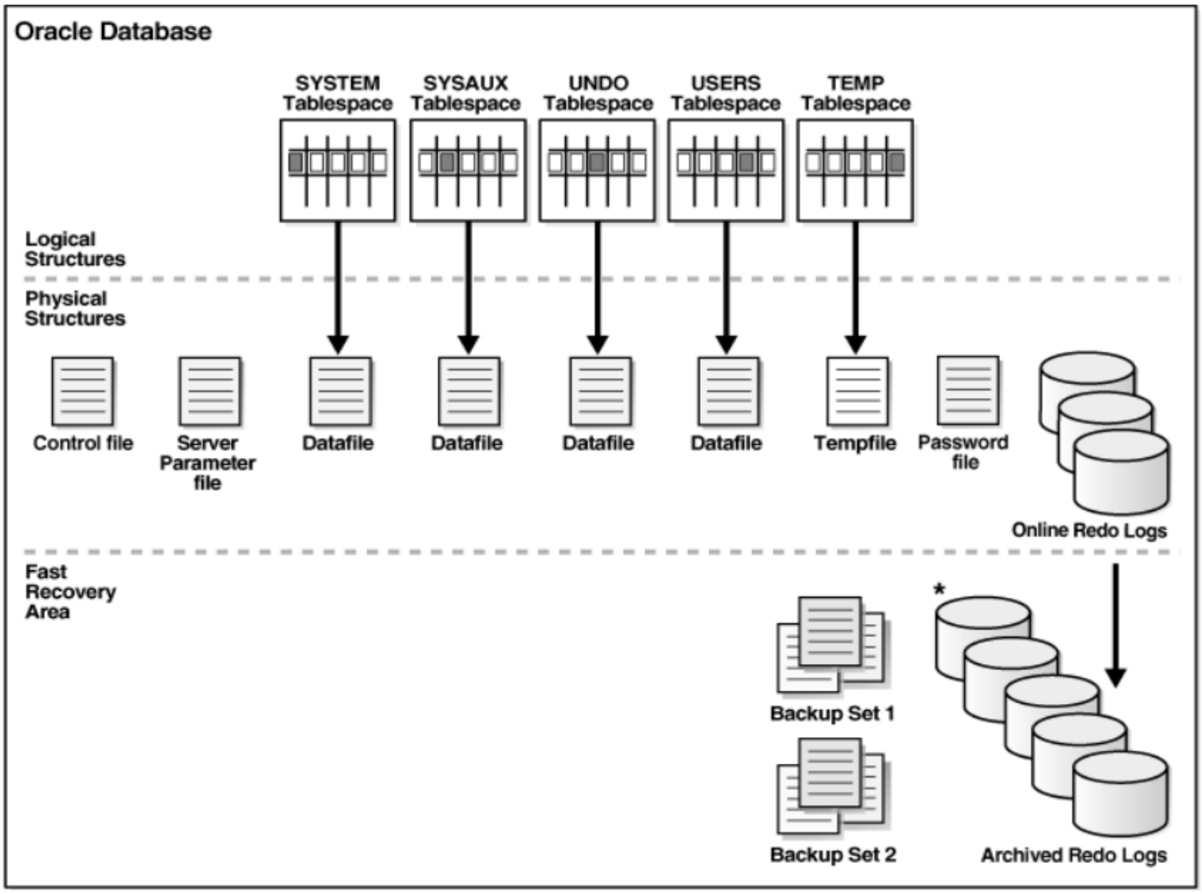
TASK3 REPORT

1. PREREQUISITE TASK

Oracle Database Storage Structures



Data files

Data files are the operating system files that store the data within the database. The data is written to these files in an Oracle proprietary format that cannot be read by other programs.

Datafiles contain the actual data stored in the database, the tables and indexes that store data, the data dictionary that maintains information about these data structures, and the rollback segments used to implement multiuser concurrency.

Data files can be broken down into the following components:

- Segment - A segment contains a specific type of database object. For example, a table is stored in a table segment, and an index is stored in an index segment. A data file can contain many segments.

