SQL Analytics for Analysis, Reporting and Modeling

ORACLE
CLOUD PLATFORM

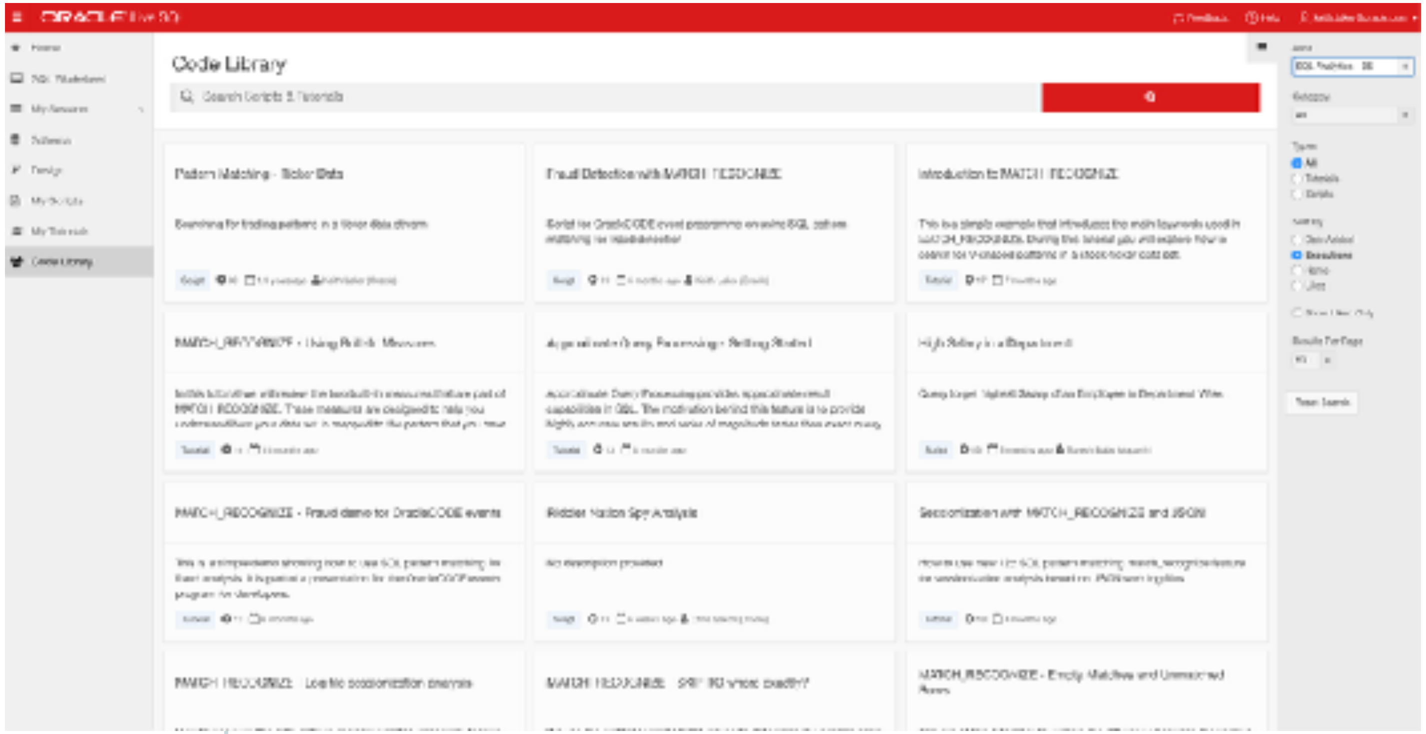
Key SQL Functionality for ANALYTICS in the cloud and on-premise with Oracle Database:

18c
12c Release 2

ORACLE

LiveSQL – The Easiest Way to Explore, Learn and Try SQL

- **Free Service – livesql.oracle.com**
- Features include:
  - Access to very latest 18c features
  - Ability to save collections of statements as a script
  - Access to growing library of tutorials
  - Share saved scripts with others
  - Embedded educational tutorials
  - Data access examples for popular languages including Java
  - Comes complete with sample schemas
    - Human Resources schema.
    - Sales History schema
    - SCOTT schema
    - World Population data
    - DinoDate demo data
    - Olympic data



Safe Harbor Statement

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.

Oracle Database 18 Release 1 – New SQL Features

- **Overview of new SQL Features**
  - SAFE HARBOR STATEMENT

- **What's new in 18 Release 1** (SAFE HARBOR STATEMENT)
  - ROUND()
  - Polymorphic Table Functions
  - Approximate Query Processing
    - Approx. Top/Bottom-N
  - Analytic Views
    - MDX Support
  - Private temporary tables
  - Inline External Tables
  - Column-Based Collation

NEW IN 18c

Oracle Database 12c Release 2 – New SQL Features

- **Overview of new SQL Features**

What's new in 12c Release 2

- LISTAGG
  - Support for larger VARCHAR2 objects
- CAST/VALIDATE
- Approximate statistics
  - APPROX_PERCENTILE
  - APPROX_MEDIAN
- Approximate aggregations
  - APPROX_xxxx_DETAIL, APPROX_xxxx_AGG
  - TO_APPROX_xxxx

- External tables
  - External table - MODIFY clause
  - Partitioned external table
  - Accessing data in Hive, HDFS etc
- Analytic Views

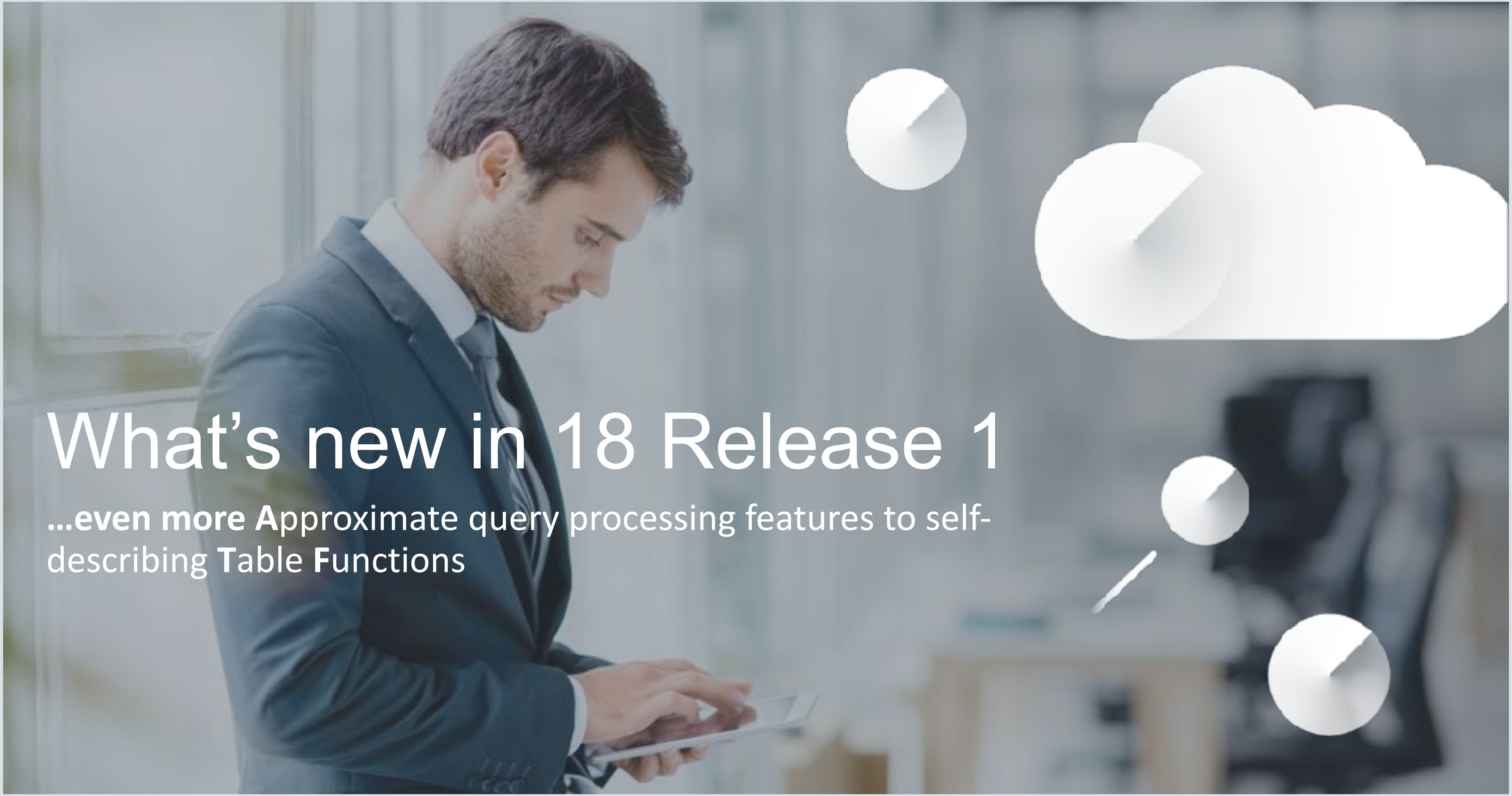
Oracle Database 12c Release 2 - Core SQL Features

- **Overview of core SQL Features**
- **Schema modeling enhancements**
  - Invisible columns
  - Default value enhancements
  - Identity columns
- **Storage optimizations**
  - Attribute Clustering
  - Zone Maps
  - Zone Maps and attribute clustering
  - Zone Maps and partitioning
  - Zone Maps and storage indexes
- **SQL for advanced analysis**
  - TOP-N
  - MATCH_RECOGNIZE
  - APPROX_COUNT_DISTINCT
- **Query rewrites**
  - Materialized views
  - In-place/out-of-place refresh
  - Synchronous refresh
- **Multilingual support**
  - Data bound collations

Analytic SQL @ #oow17

Key sessions and Labs

- Link to **Complete Data Warehouse and Big Data Guide to #oow17**
- Link to **#oow17 web app** for data warehousing and big data
- List of SQL sessions
- List of data warehouse sessions
- List of hands-on labs



What’s new in 18 Release 1

…even more Approximate query processing features to self-describing **T**able **F**unctions

New ROUND_TIES_TO_EVEN() Function in 18.1

- This enhancement will provide new rounding function

```
ROUND_TIES_TO_EVEN(n [, integer])
```

- ROUND_TIES_TO_EVEN and ROUND have the same behavior except when the rounding digit is at the mid point.
  - ROUND_TIES_TO_EVEN will return the nearest value with an even (zero) least significant digit.
  - ROUND will return nearest value above (for positive numbers) or below (for negative numbers).
- Will not support BINARY_FLOAT and BINARY_DOUBLE

NEW IN 18c

Comparing ROUND() and ROUND_TIES_TO_EVEN()

Value	ROUND (Value, 0)	ROUND_TIES_TO_EVEN (Value, 0)
1.6	2	2
-1.6	-2	-2
0.5	1	0
-0.5	-1	0
2.5	3	2
-2.5	-3	-2

NEW IN 18c

Top-N approximate aggregation

- Approximate results for common top n queries
  - How many approximate page views did the top five blog posts get last week?
  - What were the top 50 customers in each region and their approximate spending?
- Orders of magnitude faster processing with high accuracy (error rate < 0.5%)
- New approximate functions APPROX_COUNT(), APPROX_SUM(), APPROX_RANK()

Top 5 blogs with approximate hits

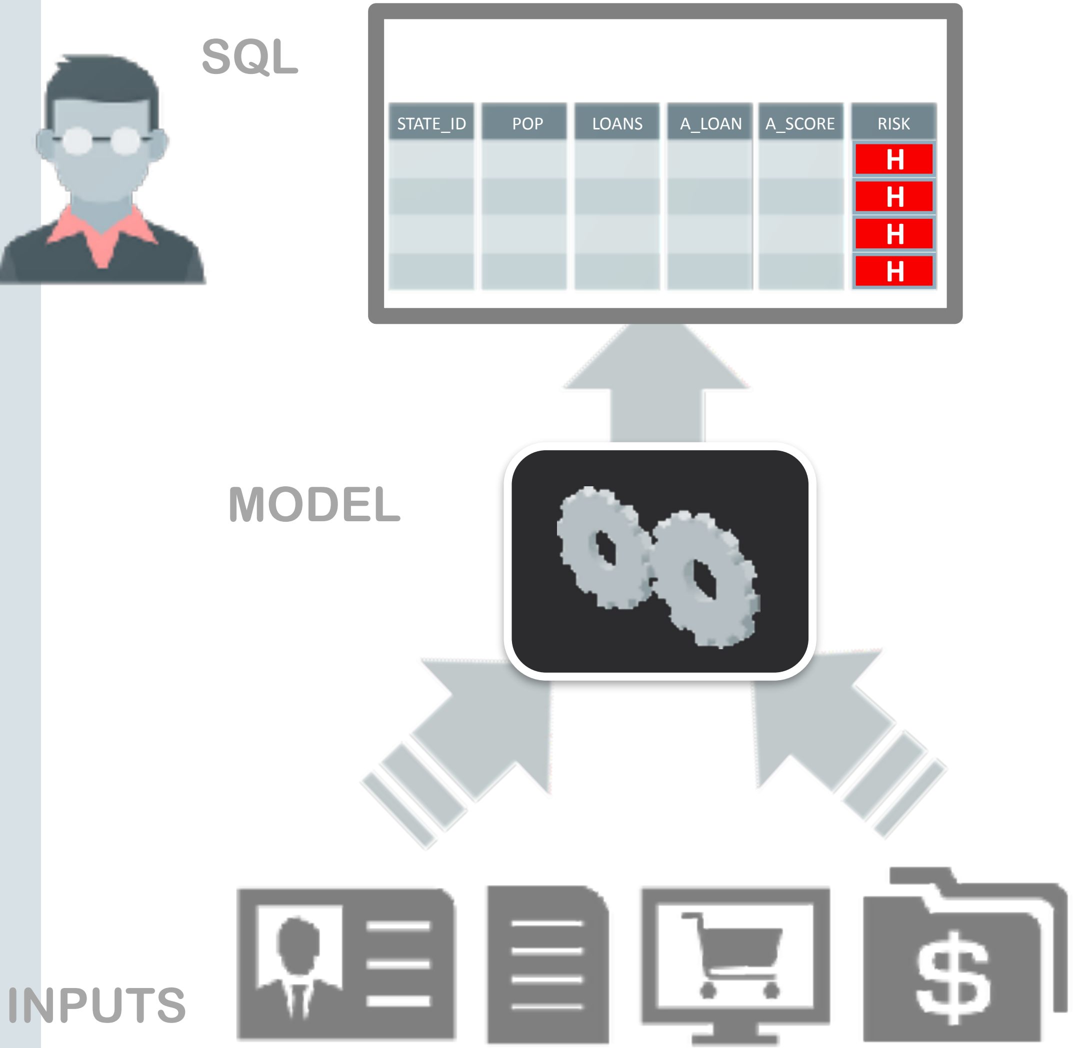
```
SELECT blog_post, APPROX_COUNT(*)
FROM   weblog
GROUP BY blog_post
HAVING
  APPROX_RANK(order by
      APPROX_COUNT(*) DESC) <= 5;
```

Top 50 customers per region with approximate spending

```
SELECT region, customer_name,
       APPROX_RANK(PARTITION BY region
                   ORDER BY APPROX_SUM(sales) DESC) appr_rank,
       APPROX_SUM(sales) appr_sales
FROM sales_transactions
GROUP BY region, customer_name
HAVING APPROX_RANK(...) <=50;
```

NEW IN 18c

Polymorphic Tables: Self-Describing, Fully Dynamic SQL



- Part of ANSI 2016
- Embed **sophisticated algorithms** in SQL
  - Hides implementation of algorithm
  - Leverage powerful, dynamic capabilities of SQL
  - Pass in any table-columns for processing
  - Returns rowset (table, JSON, XML doc, etc.)
    - Applies built-in algorithms and/or custom algorithms
    - Returns an enhanced set of rows-columns as output (table)
    - E.g. return credit score and associated risk level

NEW IN 18c

PTFs: Use Cases For Fully Dynamic SQL

```
SELECT
 state,
 AVG(credit_score)
FROM CREDIT_RISK(
  tab  => table(CUSTOMERS),
  cols => columns(DOB, ZIP, LoanDefault),
  outs => columns(credit_score, risk_level))
WHERE risk_level = 'High'
GROUP BY STATE;
```

- Embed **credit risk evaluation model**
  - Hides implementation of credit risk model
  - Pass in key columns to evaluate credit risk
  - PTF returns credit score and associated risk level

```
SELECT *
FROM HDFS_READER(
 host_port => 'http://<host>:<port>',
 path      => 'customer_reviews_2013.json',
 outs      => columns("cust_id" varchar(20),
                      "prod.id"  integer,
                      "prod.desc" varchar(500)
 ));
```

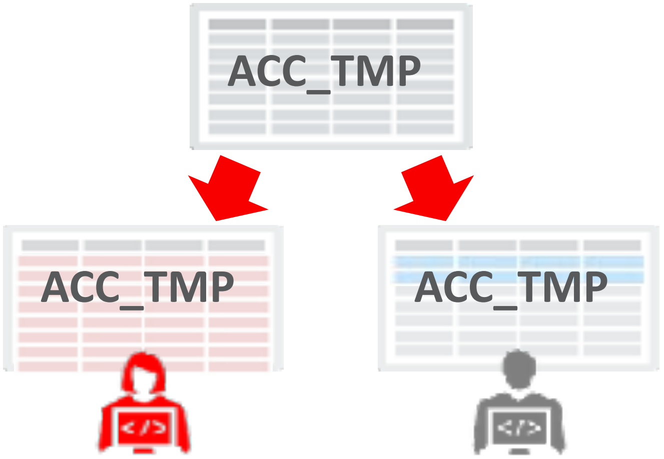
- Simplify access to **external data sets**
  - Pass in any server connection details and any source file
  - Returns row-column based formatted results

Enhancements to Analytic Views

- More calculations within Analytic Views:
  - Ranking and statistical functions
  - Hierarchical expressions
- Broader schema support for Analytic Views:
  - Snowflake schemas; flat/denormalized fact tables (in addition to star schemas)
- Dynamic definition of calculations within SQL queries
- Support for MDX

NEW IN 18c

Private Temporary Tables



Global temporary tables

- Persistent, shared (global) table definition
- Temporary, private (session-based) data content
  - Data physically exists for a transaction or session
  - Session-private statistics

Private temporary tables (18.1)

- Temporary, private (session-based) table definition
  - Private table name and shape
- Temporary, private (session-based) data content
  - Session or transaction duration

Inline External Tables

- External table definition provided at runtime
  - Similar to inline view
- No need to pre-create external tables that are used one time only
  - Increased developer productivity

NEW IN 18c

```
CREATE TABLE sales_xt
   (prod_id number, … )
    TYPE ORACLE_LOADER
    …
    LOCATION 'new_sales_kw13')
    REJECT LIMIT UNLIMITED );

INSERT INTO sales SELECT * FROM sales_xt;

DROP TABLE sales_xt;
```

```
INSERT INTO sales
SELECT sales_xt.*
FROM EXTERNAL(
        (prod_id number, … )
         TYPE ORACLE_LOADER
         …
         LOCATION 'new_sales_kw13')
         REJECT LIMIT UNLIMITED );
```

