Oracle Database 18c/19c

New Features for Database Developers

Philipp Salvisberg

@phsalvisberg https://www.salvis.com/blog

About Me

- Trivadian since April 2000
  - Senior Principal Consultant, Partner
  - Member of the Board of Directors
  - @phsalvisberg
  - https://www.salvis.com/blog
  - https://github.com/PhilippSalvisberg
- Database centric development
- Model Driven Software Development
- Author of free SQL Developer Extensions: PL/SQL Unwrapper, PL/SQL Cop, utPLSQL, plscope-utils, oddgen and Bitemp Remodeler

ORACLE ACE

trivadis makes IT easier.

Agenda

PL/SQL
- Qualified Expressions
- Polymorphic Table Functions (PTF)
- [NOT] PERSISTABLE
- Optional using clause in PTF 19c)

JSON
- SQL expression returning JSON data
- LOB results for json_value, json_query, json_table, json_object, json_array, json_arrayagg, item methods
- STRICT keyword
- Extended input data types
- New item methods: number(only), string(only), boolean(only), size, type
- JSON data guide enhancements
- JSON data dictionary views
- MV ON Statement support for materialized views
- JSON_EQUAL (ignoring whitespace differences)
- Performance improvements on JSON stored in LOBs
- Support for longer JSON names in search indexes
- Query rewrite for MVs with json_table/_exist/_value 19c)
- JSON update operations 19c)
- Mapping JSON data to/from object/collection types 19c)
- SQL/JSON syntax simplifications 19c)

SQL
- Analytic Views Enhancements
- Schema Only Accounts
- Data-bound collation support
- Private Temporary Tables
- Integration of AD Services into the Oracle Database
- Modifying Partition Strategy
- Online Merging of Partitions
- Inline External Tables
- Memoptimized RowStore (Fast Lookup)
- Approximate Top-N Functions (count, sum, rank)
- Unique Sequence Numbers Across Shards 19c)
- In-Memory External Tables on Hive and Bigdata 19c)
- Bitmap Based Count Distinct Functions (bitmap_count, ...) 19c)
- Hybrid Partitioned Tables 19c)
- Memoptimized RowStore Fast Ingest 19c)
- DISTINCT option for listagg 19c)
- Unified Auditing for Top Level Statements only 19c)

JDBC
- Accessing PL/SQL Associative Arrays (Keys and Values)
- REF Cursor as IN Bind Variable
- Java library for Reactive Streams Ingestion 19c)
- Set JDBC Properties via JDBC URL 19c)

About Old Features

Deprecated

- dbms_data_mining.get% functions
- dbms_xmlquery
- dbms_xmlsave
- Oracle Fail Safe 19c)
- Compilation parameter PLSQL_DEBUG 19c)
- …

Desupported

- Oracle Multimedia DICOM
- Oracle XML DB (dbms_xmlschema, …)
- Oracle Streams 19c)
- Oracle Multimedia 19c)
- PRODUCT_USER_PROFILE Table 19c)
- Non-CDB architecture 20c)
- …

See Database Upgrade Guide for details

Qualified Expressions

Idea – Simplify Initialization of Records and Arrays

Initialize Associative Array in Oracle Database 12c Release 2:

```
DECLARE
   TYPE t_string_type IS TABLE OF dept.loc%TYPE INDEX BY SIMPLE_INTEGER;
   t_dept t_string_type;
BEGIN
   t_dept(10) := 'Accounting';
   t_dept(20) := 'Research';
   t_dept(30) := 'Sales';
   t_dept(40) := 'Operations';
   dbms_output.put_line(t_dept(10)); -- Accounting
END;
```

Initialize Associative Array of Strings

```
DECLARE
   TYPE t_string_type IS TABLE OF dept.loc%TYPE INDEX BY SIMPLE_INTEGER;
   t_dept t_string_type := t_string_type (
                             10 => 'Accounting',
                             20 => 'Research',
                             30 => 'Sales',
                             40 => 'Operations'
                           );
BEGIN
    dbms_output.put_line(t_dept(10)); -- Accounting
END;
```

Initialize Record

```
DECLARE
   TYPE r_dept_type IS RECORD (
      deptno dept.deptno%TYPE,
      dname  dept.dname%TYPE,
      loc    dept.loc%TYPE
   );
   r_dept r_dept_type := r_dept_type (
                           deptno => 10,
                           dname  => 'Accounting',
                           loc    => 'New York'
                         );
BEGIN
   dbms_output.put_line(r_dept.dname); -- Accounting
END;
```

