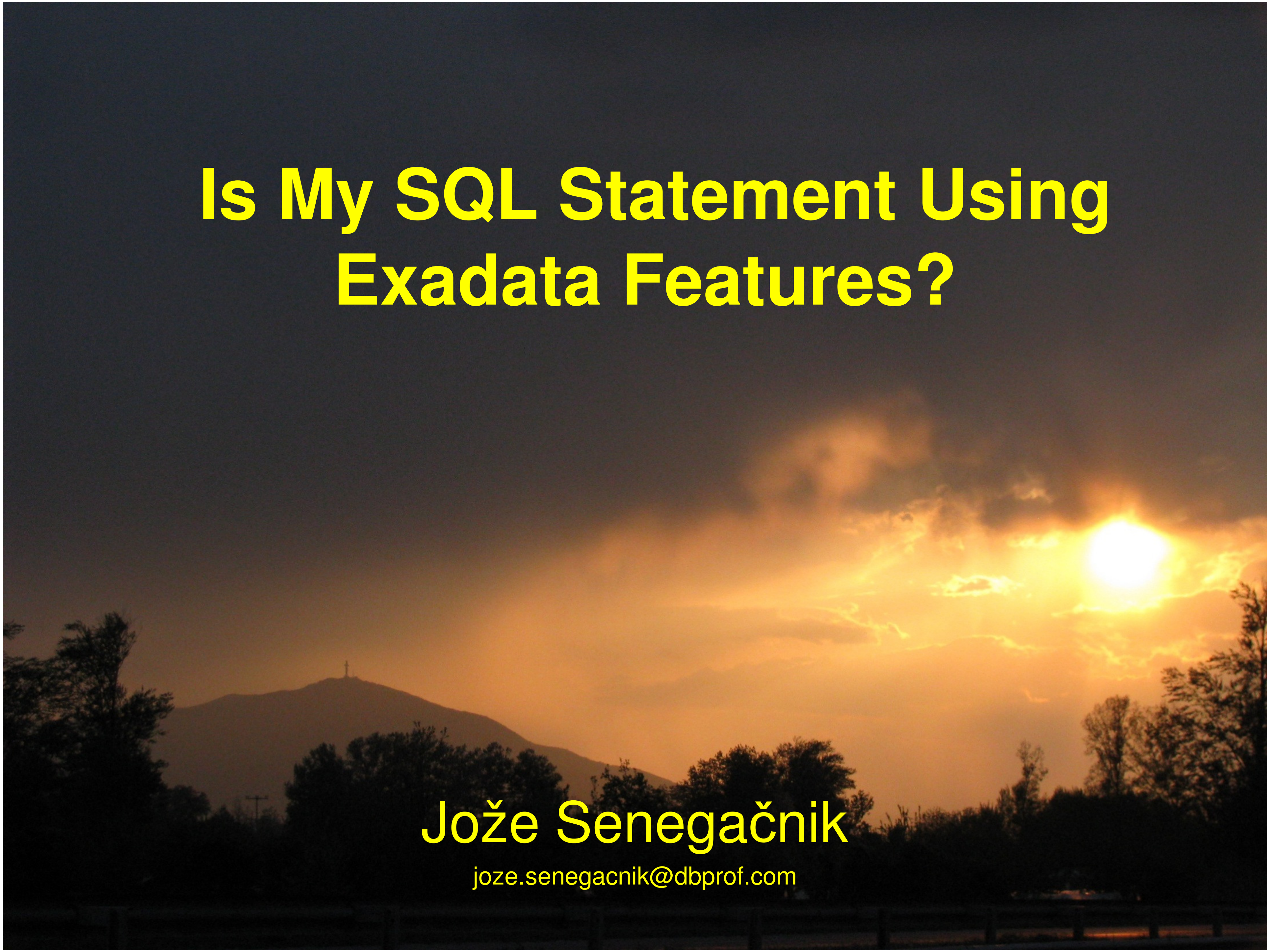


Is My SQL Statement Using Exadata Features?