Column-Based Collation

- Precise and consistent application of linguistic comparison in queries
  - Adds COLLATE clause to declare column's collation to be used in all queries
  - COLLATE operator precisely controls collation in expressions
- Case- and accent-sensitive collations (e.g. BINARY_CI) simplify implementation of case-insensitive queries
- Feature is based on ISO/IEC SQL Standard and simplifies application migration from other databases supporting the COLLATE clause

```
CREATE TABLE products
( product_code        VARCHAR2(20 BYTE)   COLLATE BINARY
, product_name        VARCHAR2(100 BYTE)  COLLATE GENERIC_M_CI
```

NEW IN 18c

What’s new in 12c Release 2

From Approximate **query processing to new V**ALIDATE Functionality to new dimensional modeling with analytic views

Pre-12.2 LISTAGG

- Pre 12.2 syntax to manage lists was relatively simple:

```
LISTAGG(c.cust_first_name||' '||c.cust_last_name, ',’)
               WITHIN GROUP (ORDER BY c.country_id) AS Customer
```

- **Issue**….key issue is overflow error:
  - `ORA-01489: result of string concatenation is too long`
- **Solutions in 12.2**
  - Increasing the VARCHAR2 size - support VARCHAR2 up to 32k
  - Handle overflow errors - New syntax support to truncate string, optionally display count of truncated items count, and set truncation indication

New Keywords For Use With LISTAGG

- With 12.2 we have made it easier to manage lists:

```
LISTAGG(<measure_column>[, <delimiter>]. . .
```

- `ON OVERFLOW ERROR` *(default)*
- `ON OVERFLOW TRUNCATE`
- `ON OVERFLOW TRUNCATE “. . .”`
- `WITH COUNT`
- `WITHOUT COUNT` *(default)*

Detecting Data Conversion Errors - **VALIDATE_CONVERSION**

Identifying invalid data in the input streams

- Useful to detect if input value can be converted to destination type. Returns 1 if conversion is successful, otherwise returns 0

- **VALIDATE_CONVERSION** ('123a' as NUMBER) --> returns **0**
- **VALIDATE_CONVERSION** ('123' as NUMBER) --> returns **1**

- Can be efficiently used as filter to avoid bad data while importing foreign data sources, ETL processing

Handling data conversion errors - TO_xxxx(), CAST()

-Replacing incorrect or missing data with default values

- Pre 12.2: TO_NUMBER('123a') --> returns invalid number error (ora-01722)

New 12.2 Features

- New syntax **DEFAULT <default_value> ON CONVERSION ERROR**
  - Replace conversion failure with user defined default value
  - TO_NUMBER('123a' DEFAULT '123' ON CONVERSION ERROR) --> returns 123

- This new syntax can be used for TO_NUMBER, TO_DATE, TO_TIMESTAMP, TO_TIMESTAMP_TZ, TO_DMINTERVAL, TO_YMINTERVAL and CAST

Review: Analytic Views in 12.2

Enhanced Analysis and Simplified Access

- Organizes data into a user and application friendly business model
  - Intuitive for the end user
- Defined with SQL DDL
  - Includes hierarchical expressions and calculated measures
  - Easy to define, supported by SQL Developer
- Easily queried with simple SQL SELECT
  - Smart Analytic View (containing hierarchies and calculations) = Simple Query

Review: Analytic Views in 12.2

Embedded Calculations

- Define centrally in the Database and access with any application
  - Single version of the truth
- Easily create new measures
  - Simplified syntax based on business model
  - Includes dimensional and hierarchical functions

Sales Year to Date

```
sales_ytd AS
 (SUM(sales)
  OVER(HIERARCHY time_hierarchy
  BETWEEN UNBOUNDED PROCEEDING
  AND 0 FOLLOWING
  WITHIN ANCESTOR AT LEVEL year)
```

Product Share of Parent

```
share_product_parent_sales AS
 (SHARE_OF (sales
  HIERARCHY product_hierachy PARENT))
```

Approximate Statistics

- Issue: `PERCENTILE_CONT, PERCENTILE_DISC, MEDIAN` functions require sorting and can consume large amounts of resources
- Solution: New approximate SQL functions use fewer resources:
  `APPROX_PERCENTILE`
  `APPROX_MEDIAN`
  - Use less memory, no sorting, no use of temp

How to get more information about result set

Additional keywords

- Each function can use different algorithms and report error rates and confidence levels:

1. `DETERMINISTIC/NONDETERMINISTIC [default]`
   - Non-deterministic is faster but results may vary, good for personal data discoveries
   - Deterministic, slightly slower; better where results are shared with other users
2. `ERROR_RATE`
   - Returns the margin of error associated with result
3. `CONFIDENCE`
   - Returned as a percentage that indicates the level of confidence

New Functions For Building *Approximate* Aggregates

1. **APPROX_xxxxxx_DETAIL**(expr [DETERMINISTIC])
   - builds summary table containing results for all dimensions in **GROUP BY** clause
   - Data stored within MV as a BLOB object
2. **APPROX_xxxxxx_AGG** (expr)
   - Builds higher level summary table based on results from table derived from **_DETAIL** function
   - Does not re-query base fact table, derives new aggregates from **_DETAIL** table
   - Data stored within MV as a BLOB object
3. **TO_APPROX_xxxxxx(detail, percentage, order)**
   - Returns results from the specified aggregated results table

```
select ... to_approx_percentile(approx_percentile_agg(detail),0.5)
```

External Tables

- Key issues:
  - Definition of external table is fixed at creation time
  - Need ability to define table once and use it multiple times, to access different external files
  - Apply same table definition to different inputs
- Solution:
  - Added EXTERNAL MODIFY clause
  - Ease of use enhancement for using external tables
  - Clause allows external table to be overridden at query time
  - Properties: DEFAULT_DIRECTORY, certain ACCESS PARAMETERS, LOCATION and REJECT LIMIT

Core SQL in 12c Release 2

From storage optimizations to SQL pattern matching to data bound collations to support multi-lingual systems

Overview of Schema Modeling Enhancements

- Invisible Columns
- DEFAULT VALUE enhancements
  - Metadata-Only Default column values for NULL'able columns
  - Default values for columns on explicit NULL insertion
  - Default values for columns based on sequences
- Multiple Indexes on the same columns
- IDENTITY columns

Attribute Clustering

Concepts and Benefits

- Orders data so that it is in close proximity based on selected columns values: “attributes”
- Attributes can be from a single table or multiple tables
  - e.g. from fact and dimension tables
- Significant IO pruning when used with zone maps
- Reduced block IO for table lookups in index range scans
- Queries that sort and aggregate can benefit from pre-ordered data
- Enable improved compression ratios
  - Ordered data is likely to compress more than unordered data

Basics of Zone Maps

- Independent access structure built for a table
  – Implemented using a type of materialized view
  – For partitioned and non-partitioned tables
- One zone map per table
  – Zone map on partitioned table includes aggregate entry per [sub]partition
- Used transparently
  – No need to change or hint queries
- Implicit or explicit creation and column selection
  – Through Attribute Clustering: CREATE TABLE … CLUSTERING
  – CREATE MATERIALIZED ZONEMAP … AS SELECT …

Pattern Recognition In Sequences of Rows

SQL Pattern Matching - Concepts

- Recognize patterns in sequences of events using SQL
  - Sequence is a stream of rows
  - Event equals a row in a stream
  - New SQL construct MATCH_RECOGNIZE
  - Logically partition and order the data
  - ORDER BY and PARTITION BY are optional – but be careful
- Pattern defined using regular expression using variables
  - Regular expression is matched against a sequence of rows
  - Each pattern variable is defined using conditions on rows and aggregate

Distinct Counts to support "How Many Unique…"

Businesses need to answers lots of different "***How many***…" type questions

- How many unique sessions today
- How many unique customers logged on
- How many unique events occurred

Most queries don't need precise answers, approximate answer good enough

- Approximate answers can be returned significantly faster
- Approximate answers consume fewer resources, leaving resources for other queries

Overview of Materialized Views in Oracle Database 12c

- Objectives
  - Improve performance of refresh operation
  - Minimize staleness time of materialized views
- Two fundamental new concepts for refresh
  - Out-of-place refresh
    - Refresh “shadow MV” and swap with original MV after refresh
  - Synchronous refresh
    - Refresh base tables and MVs synchronously, leveraging equi-partitioning of the objects

Enhancements to External Tables

- Issues:
  - Definition of external table is fixed at creation time
  - Need ability to define table once and use it multiple times, to access different external files
  - Need better integration with big data source files

- Solutions:
  - Added EXTERNAL MODIFY clause to allow overriding properties
  - Partitioned external tables for source files stored on file system, Apache Hive storage, or HDFS

Data-Bound Collations

"… a named set of rules describing how to compare and match character strings to put them in a specified order…"

- Based on the ISO/IEC/ANSI SQL standard 9075:1999
- Character set is always declared at the database level
- Collation declared for a **column**
  - Does not determine the character set of data in the column

- Why is it important?
  - it simplifies application migration to the Oracle Database from a number of non-Oracle databases implementing collation in a similar way

ROUND() Function

New financial rounding features

New ROUND_TIES_TO_EVEN() Function in 18.1

NEW IN 18c

- Formal definition for ROUND_TIES_TO_EVEN functionality

***RoundTiesToEven**: the floating-point number nearest to the infinitely precise result shall be delivered; if the two nearest floating-point numbers bracketing an unrepresentable infinitely precise result are equally near, the one with an even least significant digit shall be delivered*

New ROUND_TIES_TO_EVEN() Function in 18.1

- This enhancement will provide new rounding function

```
ROUND_TIES_TO_EVEN(n [, integer])
```

- ROUND_TIES_TO_EVEN and ROUND have the same behavior except when the rounding digit is at the mid point.
  - ROUND_TIES_TO_EVEN will return the nearest value with an even (zero) least significant digit.
  - ROUND will return nearest value above (for positive numbers) or below (for negative numbers).
- Will not support BINARY_FLOAT and BINARY_DOUBLE

NEW IN 18c

Comparing ROUND() and ROUND_TIES_TO_EVEN()

Value	ROUND (Value, 0)	ROUND_TIES_TO_EVEN (Value, 0)
1.6	2	2
-1.6	-2	-2
0.5	1	0
-0.5	-1	0
2.5	3	2
-2.5	-3	-2

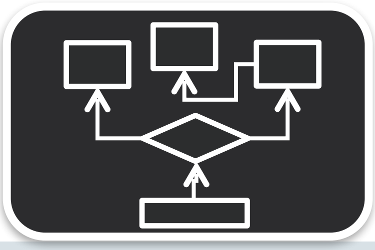
Polymorphic Table Functions

NEW IN 18c

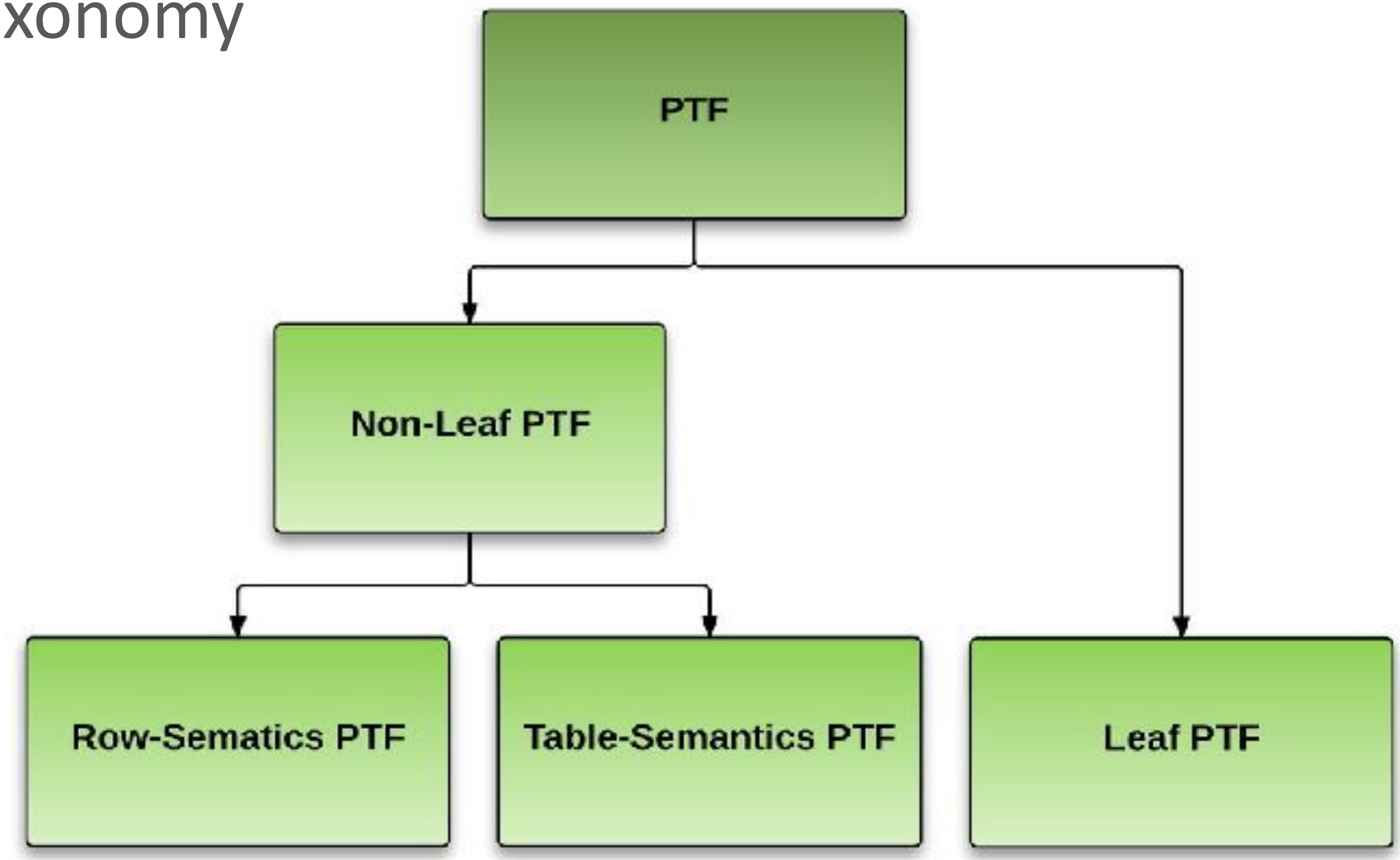
What is a Self-Describing/Polymorphic Table Function?

ANSI SQL 2016: Definition

- Polymorphic Table Functions (PTF) are user-defined functions that can be invoked in the **`FROM`** clause.
- Capable of processing any table
  - row type is not declared at definition time
  - produces a result table whose row type may/may not be declared at definition time.
- Allows application developers to leverage the long-defined dynamic SQL
  - Simple SQL access to powerful and complex custom functions.



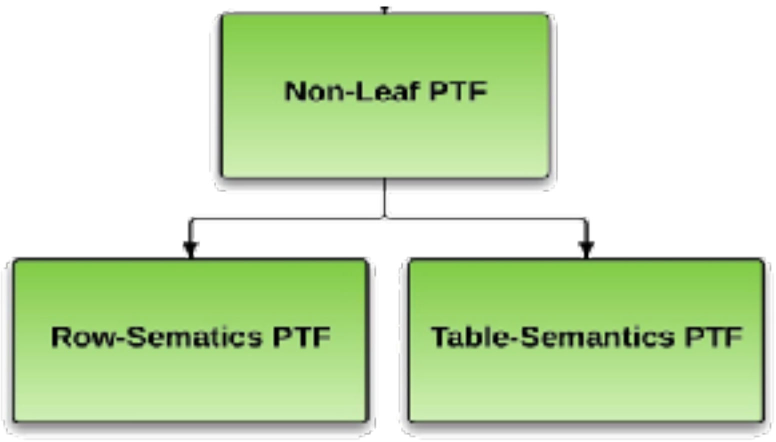
PTF Taxonomy



NEW IN 18c

PTF Taxonomy - Explained

- **Non-Leaf PTF:** Transforms an arbitrary input row stream into an output row stream.
  - Row Semantics – The PTF acts on a single row at a time, to produce its zero, one, or many output rows.
  - Table Semantics – The PTF acts on a set of rows. Where the input table is optionally partitioned into disjoint sets and each set is optionally ordered.



- **Leaf PTF:** Doesn't have input parameters of table or query type. Typically used for accessing "foreign" data sources.

On the Roadmap

Top 5 PTF Optimizations

- ✓ Pass through columns
- ✓ Projection and predicate push-down/push-through
- ✓ PTF execution in-lined with SQL execution
- ✓ Bulk data transfer into and out of PTF
- ✓ Parallel Execution

NEW IN 18c

NEW IN 18c

Part 1 - Define Implementation Package

```
CREATE OR REPLACE PACKAGE echo_package AS
      -- @Required
      procedure Describe(-- Generic Arguments:
                     newcols OUT  DBMS_TF.columns_new_t,
                     -- Specific Arguments:
                     tab    IN OUT DBMS_TF.table_t,
                     cols   IN     DBMS_TF.columns_t);
      -- @Optional
     procedure Open;

     -- @Required
     procedure Fetch_Rows;

     -- @Optional
     procedure Close;
   end;
```


Part 2 - Define Polymorphic Table Function

NEW IN 18c

```
CREATE OR REPLACE FUNCTION
  echo(tab table, cols columns)
RETURN TABLE PIPELINED ROW
POLYMORPHIC USING echo_package;
```

Part 3a - Implementation of Package Body

NEW IN 18c

```
CREATE OR REPLACE PACKAGE BODY echo_package AS
PROCEDURE Describe(

-- Generic Arguments:
      newcols OUT DBMS_TF.columns_new_t,

-- Specific Arguments:
      tab IN OUT DBMS_TF.table_t,
      cols IN DBMS_TF.columns_t)
as
      read_count pls_integer := 0;
begin
. . .

end;
```

Part 3b - Implementation of Package Body

NEW IN 18c

```
PROCEDURE Open
as
   env DBMS_TF.env_t := DBMS_TF.Get_Env();

begin
  DBMS_TF.Trace('Open()');
  DBMS_TF.Trace('Get_Col.Count = '||
         env.get_columns.count, prefix => '....');
  DBMS_TF.Trace('Put_Col.Count = '||
         env.put_columns.count, prefix => '....');
end;
```

NEW IN 18c

Part 3c - Implementation of Package Body

```
PROCEDURE Fetch_Rows
as
  Col DBMS_TF.tab_varchar2_t;
  col_count pls_integer :=
             DBMS_TF.Get_Env().get_columns.count;
begin
. . .

end;
```

Part 3d - Implementation of Package Body

NEW IN 18c

```
PROCEDURE Close
as

begin
   DBMS_TF.Trace('Close()', separator=>'*');
end;
```

NEW IN 18c

Using A Polymorphic Table

```
SELECT *
FROM ECHO(emp, COLUMNS(ename, job))
WHERE deptno = 20;
```

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO	ECHO_ENAME	ECHO_JOB
7369	SMITH	CLERK	7902	17-DEC-80	800		20	ECHO-SMITH	ECHO-CLER
7566	JONES	MANAGER	7839	02-APR-81	2975		20	ECHO-JONES	ECHO-MANA
7788	SCOTT	ANALYST	7566	19-APR-87	3000		20	ECHO-SCOTT	ECHO-ANAL
7876	ADAMS	CLERK	7788	23-MAY-87	1100		20	ECHO-ADAMS	ECHO-CLER
7902	FORD	ANALYST	7566	03-DEC-81	3000		20	ECHO-FORD	ECHO-ANAL

Explain Plan for Polymorphic Table

```
EXPLAIN PLAN FOR
SELECT *
FROM ECHO(emp, COLUMNS(ename, job))
WHERE deptno = 20;
```