Initialize Associative Array of Records

```
DECLARE
   TYPE r_dept_type IS RECORD (
      dname  dept.dname%TYPE,
      loc    dept.loc%TYPE
   );
   TYPE t_dept_type IS TABLE OF r_dept_type INDEX BY SIMPLE_INTEGER;
   t_dept t_dept_type := t_dept_type (
                            10 => r_dept_type ('Accounting', 'New York'),
                            20 => r_dept_type ('Research', 'Dallas'),
                            30 => r_dept_type ('Sales', 'Chicago'),
                            40 => r_dept_type ('Operations', 'Boston')
                         );
BEGIN
   dbms_output.put_line(t_dept(10).dname); -- Accounting
END;
```

Polymorphic Table Functions

Idea – Dynamic I/O Structure

```
SELECT * FROM ptf.hashed(dept);

DEPTNO DNAME      LOC      MD5_HASH
------ ---------- -------- --------------------------------
    10 ACCOUNTING NEW YORK 857798D8119A2ABBDA597B15E13E248D
    20 RESEARCH   DALLAS   DF8644CDEDD845BB6DBE0D7D98DCDB46
    30 SALES      CHICAGO  A64D0A97EF37533C3336C8C546D8D8A3
    40 OPERATIONS BOSTON   77EB463B523C88021037A154378BF5D0
```

```
SELECT * FROM ptf.hashed(emp);

EMPNO ENAME  JOB         MGR HIREDATE    SAL  COMM DEPTNO MD5_HASH
----- ------ --------- ----- -------- ----- ----- ------ --------------------------------
 7369 SMITH  CLERK      7902 17.12.80   800           20 EAB6774264D52D9589A9864B4E168095
 7499 ALLEN  SALESMAN   7698 20.02.81  1600   300     30 650663EFB818F4FE422027A0F8653084
…
```

Passing Named Queries

```
WITH empdept AS (
         SELECT e.empno, e.ename, e.deptno, d.dname, e.sal
           FROM emp e
           JOIN dept d
             ON d.deptno = e.deptno
          WHERE e.deptno = 10
      )
SELECT * FROM ptf.hashed(empdept);

EMPNO ENAME   DEPTNO DNAME         SAL MD5_HASH
----- ------ ------ ---------- ----- --------------------------------
 7782 CLARK      10 ACCOUNTING  2450 97BD223F2A7AD85AE0034D1828104C28
 7839 KING       10 ACCOUNTING  5000 7FECE7D987171325B6A09E87E95694D7
 7934 MILLER     10 ACCOUNTING  1300 65D9DD948B3A406B683EAD6BD86FE290
```

Predicate Pushdown

```
SELECT empno, ename, sal, md5_hash
  FROM ptf.hashed(emp)
 WHERE deptno = 10;

-------------------------------------------------------------------------------------------------
| Id  | Operation                           | Name    | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                    |         |     3 |  6111 |     2   (0)| 00:00:01 |
|   1 |  POLYMORPHIC TABLE FUNCTION         | HASHED  |     3 |   261 |            |          |
|   2 |   TABLE ACCESS BY INDEX ROWID BATCHED| EMP     |     3 |   114 |     2   (0)| 00:00:01 |
|*  3 |    INDEX RANGE SCAN                 | EMP_I01 |     3 |       |     1   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------------
```

- Pass-through columns
- PARTITION BY columns of table semantics PTF

Declaration (Interface)

```
CREATE OR REPLACE PACKAGE ptf AS
   FUNCTION describe(
      io_tab IN OUT dbms_tf.table_t
   ) RETURN dbms_tf.describe_t;

   PROCEDURE fetch_rows;

   FUNCTION hashed(in_tab IN TABLE)
      RETURN TABLE PIPELINED
      ROW POLYMORPHIC USING ptf;
END ptf;
```

- describe function
  - Called during cursor compilation
  - Defines output columns
  - Mandatory
- fetch_rows procedure
  - Called during execution 0..n times
  - Rowset size calculated at runtime (<=1024, partitions: 0..n rowsets)
  - Optional
- hashed function
  - Table argument is mandatory
  - Additional arguments are optional and passed to describe, fetch_rows
  - Row or table semantic
  - Can be a standalone function
  - Using clause can be omitted in this case (19c only)

