Oracle Database 11g New Features: Best Practices to Improve Scalability, Performance & High Availability

Session# S307729

Ami Aharonovich
Oracle Certified Professional
Independent Oracle DBA Consultant & Instructor
aaharon@gmail.com
About Myself

- Oracle Certified Professional (OCP) DBA
- Over nine years of expertise as an Oracle database developer and administrator
- Working as an independent Oracle DBA consultant and instructor
- Specializing in training Oracle University official courses
- Expertise and hands-on experience using core Oracle Database technologies in versions 8, 8i, 9i, 10g, and 11g
- Board member in Israel Oracle User Group
Agenda

• It’s All About Caching
• Virtual Columns
• Partitioning to Perfection
• Locking Enhancements
• Read Only Tables
• Miscellaneous New Features
• Real-Life Example: Using Oracle11g Interval Partitioning with Hash Sub-Partitioning
More Than 20 Reasons to Stay Here...

- SQL Query Result Cache
- PL/SQL Function Result Cache
- Virtual Columns
- Composite Partitioning Enhancements
- Interval Partitioning
- Virtual Column-Based Partitioning
- Reference Partitioning
- System Partitioning
- Partitioning Advisor
- Locking Enhancements
- DDL_LOCK_TIME
- LOCK TABLE WAIT
- Read Only Tables
- It's The Little Things...
- Default Passwords
- Case Sensitive Passwords
- DBMS_COMPARISON
- East Recovery from Loss of SPFILE
- On-Demand Segment Creation
- E-Mail Notifications
- Minimal Effort Parallel Execution – ADOP
Oracle Database 11g: Focus Areas

- Manageability
- Availability
- Performance
- Business intelligence and data warehousing
- Security
Management Automation

- Auto-tuning
- Advisory
- Instrumentation

- Storage
- Backup
- Memory
- Apps/SQL
- Schema
- RAC
- Recovery
- Replication
It’s All About Caching
It’s All About Caching

• Accessing memory is far quicker than accessing hard drives
• This fact gives rise to caching: the process of storing data in memory instead of disks
• Caching is a common principle of Oracle database architecture, in which users are fed data from the buffer cache instead of the disks on where the database resides
• Oracle database 11g enhances performance by using:
  – SQL Query Result Cache
  – PL/SQL Function Result Cache
  – Client Query Result Cache
What is SQL Query Result Cache?

• Oracle Database 11g raises the bar on data caching
• In the past, the database always cached blocks of data, the building blocks used to build result sets
• Starting with Oracle Database 11g, the database can now also cache result sets!
• If you have a query that is executed over and over again against slowly or never-changing data, you will find the new server results cache to be of great interest
• This is a feature from which virtually every application can and will benefit
SQL Query Result Cache – Overview

- Cache the result of a query for future reuse
- A dedicated memory buffer stored in the shared pool is used for storing and retrieving the cached results
- The query results become invalid when data in the database objects being accessed by the query is modified
- Multiple users can see these results without repeating the same query
- Good candidate statements:
  - Access many rows
  - Return few rows

SQLQueryResultCache.sql
Setting Up SQL Query Result Cache

- Optimizer manages result cache mechanism depending on the `RESULT_CACHE_MODE` new initialization parameter:
  - `AUTO` – the optimizer determines which results are to be stored in the cache based on repetitive executions
  - `MANUAL` (default) – you must specify, using the `RESULT_CACHE` hint, that a particular result is to be stored in the cache
  - `FORCE` – all results are stored in the cache

- You can set the `RESULT_CACHE_MODE` initialization parameter at the system and session levels

- The `NO_RESULT_CACHE` hint takes precedence over the parameter setting
Managing the SQL Query Result Cache

- By default, the database allocates memory for the result cache in the Shared Pool.
- Memory size allocated depends on the memory size of the SGA as well as the memory management system.
- You can change the memory allocated to the result cache by setting the `RESULT_CACHE_MAX_SIZE` parameter.
- The result cache is disabled if you set its value to 0.
- The `RESULT_CACHE_MAX_RESULT` parameter specifies the maximum amount of cache memory that can be used by any single result (default value is 5%).
- Changes can be implemented at system and session level.
SQL Query Result Cache
Initialization Parameters