Jože Senegačnik

joze.senegacnik@dbprof.com

About the Speaker

Jože Senegačnik

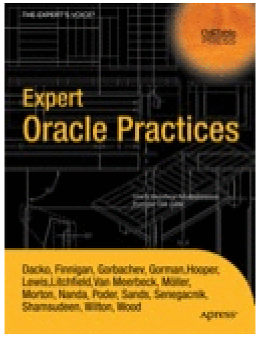
- Owner of Dbprof d.o.o.
- First experience with Oracle Version 4.1 in 1988
- 25+ years of experience with Oracle RDBMS.
- Proud member of the OakTable Network www.oaktable.net
- Oracle ACE Director
- Co-author of the OakTable book “Expert Oracle Practices” by Apress (Jan 2010)
- VP of Slovenian OUG (SIOUG) board
- CISA – Certified IS auditor
- Blog about Oracle: http://joze-senegacnik.blogspot.com

- PPL(A) / IR(SE) – private pilot license, instrument rating
- Blog about flying: http://jsenegacnik.blogspot.com
- Blog about Building Ovens, Baking and Cooking: http://senegacnik.blogspot.com

OakTable.net

ORACLE ACE Director

Oracle ACE


THE EXPERT'S VOICE
Expert
Oracle Practices
Dacko, Finnigan, Gorbachev, Gorman, Hooper,
Lewis, Litchfield, Van Meerbeeck, Möller,
Morton, Nanda, Poder, Sands, Senegacnik,
Shamsudeen, Wilton, Wood
Apress

Exadata at Oracle HQ (OOW 2011)

Speed Test

- My laptop

```
SQL> with a as (select /*+ materialize */ 1 from dual connect by level <= 14)
  2  select count(*) from a,a,a,a,a,a,a;

  COUNT(*)
----------
 105413504

Elapsed: 00:00:04.77
```

- Exadata

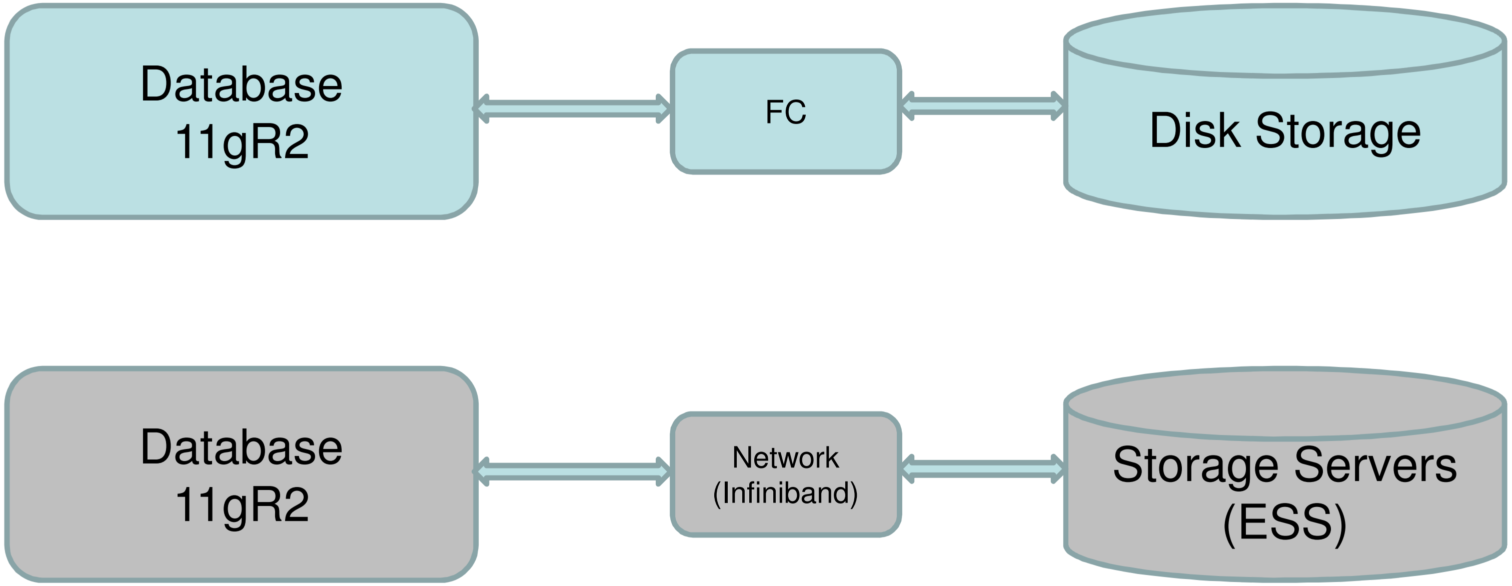
```
SQL> with a as (select /*+ materialize */ 1 from dual connect by level <= 14)
  2  select count(*) from a,a,a,a,a,a,a;

  COUNT(*)
----------
 105413504

Elapsed: 00:00:04.48
```