- Extent - An extent is a contiguous set of data blocks within a segment. Oracle Database allocates space for segments in units of one extent. When the existing extents of a segment are full, the database allocates another extent for that segment.

- Data block - A data block, also called a database block, is the smallest unit of I/O to database storage. An extent consists of several contiguous data blocks. The database uses a default block size at database creation. Oracle block sizes range from 2 KB to 32 KB.

Segments, extents, and data blocks are all logical structures. Only Oracle Database can determine how many data blocks are in a file. The operating system recognizes only files and operating system blocks, not the number of data blocks in an Oracle Database file. Each data block maps to one or more operating system blocks.

Control file

A control file tracks the physical components of the database. It is the root file that the database uses to find all the other files used by the database. Because of the importance of the control file, Oracle recommends that the control file be multiplexed, or have multiple identical copies.

If any control file fails, then your database becomes unavailable. If you have a control file copy, however, you can shut down your database and re-create the failed control file from the copy, then restart your database. Another option is to delete the failed control file from the CONTROL_FILES initialization parameter and restart your database using the remaining control files.

A control file includes:

- The database name
- Names and locations of associated datafiles and online redo log files
- The timestamp of the database creation
- The current log sequence number
- Checkpoint information

Redo log

Redo log files are operating system files used by Oracle to maintain logs of all transactions performed against the database. The primary purpose of these log files is to allow Oracle to recover changes made to the database in the case of a failure.

Redo log files are filled with redo records. A redo record, also called a redo entry, is made up of a group of change vectors, each of which is a description of a change made to a single block in the database. For example, if you change a salary value in an employee table, you generate a redo record containing change vectors that describe changes to the data segment block for the table, the undo segment data block, and the transaction table of the undo segments.

Redo entries record data that you can use to reconstruct all changes made to the database, including the undo segments. Therefore, the redo log also protects rollback data. When you recover the database using redo data, the database reads the change vectors in the redo records and applies the changes to the relevant blocks.

Redo records are buffered in a circular fashion in the redo log buffer of the SGA and are written to one of the redo log files by the Log Writer (LGWR) database background process. Whenever a transaction is committed, LGWR writes the transaction redo records from the redo log buffer of the SGA to a redo log file, and assigns a system change number (SCN) to identify the redo records for each committed transaction. Only when all redo records associated with a given transaction are safely on disk in the online logs is the user process notified that the transaction has been committed.

Redo records can also be written to a redo log file before the corresponding transaction is committed. If the redo log buffer fills, or another transaction commits, LGWR flushes all of the redo log entries in the redo log buffer to a redo log file, even though some redo records may not be committed. If necessary, the database can roll back these changes.

To maximize performance and accommodate many users, a multiprocess Oracle system uses some additional Oracle processes called **background processes**.

DBWR (DataBase Writer) is an Oracle background process created when you start a database instance. The DBWR writes modified data (dirty buffers) from the SGA into the Oracle database files. When the SGA data buffer cache fills the DBWR process selects buffers using an LRU algorithm and writes them to disk. There can be multiple database writer processes named DBWn.

LGWR (Log Writer) The log writer process (LGWR) is responsible for redo log buffer management—writing the redo log buffer to a redo log file on disk. LGWR writes all redo entries that have been copied into the buffer since the last time it wrote.

The redo log buffer is a circular buffer. When LGWR writes redo entries from the redo log buffer to a redo log file, server processes can then copy new entries over the entries in the redo log buffer that have been written to disk. LGWR normally writes fast enough to ensure that space is always available in the buffer for new entries, even when access to the redo log is heavy.

LGWR writes one contiguous portion of the buffer to disk. LGWR writes:

- A commit record when a user process commits a transaction
- Redo log buffers
    - Every three seconds
    - When the redo log buffer is one-third full
    - When a DBWn process writes modified buffers to disk, if necessary

When a user issues a COMMIT statement, LGWR puts a commit record in the redo log buffer and writes it to disk immediately, along with the transaction's redo entries. The corresponding changes to data blocks are deferred until it is more efficient to write them. This is called a fast commit mechanism. The atomic write of the redo entry containing the transaction's commit record is the single event that determines the transaction has committed. Oracle returns a success code to the committing transaction, although the data buffers have not yet been written to disk.

