How to Perform Database Health Check

Using SQL and PL/SQl Scripts

Asfaw Gedamu

2/03/2024

Table of Contents

This guide outlines the procedures for performing various queries on an Oracle Database.

Query temporary file information

Script 1:

```
SELECT
  f.file_id,
  f.file_name,
  f.file_type,
  f.tablespace_name,
  f.create_date AS created_time
FROM dba_temp_files f;
```

Script 2:

```
SET LINESIZE 200
COLUMN file_name FORMAT A70

SELECT file_id,
       file_name,
       ROUND(bytes/1024/1024/1024) AS size_gb,
       ROUND(maxbytes/1024/1024/1024) AS max_size_gb,
       autoextensible,
       increment_by,
       status
FROM   dba_temp_files
ORDER BY file_name;
```

This script utilizes the dba_temp_files data dictionary view in Oracle Database:

- **f.file_id**: Unique identifier for the temporary file.
- **f.file_name**: Name of the temporary file.
- **f.file_type**: Type of temporary file (e.g., 'TEMPORARY', 'UNDO', 'SEGMENT').
- **f.tablespace_name**: Tablespace where the temporary file resides.
- **f.create_date**: Date and time the temporary file was created.

Note:

- This script requires appropriate privileges to access the dba_temp_files view.
- You can modify the script to filter results based on specific criteria, such as file_type or tablespace_name.
- Refer to the Oracle documentation for the dba_temp_files view for a complete list of available columns and details: https://www.oracletutorial.com/oracle-administration/oracle-tablespace/

Query temp segment usage for all sessions currently using temp space

Script 1:

```
SELECT
  s.username,
  s.sid,
  s.serial#,
  s.sql_id,
  seg.segment_type,
  seg.bytes / 1024 / 1024 AS used_mb,
  seg.bytes_used / 1024 / 1024 AS used_in_sort_mb
FROM v$session s
INNER JOIN v$sort_segment_usage seg ON s.sid = seg.session_id
WHERE s.status = 'ACTIVE'
AND seg.bytes > 0;
```

Script 2:

```
SET LINESIZE 200
COLUMN username FORMAT A20

SELECT username,
       session_addr,
       session_num,
       sqladdr,
       sqlhash,
       sql_id,
       contents,
       segtype,
       extents,
       blocks
FROM   v$tempseg_usage
ORDER BY username;
```

The first script utilizes three views:

- **v$session**: Contains information about active sessions in the database.
- **v$sort_segment_usage**: Shows temporary segment usage by session for sorting operations.
- **v$sql_id**: Stores information about SQL statements executed by sessions (optional, for joining with sql_id if desired).

Explanation:

- The script filters for ACTIVE sessions (s.status) that are currently using temporary space (seg.bytes > 0).
- It joins the v$session and v$sort_segment_usage views based on the session_id to associate session information with temporary segment usage.
- The script retrieves the following information:
  - **s.username**: Username of the session owner.
  - **s.sid**: Session ID.
  - **s.serial#**: Serial number of the session.
  - **s.sql_id**: (Optional) SQL ID of the statement using temporary space (requires joining with v$sql_id).
  - **seg.segment_type**: Type of temporary segment (e.g., 'SORT', 'HASH').
  - **seg.bytes / 1024 / 1024 AS used_mb**: Total temporary space used by the session in megabytes (MB).
  - **seg.bytes_used / 1024 / 1024 AS used_in_sort_mb**: Space used by the session specifically for sorting operations in MB.

Note:

- This script provides a basic overview of temporary segment usage.
- You can modify it to filter by specific users, sessions, or segment types based on your needs.
- Refer to the Oracle documentation for detailed information about the mentioned views:
  - v$session: https://docs.oracle.com/en/database/oracle/oracle-database/19/refrn/V-SESSION.html
  - v$sort_segment_usage: https://www.dba-oracle.com/t_v$_sort_segment.htm

Query information about the top latches

Script 1:

```
SELECT
  latch_name,
  GETCOUNT(1) AS total_waits,
  TIMEWAIT / 1000000000 AS total_wait_time_sec,
  AVERAGE(TIMEWAIT / 1000000) AS avg_wait_time_ms
FROM v$latch
WHERE GETCOUNT(1) > 0
GROUP BY latch_name
ORDER BY total_waits DESC;
```

Script 2:

```
SET LINESIZE 200

SELECT l.latch#,
       l.name,
       l.gets,
       l.misses,
       l.sleeps,
       l.immediate_gets,
       l.immediate_misses,
       l.spin_gets
FROM   v$latch l
WHERE  l.misses > 0
ORDER BY l.misses DESC;
```

Explanation:

- These scripts utilizes the v$latch view, which provides information about latches in the database.
- The script filters out latches with no waits using GETCOUNT(1) > 0.
- It then groups the results by latch_name and calculates the following:
  - **GETCOUNT(1) AS total_waits**: Total number of times the latch was waited on.
  - **TIMEWAIT / 1000000000 AS total_wait_time_sec**: Total time spent waiting on the latch in seconds.

   - **AVERAGE(TIMEWAIT / 1000000) AS avg_wait_time_ms**: Average time spent waiting on the latch in milliseconds (ms).
- The script orders the results by total_waits in descending order, showing the top latches with the most waits first.