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		5	500	2 (0)	00:00:01
1	VIEW		5	500	2 (0)	00:00:01
2	**POLYMORPHIC TABLE FUNCTION**	**ECHO**				
3	VIEW		5	435	2 (0)	00:00:01
* 4	TABLE ACCESS FULL	EMP	5	435	2 (0)	00:00:01

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

   4 - filter("EMP"."DEPTNO"=20)

Note
-----
   - dynamic statistics used: dynamic sampling (level=2)
```

Explain Plan for Parallel Execution of Polymorphic Table

```
ALTER TABLE emp PARALLEL 2;
EXPLAIN PLAN FOR
SELECT *
FROM ECHO(emp, COLUMNS(ename, job))
WHERE deptno = 20;
```

```
-------------------------------------------------------------------------------------------
| Id  | Operation                        | Name      | Rows  | Bytes | Cost  (%CPU)| Time     |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                 |           |     5 |   500 |     2    (0)| 00:00:01 |
|   1 |  PX COORDINATOR                  |           |       |       |             |          |
|   2 |   PX SEND QC (RANDOM)            | :TQ10000  |     5 |   500 |     2    (0)| 00:00:01 |
|   3 |    VIEW                          |           |     5 |   500 |     2    (0)| 00:00:01 |
|   4 |     POLYMORPHIC TABLE FUNCTION   | ECHO      |       |       |             |          |
|   5 |      VIEW                        |           |     5 |   435 |     2    (0)| 00:00:01 |
|   6 |       PX BLOCK ITERATOR          |           |     5 |   435 |     2    (0)| 00:00:01 |
|*  7 |        TABLE ACCESS FULL         | EMP       |     5 |   435 |     2    (0)| 00:00:01 |
-------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------
   7 - filter("EMP"."DEPTNO"=20)
Note
-----
   - dynamic statistics used: dynamic sampling (level=2)
```

NEW IN 18c

Explain Plan for Polymorphic Table - using IMCDTs

```
EXPLAIN PLAN FOR
  WITH e AS (SELECT /*+ MATERIALIZE */ * FROM emp)
  SELECT * FROM ECHO(e, COLUMNS(ename, job)) WHERE deptno = 20;
```

```
-----------------------------------------------------------------------------------------------------------------------
| Id  | Operation                                  | Name                        | Rows   | Bytes | Cost  (%CPU)| Time     |
-----------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                           |                             |     14 |  1400 |     4    (0)| 00:00:01 |
|   1 |  TEMP TABLE TRANSFORMATION                 |                             |        |       |             |          |
|   2 |   LOAD AS SELECT (CURSOR DURATION MEMORY)| SYS_TEMP_0FD9D6612_276EFC |        |       |             |          |
|   3 |    TABLE ACCESS FULL                       | EMP                         |     14 |  1218 |     2    (0)| 00:00:01 |
|   4 |   VIEW                                     |                             |     14 |  1400 |     2    (0)| 00:00:01 |
|   5 |    POLYMORPHIC TABLE FUNCTION              | ECHO                        |        |       |             |          |
|   6 |     VIEW                                   |                             |     14 |  1218 |     2    (0)| 00:00:01 |
|*  7 |      VIEW                                  |                             |     14 |  1218 |     2    (0)| 00:00:01 |
|   8 |       TABLE ACCESS FULL                    | SYS_TEMP_0FD9D6612_276EFC   |     14 |  1218 |     2    (0)| 00:00:01 |
-----------------------------------------------------------------------------------------------------------------------
```

Explain Plan for Polymorphic Table – Using Results Cache

NEW IN 18c

```
EXPLAIN PLAN FOR
   WITH e AS (SELECT /*+ result_cache */ *
              FROM echo(emp, COLUMNS(ename, job)))
   SELECT * FROM e WHERE deptno = 20;
---------------------------------------------------------------------------------------------------------
| Id  | Operation                     | Name                        | Rows  | Bytes | Cost (%CPU)| Time     |
---------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |                             |    14 |  1400 |     2   (0)| 00:00:01 |
|*  1 |  VIEW                         |                             |    14 |  1400 |     2   (0)| 00:00:01 |
|   2 |   RESULT CACHE                | df9wucm9ak4br4mdpt7t2z1xv8  |       |       |            |          |
|   3 |    VIEW                       |                             |    14 |  1400 |     2   (0)| 00:00:01 |
|   4 |     POLYMORPHIC TABLE FUNCTION | ECHO                       |       |       |            |          |
|   5 |      VIEW                     |                             |    14 |  1218 |     2   (0)| 00:00:01 |
|   6 |       TABLE ACCESS FULL       | EMP                         |    14 |  1218 |     2   (0)| 00:00:01 |
---------------------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------
   1 - filter("DEPTNO"=20)

Result Cache Information (identified by operation id):
------------------------------------------------------
   2 - column-count=10; dependencies=(SCOTT.EMP, SCOTT.ECHO_PACKAGE, SCOTT.ECHO_PACKAGE, SCOTT.ECHO);
attributes=(dynamic); name="select /*+ result_cache */ * from ECHO(emp, columns(ename, job))"
```

Explain Plan for Polymorphic Table – Temporal Queries

```
EXPLAIN PLAN FOR
    WITH e AS (SELECT * FROM emp
           AS OF TIMESTAMP (SYSTIMESTAMP - INTERVAL '1' MINUTE))
    SELECT * FROM echo(e, COLUMNS(ename,job));
```

```
---------------------------------------------------------------------------------
| Id  | Operation                  | Name | Rows  | Bytes | Cost  (%CPU)| Time     |
---------------------------------------------------------------------------------
|   0 | SELECT STATEMENT           |      |    82 |  8200 |     2    (0)| 00:00:01 |
|   1 |  VIEW                      |      |    82 |  8200 |     2    (0)| 00:00:01 |
|   2 |   POLYMORPHIC TABLE FUNCTION | ECHO |       |       |             |          |
|   3 |    VIEW                    |      |    82 |  7134 |     2    (0)| 00:00:01 |
|   4 |     TABLE ACCESS FULL      | EMP  |    82 |  7134 |     2    (0)| 00:00:01 |
---------------------------------------------------------------------------------
```

Summary

Key Benefits of Polymorphic Tables

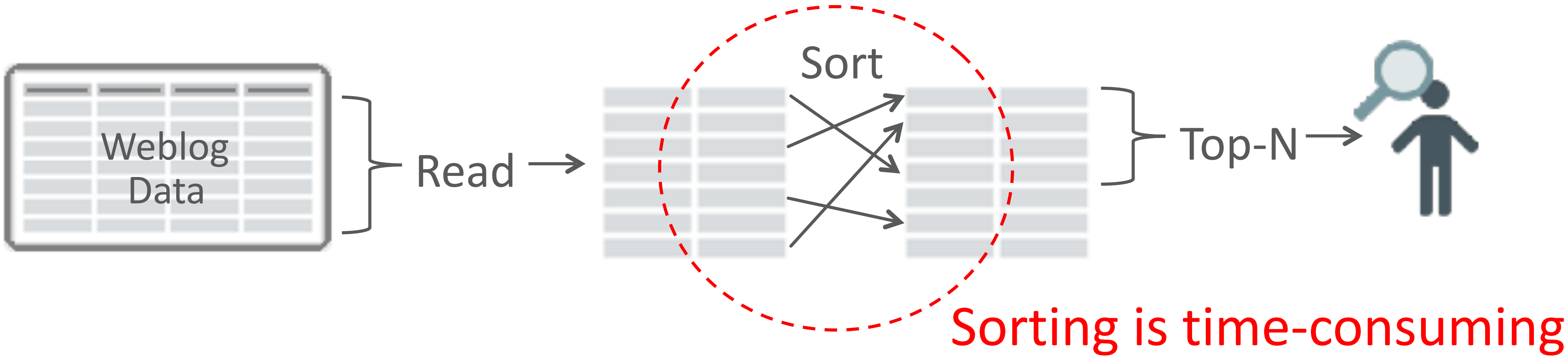
- Simpler to design and build
- Provides complete reusability
- Simpler to make parallel enabled
- Simpler to deploy
- Moves more processing back inside DB

Approximate Top-N Filtering

Top-N Queries

NEW IN 18c

- What are the top five products sold by week for the past year?
- Who are the top five earners by region?
- How many page views did the top five blog posts get last week?
- How much did my top fifty customers each spend last year?
- What components are failing most often by vehicle model?



NEW IN 18c

Top-N approximate aggregation

- Approximate results for common top n queries
  - How many approximate page views did the top five blog posts get last week?
  - What were the top 50 customers in each region and their approximate spending?
- Orders of magnitude faster processing with high accuracy (error rate < 0.5%)
- New approximate functions APPROX_COUNT(), APPROX_SUM(), APPROX_RANK()

Top 5 blogs with approximate hits

```
SELECT blog_post, APPROX_COUNT(*)
FROM   weblog
GROUP BY blog_post
HAVING
  APPROX_RANK(order by
      APPROX_COUNT(*) DESC) <= 5;
```

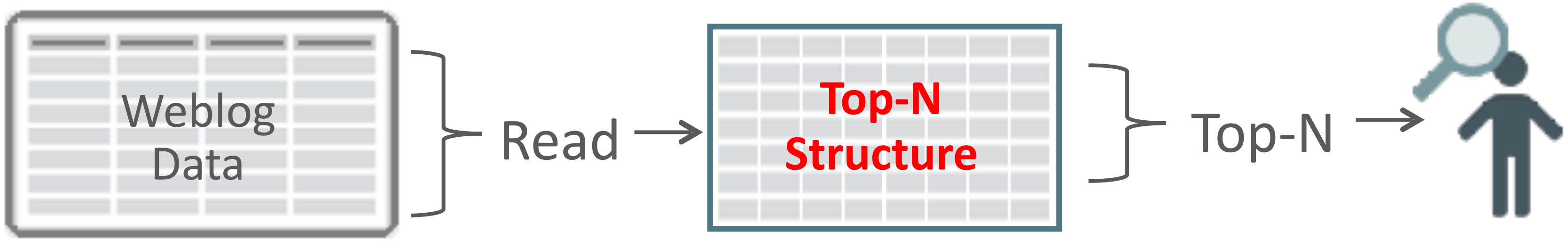
Top 50 customers per region with approximate spending

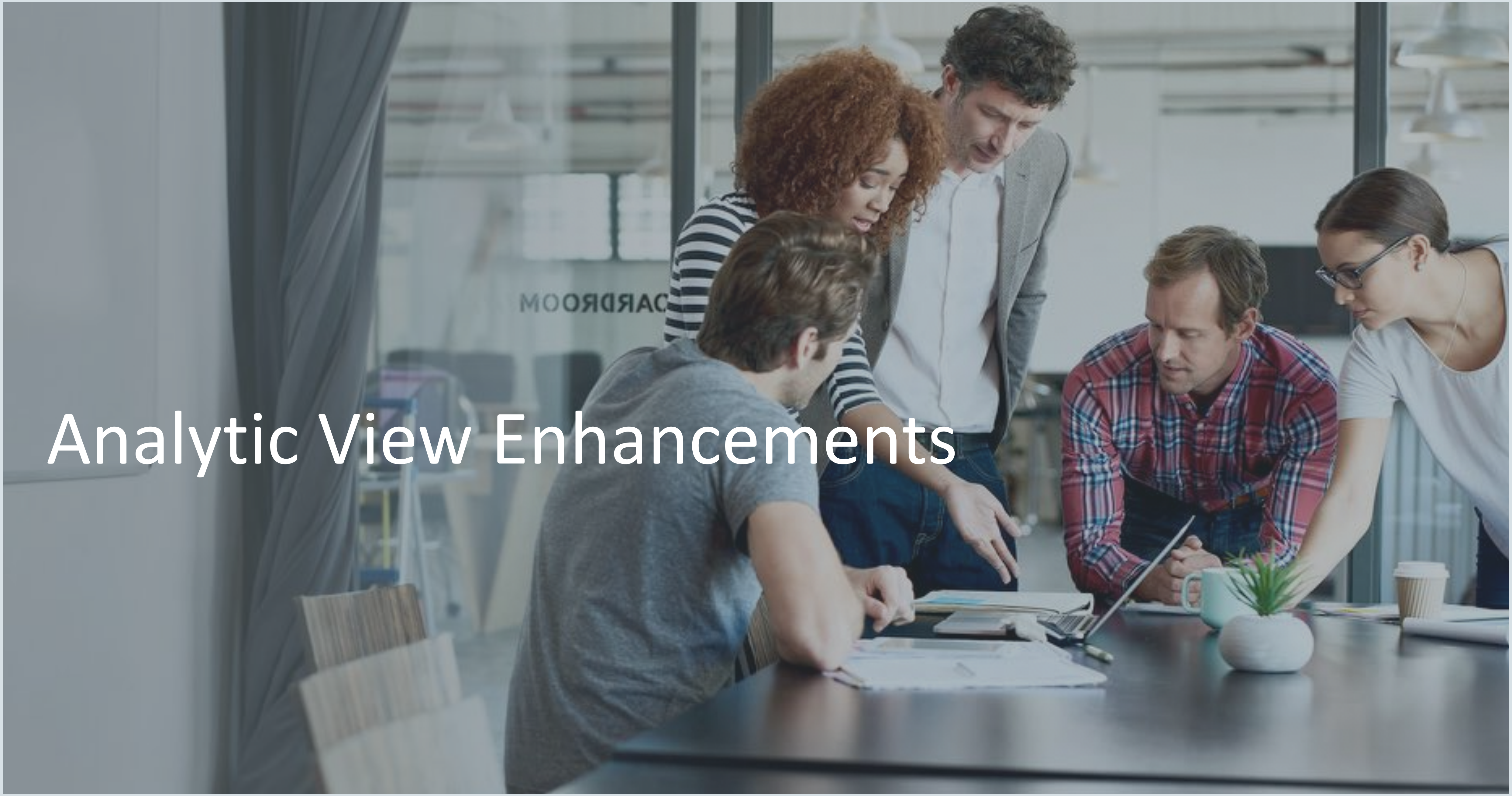
```
SELECT region, customer_name,
       APPROX_RANK(PARTITION BY region
                   ORDER BY APPROX_SUM(sales) DESC) appr_rank,
       APPROX_SUM(sales) appr_sales
FROM sales_transactions
GROUP BY region, customer_name
HAVING APPROX_RANK(...) <=50;
```

Approximate Top-N Queries

NEW IN 18c

- Approx. functions:
  - APPROX_COUNT and APPROX_RANK
- High performance
  - The benefit is most significant for large datasets
- High accuracy
  - Maximum error reporting
- "Top-N Structure" is small and memory-resident
  - No disk sorts





Analytic View Enhancements

NEW IN 18c

Enhancements to Analytic Views

- More calculations within Analytic Views:
  - Ranking and statistical functions
    - RANK_*, PERCENTILE_*, STATS_*, COVAR_*
  - Hierarchical expressions
    - HIER_DEPTH, HIER_LEVEL, HIER_MEMBER_NAME, etc
- Broader schema support for Analytic Views:
  - Snowflake schemas; flat/denormalized fact tables (in addition to star schemas)
- More powerful SQL over Analytic Views:
  - Dynamic definition of calculations within SQL queries

NEW IN 18c

MDX Query Language with Analytic Views

- Support for MDX (Multi-Dimensional Expression) query language
  - Initially certified for use by Microsoft Excel Pivot Tables
    - Support/certification for other applications to follow
  - Includes a multi-dimensional query cache
    - Similar to the SQL Result Cache

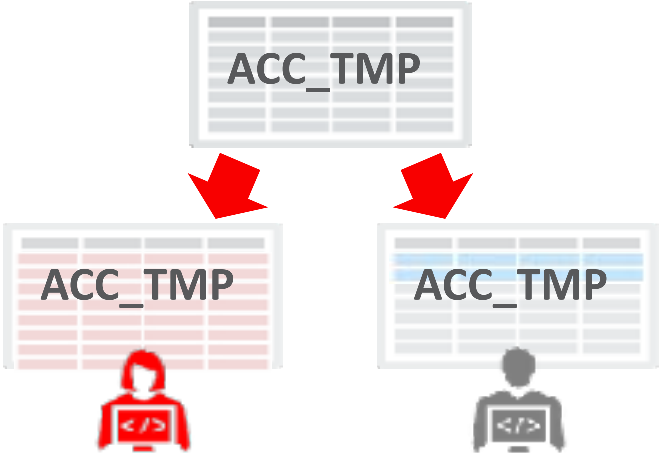
Analytic View

```
SELECT
    {[Measures].[Sales],
      [Measures].[Units_Sold]} ON COLUMNS,
    {[Time].[Calendar].[Year].&[2014],
      [Time].[Calendar].[Year].&[2015]} ON ROWS
FROM [Sales_View]
WHERE ([Customer].[Region].[North America],
      [Product].[Departments].[Category].&[Cameras])
```

Private Temporary Tables

NEW IN 18c

Private Temporary Tables



Global temporary tables

- Persistent, shared (global) table definition
- Temporary, private (session-based) data content
  - Data physically exists for a transaction or session
  - Session-private statistics

Private temporary tables (18.1)

- Temporary, private (session-based) table definition
  - Private table name and shape
- Temporary, private (session-based) data content
  - Session or transaction duration

Inline External Tables

In-lining external tables

NEW IN 18c

- External tables
  - first class object where row data resides outside database
  - maps external data to internal data (table columns)
  - access type:
    - oracle_loader (default)
    - oracle_datapump
    - oracle_hive
    - oracle_hdfs
  - default directory (directory object)
  - access parameters (opaque)
  - location list (data source)
  - reject limit

Inline external tables

- Inline external tables (inline XT)
  - don’t have to create an external table
  - query with inline XT clause, similar to inline view
  - syntax similar to external table DDL, except for column list

Inline external tables

- Example

```
select myext.*
from external
(
  (deptno number(2), dname varchar2(12), loc varchar2(13))
  type ORACLE_LOADER
  default directory scott_def_dir1
  access parameters
  (
    records delimited by newline
    badfile scott_def_dir2:'deptXT1.bad'
    logfile scott_def_dir2:'deptXT2.log'
    fields terminated by ','
    missing field values are null
  )
  location ('tkexld01.dat')
  reject limit unlimited
) myext;
```

Inline external tables

- Example, cont.

```
PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------
Plan hash value: 674205990


-------------------------------------------------
| Id  | Operation                  | Name       |
-------------------------------------------------
|   0 | SELECT STATEMENT           |            |
|   1 |  EXTERNAL TABLE ACCESS FULL| MYEXT      |
-------------------------------------------------
```

Inline external tables

- Example, cont.