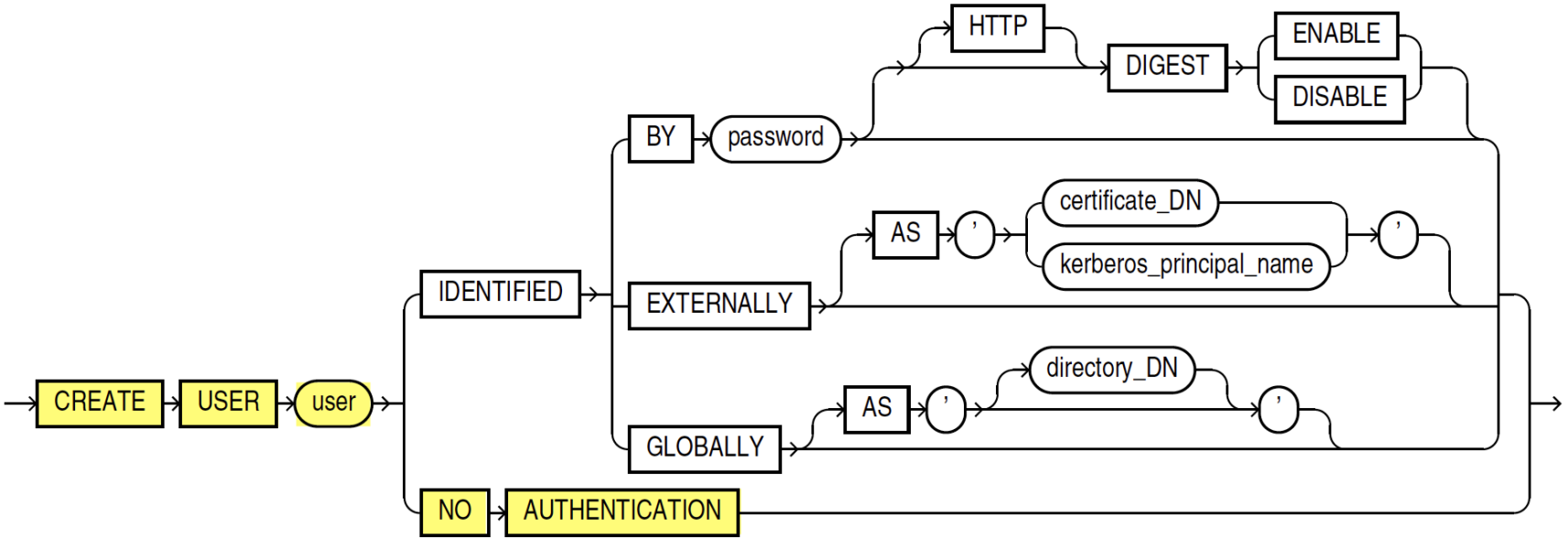
Definition (Implementation)

```
CREATE OR REPLACE PACKAGE BODY ptf AS
   FUNCTION describe(
      io_tab IN OUT dbms_tf.table_t
   ) RETURN dbms_tf.describe_t IS
      l_new_cols dbms_tf.columns_new_t;
   BEGIN
      FOR i IN 1 .. io_tab.column.count()
      LOOP
         io_tab.column(i).pass_through := TRUE;
         io_tab.column(i).for_read     := TRUE;
      END LOOP;
      l_new_cols(1) :=
         dbms_tf.column_metadata_t(
            name    => 'md5_hash',
            type    => dbms_tf.type_varchar2,
            max_len => 32
         );
      RETURN dbms_tf.describe_t(
                new_columns => l_new_cols
             );
   END describe;
```

```
   PROCEDURE fetch_rows IS
      l_row_set   dbms_tf.row_set_t;
      l_md5_hash  dbms_tf.tab_varchar2_t;
      l_row_count pls_integer;
   BEGIN
      dbms_tf.get_row_set(
         rowset    => l_row_set,
         row_count => l_row_count
      );
      FOR i IN 1 .. l_row_count LOOP
         l_md5_hash(i) :=
            dbms_crypto.hash(
               src => CAST(dbms_tf.row_to_char(
                          l_row_set, i) AS CLOB),
               typ => dbms_crypto.hash_md5
            );
      END LOOP;
      dbms_tf.put_col(1, l_md5_hash);
   END fetch_rows;
END ptf;
```