Note:

- This script provides a basic overview of latch wait information.
- You can modify it to filter by specific latch names, wait times, or other criteria based on your needs.
- Refer to the Oracle documentation for detailed information about the v$latch view: https://docs.oracle.com/en/database/oracle/oracle-database/19/refrn/V-LATCH.html

Query information on all database sessions ordered by execution

While there's no single view in Oracle Database that directly tracks the total number of executions for each session, you can combine information from different views to achieve a similar outcome. Here's an approach:

Script 1:

```
WITH session_executions AS (
  SELECT
    s.username,
    s.sid,
    s.serial#,
    s.sql_id,
    COUNT(*) AS sql_executions
  FROM v$session s
  INNER JOIN v$sql e ON s.sid = e.sid
  GROUP BY s.username, s.sid, s.serial#, s.sql_id
),
session_totals AS (
  SELECT
    username,
    sid,
    serial#,
    SUM(sql_executions) AS total_executions
  FROM session_executions
  GROUP BY username, sid, serial#
)
SELECT
  u.username,
  st.sid,
  st.serial#,
  st.total_executions
FROM session_totals st
INNER JOIN dba_users u ON st.username = u.username
ORDER BY st.total_executions DESC;
```

Script 2:

```
SET LINESIZE 500
SET PAGESIZE 1000
SET VERIFY OFF

COLUMN username FORMAT A15
COLUMN machine FORMAT A25
COLUMN logon_time FORMAT A20

SELECT NVL(a.username, '(oracle)') AS username,
       a.osuser,
       a.sid,
       a.serial#,
       c.value AS &1,
       a.lockwait,
       a.status,
       a.module,
       a.machine,
       a.program,
       TO_CHAR(a.logon_Time,'DD-MON-YYYY HH24:MI:SS') AS
logon_time
FROM   v$session a,
       v$sesstat c,
       v$statname d
```

```
WHERE  a.sid         = c.sid
AND    c.statistic# = d.statistic#
AND    d.name        = DECODE(UPPER('&1'), 'READS', 'session
logical reads',
                                          'EXECS', 'execute
count',
                                          'CPU',   'CPU used by
this session',
                                                   'CPU used by
this session')
ORDER BY c.value DESC;

SET PAGESIZE 14
```

Explanation:

1. **session_executions Common Table Expression (CTE):**
   - Joins v$session and v$sql views to associate sessions with the SQL statements they execute.
   - Groups the results by user, session ID, serial number, and SQL ID.
   - Counts the number of times each unique combination occurs (COUNT(*) AS sql_executions).
2. **session_totals CTE:**
   - Groups the results from session_executions by user, session ID, and serial number.
   - Calculates the total number of executions for each session by summing sql_executions (SUM(sql_executions) AS total_executions).
3. **Main Query:**
   - Joins session_totals with dba_users to obtain usernames.
   - Selects username, sid, serial#, and total_executions.
   - Orders the results by total_executions in descending order, showing sessions with the most executions first.

Note:

- This script provides an estimate of total executions based on the available information. It might not capture all executions, especially for short-lived sessions or statements that haven't completed yet.
- You can modify the script to filter by specific users, sessions, or SQL IDs based on your needs.
- Refer to the Oracle documentation for detailed information about the mentioned views:
  - v$session: https://docs.oracle.com/en/database/oracle/oracle-database/19/refrn/V-SESSION.html
  - v$sql: https://docs.oracle.com/en/database/oracle/oracle-database/19/refrn/V-SQL.html
  - dba_users: https://docs.oracle.com/en/database/oracle/oracle-database/19/refrn/DBA_USERS.html

Query a list of SQL statements that are using most resources

Script 1:

```
WITH sql_executions AS (
  SELECT
    s.username,
    s.sid,
    s.serial#,
    s.sql_id,
    e.sql_text,
    SUM(CASE WHEN e.elapsed_time_in_seconds > 0 THEN
e.elapsed_time_in_seconds ELSE 0 END) AS total_elapsed_time,
    SUM(CASE WHEN e.cpu_time > 0 THEN e.cpu_time ELSE 0 END) AS
total_cpu_time,
    SUM(CASE WHEN e.disk_reads > 0 THEN e.disk_reads ELSE 0 END)
AS total_disk_reads
  FROM v$session s
  INNER JOIN v$sql e ON s.sid = e.sid
  GROUP BY s.username, s.sid, s.serial#, s.sql_id, e.sql_text
),
top_statements AS (
  SELECT
    username,
    sql_id,
    sql_text,
    RANK() OVER (ORDER BY total_elapsed_time DESC) AS
elapsed_time_rank,
    RANK() OVER (ORDER BY total_cpu_time DESC) AS cpu_time_rank,
    RANK() OVER (ORDER BY total_disk_reads DESC) AS
disk_reads_rank
  FROM sql_executions
)
SELECT
  u.username,
  t.sql_id,
  t.sql_text,
  CASE WHEN t.elapsed_time_rank = 1 THEN t.total_elapsed_time
END AS top_elapsed_time,
  CASE WHEN t.cpu_time_rank = 1 THEN t.total_cpu_time END AS
top_cpu_time,
  CASE WHEN t.disk_reads_rank = 1 THEN t.total_disk_reads END AS
top_disk_reads
FROM top_statements t
INNER JOIN dba_users u ON t.username = u.username
ORDER BY top_elapsed_time DESC, top_cpu_time DESC,
top_disk_reads DESC;
```

Script 2:

```
SET LINESIZE 500
SET PAGESIZE 1000
SET VERIFY OFF

SELECT *
FROM   (SELECT Substr(a.sql_text,1,50) sql_text,

Trunc(a.disk_reads/Decode(a.executions,0,1,a.executions))
reads_per_execution,
               a.buffer_gets,
               a.disk_reads,
               a.executions,
               a.sorts,
                a.address
        FROM   v$sqlarea a
        ORDER BY 2 DESC)
WHERE  rownum <= &&1;

SET PAGESIZE 14
```

Explanation:

1. **sql_executions CTE:**
    - Joins v$session and v$sql views to associate sessions with the SQL statements they execute.
    - Includes the sql_text column for the actual SQL statement.
    - Aggregates data for each unique combination of user, session ID, serial number, and SQL ID.
    - Calculates the following for each SQL statement:
        - SUM(CASE WHEN e.elapsed_time_in_seconds > 0 THEN e.elapsed_time_in_seconds ELSE 0 END) AS total_elapsed_time: Total elapsed time spent executing the statement (seconds).
        - SUM(CASE WHEN e.cpu_time > 0 THEN e.cpu_time ELSE 0 END) AS total_cpu_time: Total CPU time used by the statement.
        - SUM(CASE WHEN e.disk_reads > 0 THEN e.disk_reads ELSE 0 END) AS total_disk_reads: Total disk reads performed by the statement.
2. **top_statements CTE:**
    - Uses window functions (RANK()) to assign ranks to each SQL statement based on the resource usage:
        - elapsed_time_rank: Rank based on total_elapsed_time (highest is 1).
        - cpu_time_rank: Rank based on total_cpu_time (highest is 1).
        - disk_reads_rank: Rank based on total_disk_reads (highest is 1).
3. **Main Query:**
    - Joins top_statements with dba_users to obtain usernames.
    - Selects username, sql_id, sql_text, and the top ranked values for each resource:

        - CASE WHEN t.elapsed_time_rank = 1 THEN t.total_elapsed_time END AS top_elapsed_time: Shows the total elapsed time only if it's the top ranked statement.
        - Similar logic is applied for top_cpu_time and top_disk_reads.
    - Orders the results by top_elapsed_time (highest first), followed by top_cpu_time, and then top_disk_reads.

Note:

- This script provides a basic overview of resource usage by SQL statements. The specific resources might vary depending on the database version and configuration.
- You can modify it depending on your specific needs.

Query details of a specified trace run

```
Script 1:
SET LINESIZE 200
SET TRIMOUT ON

COLUMN runid FORMAT 99999
COLUMN event_seq FORMAT 99999
COLUMN event_unit_owner FORMAT A20
COLUMN event_unit FORMAT A20
COLUMN event_unit_kind FORMAT A20
COLUMN event_comment FORMAT A30

SELECT e.runid,
       e.event_seq,
       TO_CHAR(e.event_time, 'DD-MON-YYYY HH24:MI:SS') AS
event_time,
       e.event_unit_owner,
       e.event_unit,
       e.event_unit_kind,
       e.proc_line,
       e.event_comment
FROM   plsql_trace_events e
```

```
WHERE  e.runid = &1
ORDER BY e.runid, e.event_seq;
```

Script 2:

```
SELECT
  tr.trace_id,
  tr.db_user,
  tr.status,
  tr.begin_interval_time,
  tr.end_interval_time,
  te.event_id,
  te.name AS event_name,
  te.description AS event_description,
  tf.session_id,
  tf.username,
  tf.sql_id,
  tf.column_value1,
  tf.column_value2,
  tf.column_value3
FROM dba_traces tr
INNER JOIN dba_trace_events te ON tr.trace_id = te.trace_id
INNER JOIN dba_trace_files tf ON te.trace_id = tf.trace_id
WHERE tr.trace_id = <specified_trace_id>;
```

Explanation:

- The script joins three tables:
  - dba_traces: Stores information about trace runs.
  - dba_trace_events: Contains details about the events captured during the trace run, including event ID, name, and description.
  - dba_trace_files: Holds data captured for each event, including session ID, username, SQL ID, and additional columns depending on the event type.
- The script filters the results based on the specified_trace_id provided.
- It retrieves the following information:

- **Trace Details:**
    - tr.trace_id: Unique identifier for the trace run.
    - tr.db_user: User who initiated the trace.
    - tr.status: Current status of the trace (e.g., 'ACTIVE', 'COMPLETED').
    - tr.begin_interval_time: Start time of the trace interval.
    - tr.end_interval_time: End time of the trace interval (if available).
- **Event Details:**
    - te.event_id: Identifier for the event type.
    - te.name: Event name (e.g., 'SQL Cursor Opened', 'Hard Parse').
    - te.description: Description of the event.
- **Trace File Data:**
    - tf.session_id: Session ID associated with the event.
    - tf.username: Username of the session owner.
    - tf.sql_id: SQL ID associated with the event (if applicable).
    - tf.column_value1, tf.column_value2, tf.column_value3: Additional data captured for the event, depending on the event type. These columns might contain various pieces of information relevant to the specific event.

Note:

- This script provides a general overview of trace details. The specific columns and information captured might differ depending on the events included in the trace definition and the Oracle Database version.
- Replace <specified_trace_id> with the actual trace ID you want to analyze.
- Refer to the Oracle documentation for detailed information about the mentioned tables and events:
    - dba_traces: [invalid URL removed]
    - dba_trace_events: [invalid URL removed]
    - dba_trace_files: [invalid URL removed]

Query information on all trace runs

```
Script 1:
SET LINESIZE 200
SET TRIMOUT ON

COLUMN runid FORMAT 99999

SELECT runid,
       run_date,
       run_owner
FROM   plsql_trace_runs
ORDER BY runid;
```

Script 2:

```
SELECT
  tr.trace_id,
  tr.db_user,
  tr.status,
  tr.begin_interval_time,
  tr.end_interval_time
FROM dba_traces tr
ORDER BY tr.trace_id;
```

Explanation:

- This script queries the dba_traces table, which stores information about trace runs in the database.
- It retrieves the following information for each trace run:
    - tr.trace_id: Unique identifier for the trace run.
    - tr.db_user: User who initiated the trace.
    - tr.status: Current status of the trace (e.g., 'ACTIVE', 'COMPLETED').
    - tr.begin_interval_time: Start time of the trace interval.
    - tr.end_interval_time: End time of the trace interval (if available).