Short Introduction to Exadata

- What is different:
  - Same database (in 11gR2 almost not aware that it runs on top of special storage)
  - Fast Network (Infiniband)
  - Storage Servers (not just regular disks/storage) - Exadata Storage Server (ESS)



Storage Servers – Exadata Specifics

- Synonymous terms:
  - Storage Server,
  - Cell Server
  - Cell
- Synonyms for Exadata Operations:
  - Smart Scan
  - Offload
- Exadata software installed on storage cells manages smart scans
  - Column projection
  - Predicate filtering
  - Storage indexes
  - Simple Joins
  - Function Offloading
  - Smart Flash Cache
  - Virtual Column Evaluation
  - Hybrid Columnar Decompression
  - Decryption

Architecture Benefits

- Infiniband 40Gbit/s throughput
- Fast File Creation
  - Only metadata is sent to storage cells
- Smart scans read encrypted data

Smart Scan

- A Smart Scan is operation on disk storage represented as a wait event at the database level
- Possible operations in a Smart Scan?
  - **Column Projection**
  - **Predicate Filtering**
  - **Storage Indexes Access**
  - Virtual Column Computation
  - Function Offloading
  - …

Smart Scan Features

- **Column Projection**
  - Only the referenced columns are returned from smart scan
    - `SQL> select empno, ename from employees;`
    - Smart scan still has to read all table data (parsing each table row)
- **Predicate Filtering**
  - Only the rows that match filter are returned
  - Smart scan still has to read all table data (parsing each row)
- **Storage Indexes**
  - Storage index is in-memory index built on the fly by a cell server using WHERE clause
  - Used for fast determination in which parts of the table data is present / not present
  - Can reduce the amount of data read from disk

Smart Scan Prerequisites

- Smart Scan is a part of the Query (reading data) which is executed on the Storage Server

- For Smart Scan the following must be true:
  - Full Table Scan / Fast Full Index Scan
  - Direct Path Read

- Direct Path Read
  - physical read
  - The blocks that are read are passed to the PGA and completely bypass the SGA (buffer cache)
  - No blocks are read from the Buffer Cache

'**direct path read**' (MOS note 793845.1)

- There have been changes in 11g in the heuristics to choose between direct path reads or reads through buffer cache for serial table scans.

- In 10g, serial table scans for "large" tables used to go through cache (by default) which is not the case anymore. In 11g, this decision to read via **direct path** or **through cache** is based on the **size of the table**, **buffer cache size** and various other stats.

- Direct path reads are faster than scattered reads and have less impact on other processes because they avoid latches.

Direct Path Read Operation

- Oracle preforms direct path read in these cases:
  - Parallel Queries
  - For all full table / fast full index scan operations if we set hidden parameter "_serial_direct_read" = TRUE
  - Since 11g when a query is made on a "big" table
    - how many table blocks are cached in the buffer cache at the execution time
    - Beyond certain percentage a table may not qualify for a smart scan
    - Storage server decide whether a smart scan will be performed, not the CBO
- The bigger your buffer cache, the smaller your tables will appear
- Smaller buffer cache means more smart scans!

Direct Path Read (continued)

Conclusions from Frits Hoogland's presentation about "About multiblock reads"

- Direct path read is decision in IO codepath of a full scan.
  - NOT an optimizer decision(!)
  - In Oracle version 11, a read is done buffered (through buffer cache), unless database decides to do a direct path read
- Direct path read decision is influenced by
  - Type of read (FTS or FFIS)
  - Size of segment (> 5 * _small_table_threshold)
  - Number of blocks cached (~ 50%)
- By default, direct path read (AIO – asynchronous IO) uses two I/O slots.
- 'autotune' scales up in steps.
- Direct path code has an 'autotune' function, which can add IO slots in order to be able to use more bandwidth
- Direct path 'autotune' works for PX reads too!