In times of high activity, LGWR can write to the redo log file using group commits. For example, assume that a user commits a transaction. LGWR must write the transaction's redo entries to disk, and as this happens, other users issue COMMIT statements. However, LGWR cannot write to the redo log file to commit these transactions until it has completed its previous write operation. After the first transaction's entries are written to the redo log file, the entire list of redo entries of waiting transactions (not yet committed) can be written to disk in one operation, requiring less I/O than do transaction entries handled individually. Therefore, Oracle minimizes disk I/O and maximizes performance of LGWR. If requests to commit continue at a high rate, then every write (by LGWR) from the redo log buffer can contain multiple commit records.

ARCH (Oracle's ARCHiver Process) is used to copy the contents of an online log file to another location, typically a disk file, when that log file becomes full. Oracle uses the online log files in a "round robin" fashion—that is, when all available online log files become full, the first file is reused. The mode of operation whereby the contents of each file are saved prior to reuse is called archivelog mode, and is controlled by the ARCHIVELOG parameter in the ALTER DATABASE statement. The ARCH process runs only when the instance is running in archivelog mode.

SMON (System MONitor) is an Oracle background process created when you start a database instance. The SMON process performs instance recovery, cleans up after dirty shutdowns and coalesces adjacent free extents into larger free extents.

SMON wakes up every 5 minutes to perform housekeeping activities. SMON must always be running for an instance. If not, the instance will terminate.

PMON (Process MONitor) is an Oracle background process created when you start a database instance. The PMON process will free up resources if a user process fails (eg. release database locks).

PMON normally wakes up every 3 seconds to perform its housekeeping activities. PMON must always be running for an instance.

RECO(Recoverer) is a background process used with the distributed database configuration that automatically resolves failures involving distributed transactions.

The RECO process of a node automatically connects to other databases involved in an in-doubt distributed transaction. When the RECO process reestablishes a connection between involved database servers, it automatically resolves all in-doubt transactions, removing from each database's pending transaction table any rows that correspond to the resolved in-doubt transactions.