```
-- inline XT in WITH clause
with dext as (
 select * from external
 ((deptno char(2), dname char(14), loc char(13))
   type oracle_loader
   default directory scott_def_dir1
   access parameters (fields terminated by ',')
   location ('tkexld01.dat')
   reject limit unlimited
 )
)
select d.dname
from dext d
where d.deptno = 10
order by 1;
```

Data Bound Collations

NEW IN 18c

Data-Bound Collation

- Precise and consistent application of linguistic comparison in queries
  - Adds COLLATE clause to declare column's collation to be used in all queries
  - COLLATE operator precisely controls collation in expressions
- Case- and accent-sensitive collations (e.g. BINARY_CI) simplify implementation of case-insensitive queries
- Feature is based on ISO/IEC SQL Standard and simplifies application migration from other databases supporting the COLLATE clause

NEW IN 18c

Column-Based Data-Bound Collation

"... a named set of rules describing how to compare and match character strings to put them in a specified order..."

- Based on the ISO/IEC/ANSI SQL standard 9075:1999
- Character set is always declared at the database level
- Collation declared for a **column**
  - Does not determine the character set of data in the column

- Why is it important?
  - it simplifies application migration to the Oracle Database from a number of non-Oracle databases implementing collation in a similar way

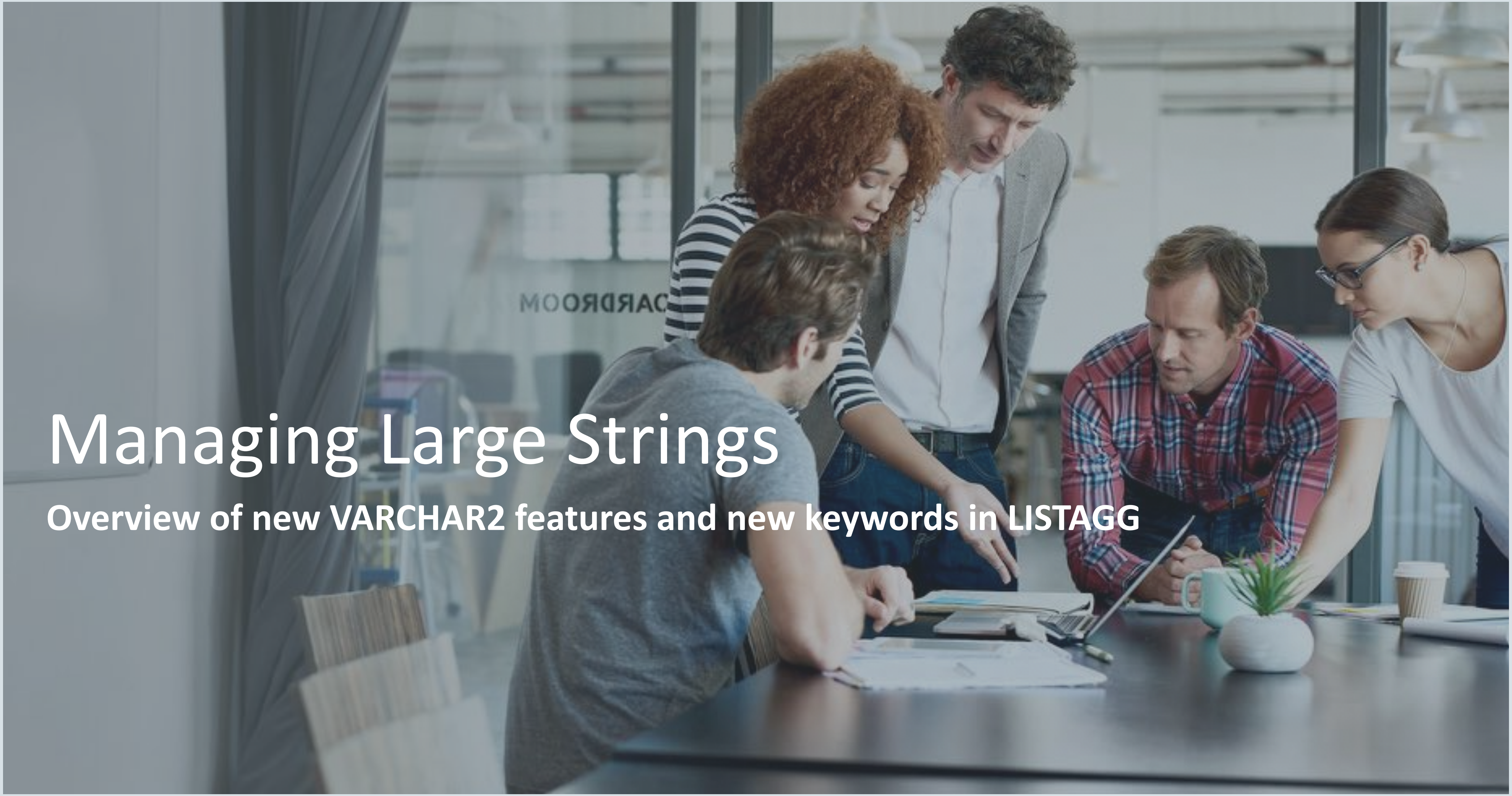
NEW IN 18c

Column-Based Data-Bound Collation

- Oracle supports around 100 linguistic collations
  - Parameterized by adding the suffix _CI or the suffix _AI
    - _CI - Specifies a case-insensitive sort
    - _AI - Specifies an accent-insensitive sort

```
CREATE TABLE products
( product_code        VARCHAR2(20 BYTE)    COLLATE BINARY
, product_name        VARCHAR2(100 BYTE)   COLLATE GENERIC_M_CI
, product_category    VARCHAR2(5 BYTE)   1 COLLATE BINARY
, product_description VARCHAR2(1000 BYTE)  COLLATE BINARY_CI
);
```

  - *Product_name is to be compared using GENERIC_M_CI - case-insensitive version of generic multilingual collation*



Managing Large Strings

Overview of new VARCHAR2 features and new keywords in LISTAGG

Pre-12.2 LISTAGG

- Pre 12.2 syntax to manage lists was relatively simple:

```
LISTAGG(c.cust_first_name||' '||c.cust_last_name, ',’)
               WITHIN GROUP (ORDER BY c.country_id) AS Customer
```

- **Issue**….key issue is overflow error:
  - `ORA-01489: result of string concatenation is too long`
- **Solutions in 12.2**
  - Increase VARCHAR2 size to support larger strings
  - Handle overflow errors - New syntax support to truncate string, optionally display count of truncated items count, and set truncation indication

Support For Larger VARCHAR2 objects

Avoids overflowing LISTAGG function by increasing size of VARCHAR(2) objects

- Introduced in **12c Release 1**
  - VARCHAR2 objects supports up to 32K

```
SQL> show parameter MAX_STRING_SIZE

     NAME                 TYPE    VALUE
     ---------------- ------ --------
     max_string_size string STANDARD

     ALTER SYSTEM SET max_string_size=extended SCOPE= SPFILE;
```

  - Need to run rdbms/admin/utl32k.sql script

New Keywords For Use With LISTAGG

- With 12.2 we have made it easier to manage lists:

```
LISTAGG(<measure_column>[, <delimiter>] . . .
```

- What to do when an overflow occurs
  - `ON OVERFLOW ERROR` *(default)*
  - `ON OVERFLOW TRUNCATE <delimiter>`
- Control to show/not-show many values were truncated
  - `WITHOUT COUNT` *(default)*
  - `WITH COUNT`

New Keywords For Use With LISTAGG WITH COUNT

```
SELECT
  g.country_region,
  LISTAGG(c.cust_first_name||' '||c.cust_last_name, ','
            ON OVERFLOW TRUNCATE WITHOUT COUNT)
        WITHIN GROUP (ORDER BY c.country_id) AS Customer
FROM customers c, countries g
WHERE g.country_id = c.country_id
GROUP BY country_region
ORDER BY country_region;
```

Keywords: ON OVERFLOW TRUNCATE WITHOUT COUNT

Query Result ×

SQL | All Rows Fetched: 6 in 0.174 seconds

	COUNTRY_REGION	CUSTOMER	
1	Africa	Abbie Venkayala,Adel Rhodes,Alfred Doctor,	
2	Americas	Abel Aaron,Abel Vance,Abel Walsh,Abigail (	ansfield,Calvert Owens,Camilla Lowe,Camilla Osgode,...
3	Asia	Abbie Kilroy,Abel Aaron,Abel Embrey,Abel E	is Gatewood,Loris Gatewood,Loris Orrson,Lorrel Barry,...
4	Europe	Aaron Kelley,Abbie Kilroy,Abel Aaron,Abel	ight,Babs Cartwright,Babs Cartwright,Babs Cartwright,...
5	Middle East	Alma Elliott,Arnold Carr,Augustus Tate,Aus	
6	Oceania	Abel Forster,Abel Vance,Abel Walsh,Abigail	: Lake,Herbert Motion,Herman Posthumus,Hesper Duke,...

New Keywords For Use With LISTAGG WITHOUT COUNT

```
SELECT
  g.country_region,
  LISTAGG(c.cust_first_name||' '||c.cust_last_name, ','
              ON OVERFLOW TRUNCATE '***' WITH COUNT)
          WITHIN GROUP (ORDER BY c.country_id) AS Customer
FROM customers c, countries g
WHERE g.country_id = c.country_id
GROUP BY country_region
ORDER BY country_region;
```

Managing Data Conversion Errors

Pre 12.2 Data Conversion Errors Parsing Data

- **Issue**: Parsing data input from a web form or loading data from external files , converting to specific data type typically generates error:

```
SQL Error: ORA-01722: invalid number
```

- **Solutions**:
  - Detect data conversion errors with new **VALIDATE_CONVERSION** function
  - Enhancements to most of conversion functions like **TO_NUMBER, TO_DATE , CAST** etc. to handle data conversion errors and replace with user provided default values

Detecting conversion errors - **VALIDATE_CONVERSION**

Identifying invalid data in the input streams

- Useful to detect if input value can be converted to destination type. Returns 1 if conversion is successful, otherwise returns 0

- **VALIDATE_CONVERSION** ('123a' as NUMBER) --> returns **0**
- **VALIDATE_CONVERSION** ('123' as NUMBER) --> returns **1**

- Can be efficiently used as filter to avoid bad data while importing foreign data sources, ETL processing

Two Methods for Dealing With Conversion Errors

Find row-column values that are causing errors: VALIDATE_CONVERSION

```
SELECT
   VALIDATE_CONVERSI
   VALIDATE_CONVERSI
   VALIDATE_CONVERSI
   VALIDATE_CONVERSI
   VALIDATE_CONVERSI
   VALIDATE_CONVERSI
FROM staging_emp;
```

Script Output × | Query Result ×

SQL | All Rows Fetched: 9 in 0.005 seconds

	IS_EMPNO	IS_MGR	IS_HIREDATE	IS_SAL	IS_COMM	IS_DEPTNO
1	0	1	1	1	1	1
2	1	1	0	1	1	1
3	1	1	1	1	1	0
4	0	1	1	1	1	1
5	1	1	1	1	1	1
6	0	1	1	1	1	1
7	1	0	1	1	1	1
8	0	1	1	1	1	1
9	1	1	1	1	1	1

Handling data conversion errors - TO_xxxx(), CAST()

-Replacing incorrect or missing data with default values

- Pre 12.2: TO_NUMBER('123a') --> returns invalid number error (ora-01722)

New 12.2 Features

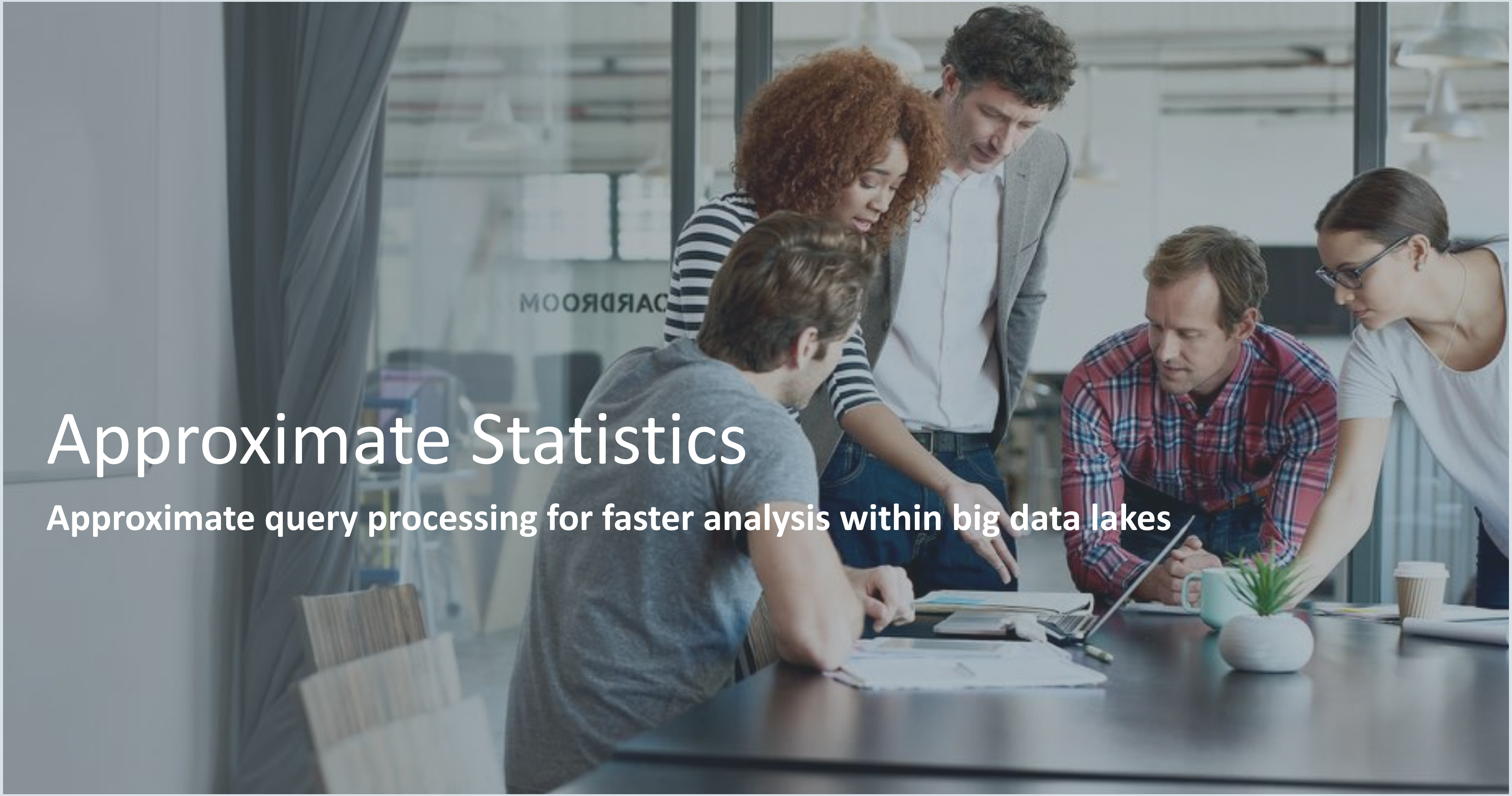
- New syntax **DEFAULT <default_value> ON CONVERSION ERROR**
  - Replace conversion failure with user defined default value
  - TO_NUMBER('123a' DEFAULT '123' ON CONVERSION ERROR) --> returns 123

- This new syntax can be used for TO_NUMBER, TO_DATE, TO_TIMESTAMP, TO_TIMESTAMP_TZ, TO_DMINTERVAL, TO_YMINTERVAL and CAST

Using CAST and TO_XXXX FUNCTIONS

Using enhanced functions to remove incorrect data types and correct conversion errors

```
INSERT INTO emp
SELECT
   empno,
   ename,
   job,
   CAST(mgr AS NUMBER DEFAULT 9999 ON CONVERSION ERROR),
   CAST(hiredate AS DATE DEFAULT sysdate ON CONVERSION ERROR),
   CAST(sal AS NUMBER DEFAULT 0 ON CONVERSION ERROR),
   CAST(comm AS NUMBER DEFAULT null ON CONVERSION ERROR),
   CAST(deptno AS NUMBER DEFAULT 99 ON CONVERSION ERROR)
FROM staging_emp
WHERE VALIDATE_CONVERSION(empno AS NUMBER) = 1
```



Approximate Statistics

Approximate query processing for faster analysis within big data lakes

Approximate Analysis

NEW IN 12.2

- `PERCENTILE_CONT, PERCENTILE_DISC, MEDIAN`
  - functions require sorting and can consume large amounts of resources

- New approximate SQL functions:

  `APPROX_PERCENTILE`
  `APPROX_MEDIAN`

- Results can be ‘**`DETERMINISTIC`**’
  - Different algorithms used for deterministic and non-deterministic result sets
  - If keyword is not present, it means deterministic results are not mandatory

Approximate Analysis

NEW IN 12.2

```
APPROX_PERCENTILE(pct_expr [DETERMINISTIC][,resulttype])
    WITHIN GROUP (ORDER BY expr [ DESC | ASC ])


APPROX_MEDIAN(expr [DETERMINISTIC][,resulttype])
```

* **pct_expr** – evaluates to a numeric value between 0 and 1, because it is a percentile value

* **resulttype** – optional. If not used then function returns the value at the specified percentile. If specified then values are 'ERROR_RATE' or 'CONFIDENCE'

NEW IN 12.2

Accuracy and Performance

Results for accuracy

- Real world customer data set *(manufacturing use case)*
- Error range around **0.1 - 1.0%**
- In general accuracy will not be a major concern

Performance Results

- Using TPC-H schema and workload
- **6-13x** improvement
- Note that major savings coming from:
  - Use of bounded memory regardless of the input size per group by key
  - Reduction in chance of spill to disk

NEW IN 12.2

Approximate Analysis

How to get more information about result set

- Queries will be able to report error rates and confidence levels as follows:

```
SELECT
   APPROX_MEDIAN (sal) AS median_sal,
   APPROX_MEDIAN (sal, 'DETERMINISTIC'),
   APPROX_MEDIAN (sal, 'ERROR_RATE') AS error_rate,
   APPROX_MEDIAN (sal, 'CONFIDENCE') as confidence,
FROM emp ;
```

Using approximate processing with **zero code changes**!

NEW IN 12.2

Converting Existing Queries To Return Approximate Answers

- Using following parameters to convert existing queries:
  - `approx_for_count_distinct = TRUE/FALSE[DEFAULT]`
    - Convert existing `COUNT(DISTINCT …)` functions to use approximate processing
  - `approx_for_percentile = 'PERCENTILE_CONT/PERCENTILE_DISC/MEDIAN/ALL'`
  - `approx_percentile_deterministic = TRUE/FALSE[DEFAULT]`
- *Can be set at session and database level*

Impact of PERCENTILE_CONT Processing

Operation	Name	Lin...	Estimated ...	Cost	Timeline(187s)	Execu...	Actual R...	Memory (...	Temp (Max)	O...
SELECT STATEMENT		0				1	1			
SORT AGGREGATE		1	1			1	1			
PX COORDINATOR		2				65	32			
PX SEND QC (RANDOM)	:TQ10001	3	1			32	32			
SORT AGGREGATE		4	1			32	32			
VIEW		5	15	5,084		32	15			
SORT GROUP BY		6	15	5,084		32	15	1GB	11GB	
PX RECEIVE		7	105M	4,974		32	105M			
PX SEND HASH	:TQ10000	8	105M	4,974		32	105M			
PX BLOCK ITERATOR		9	105M	4,974		32	105M			
TABLE ACCESS STORAGE FULL	NDV	10	105M	4,974		426	105M	97MB		

Impact of PERCENTILE_CONT Processing

Operation	Name	Lin...	Estimated ...	Cost	Timeline(187s)	Execu...	Actual R...	Memory (...	Temp (Max)	O...
SELECT STATEMENT		0				1	1			
SORT AGGREGATE		1	1			1	1			
PX COORDINATOR		2				65	32			
PX SEND QC (RANDOM)	:TQ10001	3	1			32	32			
SORT AGGREGATE		4	1			32	32			
VIEW		5	15	5,084		32	15			
SORT GROUP BY		6	15	5,084		32	15	1GB	11GB	
PX RECEIVE		7	105M	4,974		32	105M			
PX SEND HASH	:TQ10000	8	105M	4,974		32	105M			
PX BLOCK ITERATOR		9	105M	4,974		32	105M			
TABLE ACCESS STORAGE FULL	NDV	10	105M	4,974		426	105M	97MB		

1. Query accesses 105M rows from source table NDV

Impact of PERCENTILE_CONT Processing

Operation	Name	Lin...	Estimated ...	Cost	Timeline(187s)	Execu...	Actual R...	Memory (...	Temp (Max)	O...
SELECT STATEMENT		0				1	1			
SORT AGGREGATE		1	1			1	1			
PX COORDINATOR		2				65	32			
PX SEND QC (RANDOM)	:TQ10001	3	1			32	32			
SORT AGGREGATE		4	1			32	32			
VIEW		5	15	5,084		32	15			
SORT GROUP BY		6	15	5,084		32		1GB	11GB	
PX RECEIVE		7	105M	4,974		32	105M			
PX SEND HASH	:TQ10000	8	105M	4,974		32	105M			
PX BLOCK ITERATOR		9	105M	4,974		32	105M			
TABLE ACCESS STORAGE FULL	NDV	10	105M	4,974		426	105M	97MB		

2

3

1. Query accesses 105M rows from source table NDV
2. SORT GROUP BY operation consumes temp and memory: 11GB + 1GB

Benefits of APPROX_PERCENTILE Processing

Operation	Name	Lin...	Estimated ...	Cost	Timeline(14s)	Execu...	Actual R...	Memory (...	Temp (Max)
SELECT STATEMENT		0				1	1		
SORT AGGREGATE		1	1			1	1		
PX COORDINATOR		2				65	32		
PX SEND QC (RANDOM)	:TQ10001	3	1			32	32		
SORT AGGREGATE		4	1			32	32		
VIEW		5	15	5,084		32	15		
SORT GROUP BY APPROX		6	15	5,084		32	15	830KB	
PX RECEIVE		7	105M	4,974		32	105M		
PX SEND HASH	:TQ10000	8	105M	4,974		32	105M		
PX BLOCK ITERATOR		9	105M	4,974		32	105M		
TABLE ACCESS STORAGE FULL	NDV	10	105M	4,974		426	105M	97MB	

1

1. Query accesses 105M rows from source table NDV

Benefits of APPROX_PERCENTILE Processing

Operation	Name	Lin...	Estimated ...	Cost	Timeline(14s)	Execu...	Actual R...	Memory (...	Temp (Max)
SELECT STATEMENT		0				1	1		
SORT AGGREGATE		1	1			1	1		
PX COORDINATOR		2				65	32		
PX SEND QC (RANDOM)	:TQ10001	3	1			32	32		
SORT AGGREGATE		4	1			32	32		
VIEW		5	15	5,084		32	15		
SORT GROUP BY APPROX		6	15	5,084		32	[illegible]	830KB	
PX RECEIVE		7	105M	4,974		32	105M		
PX SEND HASH	:TQ10000	8	105M	4,974		32	105M		
PX BLOCK ITERATOR		9	105M	4,974		32	105M		
TABLE ACCESS STORAGE FULL	NDV	10	105M	4,974		426	105M	97MB	

1. Query accesses 105M rows from source table NDV
2. SORT GROUP BY operation consumes ZERO temp and 830KB memory

Benefits of APPROX_PERCENTILE: 13X Faster

1

Operation	Name	Lin…	Estimated …	…t	Timeline(187s)	Execu…	Actual R…	Memory (…	Temp (Max)	O…
SELECT STATEMENT		0				1	1			
SORT AGGREGATE		1	1			1	1			
PX COORDINATOR		2				65	32			
PX SEND QC (RANDOM)	:TQ10001	3	1			32	32			
SORT AGGREGATE		4	1			32	32			
VIEW		5	15	5,084		32	15			
SORT GROUP BY		6	15	5,084		32	15	1GB	11GB	

3

Operation	Name	Lin…	Estimated …	…t	Timeline(14s)	Execu…	Actual R…	Memory (…	Temp (Max)
SELECT STATEMENT		0				1	1		
SORT AGGREGATE		1	1			1	1		
PX COORDINATOR		2				65	32		
PX SEND QC (RANDOM)	:TQ10001	3	1			32	32		
SORT AGGREGATE		4	1			32	32		
VIEW		5	15	5,084		32	15		
SORT GROUP BY APPROX		6	15	5,084		32	15	830KB	

2

Approximate Aggregations

Why create a reusable approximate result set?

- **Requirement:** Support fast access to approximate answers for wide range of GROUP BY queries
- **Objective**: Avoid revisiting and re-scanning base tables

- Use cases for storing reusable approximate aggregations
  - CTAS as part of ETL process for staging data
  - CTAS as part of larger analytical process
    - pushing data into dashboards and supporting drill-down click-through analysis
  - Materialized views for query rewrite of approximate queries
  - Materialized views for transparent query rewrite to approximate queries

NEW IN 12.2

Building Reusable Approximate Result sets

COUNTRY	STATE	PRODUCT	…
US	CA	A	
US	CA	B	
…			
US	IL	A	
US	IL	C	
US	IL	D	
…			
US	TX	A	
…			
US	CO	D	
US	CO	F	
US	CO	H	
…			
US	NY	A	
US	NY	A	
US	NY	G	
…			

```
…
 APPROX_COUNT_DISTINCT_DETAIL(product) AS ac_prod
…
GROUP BY country, state
```

COUNTRY	STATE	AC_PROD (INTERNAL)
US	CA	BLOB
US	IL	BLOB
US	TX	BLOB
US	CO	BLOB
US	NY	BLOB

*Builds summary table containing results for all dimensions in **`GROUP BY`** clause*

Creating a STATE level approximation

NEW IN 12.2

Returns results from the specified ***aggregated results table***

```
…
 APPROX_COUNT_DISTINCT_DETAIL
 (product) AS ac_prod
…
GROUP BY country, state
```

COUNTRY	STATE	AC_PROD (INTERNAL)
US	CA	BLOB
US	IL	B
US	TX	B
US	CO	BLOB
US	NY	BLOB

```
…
 TO_APPROX_COUNT_DISTINCT(ac_prod)
…
WHERE state = 'CA'
```

COUNTRY	STATE	AC_PROD
US	CA	2

NEW IN 12.2

Creating a STATE level approximation

*Builds **higher level summary table** based on results from table derived from _**DETAIL** function*

```
…
 APPROX_COUNT_DISTINCT_DETAIL
 (product) AS ac_prod
…
GROUP BY country, state
```

COUNTRY	STATE	AC_PROD (INTERNAL)
US	CA	B
US	IL	B
US	TX	.OB
US	CO	BLOB
US	NY	BLOB

```
…
 APPROX_COUNT_DISTINCT_AGG(ac_prod)
…
GROUP BY country
```

COUNTRY	AC _PROD
US	3
CANANDA	3
MEXICO	.OB
BRAZIL	BLOB

NEW IN 12.2

Building Reusable Approximate Result sets

```
…
 APPROX_COUNT_DISTINCT_AGG(ac_prod)
…
GROUP BY country
```

COUNTRY	AC _PROD
US	B
CANANDA	B
MEXICO	LOB
BRAZIL	BLOB

```
…
 TO_APPROX_COUNT_DISTINCT (ac_prod)
…
```

COUNTRY	AC_PROD
US	84

NEW IN 12.2

Returning COUNTRY data from STATE level approximation

```
…
 APPROX_COUNT_DISTINCT_DETAIL
 (product) AS ac_prod
…
GROUP BY country, state
```

COUNTRY	STATE	AC_PROD (INTERNAL)
US	CA	
US	IL	
US	TX	[illegible]
US	CO	BLOB
US	NY	BLOB

```
…
 TO_APPROX_COUNT_DISTINCT(
 APPROX_COUNT_DISTINCT_AGG(ac_prod))
…
GROUP BY country
```

COUNTRY	AC_PROD
US	84

Query Rewrite with Approximate MVs

Building an MV Containing *Approximate* Results

NEW IN 12.2

```
CREATE MATERIALIZED VIEW pctl_mview
ENABLE QUERY REWRITE AS
SELECT
 state,
 count,
 APPROX_PERCENTILE_DETAIL(volume) AS pctl_detail
FROM sales_fact
GROUP BY state, county;
```

Builds materialized view containing results for all dimensions in `GROUP BY` *clause*

NEW IN 12.2

Queries Rewrite to use an *Approximate* MV

```
SELECT
 state,
 county,
 APPROX_PERCENTILE(0.1)
       WITHIN GROUP (ORDER BY volume)
FROM sales_fact
WHERE state = 'CA';
```

Query rewrites to use materialized view **PCTL_MVIEW** *containing approximate results*

NEW IN 12.2

What about non-approximate queries?

Exciting feature: Optimizer can rewrite exact functions to use MV

```
alter session set approx_for_precentile = ‘all’;

   SELECT
     state,
     county,
     MEDIAN(volume)
   FROM sales_fact
   WHERE state = 'CA'
   GROUP BY state, county;
```

Query results returned from materialized view `PCTL_MVIEW` *containing approximate results*

In-Database Dimensional Modeling

12.2

Review: Analytic Views in 12.2

Enhanced Analysis and Simplified Access

- Organizes data into a user and application friendly business model
  - Intuitive for the end user
- Defined with SQL DDL
  - Includes hierarchical expressions and calculated measures
  - Easy to define, supported by SQL Developer
- Easily queried with simple SQL SELECT
  - Smart Analytic View (containing hierarchies and calculations) = Simple Query

Review: Analytic Views in 12.2

12.2

Embedded Calculations

- Define centrally in the Database and access with any application
  - Single version of the truth
- Easily create new measures
  - Simplified syntax based on business model
  - Includes dimensional and hierarchical functions

Sales Year to Date

```
sales_ytd AS
 (SUM(sales)
  OVER(HIERARCHY time_hierarchy
  BETWEEN UNBOUNDED PROCEEDING
  AND 0 FOLLOWING
  WITHIN ANCESTOR AT LEVEL year)
```

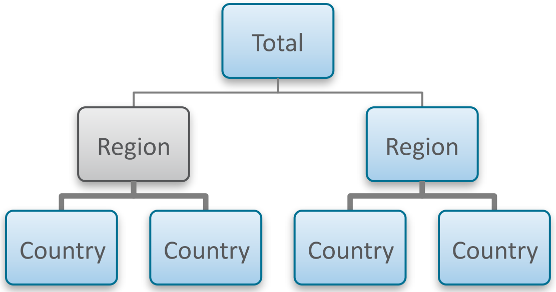
Product Share of Parent

```
share_product_parent_sales AS
 (SHARE_OF (sales
  HIERARCHY product_hierachy PARENT))
```

12.2

Review: Analytic Views in 12.2

Smart Views and Simple Queries



Sun	Mon	Tue	Wed	Thu	Fri	Sat
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31			

```
SELECT time_hierarchy.member_name AS time,
  product_hierarchy.member_name   AS product,
  customer_hierarchy.member_name  AS customer,
  sales                           AS sales,
  sales_ytd_pct_chg_yr_ago        AS sales_ytd_pct_chg,
  share_product_parent_sales      AS prod_share_sales
FROM sales_analysis hierarchies
  (time_hierarchy,
  product_hierarchy,
  customer_hierarchy)
WHERE
  time_hierarchy.level_name         = 'YEAR'
  AND product_hierarchy.level_name  = 'DEPARTMENT'
  AND customer_hierarchy.level_name = 'REGION';
```

Calculations

Aggregate data at Year, Department and Region

External Tables

Enhancements in Database 12c Release 2
MODIFY clause
Partitioned

External Tables

- Key issues:
  - Definition of external table is fixed at creation time
  - Need ability to define table once and use it multiple times, to access different external files
  - Apply same table definition to different inputs
- Solution:
  - Added EXTERNAL MODIFY clause
  - Ease of use enhancement for using external tables
  - Clause allows external table to be overridden at query time
  - Properties: DEFAULT_DIRECTORY, certain ACCESS PARAMETERS, LOCATION and REJECT LIMIT

External Tables – existing functionality

- Example….LOCATION specification is fixed

```
CREATE TABLE SALES_TRANSACTIONS_EXT
(PROD_ID NUMBER,
 CUST_ID NUMBER,
 PROMO_ID NUMBER)
ORGANIZATION EXTERNAL (
TYPE ORACLE_LOADER
DEFAULT DIRECTORY data_file_dir
ACCESS PARAMETERS
(RECORDS DELIMITED BY NEWLINE
 FIELDS (PROD_ID    (1-6) CHAR,
         CUST_ID    (7-11) CHAR,
         PROMO_ID   (12-15) CHAR))
LOCATION('sh_sales1.dat'))
REJECT LIMIT UNLIMITED
```

Override settings with EXTERNAL MODIFY clause

- Example: Override LOCATION specification (continued)

```
SELECT * FROM SALES_TRANSACTIONS_EXT
EXTERNAL MODIFY (LOCATION('sh_sales2.dat'))
```

- NOTE: LOCATION and REJECT LIMIT specifications can be specified as bind values in the EXTERNAL MODIFY clause

Partitioned External Tables

Partitioned External Table

- Similar to partitioned tables stored in Oracle database
- Source files can be stored on file system, Apache Hive storage, or HDFS

- Benefits:
  - Fast query performance
  - Enhanced data maintenance
  - Support static and dynamic(bloom, nested loop, subquery) partition pruning
  - Support full and partial partition-wise join

- Partitioning strategies supported:

Primary\Secondary	Range	List	Auto-List	Interval
Range	Y	Y	N	N
List	Y	Y	N	N
Interval	N	N	N	N

Keywords For Partitioned External Table

- Partitioning strategy determined by PARTITION clause
  - partition by range (c1)

- Partition templates define organization for each partition
  - partition p1 values less than (7655) location('./tkexpetu_p1a.dat', './tkexpetu_p1b.dat'),
  - partition p2 values less than (7845) default directory def_dir2 location('./tkexpetu_p2.dat'),
  - partition p3 values less than (7935) location(def_dir3:'./tkexpetu_p3*.dat’)

Example Partitioned External Table

```
create table salesrp_xt_hdfs
(c1 number, c2 number)
organization external (
  type oracle_hdfs
  default directory def_dir1
  access parameters (
    com.oracle.bigdata.cluster=hadoop_cl_1
    com.oracle.bigdata.fields: (c1 int, c2 int)
    com.oracle.bigdata.rowformat=delimited fields terminated by ','))
reject limit unlimited
partition by range (c1) (
  partition p1 values less than (7655)
    location('./tkexpetu_p1a.dat', './tkexpetu_p1b.dat'),
  partition p2 values less than (7845)
    default directory def_dir2 location('./tkexpetu_p2.dat'),
  partition p3 values less than (7935)
    location(def_dir3:'./tkexpetu_p3*.dat’));
```

Explain Plan For Accessing Partitioned External Table

```
Select * from salesrp_xt_hdfs partition (p2) order by c2;
--------------------------------------------------------------------
| Id  | Operation                    | Name             | Pstart| Pstop |
--------------------------------------------------------------------
|   0 | SELECT STATEMENT             |                  |       |       |
|   1 |  SORT ORDER BY               |                  |       |       |
|   2 |   PARTITION RANGE SINGLE     |                  |     2 |     2 |
|   3 |    EXTERNAL TABLE ACCESS FULL| SALESRP_XT_HDFS  |     2 |     2 |
--------------------------------------------------------------------
```


```
--------------------------------------------------------------------
| Id  | Operation                    | Name             | Pstart| Pstop |
--------------------------------------------------------------------
|   0 | SELECT STATEMENT             |                  |       |       |
```


–
```
select AVG(s.L_PARTKEY) from scott.emp e, salesrp_xt s
 where s.l_orderkey = e.sal and e.job = 'SALESMAN';
```

Creating External Tables for Big Data

Metadata: Extend Oracle External Tables

```
CREATE TABLE movielog (
  click VARCHAR2(4000))
ORGANIZATION EXTERNAL (
  TYPE ORACLE_HIVE
  DEFAULT DIRECTORY DEFAULT_DIR
  ACCESS PARAMETERS
  (
com.oracle.bigdata.tablename logs
com.oracle.bigdata.cluster mycluster
  ))
REJECT LIMIT UNLIMITED;
```

- New types of external tables
  - ORACLE_HIVE (leverage hive metadata)
  - ORACLE_HDFS (specify metadata)
- Access parameters used to describe how to identify sources and process data on the hadoop cluster

Access Parameters: HDFS Example

```
CREATE TABLE WEB_SALES_CSV
(
  WS_SOLD_DATE_SK NUMBER
, WS_SOLD_TIME_SK NUMBER
, WS_ITEM_SK NUMBER
)
ORGANIZATION EXTERNAL
(
  TYPE ORACLE_HDFS
  DEFAULT DIRECTORY DEFAULT_DIR
  ACCESS PARAMETERS
  (
    com.oracle.bigdata.cluster=orabig
    com.oracle.bigdata.fileformat=TEXTFILE
    com.oracle.bigdata.rowformat: DELIMITED FIELDS TERMINATED BY '|'
    com.oracle.bigdata.erroropt: {"action": "replace", "value": "-1"}
  )
  LOCATION ('/data/tpcds/benchmarks/bigbench/data/web_sales')
)
REJECT LIMIT UNLIMITED;
```

- Access Parameters describe source data and processing rules
- Schema-on-Read

Access Parameters: ORACLE_HIVE

