Oracle
Data Warehouse

www.lingaro.com

lingaro
The Value of Business Intelligence

★★★
★★
★

- Course contains Lingaro Standard rules
  - Rule is indicated by blue stars
  - More stars means more important rule

Training Agenda

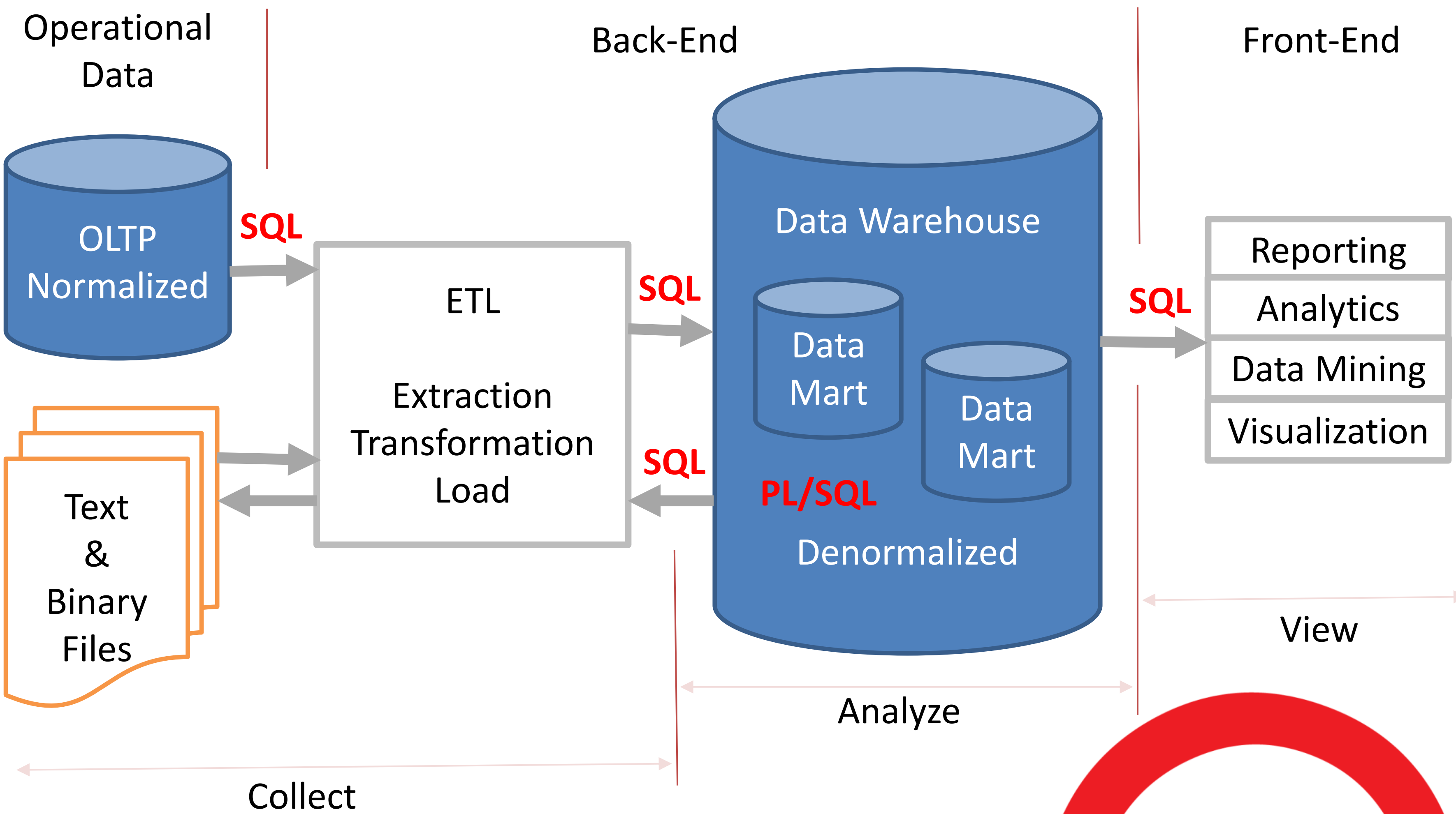
- Warehouse Introduction
- Data Model
- Constraints & Indexes
- Loading & Transforming Data
- Data Compression
- Materialized Views
  - Oracle Dimensions
- Partitioning
- SQL Parallel Execution & RAC
- Warehouse SQL