• **RESULT_CACHE_MAX_SIZE**
  – Sets the memory allocated to the result cache
  – Result cache is disabled if you set the value to 0
  – Default depends on other memory settings (0.25% of MEMORY_TARGET or 0.5% of SGA_TARGET or 1% of SHARED_POOL_SIZE)
  – Cannot be greater than 75% of shared pool

• **RESULT_CACHE_MAX_RESULT**
  – Sets maximum cache memory for a single result
  – Defaults to 5%
Using the DBMS_RESULT_CACHE Package

• Provides statistics, information, and operators that enable you to manage memory allocation for the query result cache
• Perform various operations such as viewing the status of the cache, retrieving statistics on the cache memory usage, and flushing the cache
Using the DBMS_RESULT_CACHE Package

• View the status of the cache:

```sql
SELECT DBMS_RESULT_CACHE.STATUS FROM DUAL;
```

• Retrieve statistics on the cache memory usage:

```sql
EXECUTE DBMS_RESULT_CACHE.MEMORY_REPORT;
```

• Remove all existing results and clear cache memory:

```sql
EXECUTE DBMS_RESULT_CACHE.FLUSH;
```

• Invalidate cached results depending on specified object:

```sql
EXEC DBMS_RESULT_CACHE.INVALIDATE('JFV', 'MYTAB');
```
SQL Result Cache Dictionary Information

- The following views provide information about the query result cache:

<table>
<thead>
<tr>
<th>View Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(G)V$RESULT_CACHE_STATISTICS</td>
<td>Lists the various cache settings and memory usage statistics</td>
</tr>
<tr>
<td>(G)V$RESULT_CACHE_MEMORY</td>
<td>Lists all the memory blocks and the corresponding statistics</td>
</tr>
<tr>
<td>(G)V$RESULT_CACHE_OBJECTS</td>
<td>Lists all the objects (cached results and dependencies) along with their attributes</td>
</tr>
<tr>
<td>(G)V$RESULT_CACHE_DEPENDENCY</td>
<td>Lists the dependency details between the cached results and dependencies</td>
</tr>
</tbody>
</table>
What is PL/SQL Function Result Cache?

• In the past, if you called a PL/SQL function 1,000 times and each function call consumed 1 second, the 1,000 calls would take 1,000 seconds

• With this new function results cache feature, depending on the inputs to the function and whether the data underlying the function changes, 1,000 function calls could take about 1 second, total
PL/SQL Function Result Cache

- Allows storing the results of PL/SQL functions in the SGA
- Caching mechanism is both efficient and easy to use
- Relieves you of the burden of designing and developing your own caches and cache management policies
- Provides the ability to mark a PL/SQL function to indicate that its result should be cached to allow lookup, rather than recalculation, when the same parameter values are called
- Saves significant space and time
- Done transparently using the input parameters as the lookup key
- Instancewide – all distinct sessions invoking the function benefit
Using PL/SQL Function Result Cache

• Include the `RESULT_CACHE` option in the function declaration section of a package or function definition.
• Optionally include the `RELIERS_ON` clause to specify any tables or views on which the function results depend.

```sql
CREATE OR REPLACE FUNCTION ProductName (prod_id NUMBER, lang_id VARCHAR2) RETURN NVARCHAR2 RESULT_CACHE RELIES_ON (product_descriptions) IS
   result VARCHAR2(50);
BEGIN
   SELECT translated_name INTO result FROM product_descriptions WHERE product_id = prod_id AND language_id = lang_id;
   RETURN result;
END;
```

PLSQLFunctionResultCache.sql
Virtual Columns
Virtual Columns – Overview

• Consider the following table:

<table>
<thead>
<tr>
<th>SALES_ID</th>
<th>NUMBER</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUST_ID</td>
<td>NUMBER</td>
</tr>
<tr>
<td>SALES_AMT</td>
<td>NUMBER</td>
</tr>
</tbody>
</table>

• Some users want to add an extra column SALES_CATEGORY which identifies the type of the sale:

<table>
<thead>
<tr>
<th>If sale_amt is more than:</th>
<th>And sale_amt is less than or equal to:</th>
<th>Then sale_category is:</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1000</td>
<td>LOW</td>
</tr>
<tr>
<td>10001</td>
<td>1000000</td>
<td>MEDIUM</td>
</tr>
<tr>
<td>100001</td>
<td>10000000</td>
<td>HIGH</td>
</tr>
<tr>
<td>1000001</td>
<td>Unlimited</td>
<td>ULTRA</td>
</tr>
</tbody>
</table>
Virtual Columns – Overview

- **SALESCATEGORY** column is a crucial business requirement
- You could add a new column in the table and write a trigger to populate the column
- In Oracle Database 11g, you do not need to write a single line of code in any trigger
- All you have to do instead is add a *virtual column*
- Virtual columns offer the flexibility to add columns that convey business sense without adding any complexity or performance impact
Using Virtual Columns

• Even though this column is not stored, you can refer to it as any other column in the table
• You can even create indexes on it. The result will be a function-based index
• You can even partition on this column
• You can't, however, enter a value for this column

VirtualColumn.sql
Oracle11g Partitioning to Perfection
Oracle11g – Partitioning to Perfection

• Since Oracle database version 8, you can partition a table or index into multiple segments and then place them in different tablespaces
• The table is still addressed as a logical entity while the individual partitions are stored as separate segments, which allows for easier manipulation of data
• In Oracle database version 11g, partitioning enhancements enable infinite partitioning design possibilities and boost manageability
## Oracle Database Partitioning

<table>
<thead>
<tr>
<th></th>
<th>Core functionality</th>
<th>Performance</th>
<th>Manageability</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Oracle8</strong></td>
<td>Range partitioning</td>
<td>Static partition pruning</td>
<td>Basic maintenance operations: add, drop, exchange</td>
</tr>
<tr>
<td></td>
<td>Global range indexes</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Oracle8i</strong></td>
<td>Hash and composite range-hash partitioning</td>
<td>Partitionwise joins</td>
<td>Merge operation</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Dynamic pruning</td>
<td></td>
</tr>
<tr>
<td><strong>Oracle9i</strong></td>
<td>List partitioning</td>
<td></td>
<td>Global index maintenance</td>
</tr>
<tr>
<td><strong>Oracle9i R2</strong></td>
<td>Composite range-list partitioning</td>
<td>Fast partition split</td>
<td></td>
</tr>
<tr>
<td><strong>Oracle10g</strong></td>
<td>Global hash indexes</td>
<td></td>
<td>Local Index maintenance</td>
</tr>
<tr>
<td><strong>Oracle10g R2</strong></td>
<td>1M partitions per table</td>
<td>Multidimensional pruning</td>
<td>Fast drop table</td>
</tr>
<tr>
<td><strong>Oracle Database 11g</strong></td>
<td>More composite choices</td>
<td></td>
<td>Interval Partitioning Partition Advisor</td>
</tr>
</tbody>
</table>
Oracle11g Partitioning Enhancements

- Composite partitioning enhancements
- Interval partitioning
- Virtual column-based partitioning
- REF partitioning
- System partitioning
Extended Composite Partitioning

• Prior to Oracle11g you are limited to range-hash and range-list composite partitioning
• In Oracle11g your choices are virtually limitless; you can create composite partitions in any combination:
  – Range-Range
  – Range-Hash
  – Range-List
  – List-Range
  – List-Hash
  – List-List
  – Hash-Hash (11g R2)
## Composite Partitioning Techniques