```
CREATE TABLE WEB_SALES_CSV
(
  WS_SOLD_DATE_SK NUMBER
, WS_SOLD_TIME_SK NUMBER
, WS_ITEM_SK NUMBER
)
ORGANIZATION EXTERNAL
(
  TYPE ORACLE_HIVE
  DEFAULT DIRECTORY DEFAULT_DIR
  ACCESS PARAMETERS
  (
   com.oracle.bigdata.cluster=orabig
   com.oracle.bigdata.tablename: csv.web_sales
   com.oracle.bigdata.erroropt: {"action": "replace", "value": "-1"}
   com.oracle.bigdata.datamode=automatic
  )
REJECT LIMIT UNLIMITED;
```

- Access Parameters refer to metadata description in Hive
- Add processing rules

Use ORACLE_HIVE When Possible

- Oracle Database query execution accesses Hive metadata at describe time
  - Changes to underlying Hive access parameters will not impact Oracle table (one exception… column list)
- Metadata an enabler for performance optimizations
  - Partition pruning and predicate pushdown into intelligent sources
- Utilize tooling for simplified table definitions
  - SQL Developer and DBMS_HADOOP packages

Viewing Hive Metadata from Oracle Database

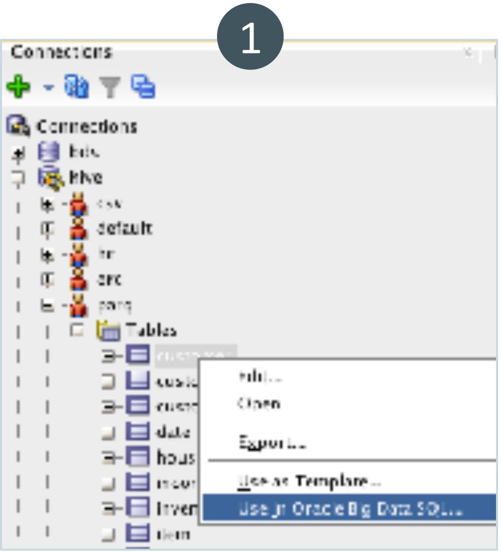
- ALL_HIVE_DATABASES, ALL_HIVE_TABLES, ALL_HIVE_COLUMNS

ALL_HIVE_COLUMNS

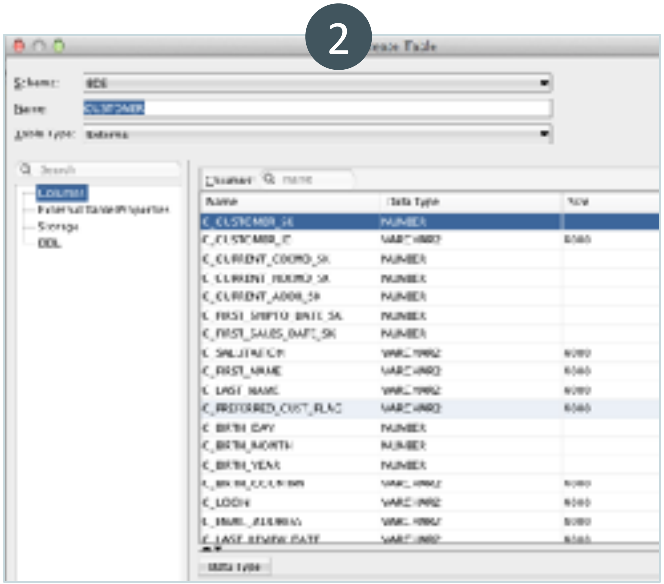
CLUSTER_ID	DATABASE_NAME	TABLE_NAME	COLUMN_NAME	HIVE_COLUMN_TYPE	ORACLE_COLUMN_TYPE	LOCAT
orabig	csv	customer	c_customer_sk	bigint	NUMBER	hdfs://h
orabig	csv	customer	c_customer_id	string	VARCHAR2(4000)	hdfs://h
orabig	csv	customer	c_current_cdemo_sk	bigint	NUMBER	hdfs://h
orabig	csv	customer	c_current_hdemo_sk	bigint	NUMBER	hdfs://h
orabig	csv	customer	c_current_addr_sk	bigint	NUMBER	hdfs://h
orabig	csv	customer	c_first_shipto_date_sk	bigint	NUMBER	hdfs://h
orabig	csv	customer	c_first_sales_date_sk	bigint	NUMBER	hdfs://h
orabig	csv	customer	c_salutation	string	VARCHAR2(4000)	hdfs://h
orabig	csv	customer	c_first_name	string	VARCHAR2(4000)	hdfs://h
orabig	csv	customer	c_last_name	string	VARCHAR2(4000)	hdfs://h
orabig	csv	customer	c_preferred_cust_flag	string	VARCHAR2(4000)	hdfs://h
orabig	csv	customer	c_birth_day	int	NUMBER	hdfs://h
orabig	csv	customer	c_birth_month	int	NUMBER	hdfs://h
orabig	csv	customer	c_birth_year	int	NUMBER	hdfs://h
orabig	csv	customer	c_birth_country	string	VARCHAR2(4000)	hdfs://h
orabig	csv	customer	c_login	string	VARCHAR2(4000)	hdfs://h
orabig	csv	customer	c_email_address	string	VARCHAR2(4000)	hdfs://h
orabig	csv	customer	c_last_review_date	string	VARCHAR2(4000)	hdfs://h
orabig	csv	customer_address	ca_address_sk	bigint	NUMBER	hdfs://h
orabig	csv	customer_address	ca_address_id	string	VARCHAR2(4000)	hdfs://h

Creating Tables

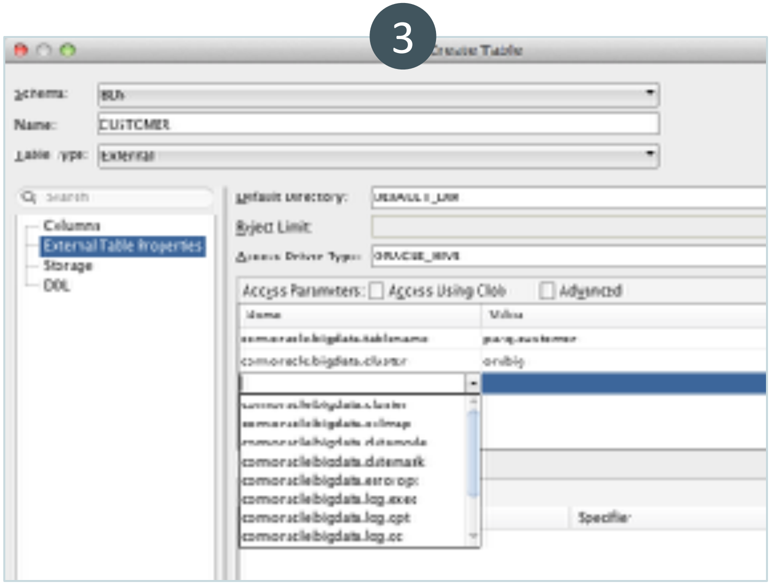
SQL Developer with Hive JDBC



Right-click on Hive Table. **Use in Oracle Big Data SQL**



Review generated columns. Update as needed - focusing on data types and precision



Add optional access parameters. Automatically generate table or save DDL.

See: https://blogs.oracle.com/datawarehousing/entry/oracle_sql_developer_data_modeler

Creating Tables

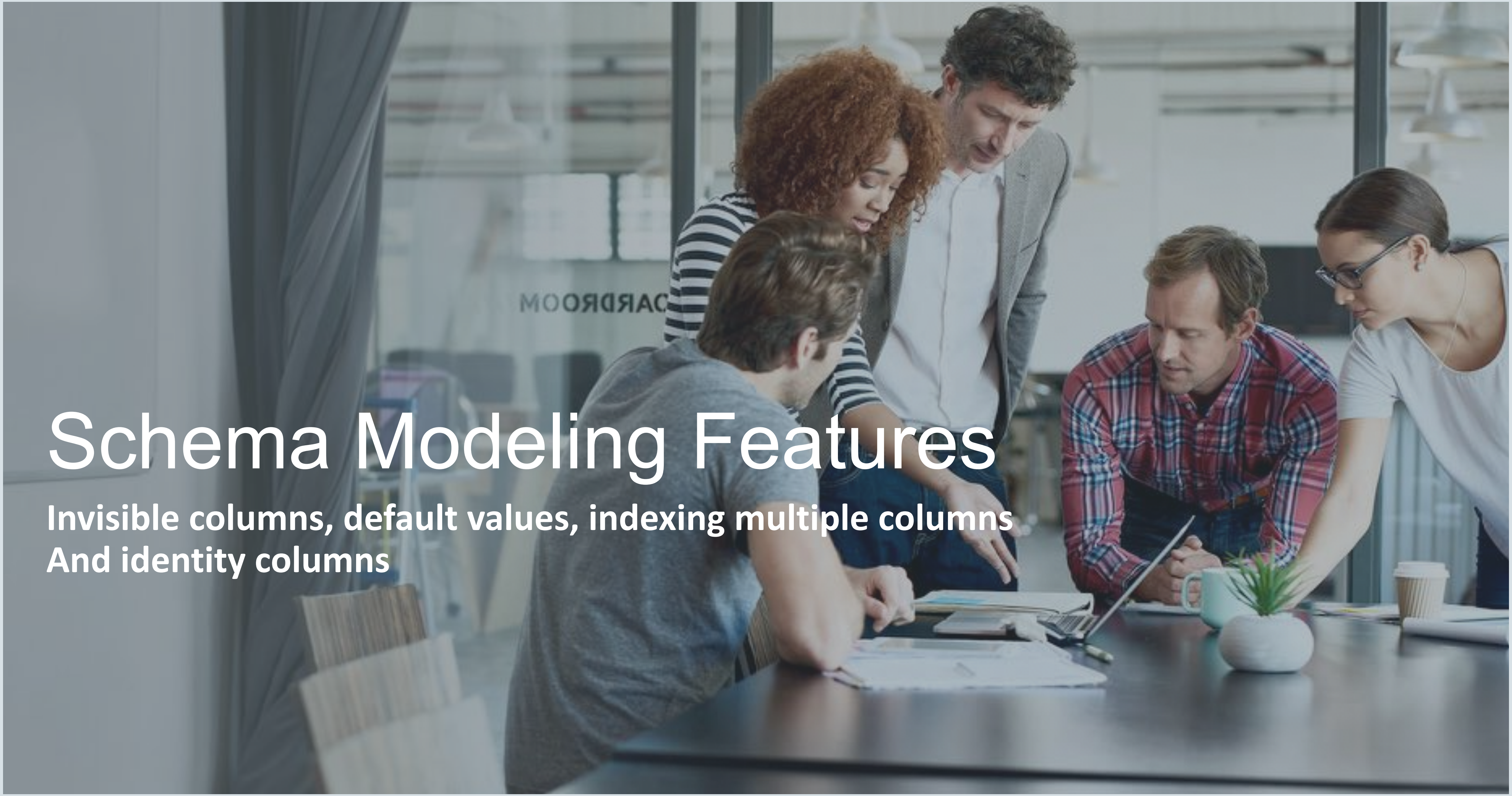
DBMS_HADOOP Package

```
declare
  DDLout VARCHAR2(4000);
begin
  DDLout := null;

  dbms_hadoop.create_extddl_for_hive (
        CLUSTER_ID=> 'orabig',
        DB_NAME=> 'parq',
        HIVE_TABLE_NAME=> 'store_sales',
        HIVE_PARTITION=>FALSE,
        TABLE_NAME=> 'store_sales_orcl',
        PERFORM_DDL=>FALSE,
        TEXT_OF_DDL=>DDLout
      );

  dbms_output.put_line(DDLout);
end;
/
```

- PL/SQL Package used to create table or generate DDL
- Combine with ALL_HIVE* dictionary views to automate creation of many tables
- Consider optimizing data type conversions - especially precision
  - string -> varchar2(?)



Schema Modeling Features

Invisible columns, default values, indexing multiple columns
And identity columns

Overview of Schema Modeling Enhancements

- Invisible Columns
- DEFAULT VALUE enhancements
  - Metadata-Only Default column values for NULL'able columns
  - Default values for columns on explicit NULL insertion
  - Default values for columns based on sequences
- Multiple Indexes on the same columns
- IDENTITY columns

Invisible Columns

Examples

- Create a simple table with invisible column

```
CREATE TABLE hr.emp
(empno NUMBER(5), name VARCHAR2(30) not null,
 status VARCHAR2(10) INVISIBLE)
TABLESPACE admin_tbs STORAGE ( INITIAL 50K));
```

- Modify to make the status column visible:

```
ALTER TABLE hr.admin_emp  MODIFY(status VISIBLE);
```

Invisible Columns – Usage in Views

- Invisible columns at the view level is supported.
- View Columns will be visible unless explicitly over-ridden by the ‘ invisible’ syntax – irrespective of the visibility of the table column.
- Invisible columns at the edition-ing view level is supported.
- Examples:
  - ```
    CREATE OR REPLACE VIEW emp (empno, ename,
                                status invisible)
    AS SELECT empno, ename, status FROM emp;
    ```
    ```

DEFAULT VALUE Enhancements

Metadata-only DEFAULT Column Values For NULL’able Columns

- New in Oracle Database 12c:
- Current Scenario when adding a NULL’able column with a default value
- Adds column to metadata
- Run as serial recursive SQL to populate existing rows with default value.
- Holds an Exclusive DML and KGL lock during the operation
- Make the entire DDL a metadata only operation

DEFAULT VALUE Enhancements

Column Defaulting for specific NULL insertion

- Allow SQL column defaulting when user specifies a NULL value on a NOT NULL column in an insert statement
- Example:

```
CREATE TABLE  test(a1 number DEFAULT ON NULL 10 NOT NULL, a2 varchar2(10);
INSERT INTO TEST (a1, a2) VALUES(NULL, ‘abc’);
SELECT a1, a2 FROM test;
       a1          a2
   --------        -------
          10         abc
```

DEFAULT VALUE Enhancements

Column Defaulting Using A Sequence

- Allow sequence [CURRVAL|NEXTVAL] to be used in SQL default expression
- Example

```
CREATE SEQUENCE s1 START WITH 1;

CREATE TABLE test (a1 number DEFAULT S1.NEXTVAL, a2 varchar2(10));

INSERT INTO test (a2) VALUES (‘abc’);

INSERT INTO test (a2) VALUES ( ‘xyz’);

SELECT * FROM test;
              C1               C2
      ------   ------------
              1           abc
              2           xyz
```

Examples of Multiple Indexes On Same Set Of Columns

- Create table and index

```
CREATE TABLE test(c1 int, c2 int);
CREATE INDEX test_idx ON test (c1,c2);
```

- Create bitmap index on the same set of columns as TEST_IDX:

```
CREATE BITMAP INDEX test_idx2 ON test(c1, c2) INVISIBLE;
```

- “Activate” new index

```
ALTER INDEX test_idx INVISIBLE;
ALTER INDEX test_idx2 VISIBLE;
```

Multiple Indexes On Same Set Of Columns

Usage Constraints

- Only one visible index on the same set of columns at any point of time
- To create a visible index , existing indexes on the same set of columns need to be invisible
- Alter index visible will only be allowed if all other indexes on the same set of columns are invisible

Identity Columns

Concept

- Identity columns enable a simple way of creating a unique identifier as part of a schema model
  - Part of ANSI Standard
- Identity Columns will default a monotone increasing integer on insert DML from a sequence generator, whose options are specified by the identity syntax
  - Note: uniqueness is not enforced as part of the IDENTITY definition

Example of Identity Columns

- Create simple table with identity column
  - Generated by default, start with 100:

```
CREATE TABLE test(C1 number GENERATED AS IDENTITY
                (START WITH 100));
```

- Add identity column, increment by 10
  - Existing rows will be updated with a value from sequence generator, but order is not deterministic

```
ALTER TABLE test ADD( C1 number GENERATED AS IDENTITY
                (INCREMENT BY 10));
```

Identity Columns

Example, cont.

- Create simple table with default identity column at NULL insertion

```
CREATE TABLE test (C1 number GENERATED BY DEFAULT ON NULL AS IDENTITY,
                   C2 varchar2(10));
```

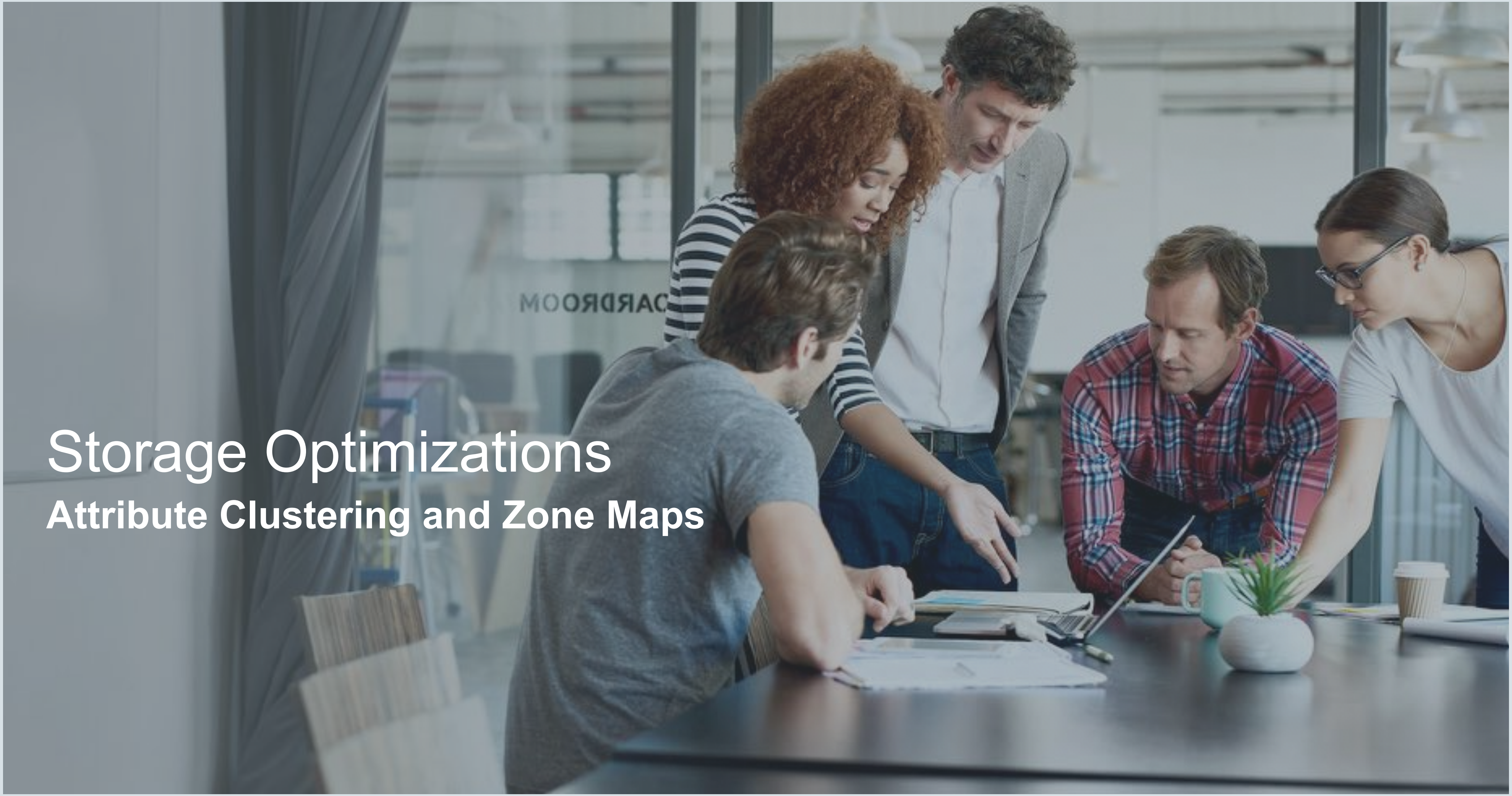
- Identity column generated by default starts with 1

```
INSERT INTO test(C2) VALUES (‘abc’);
INSERT INTO test(C1,C2) VALUES (null, ‘xyz’);


SELECT c1, c2 FROM test;
C1      C2
------   ------------
1  abc
2  xyz
```

Storage Optimizations

Attribute Clustering and Zone Maps



Attribute Clustering

Concepts and Benefits

- Orders data so that it is in close proximity based on selected columns values: “attributes”
- Attributes can be from a single table or multiple tables
  - e.g. from fact and dimension tables
- Significant IO pruning when used with zone maps
- Reduced block IO for table lookups in index range scans
- Queries that sort and aggregate can benefit from pre-ordered data
- Enable improved compression ratios
  - Ordered data is likely to compress more than unordered data

Attribute Clustering for Zone Maps

Ordered rows

```
ALTER TABLE sales
ADD CLUSTERING BY
LINEAR ORDER (category);


ALTER TABLE sales MOVE;
```

Category	Country
BOYS	AR
BOYS	JP
BOYS	SA
BOYS	US
GIRLS	AR
GIRLS	JP
GIRLS	SA
GIRLS	US
MEN	AR
MEN	JP
MEN	SA
MEN	US
WOMEN	AR
WOMEN	JP
WOMEN	SA
WOMEN	US

Ordered rows containing category values BOYS, GIRLS and MEN.

Zone maps catalogue regions of rows, or *zones*, that contain particular column value ranges.

By default, each zone is up to 1024 blocks.

For example, we only need to scan this zone if we are searching for category "GIRLS". We can skip all other zones.