Topic Agenda

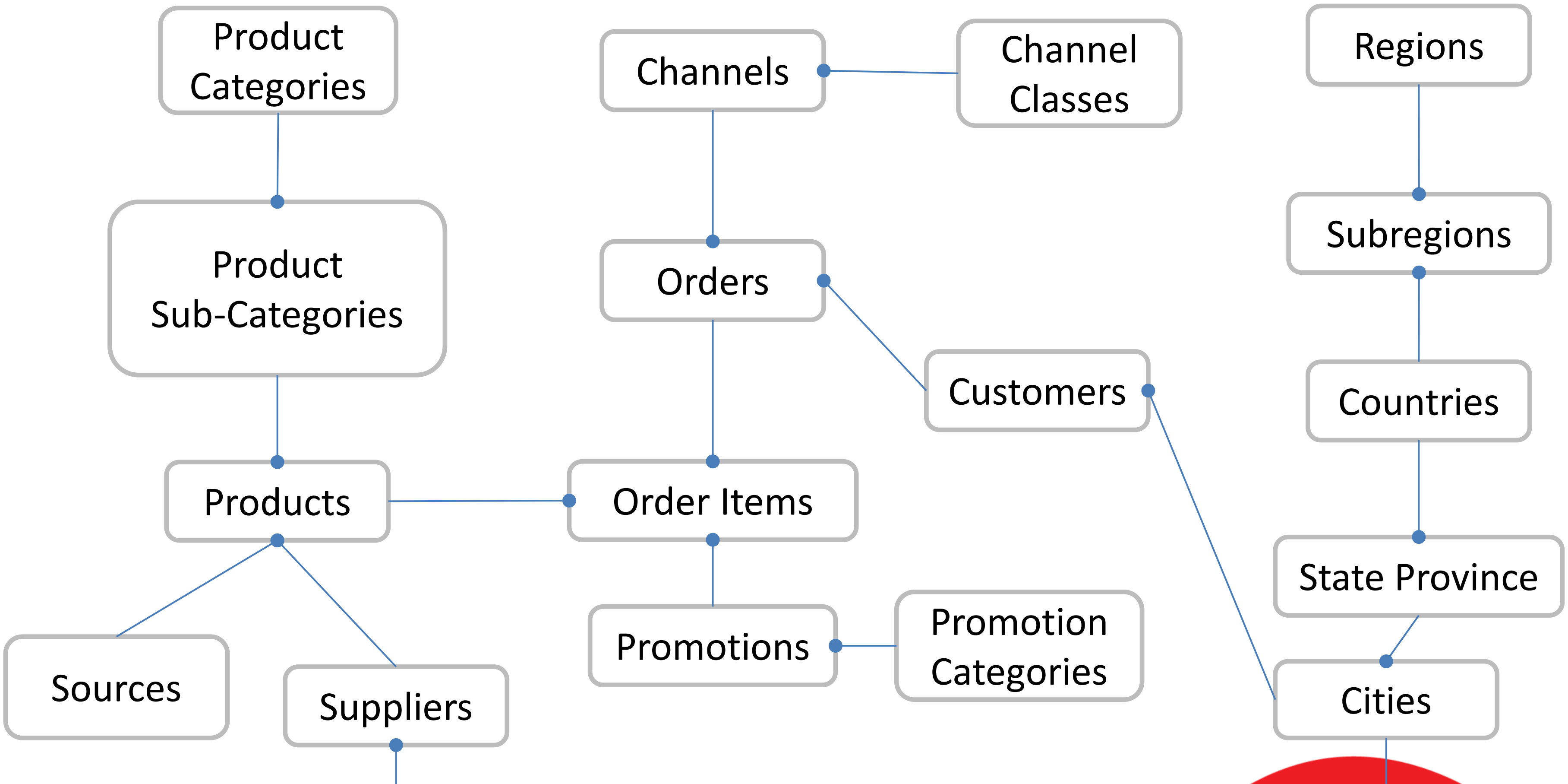
Data Warehouse Introduction

- BI Workflow
- Schema Model
  - Source OLTP Schema
  - Warehouse Star & Snowflake Schemas
- Star Query

Business Intelligence Workflow



Source OLTP - Example Normalized Model



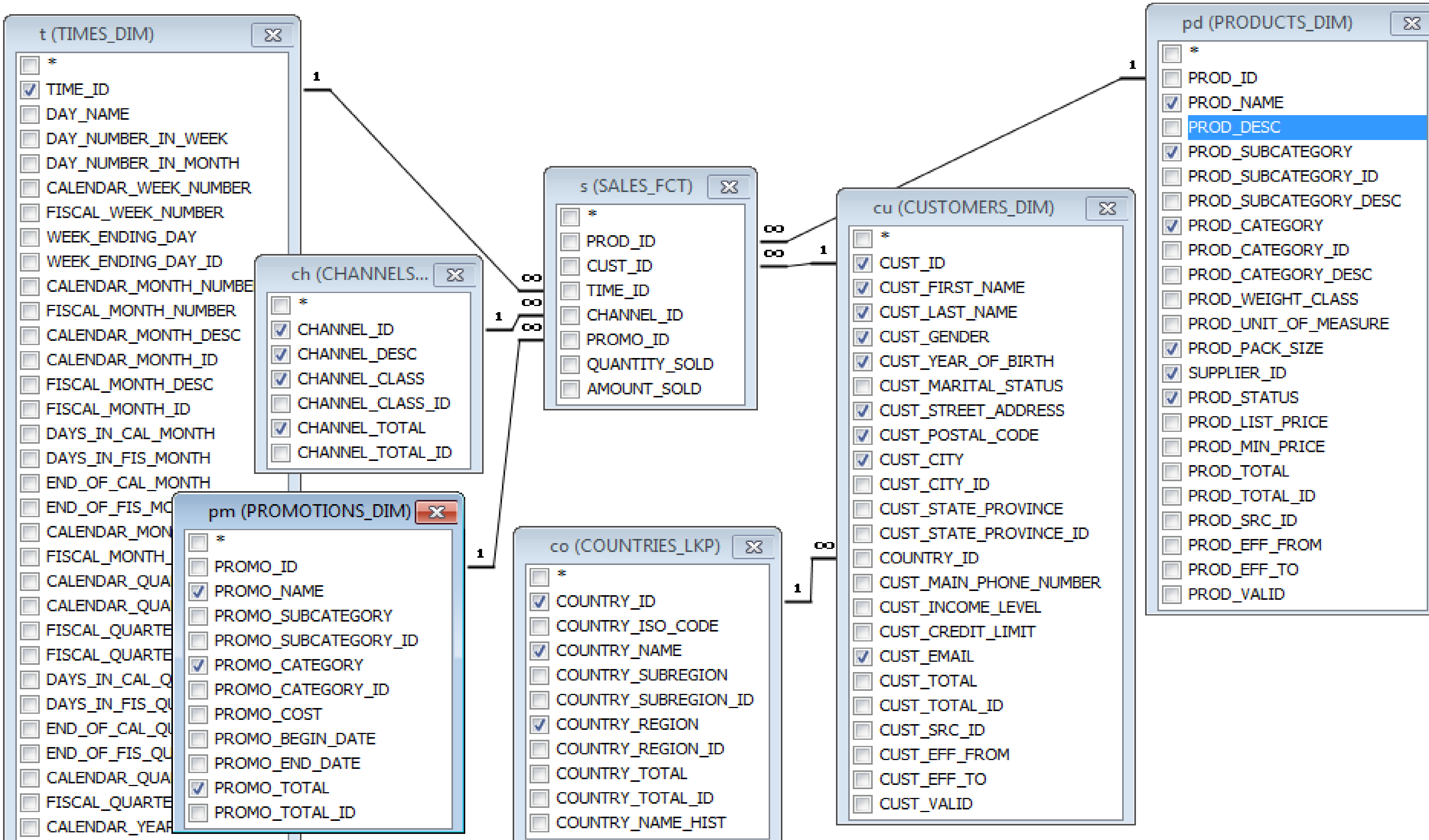
Source Data - Example Extract

"TIME_ID","PROD_ID","PROD_NAME","PROD_CATEGORY","PROD_SUBCATEGORY","PROD_PACK_SIZE","SUPPLIER_ID","PROD_STATUS","CHANNEL_ID","CHANNEL_DESC","CHANNEL_CLASS","CHANNEL_TOTAL","PROMO_ID","PROMO_NAME","PROMO_CATEGORY","PROMO_TOTAL","CUST_ID","CUST_FIRST_NAME","CUST_LAST_NAME","CUST_EMAIL","CUST_CITY","CUST_GENDER","CUST_YEAR_OF_BIRTH","CUST_STREET_ADDRESS","CUST_POSTAL_CODE","COUNTRY_ID","COUNTRY_NAME","COUNTRY_REGION","QUANTITY_SOLD","AMOUNT_SOLD„