- The script orders the results by trace_id for easier identification and exploration.

Note:

- This script displays basic information about trace runs. You can use the trace_id retrieved here to further analyze specific trace details using the script provided in the previous response.
- Refer to the Oracle documentation for detailed information about the dba_traces table: https://docs.oracle.com/cd/B10500_01/server.920/a96533/o_trace.htm

Query information about datafiles for a specified tablespace

Script 1:

```
SET LINESIZE 200
COLUMN file_name FORMAT A70

SELECT file_id,
       file_name,
       ROUND(bytes/1024/1024/1024) AS size_gb,
       ROUND(maxbytes/1024/1024/1024) AS max_size_gb,
       autoextensible,
       increment_by,
       status
FROM   dba_data_files
WHERE  tablespace_name = UPPER('&1')
ORDER BY file_id;
```

Script 2:

```
SELECT
  df.file_id AS datafile_id,
  df.file_name AS datafile_name,
  df.tablespace_name,
  df.file_size / 1024 / 1024 / 1024 AS file_size_gb,
  df.autoextend_enabled AS autoextend
FROM dba_data_files df
WHERE df.tablespace_name = '<specified_tablespace_name>';
```

Explanation:

- The script utilizes the dba_data_files view, which contains information about datafiles in the database.
- It filters the results based on the specified_tablespace_name provided.
- The script retrieves the following information for each datafile:
  - **df.file_id AS datafile_id**: Unique identifier for the datafile.
  - **df.file_name AS datafile_name**: Name of the datafile.
  - **df.tablespace_name**: Tablespace associated with the datafile.
  - **df.file_size / 1024 / 1024 / 1024 AS file_size_gb**: Size of the datafile in gigabytes (GB).
  - **df.autoextend_enabled AS autoextend**: Indicates whether autoextend is enabled for the datafile (YES or NO).

Note:

- Replace <specified_tablespace_name> with the actual name of the tablespace you want to investigate.
- This script provides basic information about datafiles. You can explore additional attributes like free space or used space using other columns in the dba_data_files view.
- Refer to the Oracle documentation for detailed information about the dba_data_files view: https://docs.oracle.com/en/database/oracle/oracle-database/19/refrn/DBA_DATA_FILES.html

Query empty space in a tablespace or specific datafile

Script 1:

```
SET SERVEROUTPUT ON SIZE 1000000
SET FEEDBACK OFF
SET TRIMOUT ON
SET VERIFY OFF

DECLARE
  l_tablespace_name VARCHAR2(30) := UPPER('&1');
  l_file_id         VARCHAR2(30) := UPPER('&2');

  CURSOR c_extents IS
    SELECT owner,
           segment_name,
           file_id,
           block_id AS start_block,
           block_id + blocks - 1 AS end_block
    FROM   dba_extents
    WHERE  tablespace_name = l_tablespace_name
    AND    file_id = DECODE(l_file_id, 'ALL', file_id,
TO_NUMBER(l_file_id))
    ORDER BY file_id, block_id;

  l_block_size     NUMBER  := 0;
  l_last_file_id   NUMBER  := 0;
  l_last_block_id  NUMBER  := 0;
  l_gaps_only      BOOLEAN := TRUE;
  l_total_blocks   NUMBER  := 0;
BEGIN
  SELECT block_size
  INTO   l_block_size
  FROM   dba_tablespaces
  WHERE  tablespace_name = l_tablespace_name;

  DBMS_OUTPUT.PUT_LINE('Tablespace Block Size (bytes): ' ||
l_block_size);
  FOR cur_rec IN c_extents LOOP
    IF cur_rec.file_id != l_last_file_id THEN
      l_last_file_id  := cur_rec.file_id;
      l_last_block_id := cur_rec.start_block - 1;
    END IF;

    IF cur_rec.start_block > l_last_block_id + 1 THEN
      DBMS_OUTPUT.PUT_LINE('*** GAP *** (' || l_last_block_id ||
' -> ' || cur_rec.start_block || ')' ||
        ' FileID=' || cur_rec.file_id ||
        ' Blocks=' || (cur_rec.start_block-l_last_block_id-1) ||
        ' Size(MB)=' || ROUND(((cur_rec.start_block-
l_last_block_id-1) * l_block_size)/1024/1024,2)
      );
      l_total_blocks := l_total_blocks + cur_rec.start_block -
l_last_block_id-1;
    END IF;
    l_last_block_id := cur_rec.end_block;
    IF NOT l_gaps_only THEN
      DBMS_OUTPUT.PUT_LINE(RPAD(cur_rec.owner || '.' ||
cur_rec.segment_name, 40, ' ') ||
                           ' (' || cur_rec.start_block || ' -> '
|| cur_rec.end_block || ')');
    END IF;
  END LOOP;
  DBMS_OUTPUT.PUT_LINE('Total Gap Blocks: ' || l_total_blocks);
  DBMS_OUTPUT.PUT_LINE('Total Gap Space (MB): ' ||
ROUND((l_total_blocks * l_block_size)/1024/1024,2));
END;
/

PROMPT
SET FEEDBACK ON
```

Script 2:

Here are two alternative approaches:

1. Analyzing Free Space with dba_data_files and dba_free_space views:

This approach utilizes two views to estimate the potential existence of gaps:

```
WITH datafile_info AS (
  SELECT
```

```
    df.file_id,
    df.tablespace_name,
    df.file_size / 1024 / 1024 / 1024 AS total_size_gb,
    SUM(fs.bytes) / 1024 / 1024 / 1024 AS used_space_gb
  FROM dba_data_files df
  LEFT JOIN dba_free_space fs ON df.file_id = fs.file_id
  GROUP BY df.file_id, df.tablespace_name, df.file_size
)
SELECT
  di.file_id,
  di.tablespace_name,
  di.total_size_gb - di.used_space_gb AS potential_gap_gb
FROM datafile_info di
WHERE di.total_size_gb - di.used_space_gb > 0
ORDER BY potential_gap_gb DESC;
```

Explanation:

- This script uses a Common Table Expression (CTE) named datafile_info to:
  - Join dba_data_files and dba_free_space views based on the file_id.
  - Calculate total_size_gb and used_space_gb for each datafile.
- The main query:
  - Selects file_id, tablespace_name, and the difference between total_size_gb and used_space_gb as potential_gap_gb.
  - Filters for datafiles with a positive potential_gap_gb, indicating potentially unused space.
  - Orders the results by potential_gap_gb in descending order.