Attribute Clustering

Basics

- Two types of attribute clustering
  - LINEAR ORDER BY
    - Classical ordering
  - INTERLEAVED ORDER BY
    - Multi-dimensional ordering
- Simple attribute clustering on a single table
- Join attribute clustering
  - Cluster on attributes derived through join of multiple tables
    - Up to four tables
    - Non-duplicating join (PK or UK on joined table is required)

Attribute Clustering With Zone Maps

Example

- CLUSTERING BY LINEAR ORDER (category, country)

- Zone map benefits are most significant with ordered data

Category	Country
BOYS	AR
BOYS	JP
BOYS	SA
BOYS	US
GIRLS	AR
GIRLS	JP
GIRLS	SA
GIRLS	US
MEN	AR
MEN	JP
MEN	SA
MEN	US
WOMEN	AR
WOMEN	JP
WOMEN	SA
WOMEN	US

LINEAR ORDER

Pruning with:

```
SELECT   ..
FROM     table
WHERE    category = 'BOYS';
```

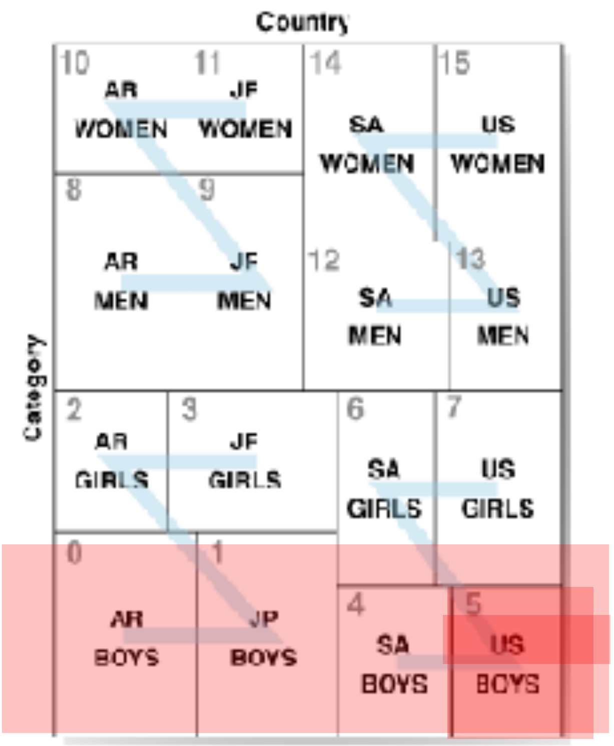
```
SELECT   ..
FROM     table
WHERE    category = 'BOYS';
AND      country  = 'US'
```

Attribute Clustering With Zone Maps

Example

- CLUSTERING BY INTERLEAVED ORDER (category, country)

- Zone map benefits are most significant with ordered data



INTERLEAVED ORDER

Pruning with:

```
SELECT   ..
FROM     table
WHERE    category = 'BOYS';
```

```
SELECT   ..
FROM     table
WHERE    country  = 'US'
```

```
SELECT   ..
FROM     table
WHERE    category = 'BOYS';
AND      country  = 'US'
```

Basics of Zone Maps

- Independent access structure built for a table
  - Implemented using a type of materialized view
  - For partitioned and non-partitioned tables
- One zone map per table
  - Zone map on partitioned table includes aggregate entry per [sub]partition
- Used transparently
  - No need to change or hint queries
- Implicit or explicit creation and column selection
  - Through Attribute Clustering: CREATE TABLE … CLUSTERING
  - CREATE MATERIALIZED ZONEMAP … AS SELECT …

Zone Maps

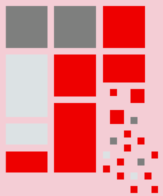
Staleness

- DML and partition operations can cause zone maps to become fully or partially stale
  - Direct path insert does not make zone maps stale
- Single table 'local' zone maps
  - Update and insert marks impacted zones as stale (and any aggregated partition entry)
  - No impact on zone maps for delete
- Joined zone map
  - DML on fact table equivalent behavior to single table zone map
  - DML on dimension table makes dependent zone maps fully stale

Refreshing Zone Maps

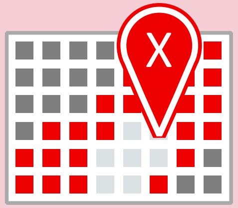
- Incremental and full refresh, as required by DML
  – Zone map refresh does require a materialized view log
    - Only stale zones are scanned to refresh the MV
  – For joined zone map
    - DML on fact table: incremental refresh
    - DML on dimension table: full refresh
- Zone map maintenance through
  – DBMS_MVIEW.REFRESH()
  – ALTER MATERIALIZED ZONEMAP <xx> REBUILD;

Zone Maps with Attribute Clustering



Attribute Clustering

Orders data so that columns values are stored together on disk



Zone maps

Stores min/max of specified columns per zone

Used to filter un-needed data during query execution

- Combined Benefits
- Improved query performance and concurrency
  - Reduced physical data access
  - Significant IO reduction for highly selective operations
- Optimized space utilization
  - Less need for indexes
  - Improved compression ratios through data clustering
- Full application transparency
  - Any application will benefit

Attribute Clustering with In-Memory Column Store

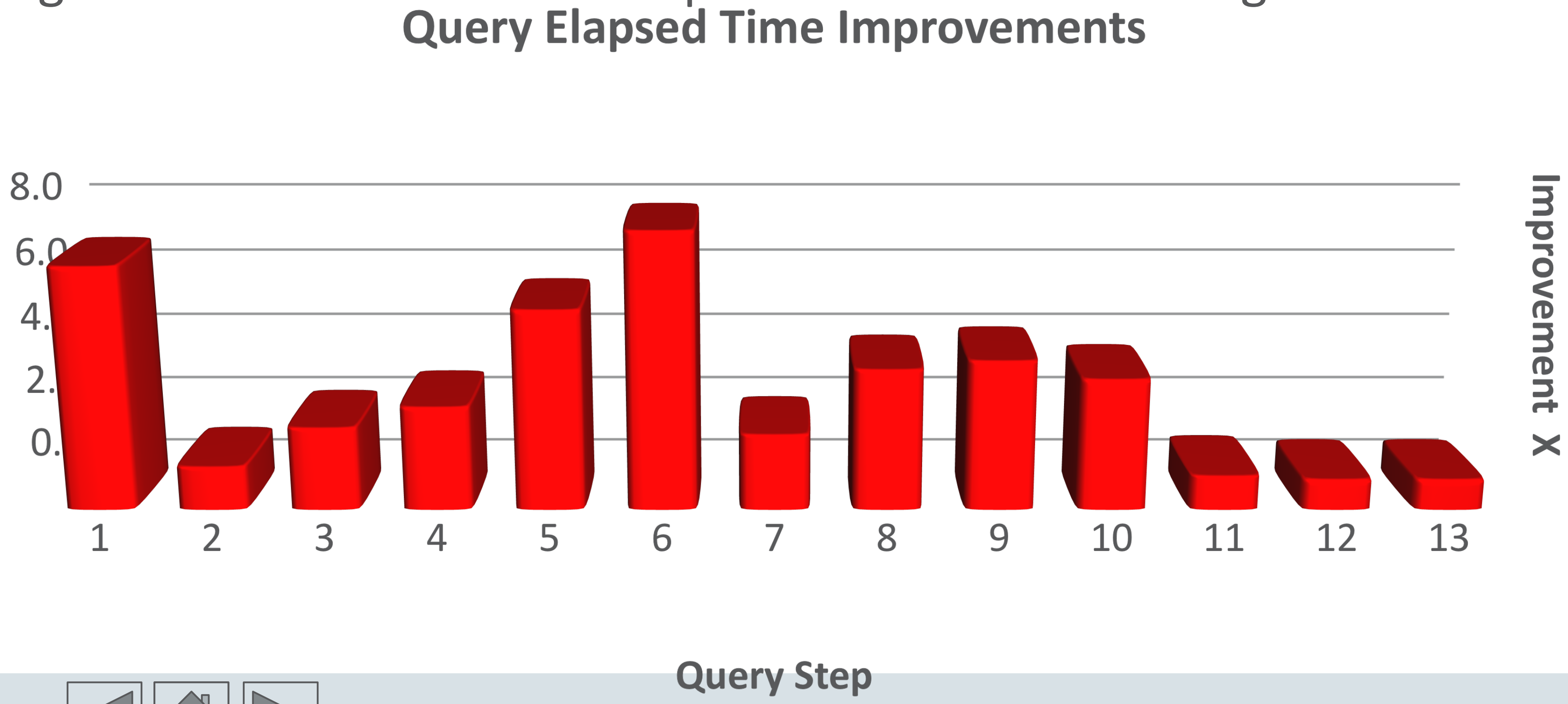
Snowflake Schema Benchmark

- Attribute clusters alone - no zone maps
- With attribute clustering versus without (baseline)
- Warehousing benchmark run on snowflake-schema
- In-Memory Column Store
- Result with attribute clustering:
  - Overall, 1.4X response time improvement over baseline
  - Improved sort and aggregation performance
    - Pre-ordered rows can require less sorting

Zone Maps With Attribute Clustering

Star Schema Benchmark

- Overall, 2.6X end-to-end elapsed time improvement
  - Comparing with and without zone map and attribute clustering



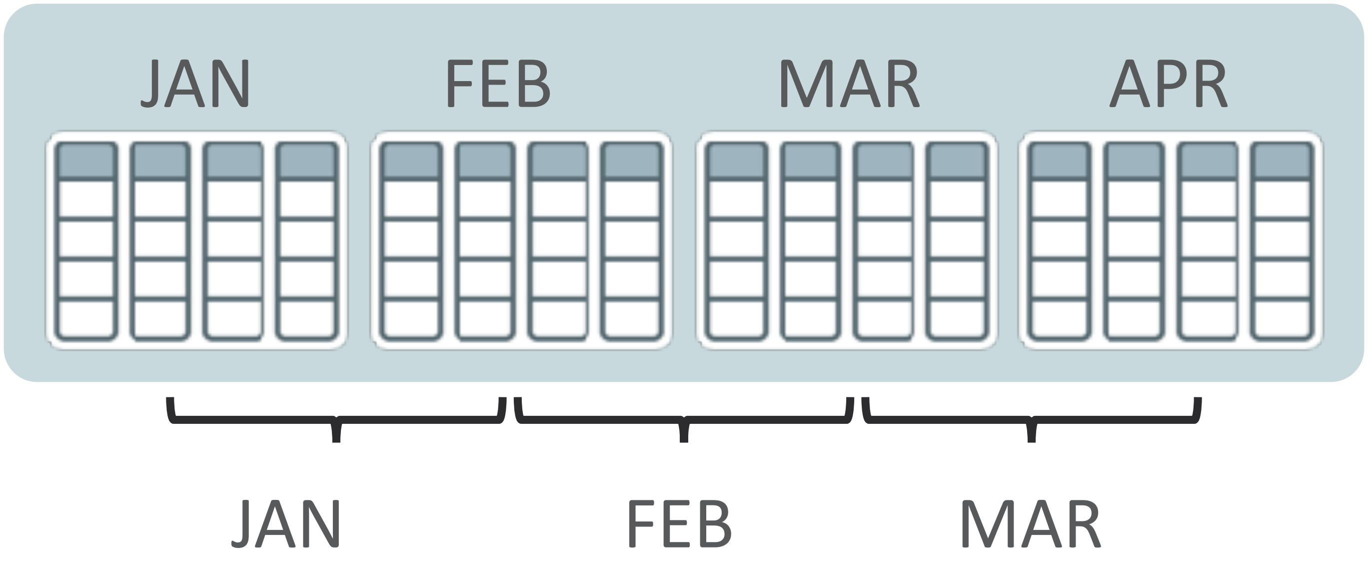
Zone Maps and Partitioning

- Zone maps can prune partitions for columns that are not included in the partition (or subpartition) key

Partition Key:
ORDER_DATE

SALES

Zone map:
SHIP_DATE



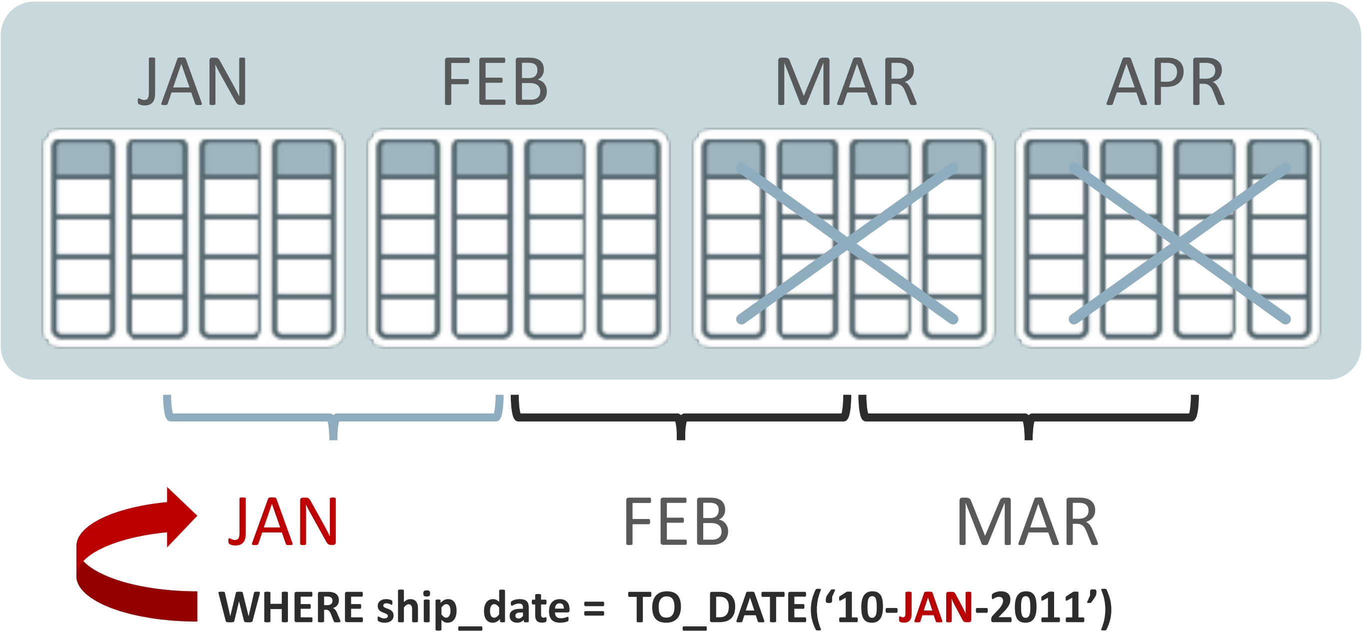
Zone map column SHIP_DATE correlates with partition key ORDER_DATE

Zone Maps and Partitioning

- Zone maps can prune partitions for columns that are not included in the partition (or subpartition) key

Partition Key:
ORDER_DATE

SALES



MAR and APR partitions are pruned

Zone map:
SHIP_DATE

WHERE ship_date = TO_DATE('10-JAN-2011')

Zone Maps and Storage Indexes

- Attribute clustering and zone maps work transparently with Exadata storage indexes
  - The benefits of Exadata storage indexes continue to be fully exploited
- In addition, zone maps (when used with attribute clustering)
  - Enable additional and significant IO optimization
    - Provide an alternative to indexes, especially on large tables
    - Join and fact-dimension queries, including dimension hierarchy searches
    - Particularly relevant in star and snowflake schemas
  - Are able to prune entire partitions and sub-partitions
  - Are effective for both direct and conventional path reads
  - Include optimizations for joins and index range scans
  - Part of the physical database design: explicitly created and controlled by the DBA

Summary

- Making I/O elimination techniques even more effective
- Attribute clustering is used to store related data in close proximity
  - Ensures that similar data falls within the same zone
- Zone maps provide I/O reduction for single tables, table joins and dimensional hierarchies

Top-N Filtering

Native Support for TOP-N Queries

New offset and fetch FIRST clause

- ANSI 2008/2011 compliant with some additional extensions
- Specify offset and number or percentage of rows to return
- Provisions to return additional rows with the same sort key as the last row (WITH TIES option)

Native Support for TOP-N Queries

Internal processing

Find 5 percent of employees with the lowest salaries

```
SELECT employee_id, last_name, salary
FROM employees
ORDER BY salary
FETCH FIRST 5 percent ROWS ONLY;
```

Native Support for TOP-N Queries

Internal processing, cont.

Find 5 percent of employees with the lowest salaries

```
SELECT employee_id, last_name, salary
FROM employee
ORDER BY sala
FETCH FIRST 5
```

- Internally the query is transformed into an equivalent query using window functions

```
SELECT employee_id, last_name, salary
FROM (SELECT employee_id, last_name, salary,
             row_number() over (order by salary) rn,
             count(*) over () total
      FROM employee)
WHERE rn <= CEIL(total * 5/100);
```

- Additional Top-N Optimization:
  - SELECT list may include expensive PL/SQL function or costly expressions
  - Evaluation of SELECT list expression limited to rows in the final result set

SQL for Advanced Analysis

Pattern matching with MATCH_RECOGNIZE

Pattern Recognition In Sequences of Rows

- Recognize patterns in sequences of events using SQL
  - Sequence is a stream of rows
  - Event equals a row in a stream
- New SQL construct MATCH_RECOGNIZE
  - Logically partition and order the data
    - ORDER BY and PARTITION BY are optional – *but be careful*
  - Pattern defined using regular expression using variables
  - Regular expression is matched against a sequence of rows
  - Each pattern variable is defined using conditions on rows and aggregates

Business Problem: Finding Suspicious Money Transfers

- Suspicious money transfer pattern for an account is:
  - 3 or more small (<2K) money transfers within 30 days
  - Large transfer (>=1M) within 10 days of last small transfer

- Report account, date of first small transfer, date of last large transfer

Finding Suspicious Money Transfers

Data Set

TIME	USER ID	EVENT	AMOUNT
1/1/2012	John	Deposit	1,000,000
1/2/2012	John	Transfer	1,000
1/5/2012	John	Withdrawal	2,000
1/10/2012	John	Transfer	1,500
1/20/2012	John	Transfer	1,200
1/25/2012	John	Deposit	1,200,000
1/27/2012	John	Transfer	1,000,000
2/2/20212	John	Deposit	500,000

Finding Suspicious Money Transfers

TIME	USER ID	EVENT	AMOUNT
1/1/2012	John	Deposit	1,000,000
1/2/2012	John	Transfer	1,000
1/5/2012	John	Withdrawal	2,000
1/10/2012	John	Transfer	1,500
1/20/2012	John	Transfer	1,200
1/25/2012	John	Deposit	1,200,000
1/27/2012	John	Transfer	1,000,000
2/2/20212	John	Deposit	500,000

Three small transfers within 30 days

Finding Suspicious Money Transfers

TIME	USER ID	EVENT	AMOUNT
1/1/2012	John	Deposit	1,000,000
1/2/2012	John	Transfer	1,000
1/5/2012	John	Withdrawal	2,000
1/10/2012	John	Transfer	1,500
1/20/2012	John	Transfer	1,200
1/25/2012	John	Deposit	1,200,000
1/27/2012	John	Transfer	1,000,000
2/2/20212	John	Deposit	500,000

Three small transfers within 30 days

Large transfer within 10 days of last small transfer

SQL Pattern Matching in action

New syntax for discovering patterns using SQL: **finding suspicious money transfers**

MATCH_RECOGNIZE ( )