Is Smart Scan Always Used?

- Is Smart Scan Always Used?
- NO! ☹
- Why not?
- Smart Scan requires:
  - Full Table Scan – not table access by ROWID
    - Queries using index access path don't benefit from smart scans
  - Direct Path Read
- SQL Functions:
  - Only some functions used in WHERE clause can use Smart Scan
  - Will discuss later
- No smart scan on index-organized table (IOT), Clustered tables
- Smart Scan stores result in PGA thus bypassing buffer cache

Exadata and Indexes

- The Rumor:
  - You should drop all your indexes on the Exadata
- The Truth:
  - You will still need indexes for index access of single rows (PK) or relatively small number of rows when access via index is cheaper.
  - Exadata is fast, but repetitive full scans on relatively big tables can bring even Exadata to the knees.
  - When index access is in question one should check the amount of work performed by Smart Scans in comparison with the amount of work performed by index access
  - Dropping indexes is beneficial as they are slowing down DML (index maintenance costs – insert, update, delete operations)

Was A Smart Scan Done?

- **We have to check SQL level “statistics” in GV$SQL**
  - **GV$SQL.IO_CELL_OFFLOAD_ELIGIBLE_BYTES** > 0 => Smart Scan was done
- This amount of IO saved by Smart Scan can be calculated

```
select last_load_time, sql_id, child_number,
decode (IO_CELL_OFFLOAD_ELIGIBLE_BYTES,0,'No','Yes')
Offloadable
from v$sql
where sql_text like '%&sql_text%'
order by last_load_time desc;
```

Storage Index Check

- Session level statistics
  - “**cell physical IO bytes saved by storage index**”

```
select s.name, m.value
from v$statname s, v$mystat m
where name like ('%storage%')
and s.statistic# = m.statistic#
```

Enabling/Disabling Exadata Features

- **Column Projection/Predicate Filtering**
  - Enable
  - alter session set cell_offload_processing = true
  - Disable
  - alter session set cell_offload_processing = false

- **Storage Indexes**
  - Enable
  - alter session set "_kcfis_storageidx_disabled"=false
  - Disable
  - alter session set "_kcfis_storageidx_disabled"=true

Column Projection

- The Storage Servers handle Column Projection
- Only columns which are referenced in the query are returned.
  - select list and join columns
- As only the required columns are returned the amount of data transferred between the Storage Cell and database is significantly reduced
  - Required PGA size for such query is also significantly reduced, thus reduced memory need at the database level
- Requires offloaded query
- Statistics available in the v$mystat view

```
SQL> select /*+ &user */  count(*)  from class_sales ;
Enter value for user: joze6
old   1: select /*+ &user */  count(*) from class_s¸aales
new   1: select /*+ joze6 */  count(*) from class_sales

  COUNT(*)
----------
  90000000

Elapsed: 00:00:04.57

SQL> @dbstat
Enter value for sql_text: joze6
old   7: where    sql_text like '%&SQL_TEXT%'
new   7: where    sql_text like '%joze6%'

LAST_LOAD_TIME      SQL_ID        CHILD_NUMBER OFF
------------------- ------------- ------------ ---
2013-05-17/00:24:45 d74f0nbwgkkfc            0 No
2013-05-17/00:24:32 1m88p87jaxvtp            0 Yes
2013-05-16/06:18:55 d74f0nbwgkkfc            0 No

Elapsed: 00:00:00.24
```

```
PLAN_TABLE_OUTPUT
------------------------------------------------------------------------------------
SQL_ID  1m88p87jaxvtp, child number 0
-------------------------------------
select /*+ joze6 */  count(*) from class_sales

Plan hash value: 3145879882

-----------------------------------------------------------------------------------
| Id  | Operation                    | Name          | Rows  | Cost (%CPU)| Time      |
-----------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |               |       |   317K(100)|           |
|   1 |  SORT AGGREGATE              |               |     1 |            |           |
|   2 |   TABLE ACCESS STORAGE FULL| CLASS_SALES |    90M|   317K   (1)| 00:41:22 |
-----------------------------------------------------------------------------------


14 rows selected.
```

```
SQL>  select /* &user */ distinct CUST_TYPE from class_sales;
```

```
C
-
Y
N

Elapsed: 00:00:08.79

SQL> @dbstat
Enter value for sql_text: joc9

LAST_LOAD_TIME       SQL_ID        CHILD_NUMBER OFF
-------------------- ------------- ------------ ---
2013-05-17/01:12:30 7am2f4kyx3396             0 No
2013-05-17/01:09:28 0z7vz5w9t8hhp             0 Yes
2013-05-16/07:06:05 7am2f4kyx3396             0 No

Elapsed: 00:00:00.23
```

```
SQL> select decode(name,
  2  'cell physical IO bytes saved by storage index', 'SI
Savings',
'cell physical IO interconnect bytes returned by smart scan',
'Smart Scan') as stat_name,
  4  value/1024/1024 as value
  5  from v$mystat s, v$statname n
  6  where s.statistic# = n.statistic#
  7  and n.name in ('cell physical IO bytes saved by storage
index',
  8   'cell physical IO interconnect bytes returned by smart
scan');

STAT_NAME       VALUE
---------- ----------
SI Savings          0
Smart Scan 1055.22408

Elapsed: 00:00:00.00
```