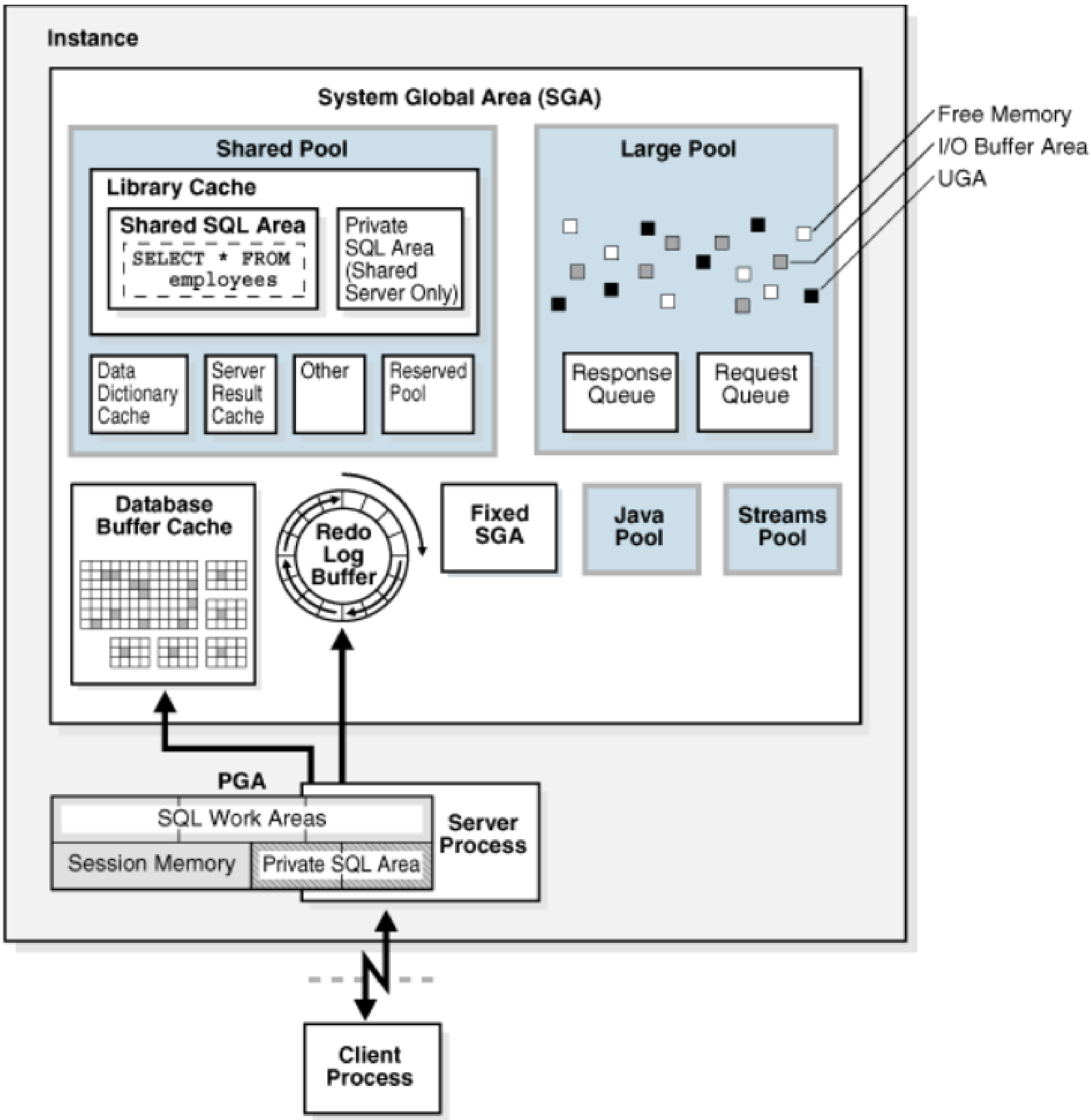
The basic **memory structures** associated with Oracle Database include:

- System global area (SGA)
  The SGA is a group of shared memory structures, known as SGA components, that contain data and control information for one Oracle Database instance. All server and background processes share the SGA. Examples of data stored in the SGA include cached data blocks and shared SQL areas.

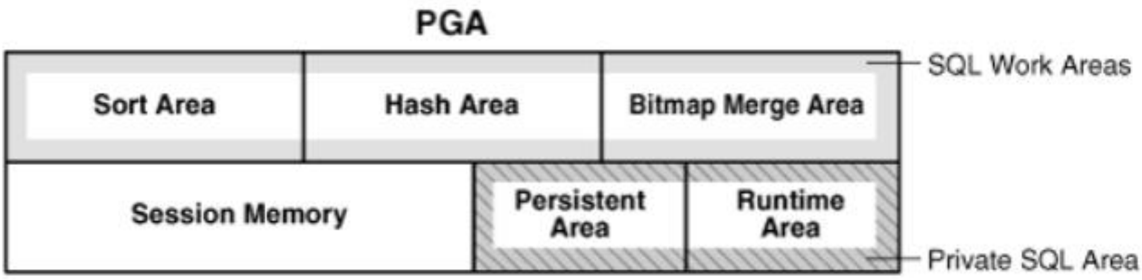
- Program global area (PGA)
  A PGA is a nonshared memory region that contains data and control information exclusively for use by an Oracle process. Oracle Database creates the PGA when an Oracle process starts.
  One PGA exists for each server process and background process. The collection of individual PGAs is the total instance PGA, or instance PGA. Database initialization parameters set the size of the instance PGA, not individual PGAs.

- User global area (UGA)
  The UGA is memory associated with a user session.

- Software code areas
  Software code areas are portions of memory used to store code that is being run or can be run. Oracle Database code is stored in a software area that is typically at a different location from user programs—a more exclusive or protected location.



The **PGA** is subdivided into different areas, each with a different purpose.