Schema Only Accounts

Idea – User Owning Objects w/o Login Credentials



Source: SQL Language Reference 18c

Create Schema

```
CREATE USER demo_schema -- no authentication is default
   DEFAULT TABLESPACE users
   TEMPORARY TABLESPACE temp
   QUOTA UNLIMITED ON users;

GRANT connect, resource TO demo_schema;
```

Create Proxy User

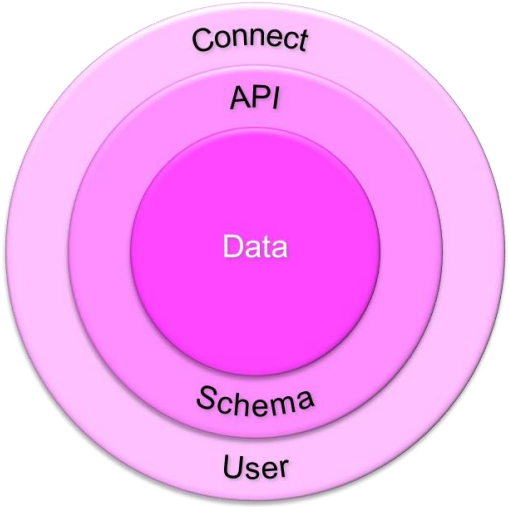
```
CREATE USER philipp identified BY philipp;

GRANT connect to philipp;

ALTER USER demo_schema GRANT CONNECT THROUGH philipp;
```

Connect Via Proxy User

```
[oracle@odb180 /]$ sqlplus philipp[demo_schema]/philipp

…
Connected to:
Oracle Database 18c Enterprise Edition Release 18.0.0.0.0 - Production
Version 18.5.0.0.0

SQL> SELECT user, sys_context('userenv', 'proxy_user') proxy FROM dual;

USER                           PROXY
------------------------------ ------------------------------
DEMO_SCHEMA                    PHILIPP
```

Advantages

- Schema account cannot be “locked” through
  - Password expiration
  - Wrong password entry
- Managed schema access
  - No password sharing
  - Proxy users have control over their passwords
- Semantic separation of users and schemas
  - Users do not own objects
  - Schemas own objects



The Pink Database Paradigm (PinkDB)

Private Temporary Tables

Idea – Create Temporary In-Memory Table

FAQs – the Differences	Global Temporary Table	Private Temporary Table
Storage type?	Disk	in Memory
Indexes?	Yes	No
Constraints (not null, check)?	Yes	No
Works in read-only environment?	No	Yes
Implicit commit on DDL (create, truncate, drop table)?	Yes	No
DML honor transactions?	Yes	Yes
Structure visible in other sessions?	Yes	No
Predefined table name prefix?	No	Yes (ora$ptt_)
Accessible via database link?	Yes	No

Example – No Implicit Commit by “CREATE PTT”

```
CREATE TABLE t1 (c1 INTEGER, c2 VARCHAR2(30));
INSERT INTO t1 VALUES (1, 'to be committed');
COMMIT;
INSERT INTO t1 VALUES (2, 'to be rollbacked');

CREATE PRIVATE TEMPORARY TABLE ora$ptt_t2 (c1 INTEGER, c2 VARCHAR2(30))
   ON COMMIT PRESERVE DEFINITION;
ROLLBACK;

SELECT * FROM t1;

        C1 C2
---------- ------------------------------
         1 to be committed
```

Memoptimized RowStore (Fast Lookup)

Idea – Fast Primary Key Access

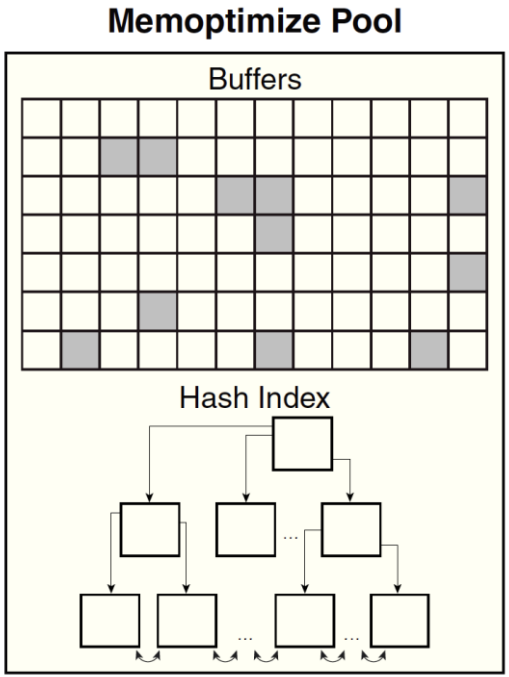
```
SELECT value
  FROM t
 WHERE key = :key
```