Storage Index Usage

```
select /* &USER */ count(*)
from   class_sales
where  currency_type is null;

COUNT(*)
----------
         1

Elapsed: 00:00:00.11
SQL> select decode(name, 'cell physical IO bytes saved by storage index', 'SI Savings',
  3  'cell physical IO interconnect bytes returned by smart scan', 'Smart Scan') as stat_name,
  4  value/1024/1024 as value
  5  from v$mystat s, v$statname n
  6  where s.statistic# = n.statistic#
  7  and n.name in ('cell physical IO bytes saved by storage index',
  8   'cell physical IO interconnect bytes returned by smart scan');

STAT_NAME       VALUE
---------- ----------
SI Savings 8173.73438
Smart Scan .001724243

Elapsed: 00:00:00.01
```

Check Offloading

```
select sql_id,
        decode(IO_CELL_OFFLOAD_ELIGIBLE_BYTES,0,'No','Yes') Offloaded,
        decode(IO_CELL_OFFLOAD_ELIGIBLE_BYTES,0,0,
        100*
        (IO_CELL_OFFLOAD_ELIGIBLE_BYTES/IO_INTERCONNECT_BYTES)/
        IO_CELL_OFFLOAD_ELIGIBLE_BYTES) as  "IO_SAVED"
from v$sql
where sql_text like '%&sql_text%';
```

*True most of the time

Source: Kerry Osborne

Predicate Filtering

- Storage server evaluates the query predicates.
- Only rows which pass the predicates are returned from storage server.
- Normal database reads all required rows (blocks) and predicates are applied after.
- Result from storage server is stored in PGA.
- Result is already at least partially processed so there is less work on database level.

Offloadable Functions

- Not every function can be offloaded.
- Offloadable functions are defined in **v$sqlfn_metadata**

```
SQL> select OFFLOADABLE, count(*) from v$sqlfn_metadata group by
OFFLOADABLE;

OFF    COUNT(*)
--- ----------
NO          530
YES         393
```

- When function is non-offloadable the data is returned to the SGA as in traditional systems
- Functions used in where clause are executed for every single row (at partucular step of the execution plan), so offloading is very beneficial to filter out as many rows as possible.

Offloadable Functions (cont.)

```
SQL> select distinct name from v$sqlfn_metadata where ANALYTIC='YES';

NAME
------------------------------
STDDEV
MAX
LAG
RANK
MIN
COUNT
SUM
AVG
LEAD
…

42 rows selected.

SQL> select name from v$sqlfn_metadata where ANALYTIC='YES' and OFFLOADABLE='YES';

no rows selected
```

Storage Indexes

- Used to eliminate disk I/O operations.
- The only Exadata feature which really eliminates disk reads.
- Built automatically by the cell servers for a maximum of 8 columns per table.
- No documented way to alter or tune them.
- Storage index store the min and max column values for disk storage units (1MB by default)
- So during the query the storage units which don't contain the requested values are skipped
- Highly dependent on data distribution, therefore very effective on sorted or partitioned data
- There were many bugs related to their usage in past.
- Problematic because they are created on the fly and may or may not be present so the response time can differ significantly.