"1999.07.10",23,"External 101-key keyboard","Software/Other","Accessories","P",1,"STATUS",2,"Partners","Others","Channel total",999,"NO PROMOTION #","NO PROMOTION","Promotion total",24561,"Abigail","Ruddy","Ruddy@company.com","Yokohama","M",1978,"77 North Packard Avenue","37400",52782,"Japan","Asia",1,22.34

"1999.05.10",23,"External 101-key keyboard","Software/Other","Accessories","P",1,"STATUS",2,"Partners","Others","Channel total",999,"NO PROMOTION #","NO PROMOTION","Promotion total",24561,"Abigail","Ruddy","Ruddy@company.com","Yokohama","M",1978,"77 North Packard Avenue","37400",52782,"Japan","Asia",1,22.34

Example Star Schema Model



Star Query Example

```
CREATE VIEW SALES_VW AS
SELECT
  to_char(t.time_id, 'YYYY.MM.DD') AS time_id,
  pd.prod_id,
  pd.prod_name,
  pd.prod_category,
  pd.prod_subcategory,
  pd.prod_pack_size,
  pd.supplier_id,
  pd.prod_status,
  ch.channel_id,
  ch.channel_desc,
  ch.channel_class,
  ch.channel_total,
  pm.promo_id,
  pm.promo_name,
  pm.promo_category,
  pm.promo_total,
  cu.cust_id,
  cu.cust_first_name,
  cu.cust_last_name,
  cu.cust_email,
  cu.cust_city,
  cu.cust_gender,
  cu.cust_year_of_birth,
  cu.cust_street_address,
  cu.cust_postal_code,
  co.country_id,
  co.country_name,
  co.country_region,
  s.quantity_sold,
  s.amount_sold
```

```
FROM channels_dim ch
INNER JOIN sales_fct s
  ON ch.channel_id = s.channel_id
INNER JOIN products_dim pd
  ON pd.prod_id    = s.prod_id
INNER JOIN promotions_dim pm
  ON pm.promo_id   = s.promo_id
INNER JOIN customers_dim cu
  ON cu.cust_id    = s.cust_id
INNER JOIN countries_lkp co
  ON co.country_id = cu.country_id
```

Star Query - Simple Groping

```
SELECT
      t.fiscal_year,
      c.cust_city,
      SUM(s.amount_sold) AS amount_sold
    FROM sales_fct s
      INNER JOIN times_dim t
        ON t.time_id  = s.time_id
      INNER JOIN products_dim p
        ON p.prod_id  = s.prod_id
      INNER JOIN customers_dim c
        ON c.cust_id  = s.cust_id
     GROUP BY
       t.fiscal_year,
       c.cust_city
     ORDER BY 1,3 DESC
```

Topic Agenda

Data Model

- Fact Tables
  - Junk & Degenerated Dimensions
- Dimension Tables
  - Conformed, Slowly Changing or Growing
- Dimension Hierarchies
- PG ADW Naming Standard
- PG ADW Table Types

Data Model

Fact Tables

- Contains
  - FK [SK]ID key columns
    * Contains references to dimension or lookup PKs
    * Set of FKs columns concatenation constitute PK
  - Measure columns
    * Amount (e.g. currency), quantity, count
    * Factless fact
      - contains no measures
      - only captures many-to-many relationships between dimensions
  - No dimensional key columns
    e.g. indicator to differentiate estimated and real values
  - Attributes not related to any dimension
    * If used in row aggregation or filtering
      - all should be placed in one additional **junk dimension table**
      - junk PK should be referenced from fact using additional FK
    * If not used in row aggregation or filtering
      - are high cardinality
      - stay in fact table
      - **are degenerated dimensions**
  - **Replace NULL in PK columns to dummy values**

sh_sales_fct

COLUMN_NAME	DATA_TYPE	COMMENTS
PROD_ID	NUMBER	FK to the products di
CUST_ID	NUMBER	FK to the customers d
TIME_ID	DATE	FK to the times dimen
CHANL_ID	NUMBER	FK to the channels di
PROMO_ID	NUMBER	promotion identifier,
SOLD_QTY	NUMBER(10,2)	product quantity sold
SOLD_AMT	NUMBER(10,2)	invoiced amount to th

EST_IND_ID VARCHAR2(1)

REP_IND VARCHAR2(1)
PRCSD_IND VARCHAR2(1)

EVENT_DESC VARCHAR2(100)

Data Model

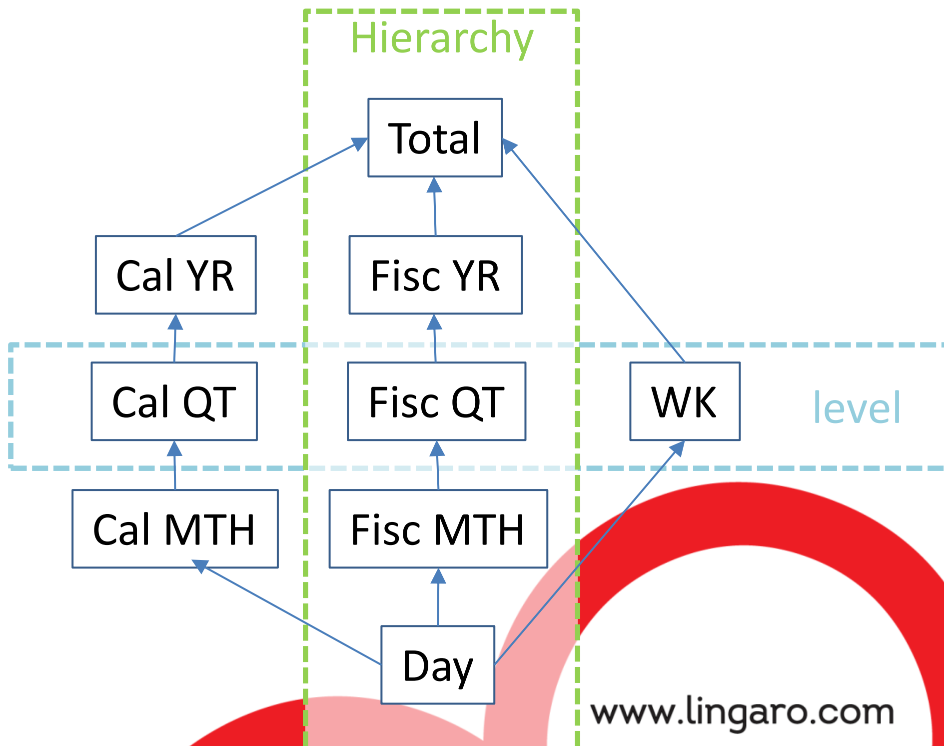
Dimension Tables

- Stores distinct combination of related fact attributes
- Few of attribute columns are used to
  - define levels of at least one hierarchy using

```
* oracle dimension object – for static hierarchies when QR needed
* hierarchy table – if hierarchies are dynamic
```