<table>
<thead>
<tr>
<th>1st Level Partitioning</th>
<th>2nd Level Partitioning</th>
<th>Database Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Range</td>
<td>Hash</td>
<td>Oracle8i</td>
</tr>
<tr>
<td></td>
<td>List</td>
<td>Oracle9i</td>
</tr>
<tr>
<td></td>
<td>Range</td>
<td>Oracle11g R1</td>
</tr>
<tr>
<td>List</td>
<td>Range</td>
<td>Oracle11g R1</td>
</tr>
<tr>
<td></td>
<td>List</td>
<td>Oracle11g R1</td>
</tr>
<tr>
<td></td>
<td>Hash</td>
<td>Oracle11g R1</td>
</tr>
<tr>
<td>Interval</td>
<td>Range</td>
<td>Oracle11g R1</td>
</tr>
<tr>
<td></td>
<td>List</td>
<td>Oracle11g R1</td>
</tr>
<tr>
<td></td>
<td>Hash</td>
<td>Oracle11g R1</td>
</tr>
<tr>
<td>Hash</td>
<td>Hash</td>
<td>Oracle11g R2</td>
</tr>
</tbody>
</table>
Oracle11g Partitioning New Features

• Interval Partitioning
  – Automates the creation of range partitions
  – Extends the capabilities of the range method to define equal-partitioned ranges using an interval definition
  – Oracle will create any partition automatically as-needed whenever data for a partition is inserted for the very first time
  – Greatly improves the manageability of a ranged partitioned table
  – Available techniques are interval, interval-list, interval-hash and interval-range
  – You must specify at least one range partition
  – Partitioning key column must be of NUMBER or DATE type
  – Cannot be used for index-organized tables
  – Not supported at the sub-partition level
Oracle11g R1 Interval Partitioning

CREATE TABLE INTERVAL_PARTITIONING_DEMO
(serial_num NUMBER,
  name VARCHAR2(32))
PARTITION BY RANGE (serial_num) INTERVAL (10000)
( PARTITION p1 VALUES LESS THAN (10000),
  PARTITION p2 VALUES LESS THAN (20000),
  PARTITION p3 VALUES LESS THAN (30000));

Automatically created when inserting data

The diagram shows the partitioning structure:
- **Range section**: P1, P2, P3
- **Interval section**: Pi1
- **Transition point**

Interval.sql
Oracle11g Partitioning New Features

• **Virtual Column-Based Partitioning:**
  – Allows the partitioning key to be defined by an expression, using one or more existing columns of a table and storing the expression as metadata only
  – Enables a more comprehensive match of the business requirements
  – Supported with all basic partitioning strategies
  – Can also be used with interval partitioning as well as the partitioning key for REF partitioned tables
  – Virtual columns are treated as real columns except no DML operations are allowed
  – One of the most useful innovations in Oracle11g R1
Oracle11g R1 Virtual Column-Based Partitioning

CREATE TABLE SALES
(
    PROD_ID NUMBER NOT NULL,
    CUST_ID NUMBER NOT NULL,
    TIME_ID DATE NOT NULL,
    CHANNEL_ID NUMBER NOT NULL,
    PROMO_ID NUMBER NOT NULL,
    QUANTITY_SOLD NUMBER(10,2) NOT NULL,
    AMOUNT_SOLD NUMBER(10,2) NOT NULL,
    PROD_TYPE NUMBER(1) AS
    (TO_NUMBER(SUBSTR(TO_CHAR(PROD_ID),1,1))))
TABLESPACE USERS
PARTITION BY RANGE (PROD_TYPE) INTERVAL (1)
(PARTITION p1 VALUES LESS THAN (1));
REF Partitioning:

- Allows to partition a table by leveraging an existing parent-child relationship
- The partitioning strategy of the parent table is inherited to its child table without the necessity to store the parent’s partitioning key column in the child table
- Transparently inherits all partition maintenance operations that change the logical shape of a table from the parent table to the child table (for example when you drop/add/split partitions)
- Automatically enables partition-wise joins for the equal-partitions of the parent and child table
Oracle11g R1 REF Partitioning

Without using reference partitioning:
- Table ORDERS
- Table ORDER_ITEMS
- Redundant storage/maintenance of ORDER_DATE

Reference partitioning:
- Range(ORDER_DATE)
- Primary key (ORDER_ID)
- Range(ORDER_DATE)
- Foreign key (ORDER_ID)
- Partition key inherited through PK/FK relationship
Oracle11g Partitioning New Features