The Program Global Area contains the following two types of information:

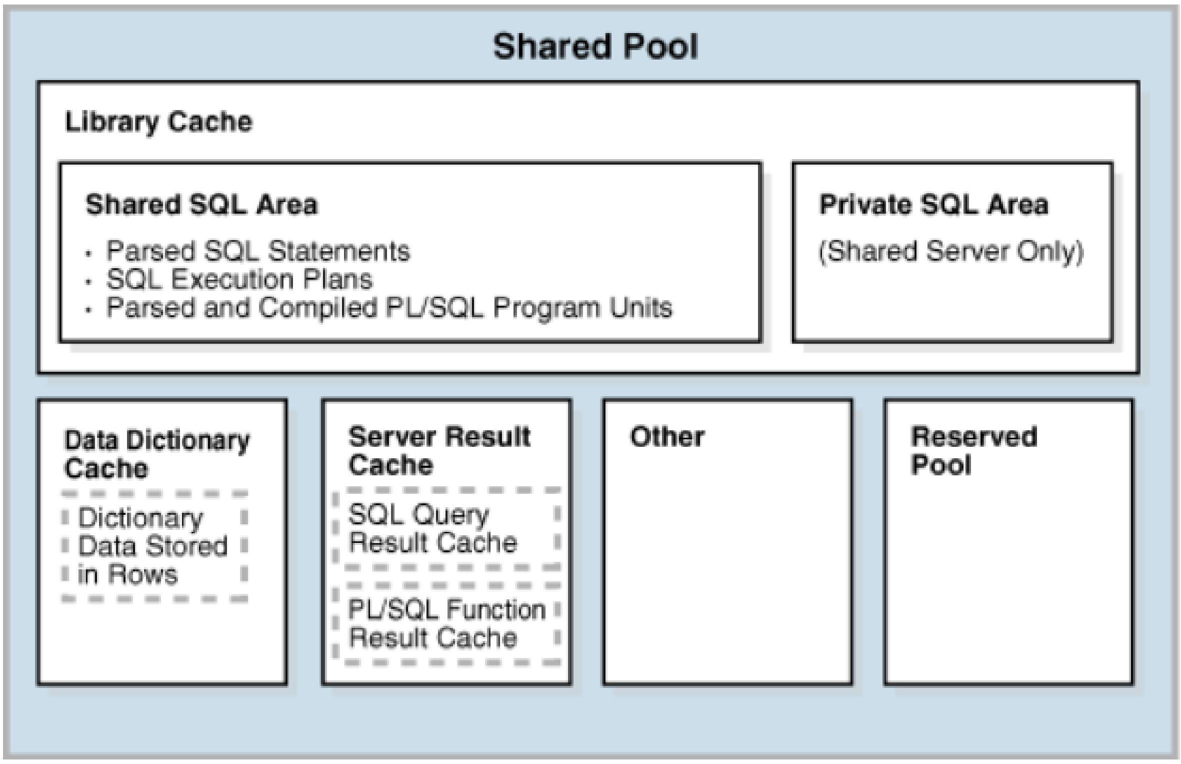
1. Stack space - The stack space is used to hold process variables, arrays, and other similar information.

2. Session information - Session information includes PL/SQL variables and the private SQL areas. Under the multi-threaded server configuration, the session information is contained in the SGA.

Each database instance has its own **SGA**. Oracle Database automatically allocates memory for an SGA at instance startup and reclaims the memory at instance shutdown.

SGA components are:

- Database Buffer Cache - is the memory area that stores copies of data blocks read from data files.The goals include:
  - Optimize physical I/O
    The database updates data blocks in the cache and stores metadata about the changes in the redo log buffer. After a COMMIT, the database writes the redo buffers to the online redo log but does not immediately write data blocks to the data files. Instead, database writer (DBW) performs lazy writes in the background.
  - Keep frequently accessed blocks in the buffer cache and write infrequently accessed blocks to disk
- Redo Log Buffer - is a circular buffer in the SGA that stores redo entries describing changes made to the database. A redo record is a data structure that contains the information necessary to reconstruct, or redo, changes made to the database by DML or DDL operations. Database recovery applies redo entries to data files to reconstruct lost changes.
- Shared Pool caches various types of program data.For example, the shared pool stores parsed SQL, PL/SQL code, system parameters, and data dictionary information. The shared pool is involved in almost every operation that occurs in the database. For example, if a user executes a SQL statement, then Oracle Database accesses the shared pool.