PROD_HIER

PROD_HIER_ID
HIER_NAME
HIER_LVL
LVL_COL_NAME
LVL_NAME



- Use already existed referential
  - if available

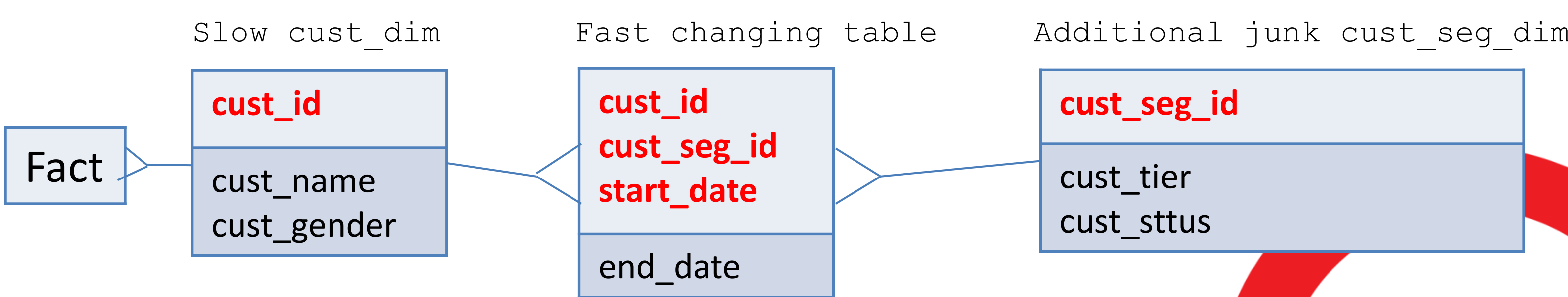
Data Model

Dimension Types

- Conformed
  - Is shared across many or all fact tables
  - Possible Drill Across

    `adding to star query additional data from another fact`
  - Examples: Customer, Location, Time, Product.
- Slow changing or growing
  - Static or changes are unpredictable
  - Change not need to preserve old values
  - Old values can be stored in history records in the same tables

    `additional columns needed – start_date, end_date`
  - Oracle Flashback Archive can be used
  - Fast change generate huge number of history rows - use snowflake with junk



PG ADW Data Model

Naming Standards ★ ★ ★

from PG ADW DM Naming Standards document

- Table and column names
  - Abbreviations with maximum 5 characters separated using underscore
  - Use abbreviation from glossary
  - If not available create by removing 1-vowels 2-duplicates from left
  - Finish column name with **suffix** indicating kind of data

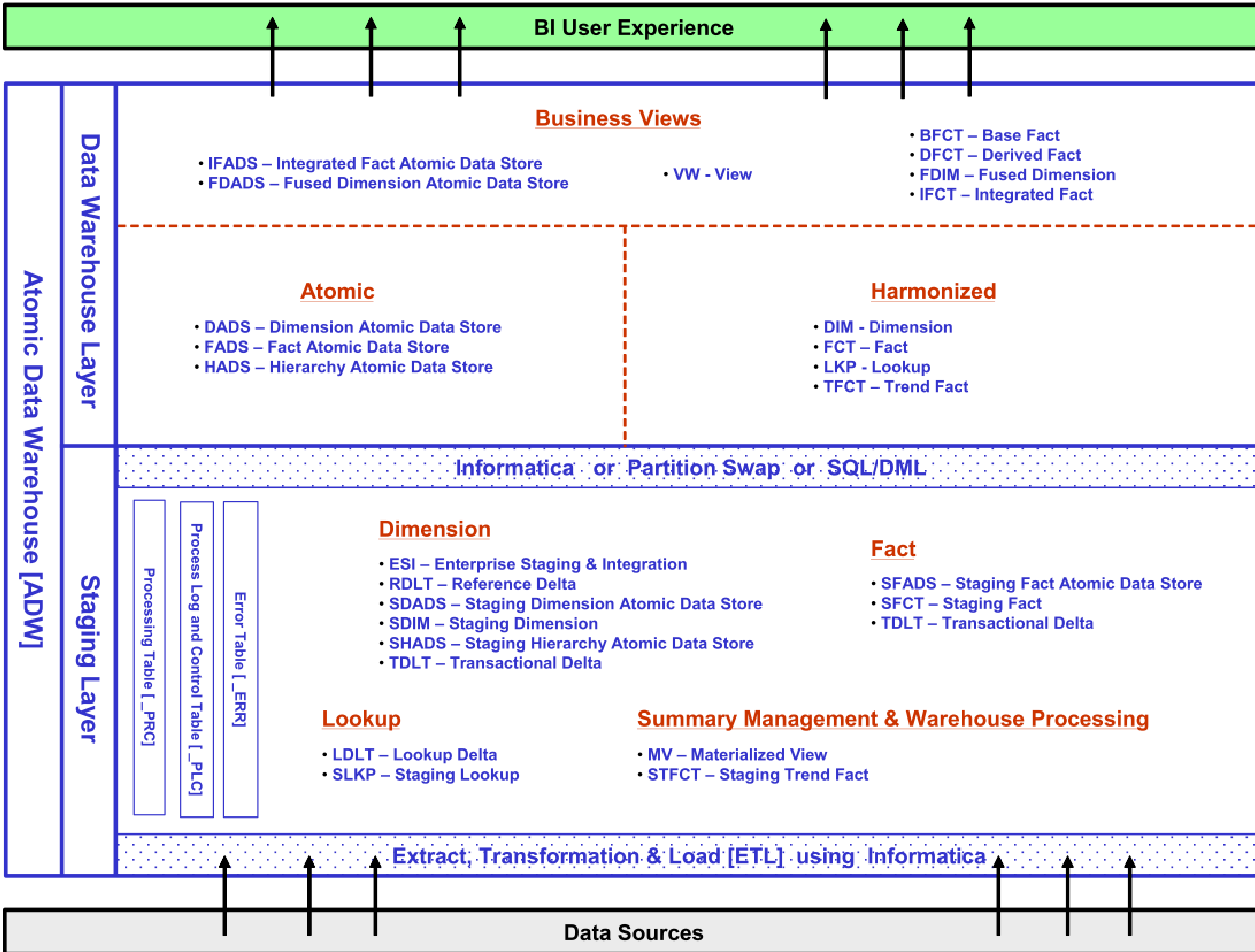
```
ID – key identifier based on business value
SKID – key identifier based on number from sequences – surrogate key
NAME – short name
DESC – long description
CODE – very short, typically abbreviated identifier
NUM – numeric integer value
CNT – integer with count value
AMT – amount, typical for currency
DATE – day date without time
DATETM – date with time
TIME_STAMP – only time stamp
IND – logical value like 0/1 but with one character values Y/N
LABEL – describe contents of generic (typical next) column
LVL – hierarchy level numeric value
LOC – geo location
PERD - period
SEQ_NUM – sequence number
TXT – source code text
VAL – last resort - other kind and unclassified data
```

