Author

### [kzhangkzhang](https://gist.github.com/kzhangkzhang)commented [on Aug 17, 2021](https://gist.github.com/kzhangkzhang/2866c2530c5a0ec337c475879eecabc0?permalink_comment_id=3863807#gistcomment-3863807)

[](https://user-images.githubusercontent.com/17183068/129757770-ca86245e-c3a9-4de7-ab27-c143847aef12.png)

<https://www.youtube.com/watch?v=r043wIjtT-U&list=PLR0UXVQ7j2yxBfC7a_As-d756LBLpkZra&index=6>

Top of Form

Bottom of Form

[](https://gist.github.com/kzhangkzhang)

Author

### [kzhangkzhang](https://gist.github.com/kzhangkzhang)commented [on Aug 17, 2021](https://gist.github.com/kzhangkzhang/2866c2530c5a0ec337c475879eecabc0?permalink_comment_id=3863840#gistcomment-3863840) •

### edited

[](https://user-images.githubusercontent.com/17183068/129761409-4260af33-6aef-452b-a410-0a65e6932456.png)

* Inner join
* Left/right outer join
* Full outer join
* Anti join/Semi join
* Grouped outer join

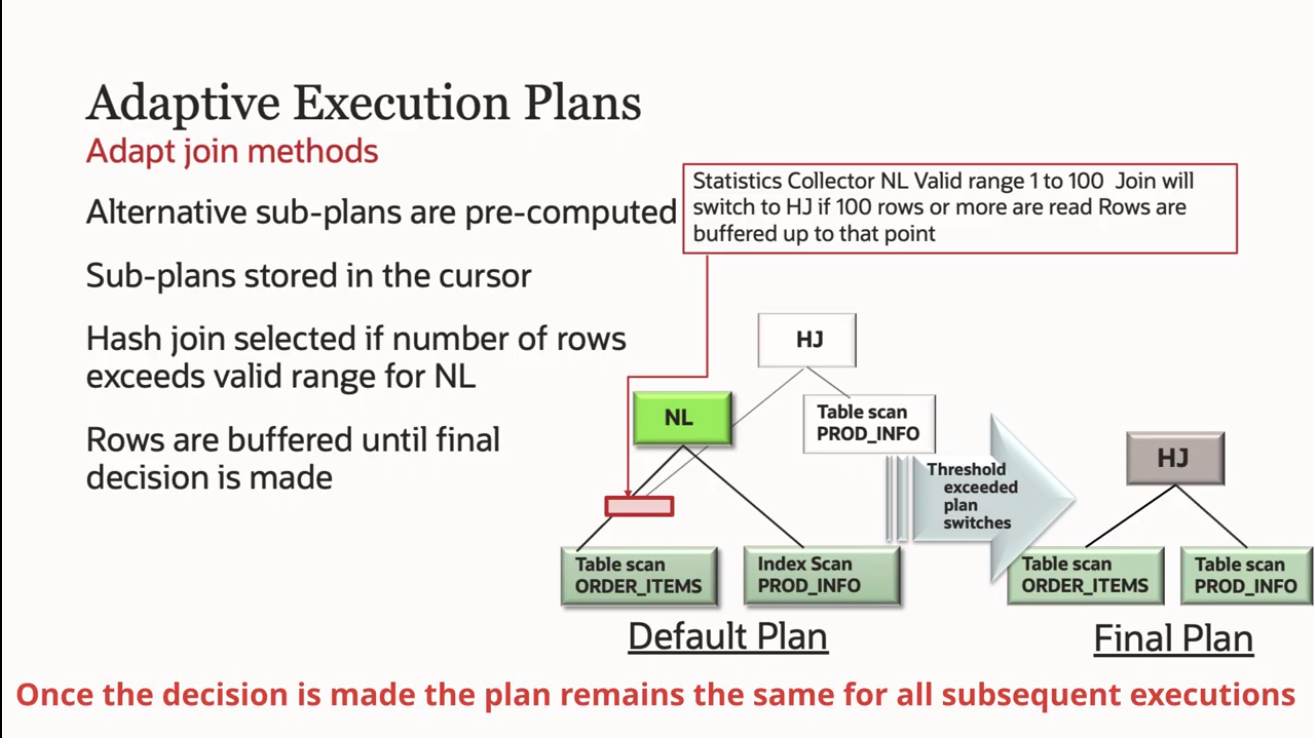
Top of Form

Bottom of Form

[](https://gist.github.com/kzhangkzhang)

Author

### [kzhangkzhang](https://gist.github.com/kzhangkzhang)commented [on Aug 17, 2021](https://gist.github.com/kzhangkzhang/2866c2530c5a0ec337c475879eecabc0?permalink_comment_id=3863871#gistcomment-3863871)

[](https://user-images.githubusercontent.com/17183068/129767106-163ab002-4e87-4d2a-ae6d-75fb7d4ab9ea.png)

<https://www.youtube.com/watch?v=r043wIjtT-U&list=PLR0UXVQ7j2yxBfC7a_As-d756LBLpkZra&index=6>

Top of Form

Bottom of Form

[](https://gist.github.com/kzhangkzhang)

Author

### [kzhangkzhang](https://gist.github.com/kzhangkzhang)commented [on Aug 27, 2021](https://gist.github.com/kzhangkzhang/2866c2530c5a0ec337c475879eecabc0?permalink_comment_id=3874087#gistcomment-3874087)

[](https://user-images.githubusercontent.com/17183068/131155514-80309d53-0004-4ca9-b486-fa8f17e65bc6.png)