- Large Pool is an optional memory area intended for memory allocations that are larger than is appropriate for the shared pool.The large pool can provide large memory allocations for the following:
  - UGA for the shared server and the Oracle XA interface (used where transactions interact with multiple databases)
  - Message buffers used in the parallel execution of statements
  - Buffers for Recovery Manager (RMAN) I/O slaves
- Java Pool - is an area of memory that stores all session-specific Java code and data within the Java Virtual Machine (JVM). This memory includes Java objects that are migrated to the Java session space at end-of-call. For dedicated server connections, the Java pool includes the shared part of each Java class, including methods and read-only memory such as code vectors, but not the per-session Java state of each session. For shared server, the pool includes the shared part of each class and some UGA used for the state of each session. Each UGA grows and shrinks as necessary, but the total UGA size must fit in the Java pool space.
- Streams Pool l stores buffered queue messages and provides memory for Oracle Streams capture processes and apply processes. The Streams pool is used exclusively by Oracle Streams.Unless you specifically configure it, the size of the Streams pool starts at zero. The pool size grows dynamically as required by Oracle Streams.
- Fixed SGA - The fixed SGA is an internal housekeeping area. The size of the fixed SGA is set by Oracle Database and cannot be altered manually. The fixed SGA size can change from release to release.For example, the fixed SGA contains:
  - General information about the state of the database and the instance, which the background processes need to access
  - Information communicated between processes, such as information about locks

2. UNDERSTANDING ORACLE BACKGROUND PROCESSES

I used table ce_products for this task.

*SELECT **

FROM bl_3nf.ce_products

WHERE unit_cost<100;

	PRODUCT_ID	PRODUCT_SRCID	PRODUCT_SOURCE_SYSTEM	PRODUCT_SOURCE_ENTITY	PRODUCT	UNIT_PRICE	UNIT_COST	PRODUCT_BRAND_ID	PRODUCT_TYPE_ID	TA_UPDATE_DT	TA_INSERT_DT
1	108496	76	personnel_sales	src_products	Long-Wear Gel Eyeliner	792.51	634.01	6249	1705	26-FEB-22	26-FEB-22
2	108497	77	personnel_sales	src_products	Metallic Eye Shadow	941.92	753.54	6249	1702	26-FEB-22	26-FEB-22
3	108498	78	personnel_sales	src_products	Metallic Long-Wear Cream Shadow	277.17	221.74	6249	1602	26-FEB-22	26-FEB-22
4	108499	79	personnel_sales	src_products	Nail Lacquer	188.56	150.85	6249	1613	26-FEB-22	26-FEB-22
5	108500	80	personnel_sales	src_products	Natural Brow Shaper & hair Touch up	93.72	74.98	6249	1705	26-FEB-22	26-FEB-22
6	108501	81	personnel_sales	src_products	No Smudge Mascara	1103.34	882.67	6249	1693	26-FEB-22	26-FEB-22
7	108502	82	personnel_sales	src_products	Pot Rouge for Lips & Cheeks	2139.74	1711.79	6249	1598	26-FEB-22	26-FEB-22
8	108503	83	personnel_sales	src_products	Powder	1243.86	995.09	6249	1652	26-FEB-22	26-FEB-22
9	108504	84	personnel_sales	src_products	Protective Face Lotion	1762.01	1409.61	6249	1612	26-FEB-22	26-FEB-22
10	108505	85	personnel_sales	src_products	Retouching Powder	1457.99	1166.39	6249	1646	26-FEB-22	26-FEB-22
11	108506	86	personnel_sales	src_products	Retractable Lip	412.67	330.14	6249	1641	26-FEB-22	26-FEB-22
12	108507	87	personnel_sales	src_products	Rich Color Gloss	1201.32	961.06	6249	1667	26-FEB-22	26-FEB-22
13	108508	88	personnel_sales	src_products	Rich Lip Color	1635.03	1308.02	6249	1674	26-FEB-22	26-FEB-22
14	108509	89	personnel_sales	src_products	Sheer Color Cheek Tint	1225.81	980.65	6249	1602	26-FEB-22	26-FEB-22
15	108510	90	personnel_sales	src_products	Sheer Color Lip Gloss	27.31	21.85	6249	1667	26-FEB-22	26-FEB-22
16	108511	91	personnel_sales	src_products	Sheer Finish Loose Powder	1333.99	1067.19	6249	1646	26-FEB-22	26-FEB-22
17	108512	92	personnel_sales	src_products	Sheer Finish Pressed Powder	367.13	293.7	6249	1636	26-FEB-22	26-FEB-22