- **System Partitioning:**
  - Enables application-controlled partitioning
  - Allows the application to explicitly map rows to arbitrary partitions
  - Provides the ability to break down a table into meaningless partitions
  - All aspects of partitioning are controlled by the application
  - Usual performance benefits of partitioned tables are not available (do not have a partitioning key)
  - No support for traditional partition pruning, partition wise joins, and so on
SQL Access Advisor – Overview

• Part of the manageability improvements in Oracle10g and Oracle11g
• Advices on defining appropriate access structures to optimize SQL queries
• Identifies and helps resolve performance problems that relate to the execution of SQL statements
• Recommends which indexes, materialized views or partitions to create, drop, or retain
• Can be run using Database Control or PL/SQL procedures
## SQL Access Advisor – Recommendations

<table>
<thead>
<tr>
<th>Recommendation</th>
<th>Comprehensive</th>
<th>Limited</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add new <em>(partitioned)</em> index on table or materialized view</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Drop an unused index</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Modify an existing index by changing the index type</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Modify an existing index by adding columns at the end</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Add a new <em>(partitioned)</em> materialized view</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Drop an unused materialized view (log)</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Add a new materialized view log</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Modify an existing materialized view log to add new columns or clauses</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Partition an existing unpartitioned table or index</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Partition Advisor

• Beginning with Oracle Database 11g Release 2, the SQL Access Advisor has been enhanced to generate partitioning recommendations
• Recommendations will show the anticipated performance gains that will result if they are implemented
• Generated scripts can either be implemented manually or submitted onto a queue within Oracle Enterprise Manager
• Part of Oracle’s Tuning Pack, an extra licensable option
• Can be used from within Enterprise Manager or via a command line interface
Locking Enhancements
Locking Enhancements

• Limit the time that DDL commands wait for DML locks before failing by setting `DDL_LOCK_TIMEOUT` parameter at the system or session level.
• Default is 0, that is `NOWAIT`, which ensures backward compatibility.
• Range of values is 0–1,000,000 (in seconds).
Locking Enhancements

- The `LOCK TABLE` command has new syntax to specify the maximum number of seconds the statement should wait to obtain a DML lock on the table.
- Use the `WAIT` clause to indicate that the `LOCK TABLE` statement should wait up to the specified number of seconds to acquire a DML lock.
- There is no limit on the value of the integer.
Locking Enhancements

• The following commands will no longer acquire exclusive locks (X), but shared exclusive locks (SX):
  – CREATE INDEX ONLINE
  – CREATE MATERIALIZED VIEW LOG
  – ALTER TABLE ENABLE CONSTRAINT NOVALIDATE

• In highly concurrent environments, the requirement of acquiring an exclusive lock could lead to a spike of waiting DML operations and a short drop and spike of system usage

• The commands listed above no longer require exclusive locks
Read Only Tables
Read Only Tables

• The DBA in a data warehouse system has a classic problem
  – As a part of the ETL process, several tables are updated with different periodicities
  – When updated, the tables are opened up to the users per business rules, even though the users shouldn't modify them
  – Revoking DML privilege from the users on these tables is not an option

• We need a functionality that acts as a switch, to make a table update-able and then not so

• The implementation of this trivial-sounding operation is actually quite difficult:
  – Create a trigger on the table that raises an exception
  – Create a Virtual Private Database (VPD) policy that always returns a false string
Read Only Tables

• In Oracle Database 11g, all you need to do is:
  – ALTER TABLE table_name READ ONLY;

• Now when a user tries to issue a DML statement the database throws an error right away (ORA-12081)

• When you want to make the table read/write:
  – ALTER TABLE table_name READ WRITE;

• While a table is in read-only mode only DMLs are disallowed; you can still perform all DDL operations (such as create indexes, maintain partitions, and so on)

• To see the status of the table, look for the READ_ONLY column in the data dictionary view DBA_TABLES
Miscellaneous New Features
“It’s the little things in life that really make a difference”

Here's a little thing that will make your coding life a bit easier and the PL/SQL language a bit more complete:

```sql
CREATE SEQUENCE this_is_nice;
DECLARE
    n NUMBER;
BEGIN
    n := this_is_nice.NEXTVAL;
END;
/
```
Who’s Using the Default Password?