Storage Indexes Usage

- Requirement for Storage Index usage besides smart scan:
  - A where clause with at least one predicate
  - Smart Scan (of course)
  - A simple comparison operator (=, <, >, BETWEEN, IS (not) NULL, etc.)

- Storage Indexes are used:
  - Multi-column predicates
  - Joins
  - Parallel Queries
  - HCC tables
  - Partitions

When Storage Indexes Are Not Used

- Not Equals (!=) operators
  - Logical, isn't it
- LOB's / CLOBs
- Column encryption
- Wildcards
- Where clauses with sub-queries
- Clustered Tables / IOT

Not Equals (!=) operator

```
select count(trans_id)
from   class_sales
where  trans_id <>'a‘;

COUNT(TRANS_ID)
---------------
       90000000

Elapsed: 00:00:04.67

SQL> select decode(name, 'cell physical IO bytes saved by storage index', 'SI Savings',
    'cell physical IO interconnect bytes returned by smart scan', 'Smart Scan') as stat_name,
     value/1024/1024 as value
     from v$mystat s, v$statname n
     where s.statistic# = n.statistic#
     and n.name in ('cell physical IO bytes saved by storage index',
     'cell physical IO interconnect bytes returned by smart scan');

STAT_NAME       VALUE
---------- ----------
SI Savings          0
Smart Scan 1055.12666
```

Storage Indexes (cont.)

- Storage Indexes are persisted in the memory of the storage cells, not in Smart Flash Cache, and not on disk
- Rebuilt after a storage cell restart or when a column for which a storage index was built was updated.
- Able to detect NULL values – unlike b-tree indexes
- New feature with undocumented “features” – bugs
  - Date strings which use 2 digit years may ignore a Storage Index
- No Smart Scan, no Storage Index usage!

Conclusions

- Exadata Features are Cumulative
- An offloaded full table scan or fast full index scan which is offloaded can take advantage:
  - Column Projection which reduces the size of the return set
  - Predicate Filtering further reduces the return set
  - Smart scan can be used on partitioned tables
  - Tables which are pinned in the Exadata Smart Flash Cache (ESFC) can be smart scanned

How To Get Things Working

- Queries must be executed in Direct Path mode. Force Direct Path mode by:
  - Forcing Parallelization
  - Forcing Full Table Scans (drop indexes / make them invisible)
  - Using appropriate hints (PARALLEL / FULL / INDEX_FFS)
  - Setting the instance parameter _serial_direct_read = true

- Verifying that Smart Scans were performed
  - Verify that query was OFFLOADED
  - Verify that Storage Indexes are being used
  - SQL trace shows “smart table/index scan”, Active session history,…

Exadata Smart Flash Cache

- Objects are automatically cached unless the object level parameters are used:
  - NONE - Never cache this object
  - DEFAULT - Automatic caching (default setting for single block reads)
  - KEEP - Pin an object in ESFC and allow SmartScans to find it

- To pin a table in ESFC use the following command:

```
SQL> ALTER TABLE customers STORAGE (CELL_FLASH_CACHE KEEP);
```

- To un-pin a table in ESFC use the following command:

```
SQL> ALTER TABLE customers STORAGE (CELL_FLASH_CACHE DEFAULT);
```

ESFC (cont.)

- Only 80% of ESFC may be used for table pinning, the rest is used for automated caching.

- Pinned objects are removed from ESFC when:
  - Object is dropped or truncated
  - Object is not accessed for 48 hours
  - Object is downgraded to DEFAULT or NONE

References

- Frits Hoogland: About multiblock reads
  - http://fritshoogland.files.wordpress.com/2012/06/about-multiblock-reads-v2.pdf
- Enkitec Exadata Education – Optimizing Exadata Performance

Thank you for your interest!

Q&A