Step 1: create index

CREATE INDEX idx_btree ON bl_3nf.ce_products (unit_cost);

Step 2: execution plan

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			572	3
INDEX	IDX_BTREE	RANGE SCAN	572	3
Access Predicates				
UNIT_COST<100				
Other XML				
{info}				
info type="has_user_tab"				
yes				
info type="db_version"				
21.0.0.0				
info type="parse_schema"				
"BL_3NF"				
info type="plan_hash_full"				
3408659901				
info type="plan_hash"				
444855519				
info type="plan_hash_2"				

Step 3: update rows

Update bl_3nf.ce_products

set unit_cost=unit_cost+1;

commit;

Step 4: explanation

Update a row in the b-tree index table:

- check existing of query in library cache, if not exists - create an execution plan and add it to library cache
- place blocks from disk to cache
- update rows and index
- add information about transaction to redo log buffer
- write blocks on disk

3. RESOURCE CONSUMPTION

Step 1: run query

SELECT s.sale_id, c.channel_description, p.product, b.product_brand, pt.product_type, p.unit_cost

FROM CE_SALES s

Left JOIN CE_products p

on s.product_id=p.product_id

Left join ce_brands b

on p.product_brand_id=b.product_brand_id

Left join ce_product_types pt

on p.product_type_id=pt.product_type_id

left join ce_channels c

on s.channel_id=c.channel_id

Where p.unit_cost<100;

Step 2: select query id

select prev_sql_id from v$session where sid=sys_context('userenv','sid');--dnpvjvxvfy7rj

Step 3: check id

select sql_id, sql_text from v$sqltext where sql_id in ('dnpvjvxvfy7rj');

	SQL_ID	SQL_TEXT
1	dnpvjvxvfy7rj	els c on s.channel_id=c.channel_id Where p.unit_cost<100
2	dnpvjvxvfy7rj	es pt on p.product_type_id=pt.product_type_id left join ce_chann
3	dnpvjvxvfy7rj	p.product_brand_id=b.product_brand_id Left join ce_product_typ
4	dnpvjvxvfy7rj	roducts p on s.product_id=p.product_id Left join ce_brands b on
5	dnpvjvxvfy7rj	and, pt.product_type, p.unit_cost FROM CE_SALES s Left JOIN CE_p
6	dnpvjvxvfy7rj	SELECT s.sale_id, c.channel_description, p.product, b.product_br

Step 4: show resource consumption

I used V$SQL. It lists statistics on shared SQL area without the GROUP BY clause and contains one row for each child of the original SQL text entered. Statistics displayed in V$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.

*SELECT * FROM V$SQL WHERE SQL_ID = 'dnpvjvxvfy7rj';*

Query Result × | Script Output × | Query Result 1 ×

SQL | All Rows Fetched: 1 in 0.012 seconds

	SQL_TEXT	SQL_...	SQL_ID	SHARABLE_MEM	PERSISTENT_MEM	RUNTIME_MEM	SORTS	LOADED_VERSIONS	OPEN_VERSIONS	USERS_OPENING	FETCHES	EXECUTIONS	PX_SERVERS_EXECUTIONS	END_OF_FETCH_COUNT
1	SELECT s.sal...	SEL...	dnpvjvxvfy7rj	82513	311320	309304	0	1	0	0	2	2	0	0