- Oracle Database 11g now offers a way to quickly identify users with default passwords, implemented in the rather ludicrously simple way of checking a single data dictionary view: `DBA_USERS_WITH_DEFPWD`.

- You can simply identify these users by issuing:

  ```sql
  SELECT * FROM dba_users_with_defpwd;
  ```
Oracle11g Passwords are Case Sensitive

- Starting with Oracle11g, database passwords have become case sensitive
  
  `SQL> ALTER USER SCOTT IDENTIFIED BY TIGER;`
  `User altered.`
  `SQL> CONN SCOTT/tiger`
  `ERROR: ORA-01017:invalid username/password; logon denied`
  `SQL> CONN SCOTT/TIGER`
  `Connected.`

- Behavior can be changed using the system parameter:
  `SEC_CASE_SENSITIVE_LOGON=FALSE`
DBMS_COMPARISON

- New in Oracle11g, a PL/SQL supplied package which can be used as a fast and easy way to compare between tables, views and materialized views
- DBMS_COMPARISON.COMPARISON – performs a comparison
- DBMS_COMPARISON.CONVERGE – execute compensating DML to get the two objects to converge
Easy Recovery from Loss of SPFILE

- In Oracle Database 11g, the `FROM MEMORY` clause creates a `pfile` or `spfile` using the current system-wide parameter settings.
- During startup, all parameters are logged to the `alert.log` file.
- The `alert.log` parameter dump text is written in valid parameter syntax which facilitates cutting and pasting of parameters into a separate file, then used as a `pfile`.
- The `pfile` or `spfile` name is written to the `alert.log` at startup and in cases when an unknown client-side `pfile` is used, the alert log indicates this as well.
- To support this additional functionality, the `COMPATIBLE` initialization parameter must be set to 11.0.0.0 or higher.
Easy Recovery from Loss of SPFILE

- The **FROM MEMORY** clause allows the creation of current system-wide parameter settings

```sql
CREATE PFILE [= 'pfile_name' ]
  FROM { { SPFILE [= 'spfile_name'] } | MEMORY } ;
```

```sql
CREATE SPFILE [= 'spfile_name' ]
  FROM { { PFILE [= 'pfile_name' ] } | MEMORY } ;
```
Oracle11g R2 –
Minimal Effort Parallel Execution (ADOP)

• When activated, Oracle automatically determines the DOP for any given SQL operation based on the size of the objects, the complexity of a statement, and the existing hardware resources
• Database compensates for wrong or missing user settings for parallel execution
• Ensures a more optimal resource consumption and overall system behavior
Oracle11g R2 – On-Demand Segment Creation

• Segment creation on demand: initial segment creation for non-partitioned tables and indexes can be delayed until data is first inserted into an object so that empty database objects do not consume any space, reducing the installation footprint and speeding up the installation
Real-Life Example
using Oracle11g Interval Partitioning
with Hash Sub-Partitioning
Next-Generation Application Performance Management

“A New Technology Platform that Redefines End User Experience Management”

- The Aternity Frontline Performance Intelligence (FPI) Platform uniquely monitors, aggregates, analyzes, and correlates the three components that dynamically interact, define, and constantly impact end user IT experience – in real-time: application performance, real and virtual desktop performance, and end user productivity

- “Aternity provides in-depth visibility into how applications are performing from the perspective of the end-users on the business frontline, with the information and metrics needed to rapidly identify and address performance challenges”

(Moshe Horev, Vice President and Managing Director, Oracle)
• **Preemptive Problem Detection**
  – Automatically identifies impacted users
  – Pinpoints probable cause
  – Dramatically reduces business disruptions

• **360° View of End User Experience**
  – How users consume IT services
  – Correlation of real/virtual desktop performance and Applications usage
  – End-to-end response times
  – Optimization of IT infrastructure/software licensing costs

• **Right-Time Decision Support**
  – Analyzes, correlates, transforms data into actionable intelligence
  – Supports effective business/IT decisions
Aternity Historical Data Tables

• Real-time user activities across all desktop applications (office, ERP, CRM etc)