Note:

- This approach estimates gaps based on the difference between the total file size and the sum of free space extents. It may not accurately reflect actual gaps due to fragmentation or how free space is managed internally.
- You can modify the script to focus on a specific tablespace by adding a filter on tablespace_name in the datafile_info CTE.

2. Analyzing Extent Usage with dba_extents view:

This approach analyzes extent usage within a datafile:

```
SELECT
  df.file_id,
  df.tablespace_name,
  fe.block_id,
  fe.segment_type,
  fe.segment_id,
  fe.block_status,
  fe.extent_id,
  fe.block_size * fe.num_blocks / 1024 / 1024 AS extent_size_mb
FROM dba_data_files df
INNER JOIN dba_file_extents fe ON df.file_id = fe.file_id
ORDER BY df.file_id, fe.block_id;
```

Explanation:

- This script joins the dba_data_files and dba_file_extents views:
    - dba_data_files: Provides information about datafiles.
    - dba_file_extents: Shows details about extents within datafiles, including block status (used or free).
- The script retrieves information about each extent, including:
    - file_id and tablespace_name: Identify the datafile and tablespace.
    - block_id: Block ID within the extent.
    - segment_type: Type of segment associated with the extent (e.g., data, index).
    - segment_id: Segment ID (if applicable).
    - block_status: Indicates whether the block (and consequently the extent) is used or free.
    - extent_size_mb: Size of the extent in megabytes (MB), calculated from block_size and num_blocks.
- By analyzing the sequence of block_status and the gaps between extent IDs, you might identify potential gaps within the datafile. However, this requires manual analysis and interpretation.

Note:

- This approach requires manual inspection of the results to identify potential gaps. It doesn't provide a direct and automated way to quantify empty space.
- Analyzing individual extents can be complex and time-consuming for large datafiles.

Query a list of tablespaces and their used/full status

Script 1:

```
SET PAGESIZE 140 LINESIZE 200
COLUMN used_pct FORMAT A11

SELECT tablespace_name,
       size_mb,
       free_mb,
       max_size_mb,
       max_free_mb,
       TRUNC((max_free_mb/max_size_mb) * 100) AS free_pct,
       RPAD(' '|| RPAD('X',ROUND((max_size_mb-
max_free_mb)/max_size_mb*10,0), 'X'),11,'-') AS used_pct
FROM   (
        SELECT a.tablespace_name,
               b.size_mb,
               a.free_mb,
               b.max_size_mb,
               a.free_mb + (b.max_size_mb - b.size_mb) AS
max_free_mb
        FROM   (SELECT tablespace_name,
                       TRUNC(SUM(bytes)/1024/1024) AS free_mb
                FROM   dba_free_space
                GROUP BY tablespace_name) a,
               (SELECT tablespace_name,
                       TRUNC(SUM(bytes)/1024/1024) AS size_mb,

TRUNC(SUM(GREATEST(bytes,maxbytes))/1024/1024) AS max_size_mb
                FROM   dba_data_files
                GROUP BY tablespace_name) b
        WHERE  a.tablespace_name = b.tablespace_name
       )
ORDER BY tablespace_name;

SET PAGESIZE 14
```

Script 2:

```
SELECT
  ts.tablespace_name,
  ts.total_bytes / 1024 / 1024 / 1024 AS total_size_gb,
  ts.used_bytes / 1024 / 1024 / 1024 AS used_space_gb,
  (ts.used_bytes * 100) / ts.total_bytes AS used_percentage,
  CASE
    WHEN ts.used_bytes >= ts.max_bytes THEN 'FULL'
    ELSE 'NOT FULL'
  END AS status
FROM dba_tablespaces ts;
```

Explanation:

- The script utilizes the dba_tablespaces view, which contains information about tablespaces in the database.
- It calculates the following for each tablespace:
    - **ts.total_bytes / 1024 / 1024 / 1024 AS total_size_gb**: Total size of the tablespace in gigabytes (GB).
    - **ts.used_bytes / 1024 / 1024 / 1024 AS used_space_gb**: Used space in the tablespace in gigabytes (GB).
    - **(ts.used_bytes * 100) / ts.total_bytes AS used_percentage**: Percentage of used space compared to the total size.
    - **CASE WHEN ts.used_bytes >= ts.max_bytes THEN 'FULL' ELSE 'NOT FULL' END AS status**: Status of the tablespace, indicating whether it's 'FULL' or 'NOT FULL' based on the used space exceeding the maximum space allowed (max_bytes).