TIME	USER ID	EVENT	AMOUNT
1/1/2012	John	Deposit	1,000,000
1/2/2012	John	Transfer	1,000
1/5/2012	John	Withdrawal	2,000
1/10/2012	John	Transfer	1,500
1/20/2012	John	Transfer	1,200
1/25/2012	John	Deposit	1,200,000
1/27/2012	John	Transfer	1,000,000
2/2/20212	John	Deposit	500,000

```
SELECT . . .
FROM (SELECT * FROM event_log WHERE event = 'transfer')
MATCH_RECOGNIZE (
  . . .









 )
```

Define the how data is to be processed

STEP 1

Set the **PARTITION BY** and **ORDER BY** clauses

TIME	USER ID	EVENT	AMOUNT
1/1/2012	John	Deposit	1,000,000
1/2/2012	John	Transfer	1,000
1/5/2012	John	Withdrawal	2,000
1/10/2012	John	Transfer	1,500
1/20/2012	John	Transfer	1,200
1/25/2012	John	Deposit	1,200,000
1/27/2012	John	Transfer	1,000,000
2/2/20212	John	Deposit	500,000

```
SELECT . . .
FROM (SELECT * FROM event_log WHERE event = 'transfer')
MATCH_RECOGNIZE (
   PARTITION BY userid ORDER BY time







  )
```

Define PATTERN clause

TIME	USER ID	EVENT	AMOUNT
1/1/2012	John	Deposit	1,000,000
1/2/2012	John	Transfer	1,000
1/5/2012	John	Withdrawal	2,000
1/10/2012	John	Transfer	1,500
1/20/2012	John	Transfer	1,200
1/25/2012	John	Deposit	1,200,000
1/27/2012	John	Transfer	1,000,000
2/2/20212	John	Deposit	500,000

STEP 2

Define the **PATTERN** –

Three or more small amount (<2K) money transfers within 30 days

```
SELECT . . .
FROM (SELECT * FROM event_log WHERE event = 'transfer')
MATCH_RECOGNIZE (
 PARTITION BY userid ORDER BY time



 PATTERN ( X{3,}  )



 )
```

Define PATTERN clause

STEP 2 Define the **PATTERN** variables:

Large transfer (>=1M) within 10 days of last small transfer

TIME	USER ID	EVENT	AMOUNT
1/1/2012	John	Deposit	1,000,000
1/2/2012	John	Transfer	1,000
1/5/2012	John	Withdrawal	2,000
1/10/2012	John	Transfer	1,500
1/20/2012	John	Transfer	1,200
1/25/2012	John	Deposit	1,200,000
1/27/2012	John	Transfer	1,000,000
2/2/20212	John	Deposit	500,000

```
SELECT . . .
FROM (SELECT * FROM event_log WHERE event = 'transfer')
MATCH_RECOGNIZE (
 PARTITION BY userid ORDER BY time



 PATTERN ( X{3,} Y)



 )
```

Define PATTERN clause

STEP 2 Define the **PATTERN** variables:

Describe the details of each pattern – small amount is less than 2K and within 30 days

TIME	USER ID	EVENT	AMOUNT
1/1/2012	John	Deposit	1,000,000
1/2/2012	John	Transfer	1,000
1/5/2012	John	Withdrawal	2,000
1/10/2012	John	Transfer	1,500
1/20/2012	John	Transfer	1,200
1/25/2012	John	Deposit	1,200,000
1/27/2012	John	Transfer	1,000,000
2/2/20212	John	Deposit	500,000

```
SELECT . . .
FROM (SELECT * FROM event_log WHERE event = 'transfer')
MATCH_RECOGNIZE (
 PARTITION BY userid ORDER BY time



 PATTERN ( X{3,} Y)
  DEFINE
   X as (amount < 2000) AND
     LAST(X.time) - FIRST(X.time) < 30,

 )
```

Define PATTERN clause

STEP 2 Define the **PATTERN** variables:

Describe the details of each pattern – large amount is more than 1M

TIME	USER ID	EVENT	AMOUNT
1/1/2012	John	Deposit	1,000,000
1/2/2012	John	Transfer	1,000
1/5/2012	John	Withdrawal	2,000
1/10/2012	John	Transfer	1,500
1/20/2012	John	Transfer	1,200
1/25/2012	John	Deposit	1,200,000
1/27/2012	John	Transfer	1,000,000
2/2/20212	John	Deposit	500,000

```
SELECT . . .
FROM (SELECT * FROM event_log WHERE event = 'transfer')
MATCH_RECOGNIZE (
 PARTITION BY userid ORDER BY time




 PATTERN ( X{3,} Y)
  DEFINE
   X as (amount < 2000) AND
     LAST(X.time) - FIRST(X.time) < 30,
   Y as (amount >= 1000000
 )
```

Define PATTERN clause

TIME	USER ID	EVENT	AMOUNT
1/1/2012	John	Deposit	1,000,000
1/2/2012	John	Transfer	1,000
1/5/2012	John	Withdrawal	2,000
1/10/2012	John	Transfer	1,500
1/20/2012	John	Transfer	1,200
1/25/2012	John	Deposit	1,200,000
1/27/2012	John	Transfer	1,000,000
2/2/20212	John	Deposit	500,000

STEP 2 Define the **PATTERN** variables:

Large transfer within 10 days of last small transfer

```
SELECT . . .
FROM (SELECT * FROM event_log WHERE event = 'transfer')
MATCH_RECOGNIZE (
 PARTITION BY userid ORDER BY time



 PATTERN ( X{3,} Y)
  DEFINE
   X as (amount < 2000) AND
     LAST(X.time) - FIRST(X.time) < 30,
   Y as (amount >= 1000000 AND
     Y.time - LAST(X.time) < 10 ))
```

Define Measures To Be Calculated

STEP 3 Define the **MEASURES**:

Report account, date of first small transfer, date of last large transfer

TIME	USER ID	EVENT	AMOUNT
1/1/2012	John	Deposit	1,000,000
1/2/2012	John	Transfer	1,000
1/5/2012	John	Withdrawal	2,000
1/10/2012	John	Transfer	1,500
1/20/2012	John	Transfer	1,200
1/25/2012	John	Deposit	1,200,000
1/27/2012	John	Transfer	1,000,000
2/2/20212	John	Deposit	500,000

```
SELECT . . .
FROM (SELECT * FROM event_log WHERE event = 'transfer')
MATCH_RECOGNIZE (
 PARTITION BY userid ORDER BY time
 MEASURES FIRST(x.time) first_t,
        y.time last_t,
         y.amount amount

  PATTERN ( X{3,} Y)
  DEFINE
   X as (amount < 2000) AND
     LAST(X.time) - FIRST(X.time) < 30,
   Y as (amount >= 1000000 AND
     Y.time - LAST(X.time) < 10 ))
```

Define How Much Data Is Returned

STEP 4 Control the output:

Output **one row** each time we find a match to our pattern

TIME	USER ID	EVENT	AMOUNT
1/1/2012	John	Deposit	1,000,000
1/2/2012	John	Transfer	1,000
1/5/2012	John	Withdrawal	2,000
1/10/2012	John	Transfer	1,500
1/20/2012	John	Transfer	1,200
1/25/2012	John	Deposit	1,200,000
1/27/2012	John	Transfer	1,000,000
2/2/20212	John	Deposit	500,000

```
SELECT . . .
FROM (SELECT * FROM event_log WHERE event = 'transfer')
MATCH_RECOGNIZE (
 PARTITION BY userid ORDER BY time
 MEASURES FIRST(x.time) first_t,
        y.time last_t,
         y.amount amount
  ONE ROW PER MATCH
 PATTERN ( X{3,} Y)
  DEFINE
   X as (amount < 2000) AND
     LAST(X.time) - FIRST(X.time) < 30,
   Y as (amount >= 1000000 AND
     Y.time - LAST(X.time) < 10 ))
```

Define output columns

TIME	USER ID	EVENT	AMOUNT
1/1/2012	John	Deposit	1,000,000
1/2/2012	John	Transfer	1,000
1/5/2012	John	Withdrawal	2,000
1/10/2012	John	Transfer	1,500
1/20/2012	John	Transfer	1,200
1/25/2012	John	Deposit	1,200,000
1/27/2012	John	Transfer	1,000,000
2/2/20212	John	Deposit	500,000

```
SELECT userid, first_t, last_t, amount
FROM (SELECT * FROM event_log WHERE event = 'transfer')
MATCH_RECOGNIZE (
 PARTITION BY userid ORDER BY time
 MEASURES FIRST(x.time) first_t,
       y.time last_t,
        y.amount amount
  ONE ROW PER MATCH
 PATTERN ( X{3,} Y)
  DEFINE
   X as (amount < 2000) AND
     LAST(X.time) - FIRST(X.time) < 30,
   Y as (amount >= 1000000 AND
     Y.time - LAST(X.time) < 10 ))
```

Finally list columns to return as part of the query result set…

Adding New Requirements

Using SQL makes it very easy to extend pattern for new requirements

- Additional requirement:
  - Check for transfers across different accounts
    - total sum of small transfers must be less than 20K

TIMESTAMP	USER ID	EVENT	TRANSFER_TO	AMOUNT
1/1/2012	John	Deposit	-	1,000,000
1/2/2012	John	Transfer	Bob	1,000
1/5/2012	John	Withdrawal	-	2,000
1/10/2012	John	Transfer	Allen	1,500
1/20/2012	John	Transfer	Tim	1,200
1/25/2012	John	Deposit		1,200,000
1/27/2012	John	Transfer	Tim	1,000,000
2/2/20212	John	Deposit	-	500,000

Adding New Requirements

Using SQL makes it very easy to extend pattern for new requirements

- Additional requirement:
  – Check for transfers across different accounts
    - total sum of small transfers must be less than 20K

TIMESTAMP	USER ID	EVENT	TRANSFER_TO	AMOUNT
1/1/2012	John	Deposit	-	1,000,000
1/2/2012	John	Transfer	Bob	1,000
1/5/2012	John	Withdrawal	-	2,000
1/10/2012	John	Transfer	Allen	1,500
1/20/2012	John	Transfer	Tim	1,200
1/25/2012	John	Deposit		1,200,000
1/27/2012	John	Transfer	Tim	1,000,000
2/2/20212	John	Deposit	-	500,000

Three small transfers within 30 days
to different acct and total sum < 20K

Large transfer within 10 days of last small transfer

Adding New Requirements

Modify the pattern variables
DEFINE

- Check the transfer account

```
SELECT userid, first_t, last_t, amount
FROM (SELECT * FROM event_log WHERE event = 'transfer')
MATCH_RECOGNIZE (
 PARTITION BY userid ORDER BY time
 MEASURES FIRST(x.time) first_t,
        y.time last_t,
         y.amount amount
  ONE ROW PER MATCH
 PATTERN (X{3,} Y)
  DEFINE
   X as (amount < 2000) AND
         LAST(X.time) - FIRST(X.time) < 30 AND
     PREV(X.transfer_to) <> X.transfer_to
```

Adding New Requirements

Modify the pattern variables **DEFINE**

- Check the total of the small transfers is less than 20K

```
SELECT userid, first_t, last_t, amount
FROM (SELECT * FROM event_log WHERE event = 'transfer')
MATCH_RECOGNIZE (
 PARTITION BY userid ORDER BY time
 MEASURES FIRST(x.time) first_t,
        y.time last_t,
         y.amount amount
  ONE ROW PER MATCH
 PATTERN (X{3,} Y)
  DEFINE
   X as (amount < 2000) AND
         LAST(X.time) - FIRST(X.time) < 30 AND
     PREV(X.transfer_to) <> X.transfer_to
   Y as (amount >= 1000000 AND
     y.time - LAST(X.time) < 10 AND
     SUM(X.amount) < 20,000 );
```

SQL for Advanced Analysis

Approximate count distinct

Exploring Today's Big Data Lakes

- **Key business challenges**
  - Many queries rely on counts and/or statistical calculations
    - NDVs, Pareto's 80:20 rule, identifying outliers etc.
  - Exact processing of large data sets is resource intensive
  - Exploratory queries don't require completely accurate result
    - Trending analysis, social analysis, sessionization analytics
- **Oracle's solutions**
  - Provide "approximate result" capabilities in SQL
  - **Key objectives**
    - Return approximate results faster, minimal deviation from actual
    - Use fewer resources, allowing more queries to run

Getting *Approximate* Counts

Answer "***How many***…" type questions

- How many unique sessions today
- How many unique customers logged on
- How many unique events occurred

COUNT (DISTINCT *expr*)

- returns the exact number of rows that contain distinct values of specified expression
- Can be resource intensive because requires sorting

APPROX_COUNT_DISTINCT(*expr*)

- processes large amounts of data significantly faster
- uses HyperLogLog algorithm
- negligible deviation from exact result
  - ignores rows containing null values
- supports any scalar data type
  - Does not support BFILE, BLOB, CLOB, LONG, LONG RAW, or NCLOB

… significantly faster solution

Performance and Accuracy of APPROX_COUNT_DISTINCT

Performance Results

- Real world customer workload
- **5-50x** improvement

Notes:

this approach does not use sampling, it uses a hash-based approach ignores rows that contain a null value for specified expression

Supports any scalar data type other than BFILE, BLOB, CLOB, LONG, LONG RAW, or NCLOB

Results for accuracy

- Real world customer workload
- Accuracy that is typically **97%** with **95% confidence**

COUNT(DISTINCT) Processing

Operation	Name	Li...	Estimated...	Cost	Timeline(3059s)	Exec...	Actual R...	Memory...	Temp (...	O..	IO Req...	IO ...
SELECT STATEMENT		0				1	1					
SORT GROUP BY		1	1			1	1					
PX COORDINATOR		2				161	454					
PX SEND QC (RANDOM)	:TQ10001	3	1			80	454					
SORT GROUP BY		4	1			80	[illegible]	8GB	164GB		1,418K	328GB
PX RECEIVE		5	1			80	48G					
PX SEND HASH	:TQ10000	6	1			80	48G					
SORT GROUP BY		7	1			80	48G	8GB				
PX PARTITION HASH ALL		8	6,000M	15K		80	6,000M					
TABLE ACCESS STORAGE F...	LINEITEM	9	6,000M	15K		13K	[illegible]	541MB			589K	192GB

1. Query is processing all 6,00M rows in table LINEITEM
2. Table access consumes 541MB memory
3. Sort operation to manage count + distinct operations
4. Distinct + Count processing consumes **8GB** of memory and **164GB** of temp

Benefits of APPROX_COUNT_DISTINCT processing

Operation	Name	Li...	Estimated...	Cost	Timeline(69s)	Exec...	Actual ...	Memory...	Temp (...	O..	IO Req...	IO ...
SELECT STATEMENT		0				1	1					
SORT AGGREGATE APPROX		1	1			1	1					
PX COORDINATOR		2				81	80					
PX SEND QC (RANDOM)	:TQ10000	3	1			80	80					
SORT AGGREGATE APPROX		4	1			80	80					
PX PARTITION HASH ALL		5	6,000M	16K		80	6,000M					
TABLE ACCESS STORAGE FULL	LINEITEM	6	6,000M	16K		13K	...0M	542MB			588K	192G...

1. Query is processing all 6,00M rows in table LINEITEM
2. Table access consumes 542MB memory
3. Only sort operation is now AGGREGATE APPROX
4. Approximate processing ONLY consumes **524MB** of memory and **zero GB** of temp

Benefits of APPROX_COUNT_DISTINCT: 50X Faster

Operation	Name	Li...	Estimated...		Timeline(3059s)	Exec...	Actual R...	Memory...	Temp (...	O..	IO Req...	IO ...
SELECT STATEMENT		0				1	1					
SORT GROUP BY		1	1			1	1					
PX COORDINATOR		2				161	454					
PX SEND QC (RANDOM)	:TQ10001	3	1			80	454					

Operation	Name	Li...	Estimated...		Timeline(69s)	Exec...	Actual ...	Memory...	Temp (...	O..	IO Req...	IO ...
SELECT STATEMENT		0				1	1					
SORT AGGREGATE APPROX		1	1			1	1					
PX COORDINATOR		2				81	80					
PX SEND QC (RANDOM)	:TQ10000	3	1			80	80					

1. COUNT(DISTINCT…) timeline 3,059 seconds on 6,000M rows
2. APPROX_COUNT_DISTINCT indicator in explain plan
3. APPROX_COUNT_DISTINCT timeline 69 seconds on 6,000M rows

50X FASTER

Using Statistical Analytics For Intelligent Analysis

NEW IN 12.2

- **Key business requirements**
  - Searching for outliers within a data set
  - Pareto (80/20) analysis
  - Data points 3 SDs from mean
    - Data outside 3 SDs is often considered an anomaly
- Typical use cases include
  - Quality monitoring and assurance
  - Monitoring SLA performance
  - Anomaly/outlier detection
  - Tracking activity/visibility on social media sites

Materialized Views

Overview of Materialized Views in Oracle Database 12c

- Objectives
  - Improve performance of refresh operation
  - Minimize staleness time of materialized views
- Two fundamental new concepts for refresh
  - Out-of-place refresh
    - Refresh “shadow MV” and swap with original MV after refresh
  - Synchronous refresh
    - Refresh base tables and MVs synchronously, leveraging equi-partitioning of the objects

Materialized Views: In-Place vs. Out-of-Place Refresh

In-place refresh

- Apply refresh statement to MV directly
- MV remains unusable during execution of refresh statement
- Potential suboptimal processing
  - Conventional DMLs don't scale well
  - Truncate and direct path load only used in limited cases
- MV becomes fragmented after certain numbers of refreshes

Out-of-place refresh

- Create outside table(s)
  - Populate outside table(s)
  - Switch outside table(s) to become new MV or MV partition
- High MV availability
- Efficiency due to direct load
- Addresses fragmentation problem

Overview of Synchronous Refresh

- Materialized View and base-tables refreshed together
  - Materialized View and base-tables always "in sync"
  - Materialized Views always fresh
- Improved availability of Materialized View for rewrite
- Materialized View and fact tables must be equi-partitioned
  - Partition key of fact table must functionally determine partition-key of MV
- Synchronous Refresh uses partition exchange of changed fact table and Materialized View partitions

Data Bound Collations

Enhancements to tables/views to support searching multilingual text strings

Data-Bound Collations

"… a named set of rules describing how to compare and match character strings to put them in a specified order…"

- Based on the ISO/IEC/ANSI SQL standard 9075:1999
- Character set is always declared at the database level
- Collation declared for a **column**
  - Does not determine the character set of data in the column

- Why is it important?
  - it simplifies application migration to the Oracle Database from a number of non-Oracle databases implementing collation in a similar way

Data-Bound Collations

- Oracle supports around 100 linguistic collations
  - Parameterized by adding the suffix _CI or the suffix _AI
    - _CI - Specifies a case-insensitive sort
    - _AI - Specifies an accent-insensitive sort

```
CREATE TABLE products
( product_code        VARCHAR2(20 BYTE)   COLLATE BINARY
, product_name        VARCHAR2(100 BYTE)  COLLATE GENERIC_M_CI
, product_category    VARCHAR2(5 BYTE)    COLLATE BINARY
, product_description VARCHAR2(1000 BYTE) COLLATE BINARY_CI
);
```

- *Product_name is to be compared using GENERIC_M_CI - case-insensitive version of generic multilingual collation*

#oow17

Get the most from #oow17 - Must-See DW and Big Data Sessions and Hands-on Labs

Big Data Warehousing Review of #oow17

Download our complete review of all the key Big DW sessions, presenters, keynotes and links to social media sites etc:

https://oracle-big-data.blogspot.co.uk/2017/10/review-of-big-data-warehousing-at.html

Safe Harbor Statement

The preceding is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.