• Diverse data usage patterns:
  – Workflow of a user for the last 30 minutes
  – Application performance trends for the last 30 days
  – Drill-down to slowest performing users across the organization in the last week

• Optimal performance for all queries:
  – Short timeframes need granular data
  – Long timeframes need to go through huge amounts of data
Aternity Historical Data Tables

- Schema includes 3 main tables:
  - HISTORICAL_DATA_1 – 1st aggregation level, daily partitions
  - HISTORICAL_DATA_2 – 2nd aggregation level, weekly partitions
  - HISTORICAL_DATA_3 – 3rd aggregation level, monthly partitions
- Queries based on TIMEFRAME (date and time) and MA_ID (basic unit of monitoring; response time, latency etc)
- Range partitioned on TIMEFRAME
- Hash sub-partitioned on MA_ID
- Number of hash sub-partitions depends on customer implementation
Example Script – HISTORICAL_DATA_1 Table

```sql
CREATE TABLE HISTORICAL_DATA_1
(
  TIMEFRAME DATE,
  MA_ID NUMBER(10),
  EP_COMBO_ID NUMBER(10),
  CTX_COMBO_ID NUMBER(10),
  ORIGINATING_PATH_INDEX NUMBER(2),
  VARIANCE NUMBER(37,10),
  AVERAGE NUMBER(37,10),
  VALUE_COUNT NUMBER(10,0)
)
TABLESPACE %TABLESPACE_HD%
PARTITION BY RANGE (TIMEFRAME)
SUBPARTITION BY HASH (MA_ID) SUBPARTITIONS %SUBPARTITIONS_HIST%
(PARTITION HD_AL1_30012000 VALUES LESS THAN
(TO_DATE('01-02-2000','DD-MM-YYYY')));
```

HD.sql
Historical Data Procedure

- PL/SQL procedure designed to handle historical data tables
- Designed to work with all three aggregation levels
- Used for creating, managing and dropping table and index partitions based on customer implementation
- Creating new partitions is a must otherwise application fails with ORA-14400: inserted partition key does not map to any partition
- Called from within the Java application code
Historical Data Tables in Oracle11g

CREATE TABLE HISTORICAL_DATA_1.....
PARTITION BY RANGE (TIMEFRAME)
INTERVAL (NUMTODSINTERVAL (1, 'DAY'))
SUBPARTITION BY HASH (MA_ID) SUBPARTITIONS %SUBPARTITIONS_HIST%

CREATE TABLE HISTORICAL_DATA_2.....
PARTITION BY RANGE (TIMEFRAME)
INTERVAL (NUMTODSINTERVAL (7, 'DAY'))
SUBPARTITION BY HASH (MA_ID) SUBPARTITIONS %SUBPARTITIONS_HIST%

CREATE TABLE HISTORICAL_DATA_3.....
PARTITION BY RANGE (TIMEFRAME)
INTERVAL (NUMTOYMINTERVAL (1, 'MONTH'))
SUBPARTITION BY HASH (MA_ID) SUBPARTITIONS %SUBPARTITIONS_HIST%
What About Statistics?

• PL/SQL package designed to handle all schema statistics
• Three main procedures:
  – Gathering statistics on all “regular” tables and indexes
  – Gathering statistics on transient tables using a high-water mark
  – Gathering statistics on historical data tables and indexes partitions
• All statistics are gathered using the DBMS_STATS Oracle supplied package
• Procedures are executed automatically using jobs defined with DBMS_SCHEDULER Oracle supplied package
No Need to Gather – Just Copy

- `DBMS_STATS.COPY_TABLE_STATS` – used to copy statistics between different partitions of the same historical data table (available since Oracle10g 10.2.0.4)
- `DBMS_STATS.GET_INDEX_STATS` and `SET_INDEX_STATS` – used to copy statistics between different index partitions of the same historical data table index
- Statistics are copied between tables and indexes partitions of the same aggregation level (between days, weeks and months)
Thank You!

Ami Aharonovich
Oracle Certified Professional
Independent Oracle DBA Consultant & Instructor
aaharon@gmail.com