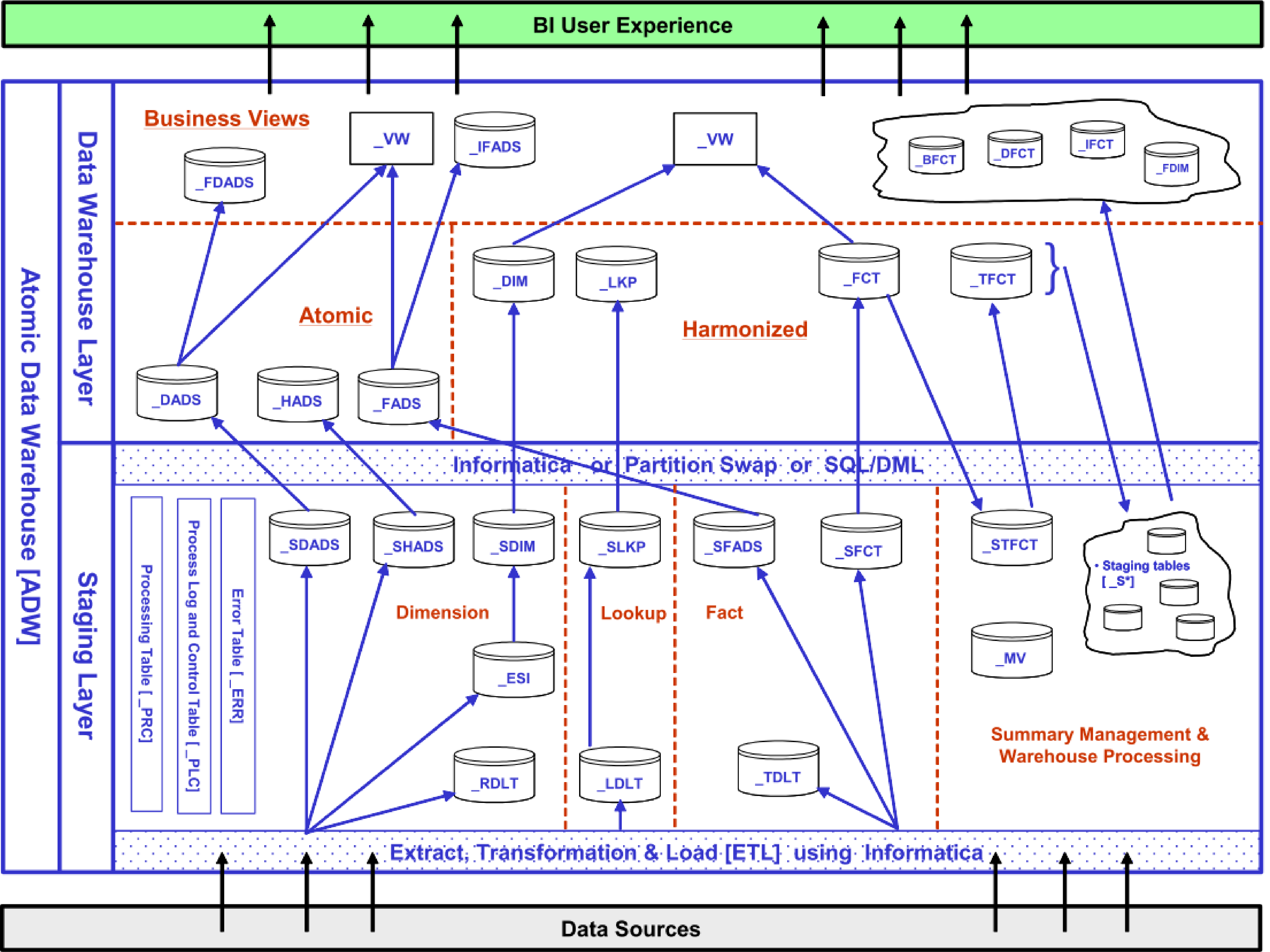
PG ADW Data Model

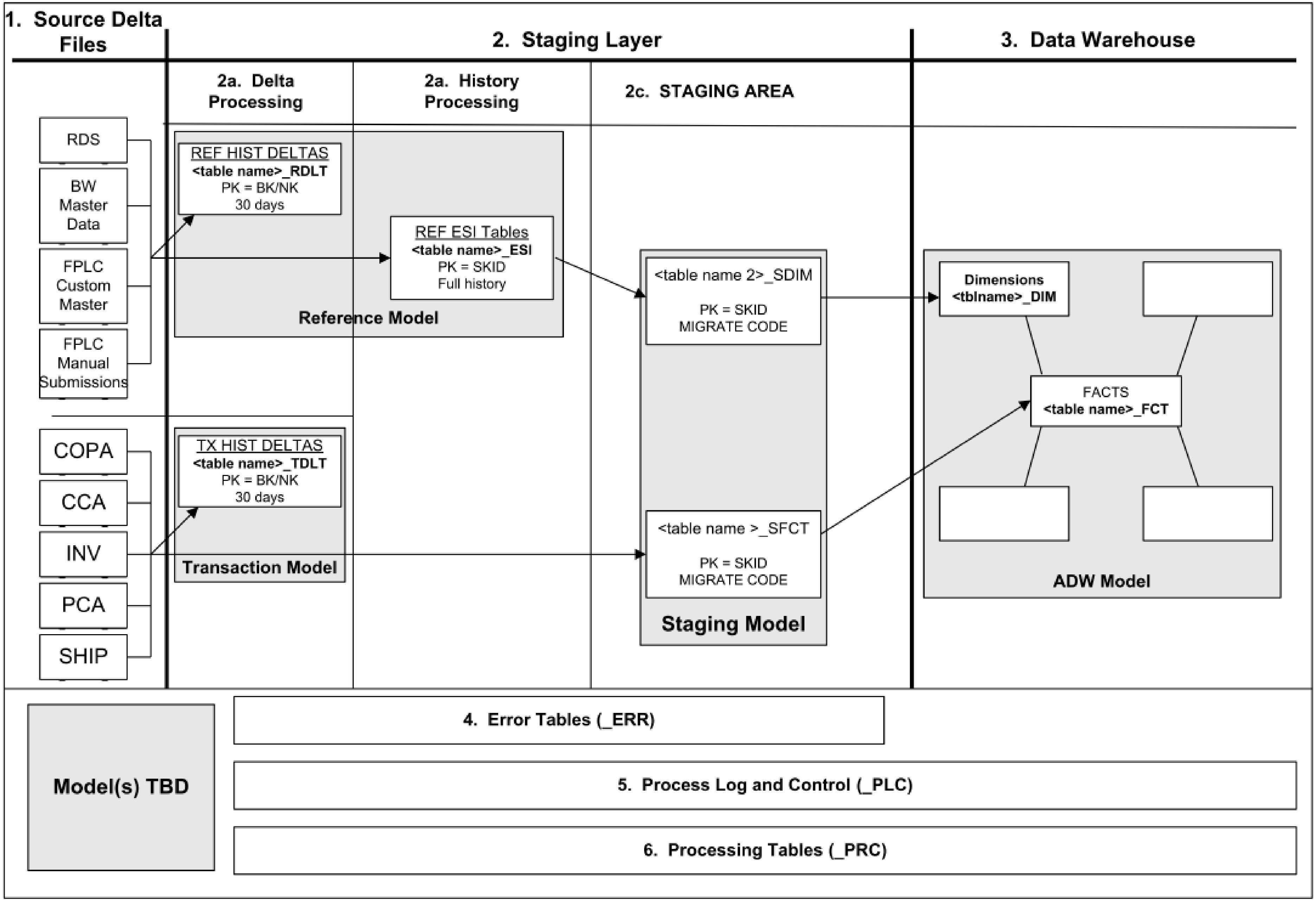
Table Types

Warehouse Layer

- Finish table name with suffix indicating table type
- Staging Layer (before Warehouse Layer in Atomic DW)
  - * Deltas: _RDLT, _LDLT, _TDLT – Reference, Lookup, Transaction
    - - Tracking 30 days changes from dimension, lookup and fact tables in data source
    - - Uses source table name before suffix. - Help in resolving processing issue
  - * History: _ESI - Enterprise Staging & Integration - dimension like - fast dimensions refresh
  - * Stage tables for Warehouse Layer _S* - loading dock - not use in reports
  - * Error: _ERR, Metadata: _PRC, Log: _PLC,
- Atomic - _*ADS - Atomic Data Store - finest grain data available is source (e.g. daily)
- Harmonized - transformed, standardized depending on how data differ in source
  - no business logic transformations, are: _DIM - dimension, _FCT - fact,
  - _LKP - Lookup Table - similar to dimension but can't be used to build any hierarchy
  - _TFCT - trend fact - time denormalized aggregated measures like current and previous year sales
- Business Ready - integrated, summarized, interpreted data for reporting
  - _BFCT – Base Fact - first derivation step - moderate app-specific filtering, aggregation etc.
  - _DFCT – Derived Fact - second step - derived from _BFCT – not integrated with other facts
  - _IFCT – Integrated Fact - integrated to view in reports with other facts
  - <DIM_abbr>_<HIER_num>_<PROJ_abbr>_<CAL_abbr>_FDIM – Fused Dimension - flattened format of a hierarchy - levels columns instead of parent child columns


BI User Experience
Atomic Data Warehouse [ADW]
Data Warehouse Layer
Business Views
• IFADS – Integrated Fact Atomic Data Store
• FDADS – Fused Dimension Atomic Data Store
• VW - View
• BFCT – Base Fact
• DFCT – Derived Fact
• FDIM – Fused Dimension
• IFCT – Integrated Fact
Atomic
• DADS – Dimension Atomic Data Store
• FADS – Fact Atomic Data Store