Query Result × | Script Output × | Query Result 1 ×

SQL | All Rows Fetched: 1 in 0.012 seconds

	NT	USERS_EXECUTING	LOADS	FIRST_LOAD_TIME	INVALIDATIONS	PARSE_CALLS	DISK_READS	DIRECT_WRITES	DIRECT_READS	BUFFER_GETS	APPLICATION_WAIT_TIME	CONCURRENCY_WAIT_TIME	CLUSTER_WAIT_TIME
1	0	0	3	2022-03-03/11:31:03	2	2	70	0	70	416	527	0	0

Step 5: show execution plan

PLAN_TABLE_OUTPUT

```
Plan hash value: 1829196039

---------------------------------------------------------------------------------------
| Id  | Operation                | Name             | Rows  | Bytes | Cost (%CPU)| Time     |
---------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT         |                  | 57484 |  5613K|  7343   (1)| 00:00:01 |
|*  1 |  HASH JOIN RIGHT OUTER   |                  | 57484 |  5613K|  7343   (1)| 00:00:01 |
|   2 |   TABLE ACCESS FULL      | CE_CHANNELS      |     9 |   117 |     3   (0)| 00:00:01 |
|*  3 |   HASH JOIN              |                  | 57484 |  4883K|  7340   (1)| 00:00:01 |
|*  4 |    HASH JOIN RIGHT OUTER |                  |   572 | 41184 |    74   (0)| 00:00:01 |
|   5 |     TABLE ACCESS FULL    | CE_BRANDS        |   424 |  6360 |     3   (0)| 00:00:01 |
|*  6 |     HASH JOIN RIGHT OUTER|                  |   572 | 32604 |    71   (0)| 00:00:01 |
|   7 |      TABLE ACCESS FULL   | CE_PRODUCT_TYPES |   123 |  1722 |     3   (0)| 00:00:01 |
|*  8 |      TABLE ACCESS FULL   | CE_PRODUCTS      |   572 | 24596 |    68   (0)| 00:00:01 |
|   9 |    TABLE ACCESS FULL     | CE_SALES         |  1071K|    15M|  7258   (1)| 00:00:01 |
---------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   1 - access("S"."CHANNEL_ID"="C"."CHANNEL_ID"(+))
   3 - access("S"."PRODUCT_ID"="P"."PRODUCT_ID")
   4 - access("P"."PRODUCT_BRAND_ID"="B"."PRODUCT_BRAND_ID"(+))
   6 - access("P"."PRODUCT_TYPE_ID"="PT"."PRODUCT_TYPE_ID"(+))
   8 - filter("P"."UNIT_COST"<100)

Note
-----
   - this is an adaptive plan
```

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			57484	7343
HASH JOIN		RIGHT OUTER	57484	7343
Access Predicates				
S.CHANNEL_ID=C.CHANNEL_ID(+)				
TABLE ACCESS	CE_CHANNELS	FULL	9	3
HASH JOIN			57484	7340
Access Predicates				
S.PRODUCT_ID=P.PRODUCT_ID				
NESTED LOOPS			57484	7340
NESTED LOOPS				
STATISTICS COLLECTOR				
HASH JOIN		RIGHT OUTER	572	74
Access Predicates				
P.PRODUCT_BRAND_ID=B.PRODUCT_BRAND_ID(+)				
TABLE ACCESS	CE_BRANDS	FULL	424	3
HASH JOIN		RIGHT OUTER	572	71
Access Predicates				
P.PRODUCT_TYPE_ID=PT.PRODUCT_TYPE_ID(+)				
TABLE ACCESS	CE_PRODUCT_TYPES	FULL	123	3
TABLE ACCESS	CE_PRODUCTS	FULL	572	68
Filter Predicates				
P.UNIT_COST<100				
INDEX	IDX_BTREE3	RANGE SCAN		
Access Predicates				
S.PRODUCT_ID=P.PRODUCT_ID				
TABLE ACCESS	CE_SALES	BY INDEX ROWID	100	7258
TABLE ACCESS	CE_SALES	FULL	1071769	7258

Other XML
{info}
info type="has_user_tab"
yes
info type="db_version"
21.0.0.0
info type="parse_schema"
"BL_3NF"