Note:

- This script provides a basic overview of tablespace usage. You can modify it to filter specific tablespaces, format the output differently, or calculate additional metrics based on your needs.
- Refer to the Oracle documentation for detailed information about the dba_tablespaces view: https://docs.oracle.com/en/database/oracle/oracle-database/19/refrn/DBA_TABLESPACES.html

Query a list of tablespaces that are nearly full

Script 1:

```
SET PAGESIZE 100

PROMPT Tablespaces nearing 0% free
PROMPT ***************************
SELECT tablespace_name,
       size_mb,
       free_mb,
       max_size_mb,
       max_free_mb,
       TRUNC((max_free_mb/max_size_mb) * 100) AS free_pct
FROM   (
        SELECT a.tablespace_name,
               b.size_mb,
               a.free_mb,
               b.max_size_mb,
               a.free_mb + (b.max_size_mb - b.size_mb) AS
max_free_mb
        FROM   (SELECT tablespace_name,
                       TRUNC(SUM(bytes)/1024/1024) AS free_mb
                FROM   dba_free_space
                GROUP BY tablespace_name) a,
               (SELECT tablespace_name,
                       TRUNC(SUM(bytes)/1024/1024) AS size_mb,

TRUNC(SUM(GREATEST(bytes,maxbytes))/1024/1024) AS max_size_mb
                FROM   dba_data_files
                GROUP BY tablespace_name) b
        WHERE  a.tablespace_name = b.tablespace_name
       )
WHERE  ROUND((max_free_mb/max_size_mb) * 100,2) < 10;

SET PAGESIZE 14
```

Script 2:

```
SELECT
  ts.tablespace_name,
  ts.total_bytes / 1024 / 1024 / 1024 AS total_size_gb,
  ts.used_bytes / 1024 / 1024 / 1024 AS used_space_gb,
  (ts.used_bytes * 100) / ts.total_bytes AS used_percentage,
  CASE
    WHEN ts.used_bytes >= ts.max_bytes THEN 'FULL'
    ELSE 'NEARLY FULL'
  END AS status
FROM dba_tablespaces ts
WHERE (ts.used_bytes * 100) / ts.total_bytes >=
<threshold_percentage>;
```

Explanation:

- This script builds upon the previous script and adds a filter to identify nearly full tablespaces.
- It utilizes the same calculations:
    - Total size, used space, and used percentage.
    - Status is modified to indicate 'FULL' if the used space is greater than or equal to the maximum (max_bytes) and 'NEARLY FULL' if the used space as a percentage of the total size is greater than or equal to a specified threshold_percentage.
- Replace <threshold_percentage> with the desired percentage threshold for considering a tablespace nearly full (e.g., 90, 95).

Note:

- This script provides a customizable way to identify tablespaces nearing capacity based on a user-defined threshold.
- You can adjust the threshold_percentage to suit your specific needs and risk tolerance.

Query tablespace threshold information

Oracle 11g Release 2 introduced the DBA_TABLESPACE_THRESHOLDS view, which displays the settings for all tablespaces, including default thresholds if no table-specific threshold is set. Here's an SQL script to display threshold information for tablespaces:

Script 1:

```
SET LINESIZE 200

COLUMN metrics_name FORMAT A30
COLUMN warning_value FORMAT A30
COLUMN critical_value FORMAT A15

SELECT tablespace_name,
       contents,
       extent_management,
       threshold_type,
       metrics_name,
       warning_operator,
       warning_value,
       critical_operator,
       critical_value
FROM   dba_tablespace_thresholds
ORDER BY tablespace_name, metrics_name;

SET LINESIZE 80
```

Script 2:

```
SELECT
  tablespace_name,
  contents,
  extent_management,
  threshold_type,
  metrics_name,
  warning_operator,
  warning_value,
  critical_operator,
  critical_value
FROM dba_tablespace_thresholds
ORDER BY tablespace_name, threshold_type, metrics_name;
```

Explanation:

- The script utilizes the dba_tablespace_thresholds view to retrieve information about thresholds for each tablespace.
- It displays the following information:
    - **tablespace_name**: Name of the tablespace.
    - **contents**: Type of contents stored in the tablespace (e.g., 'PERMANENT', 'TEMPORARY').
    - **extent_management**: Extent management type for the tablespace (e.g., 'LOCAL', 'DICTIONARY').
    - **threshold_type**: Type of threshold (e.g., 'SPACE USAGE', 'EXTENT USAGE').
    - **metrics_name**: Specific metric used for the threshold (e.g., 'BYTES').
    - **warning_operator**: Operator used for the warning threshold (e.g., 'GE', 'GT').
    - **warning_value**: Value that triggers a warning notification (e.g., percentage used space).
    - **critical_operator**: Operator used for the critical threshold (e.g., 'GE', 'GT').
    - **critical_value**: Value that triggers a critical notification (e.g., percentage used space).
- The script orders the results by tablespace_name, threshold_type, and metrics_name for better organization and identification of specific thresholds.

Note:

- This script provides an overview of configured thresholds. You can use the information obtained here to understand how tablespaces are monitored and when alerts might be triggered based on resource usage.
- Refer to the Oracle documentation for detailed information about the dba_tablespace_thresholds view: https://oracle-base.com/articles/misc/tablespace-thresholds-and-alerts

PL SQL script that displays tablespace threshold information

While displaying existing thresholds is possible via querying the dba_tablespace_thresholds view, modifying or managing thresholds requires higher privileges and often involves using PL/SQL procedures. Here's an example PL/SQL script that demonstrates setting a new warning threshold for a specific tablespace using the DBMS_SPACE_ADMIN package:

Script 1:

```
SET VERIFY OFF

DECLARE
  g_warning_value       VARCHAR2(4) := '&1';
  g_warning_operator    VARCHAR2(4) :=
DBMS_SERVER_ALERT.OPERATOR_GE;
  g_critical_value      VARCHAR2(4) := '&2';
  g_critical_operator   VARCHAR2(4) :=
DBMS_SERVER_ALERT.OPERATOR_GE;

  PROCEDURE set_threshold(p_ts_name  IN VARCHAR2) AS
  BEGIN
    DBMS_SERVER_ALERT.SET_THRESHOLD(
      metrics_id              =>
DBMS_SERVER_ALERT.TABLESPACE_PCT_FULL,
      warning_operator        => g_warning_operator,
      warning_value           => g_warning_value,
      critical_operator       => g_critical_operator,
      critical_value          => g_critical_value,
      observation_period      => 1,
      consecutive_occurrences => 1,
      instance_name           => NULL,
      object_type             =>
DBMS_SERVER_ALERT.OBJECT_TYPE_TABLESPACE,
      object_name             => p_ts_name);
  END;
BEGIN
  IF g_warning_value  = 'NULL' THEN
    g_warning_value    := NULL;
    g_warning_operator := NULL;
  END IF;
  IF g_critical_value = 'NULL' THEN
    g_critical_value    := NULL;
    g_critical_operator := NULL;
  END IF;

  FOR cur_ts IN (SELECT tablespace_name
                 FROM   dba_tablespace_thresholds
                 WHERE  warning_operator != 'DO NOT CHECK'
                 AND    extent_management = 'LOCAL')
  LOOP
    set_threshold(cur_ts.tablespace_name);
  END LOOP;
END;
/
```

Script 2:

```
DECLARE
  l_tablespace_name VARCHAR2(30) := 'USERS'; -- Replace with
desired tablespace name
  l_threshold_type VARCHAR2(30) := 'SPACE USAGE';
  l_metrics_name VARCHAR2(30) := 'BYTES';
  l_operator VARCHAR2(2) := 'GE'; -- Operator (e.g., GE, GT, LT,
etc.)
  l_warning_value NUMBER := 90; -- Warning threshold value
(e.g., percentage used space)
BEGIN
  DBMS_SPACE_ADMIN.SET_TABLESPACE_THRESHOLD(
    tablespace_name => l_tablespace_name,
    threshold_type => l_threshold_type,
    metrics_name => l_metrics_name,
    operator => l_operator,
    warning_value => l_warning_value
  );

  DBMS_OUTPUT.PUT_LINE('Warning threshold set for tablespace '
|| l_tablespace_name || '.');
EXCEPTION
  WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE('Error setting warning threshold: ' ||
SQLERRM);
END;
/
```

Explanation:

- The script declares variables:
    - l_tablespace_name: Replace with the actual tablespace name you want to modify.
    - l_threshold_type: Set to 'SPACE USAGE' as we're modifying a space usage threshold.
    - l_metrics_name: Set to 'BYTES' to use bytes used as the metric.
    - l_operator: Operator used for the threshold comparison (e.g., 'GE' for greater than or equal).
    - l_warning_value: Threshold value (e.g., 90% used space) that triggers a warning.
- The script calls the DBMS_SPACE_ADMIN.SET_TABLESPACE_THRESHOLD procedure:
    - It provides the tablespace name, threshold type, metrics name, operator, and warning value.
- Upon successful execution, it displays a confirmation message.
- An exception block handles potential errors and displays an error message if encountered.

Important Note:

- This script requires appropriate privileges (e.g., ALTER ANY TABLESPACE) to modify tablespace thresholds. Use this script cautiously in a test environment before applying it to a production system.
- Refer to the Oracle documentation for detailed information about the DBMS_SPACE_ADMIN package and managing tablespace thresholds: [invalid URL removed]