• HADS – Hierarchy Atomic Data Store
Harmonized
• DIM - Dimension
• FCT – Fact
• LKP - Lookup
• TFCT – Trend Fact
Informatica or Partition Swap or SQL/DML
Staging Layer
Processing Table [_PRC]
Process Log and Control Table [_PLC]
Error Table [_ERR]
Dimension
• ESI – Enterprise Staging & Integration
• RDLT – Reference Delta
• SDADS – Staging Dimension Atomic Data Store
• SDIM – Staging Dimension
• SHADS – Staging Hierarchy Atomic Data Store
• TDLT – Transactional Delta
Fact
• SFADS – Staging Fact Atomic Data Store
• SFCT – Staging Fact
• TDLT – Transactional Delta
Lookup
• LDLT – Lookup Delta
• SLKP – Staging Lookup
Summary Management & Warehouse Processing
• MV – Materialized View
• STFCT – Staging Trend Fact
Extract, Transformation & Load [ETL] using Informatica
Data Sources


BI User Experience
Atomic Data Warehouse [ADW]
Data Warehouse Layer
Business Views
_FDADS
_VW
_IFADS
_VW
_BFCT
_DFCT
_IFCT
_FDIM
Atomic
_DADS
_HADS
_FADS
Harmonized
_DIM
_LKP
_FCT
_TFCT
Informatica or Partition Swap or SQL/DML
Staging Layer
Processing Table [_PRC]
Process Log and Control Table [_PLC]
Error Table [_ERR]
_SDADS
_SHADS
_SDIM
_SLKP
_SFADS
_SFCT
_STFCT
• Staging tables [_S*]
Dimension
Lookup
Fact
_ESI
_MV
_RDLT
_LDLT
_TDLT
Summary Management & Warehouse Processing
Extract, Transformation & Load [ETL] using Informatica
Data Sources


1. Source Delta Files
2. Staging Layer
3. Data Warehouse
2a. Delta Processing
2a. History Processing
2c. STAGING AREA
RDS
BW Master Data
FPLC Custom Master
FPLC Manual Submissions
REF HIST DELTAS
<table name>_RDLT
PK = BK/NK
30 days
REF ESI Tables
<table name>_ESI
PK = SKID
Full history
Reference Model
<table name 2>_SDIM
PK = SKID
MIGRATE CODE
Dimensions
<tblname>_DIM
FACTS
<table name>_FCT
COPA
CCA
INV
PCA
SHIP
TX HIST DELTAS
<table name>_TDLT
PK = BK/NK
30 days
Transaction Model
<table name >_SFCT
PK = SKID
MIGRATE CODE
Staging Model
ADW Model
Model(s) TBD
4. Error Tables (_ERR)
5. Process Log and Control (_PLC)
6. Processing Tables (_PRC)