- Store table in memory
- Bypass the SQL execution layer
- Execute directly in the data access layer

Configure Database

- Store table completely in the SGA
- Memoptimized Pool
  - 75% for Table Buffers in Memoptimize Buffer Area
  - 25% for Primary Keys in Hash Index
- Configure Memoptimized Pool:

```
ALTER SYSTEM
   SET memoptimize_pool_size = 100M
   SCOPE=SPFILE;
SHUTDOWN IMMEDIATE
STARTUP
```



Source: Database Concepts 18c

Configure Memoptimized Table

```
CREATE TABLE t4 (
   key     INTEGER              NOT NULL,
   value   VARCHAR2(30 CHAR)    NOT NULL,
   CONSTRAINT t4_pk PRIMARY KEY (key)
)
SEGMENT CREATION IMMEDIATE
MEMOPTIMIZE FOR READ
```

- Delayed segment creation is not supported
- Additional step required to populate data into the Memoptimized Pool

Populate Table Into Memoptimized Pool

```
BEGIN
   dbms_memoptimize.populate(
     schema_name => USER,
     table_name  => 'T4'
   );
END;
```

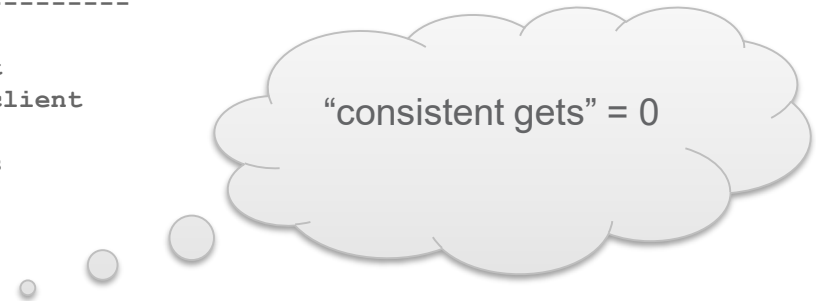
- Memoptimized Pool is populated in the background
- The “memopt r rows populated” statistics indicates the progress
- In a multitenant architecture the population is managed in the CDB!