Query performance indicators that help tuning databases

Script 1:

```
SET SERVEROUTPUT ON
SET LINESIZE 1000
SET FEEDBACK OFF

SELECT *
FROM   v$database;
PROMPT

DECLARE
  v_value  NUMBER;

  FUNCTION Format(p_value  IN  NUMBER)
    RETURN VARCHAR2 IS
  BEGIN
    RETURN LPad(To_Char(Round(p_value,2),'990.00') || '%',8,' ')
|| '  ';
  END;

BEGIN

  -- --------------------------
  -- Dictionary Cache Hit Ratio
  -- --------------------------
  SELECT (1 - (Sum(getmisses)/(Sum(gets) + Sum(getmisses)))) *
100
  INTO   v_value
  FROM   v$rowcache;
  DBMS_Output.Put('Dictionary Cache Hit Ratio       : ' ||
Format(v_value));
  IF v_value < 90 THEN
    DBMS_Output.Put_Line('Increase SHARED_POOL_SIZE parameter to
bring value above 90%');
  ELSE
    DBMS_Output.Put_Line('Value Acceptable.');
  END IF;

  -- -----------------------
  -- Library Cache Hit Ratio
  -- -----------------------
  SELECT (1 -(Sum(reloads)/(Sum(pins) + Sum(reloads)))) * 100
  INTO   v_value
  FROM   v$librarycache;

  DBMS_Output.Put('Library Cache Hit Ratio          : ' ||
Format(v_value));
  IF v_value < 99 THEN
    DBMS_Output.Put_Line('Increase SHARED_POOL_SIZE parameter to
bring value above 99%');
  ELSE
    DBMS_Output.Put_Line('Value Acceptable.');
  END IF;

  -- -------------------------------
  -- DB Block Buffer Cache Hit Ratio
  -- -------------------------------
  SELECT (1 - (phys.value / (db.value + cons.value))) * 100
  INTO   v_value
  FROM   v$sysstat phys,
         v$sysstat db,
         v$sysstat cons
  WHERE  phys.name  = 'physical reads'
  AND    db.name    = 'db block gets'
  AND    cons.name  = 'consistent gets';

  DBMS_Output.Put('DB Block Buffer Cache Hit Ratio  : ' ||
Format(v_value));
  IF v_value < 89 THEN
    DBMS_Output.Put_Line('Increase DB_BLOCK_BUFFERS parameter to
bring value above 89%');
  ELSE
```

```
    DBMS_Output.Put_Line('Value Acceptable.');
  END IF;

  -- ---------------
  -- Latch Hit Ratio
  -- ---------------
  SELECT (1 - (Sum(misses) / Sum(gets))) * 100
  INTO   v_value
  FROM   v$latch;

  DBMS_Output.Put('Latch Hit Ratio                  : ' ||
Format(v_value));
  IF v_value < 98 THEN
    DBMS_Output.Put_Line('Increase number of latches to bring
the value above 98%');
  ELSE
    DBMS_Output.Put_Line('Value acceptable.');
  END IF;

  -- -----------------------
  -- Disk Sort Ratio
  -- -----------------------
  SELECT (disk.value/mem.value) * 100
  INTO   v_value
  FROM   v$sysstat disk,
         v$sysstat mem
  WHERE  disk.name = 'sorts (disk)'
  AND    mem.name  = 'sorts (memory)';

  DBMS_Output.Put('Disk Sort Ratio                  : ' ||
Format(v_value));
  IF v_value > 5 THEN
    DBMS_Output.Put_Line('Increase SORT_AREA_SIZE parameter to
bring value below 5%');
  ELSE
    DBMS_Output.Put_Line('Value Acceptable.');
  END IF;

  -- ----------------------
  -- Rollback Segment Waits
  -- ----------------------
  SELECT (Sum(waits) / Sum(gets)) * 100
  INTO   v_value
  FROM   v$rollstat;
  DBMS_Output.Put('Rollback Segment Waits            : ' ||
Format(v_value));
  IF v_value > 5 THEN
    DBMS_Output.Put_Line('Increase number of Rollback Segments
to bring the value below 5%');
  ELSE
    DBMS_Output.Put_Line('Value acceptable.');
  END IF;

  -- --------------------
  -- Dispatcher Workload
  -- --------------------
  SELECT NVL((Sum(busy) / (Sum(busy) + Sum(idle))) * 100,0)
  INTO   v_value
  FROM   v$dispatcher;

  DBMS_Output.Put('Dispatcher Workload               : ' ||
Format(v_value));
  IF v_value > 50 THEN
    DBMS_Output.Put_Line('Increase MTS_DISPATCHERS to bring the
value below 50%');
  ELSE
    DBMS_Output.Put_Line('Value acceptable.');
  END IF;

END;
/

PROMPT
SET FEEDBACK ON
```

Script 2:

```
SELECT
  -- Database level
  s.sid AS session_id,
  s.serial# AS serial_number,
  s.username,
  s.machine AS client_machine,
  s.status AS session_status,
  s.sql_id AS current_sql_id,
  e.wait_class,
  e.wait_time_micro / 1000000 AS wait_time_seconds,
  -- Comment on wait time
  CASE
    WHEN wait_time_seconds > 1 THEN 'Potential performance
bottleneck, investigate further.'
    ELSE 'Normal wait time.'
  END AS wait_time_comment,
  -- CPU and I/O utilization
  s.cpu_time / 1000000 AS cpu_time_seconds,
  s.elapsed_time / 1000000 AS elapsed_time_seconds,
  s.disk_reads,
  -- Comment on I/O utilization
  CASE
    WHEN disk_reads > 1000 THEN 'High disk reads, consider
optimizing queries or adding indexes.'
    ELSE 'Normal disk reads.'
  END AS disk_reads_comment
FROM v$session s
INNER JOIN v$session_longops e ON s.sid = e.sid
ORDER BY wait_time_seconds DESC;
```

Explanation:

- The script joins v$session and v$session_longops views:
  - v$session: Provides information about active sessions.
  - v$session_longops: Captures details about long-running operations associated with sessions.
- It retrieves the following information for each session:
  - **Session Details:**
    - s.sid, s.serial#, s.username: Identify the session.
    - s.machine: Client machine where the session originated.
    - s.status: Current status of the session (e.g., 'ACTIVE', 'WAITING').
    - s.sql_id: Current SQL ID being executed (if applicable).
  - **Wait Information:**
    - e.wait_class: Class of wait event the session is currently experiencing.

        - e.wait_time_micro / 1000000 AS wait_time_seconds: Total wait time in seconds.
        - **CASE WHEN wait_time_seconds > 1 THEN 'Potential performance bottleneck, investigate further.' ELSE 'Normal wait time.' END AS wait_time_comment**: Provides a comment based on the wait time, suggesting further investigation for high wait times.
    - **Resource Usage:**
        - s.cpu_time / 1000000 AS cpu_time_seconds: Total CPU time used by the session in seconds.
        - s.elapsed_time / 1000000 AS elapsed_time_seconds: Total elapsed time for the session in seconds.
        - s.disk_reads: Total number of disk reads performed by the session.
        - **CASE WHEN disk_reads > 1000 THEN 'High disk reads, consider optimizing queries or adding indexes.' ELSE 'Normal disk reads.' END AS disk_reads_comment**: Provides a comment based on the number of disk reads, suggesting optimization for high values.
- The script orders the results by wait time in descending order, prioritizing sessions experiencing the most wait.

Note:

- The specific threshold values used for comments can be adjusted based on your system characteristics and performance expectations.
- This script provides a starting point for identifying potential performance issues. Further analysis and investigation might be needed to pinpoint the root cause of any observed problems.
- Refer to the Oracle documentation for detailed information about the mentioned views:
    - v$session: https://docs.oracle.com/en/database/oracle/oracle-database/19/refrn/V-SESSION.html
    - v$session_longops: https://docs.oracle.com/en/database/oracle/oracle-database/19/refrn/V-SESSION_LONGOPS.html