Topic Agenda

Constraints

- PK, FK, NN, UQ, Check
- Using in Warehouse

Indexes

- Bitmap Indexes
- Bitmap Join Indexes
- B-Tree Indexes
- Index Organized Tables

Constraints Usage

Warehouse Recommendations ★★★

- NN - Not Null
  - cheep – direct load can use it with very small impact
  - use for all mandatory columns
- UQ – Unique & PK – Primary Key
  - If ENABLE uses auto-created B-Tree index
  - Do not create PK & UQ on fact tables or use only DISABLE RELY state
- FK – Foreign Key - referential
  - Use DISABLE RELY state in fact tables (use always when column is related do PK)
- CK – Check – logical expression
  - Do not use on fact tables at all – use validation processes after load intead
- Enabled UQ, PK, FK
  - Only on dimension, lookup and metadata tables
  - Constraints and UQ and PK auto-created indexes should be named with convention

```
column_name data_type CONSTRAINT tab_col_pk PRIMARY KEY
```

Bitmap Index
Fundamentals

```
CREATE BITMAP INDEX sales_fct_bx1 ON sales_fct (cntry_code);
```

- Each key column value uses separate bitmap
  - bit position describes table row physical address - ROWID
  - “1” means - row contains this key value in key column
- Bitmaps are **compressed**
  - compression is done during index creation and modification
  - decompression is needed before bitmaps marge (AND, OR)
  - work area PGA memory structures are used for this
- Best in DW – but drop before and recreate after table load
  - smaller than B-tree index
  - faster to find large number off rows
- Problematic in OLTP
  - index modifications are expensive (CPU)
  - after UPDATE on one row many rows are locked
  - can’t be unique (PK and UQ columns use B-tree indexes)
- Problematic in same cases in Exadata
  - preventing of use storage indexes

sales_fct table			bitmap		
id	cntry_code	ROWID	US	CA	UK
10	US	...Z4d1	1	0	0
11	US	...Z4d2	1	0	0
12	CA	...Z4d3	0	1	0
13	CA	...Z4d4	0	1	0
14	UK	...Z4d5	0	0	1
15	UK	...Z4d6	0	0	1
16	UK	...Z4d7	0	0	1
17	UK	...Z4d8	0	0	1
18	US	...Z4d9	1	0	0
19	US	...Z4e0	1	0	0
20	US	...Z4e1	1	0	0

contains info about NULL
e.g. count(*) use it

Bitmap Join Index

Fundamentals

```
CREATE BITMAP INDEX sales_cust_gndr_bx1
ON sales_fct(c.cust_gendr_code)
FROM sales_fct s, cust_dim c
WHERE s.cust_id = c.cust_id
LOCAL NOLOGGING;
```

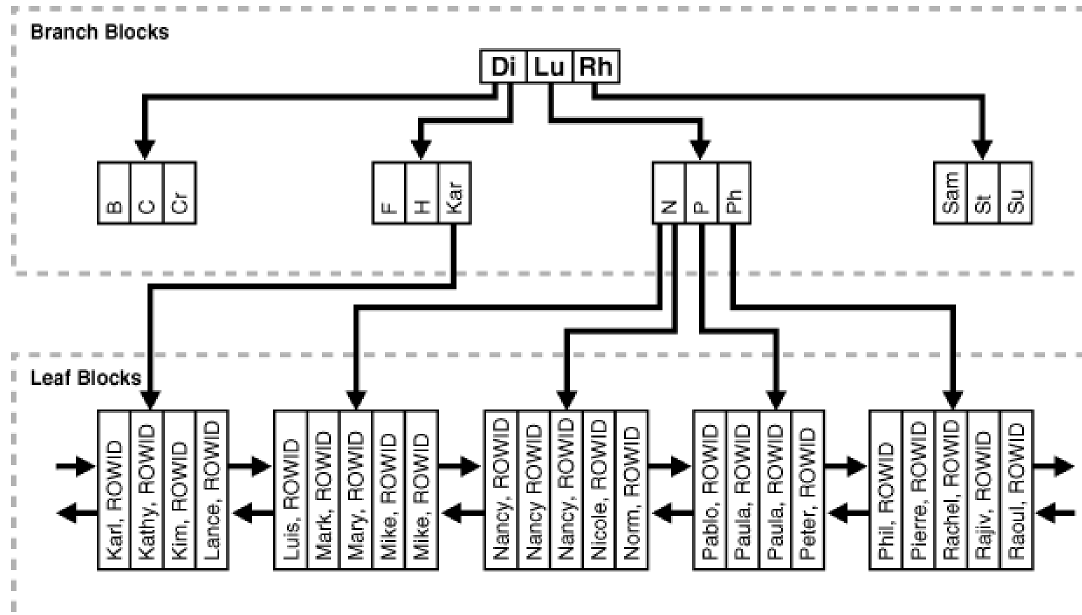
sales_fct cust_dim			bitmap	
id	**gender**	ROWID	**M**	**F**
10	**M**	...Z4d1	1	0
11	**M**	...Z4d2	1	0
12	**F**	...Z4d3	0	1
13	**F**	...Z4d4	0	1
14	**F**	...Z4d5	0	1
15	**F**	...Z4d6	0	1
16	**F**	...Z4d7	0	1
17	**F**	...Z4d8	0	1
18	**M**	...Z4d9	1	0
19	**M**	...Z4e0	1	0
20	**M**	...Z4e1	1	0

- Rowids from fact but key from dimension table
- Cen use many dimension tables – star or snowflake
- Star join query not scan dimension tables
  - Only index and fact table is scanned
  - Performance improve
  - Best for large dimension tables
- Parallel DML on dimension table mark the index as unusable

- Create separate single column bitmap indexes instead of multicolumn

B-Tree Index
Fundamentals

```
CREATE [UNIQUE] INDEX emp_fname_ix1
  ON employee(first_name)
  [REVERSE];
```



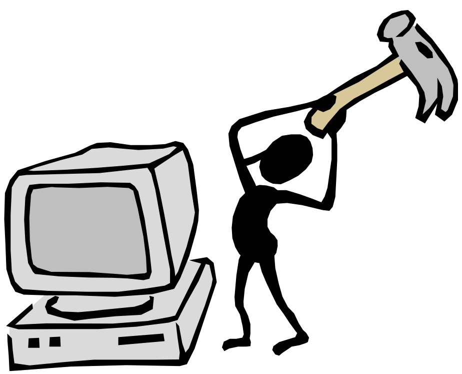
- Not good on fact tables
- Full scan & Fast Full scan
- Auto-created on PK and UQ
- Add NN on key column
  - if possible

- Index Organized Tables
  - Looks like B-Tree with all columns
  - Sorted on mandatory PK
  - Good when predicates only on PK

```
CREATE table employees
  ...
  ORGANIZATION INDEX;
```

Bitmap Join Index
Workshop

- Create one bitmap join index for sales_fct and cust_dim tables
- Check is used in execution plan instead of dimension table
  - sales_fct should be partitioned copy of sh_sales_fct but with only channel 4 used.