Query with Autotrace – Execution Plan

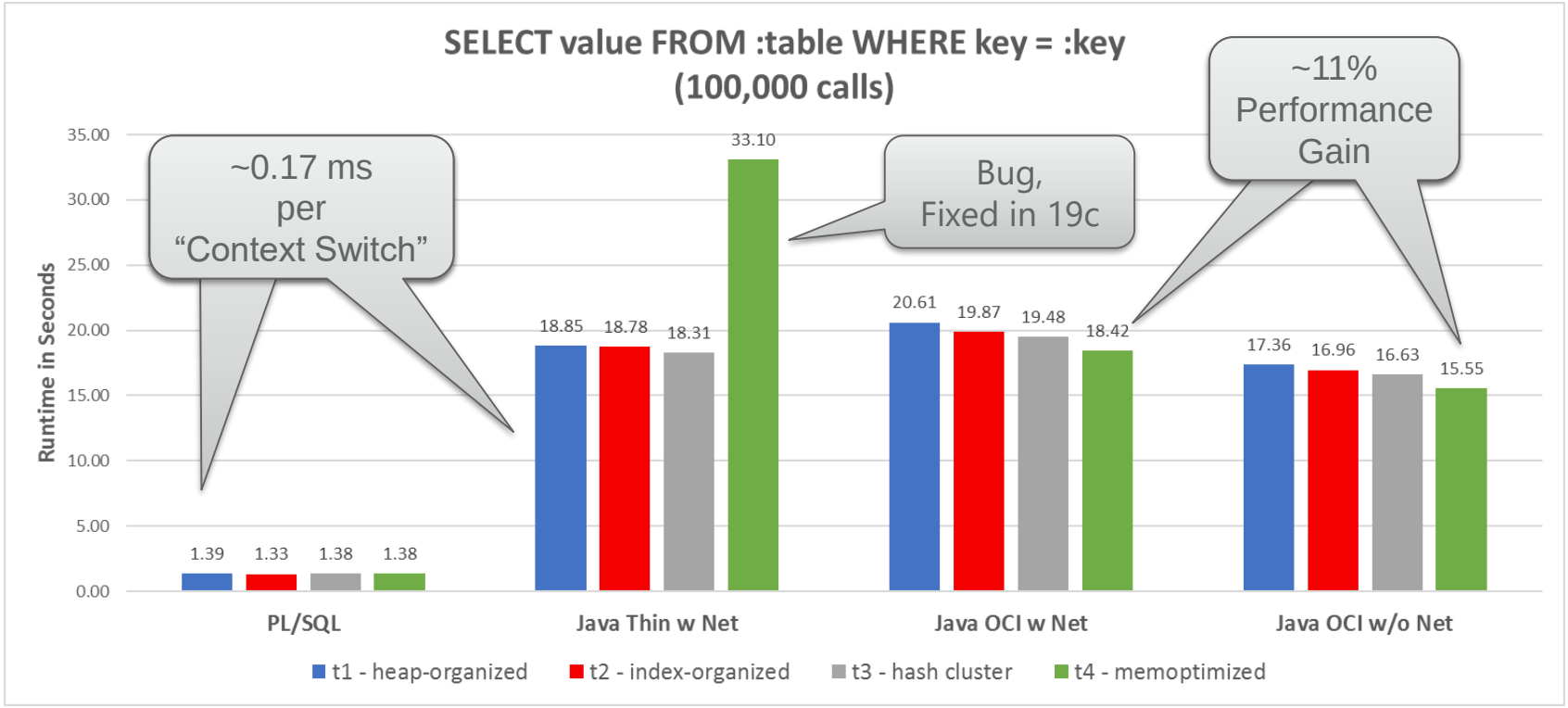
```
SELECT * FROM t4 WHERE key = 42;

       KEY VALUE
---------- ------------------------------
        42 UKPBW05FQ1

-------------------------------------------------------------------------------------------------
| Id  | Operation                                | Name  | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                         |       |     1 |    24 |     2   (0)| 00:00:01 |
|   1 |  TABLE ACCESS BY INDEX ROWID READ OPTIM| T4    |     1 |    24 |     2   (0)| 00:00:01 |
|*  2 |   INDEX UNIQUE SCAN READ OPTIM         | T4_PK |     1 |       |     1   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------------

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

   2 - access("KEY"=42)
```

Query with Autotrace – Statistics

```
Statistics
----------------------------------------------------------
              43  Requests to/from client
              43  SQL*Net roundtrips to/from client
             605  bytes received via SQL*Net from client
           83447  bytes sent via SQL*Net to client
               3  calls to get snapshot scn: kcmgss
               2  calls to kcmgcs
               2  execute count
               1  memopt r hits
               1  memopt r lookups
              44  non-idle wait count
               2  opened cursors cumulative
               1  opened cursors current
               2  parse count (total)
               1  session cursor cache count
               1  sorts (memory)
            2011  sorts (rows)
              46  user calls
```



Performance



Run in Docker Container with Oracle Database 18c, Version 18.5 and "_exadata_feature_on=TRUE"

Availability

This feature is available for the following Oracle Database offerings only:

- Oracle Database Enterprise Edition on Engineered Systems (EE-ES)
- Oracle Database Exadata Cloud Service (ExaCS)
- Oracle Database Cloud Service Enterprise Edition – Extreme Performance (DBCS EE-EP)

Prerequisites

- Table marked as MEMOPTIMIZE FOR READ
- Table is heap-organized
- Table has a primary key
- Primary key is not an identity column
- Table is not compressed
- Table is not reference-partitioned
- Table has at least one segment
- Table loaded into the memoptimized pool using dbms_memoptimize.populate
- Table fits completely in the memoptimized pool
- Query is in the format SELECT <column_list> FROM <table> WHERE <pk> = <value>
- STATISTICS_LEVEL is not set to ALL
- No GATHER_PLAN_STATISTICS hint
- SQL trace is not enabled
- Query is not executed from PL/SQL
- Query is not executed from a Java stored procedure with default database connection
- Query is executed via OCI (18c, fixed in 19c)

Core Messages

Upgrade – The New Features Are Worth It

- Polymorphic Table Functions
- Private Temporary Tables
- Schema Only Accounts
- Qualified Expressions
- Memoptimized RowStore
- … and more …

Questions and Answers...

Philipp Salvisberg
Senior Principal Consultant

Tel. +41 58 459 52 31
philipp.salvisberg@trivadis.com

@phsalvisberg