Topic Agenda

ETL

Data Extraction

Transporting Data

- Oracle DB -> Oracle DB
  - Direct Load
  - Database Links
  - Transporting Tablespaces
  - Data Pump - External Table
- File -> Oracle DB
  - SQL Loader - External Tables

Transforming Data

- Multistage
- Pipelined

Data Extraction

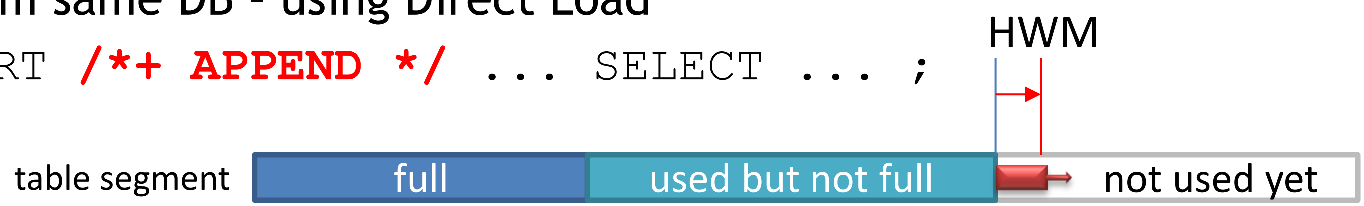
- Interactive
  - SQL Developer – export from query
    http://www.bryansgeekspeak.com/2011/06/exporting-oracle-table-data-using-sql.html
  - MS Office – using ODBC Oracle driver
    https://itkbs.wordpress.com/2014/07/28/how-to-install-odbc-driver-for-oracle-in-windows-7/
    https://community.office365.com/en-us/f/172/t/206131
- Batch
  - sqlplus – spool query results
    http://lenguyenthedat.blogspot.com/2014/01/dumping-oracle-dbs-table-into-csv-with.html
  - utl_file package – write from dbms_sql cursor into file in directory
    https://asktom.oracle.com/pls/apex/f?p=100:11:0::::P11_QUESTION_ID:88212348059
  - Pro*C

Transporting Data

Oracle DB -> Oracle DB

- Loading from same DB – using Direct Load

```
INSERT /*+ APPEND */ ... SELECT ... ;
```



- Between DBs - Database Link

```
CREATE DATABASE LINK link_name
  CONNECT TO user_name IDENTIFIED BY password USING 'DB service';
INSERT /*+ APPEND */ ... SELECT ... FROM table_name@link_name;
```

← Synonym recommended

- Between DBs - Transporting Tablespace
  - Turn TS into read only mode – SQL> ALTER TABLESPACE tsname READ ONLY;
  - Export TS metadata from source DB

```
$ expdp system/pass DUMPFILE=expdat.dmp DIRECTORY=dir1 TRANSPORT_TABLESPACES = sales1,sales2
```

  - Copy expdat.dmp and tablespace datafiles to target server – not needed if shared storage
  - Import TS metadata into target DB

```
$ impdp system/password DUMPFILE=expdat.dmp DIRECTORY=dir1
        TRANSPORT_DATAFILES=/salesdb/sales01.dbf,
                            /salesdb/sales02.dbf
```

- For files outside DB DBA need to create directory

```
CREATE DIRECTORY dir1 AS '/dmpfiles/';
GRANT ALL ON dir1 TO dev_role;
```

Transporting Data
Oracle DB -> Oracle DB - continued

- Between DBs – Data Pump External Table
  - Unload data outside source DB on filesystem into DMP file

```
CREATE TABLE inventories_xt
  ORGANIZATION EXTERNAL
    (TYPE ORACLE_DATAPUMP DEFAULT DIRECTORY dir1 LOCATION ('inv_xt.dmp'))
  AS SELECT * FROM inventories_fct;
```

  - Copy expdat.dmp to target server – not needed if shared storage
  - Create external table in target DB

```
CREATE TABLE inventories_xt
  ORGANIZATION EXTERNAL
    (TYPE ORACLE_DATAPUMP DEFAULT DIRECTORY dir1 LOCATION ('inv_xt.dmp'));
```

  - Load from external table as from normal local table
  - Can be used among same DB to decrease DB size
- Alternative for data transport
  - Data Pump without external table

```
$ expdp user/pass DUMPFILE=expdat.dmp DIRECTORY=dir1 TABLES=table
$ impdp user/pass DUMPFILE=expdat.dmp DIRECTORY=dir1
```

  - Data Pump can use DB links instead of DMP file

```
$ impdb user/pass NETWORK_LINK=db_link
```

Loading Data

File -> Oracle DB

- From flat text file – SQL Loader External Table
  - Create external table in target DB

```
CREATE TABLE dept_ext
  (DEPTNO NUMBER(2),
   DNAME CHAR(14),
   LOC CHAR(13))
  ORGANIZATION external
    (TYPE oracle_loader
     DEFAULT dir1
     ACCESS PARAMETERS
       (RECORDS DELIMITED BY NEWLINE BADFILE 'dir1:data1.bad' SKIP 20
        FIELDS TERMINATED BY "," OPTIONALLY ENCLOSED BY '"' LDRTRIM
       (DEPTNO CHAR(255) TERMINATED BY "," OPTIONALLY ENCLOSED BY '"',
        DNAME  CHAR(255) TERMINATED BY "," OPTIONALLY ENCLOSED BY '"',
        LOC    CHAR(255) TERMINATED BY "," OPTIONALLY ENCLOSED BY '"')
     ) LOCATION('data1.dat')
) REJECT LIMIT UNLIMITED PARALLEL;
```

  - Load from external table as from normal local table
  - Alternative - SQL Loader without external table

```
$ sqlldr userid=user/pass control=data1.ctl log=loader.log
```

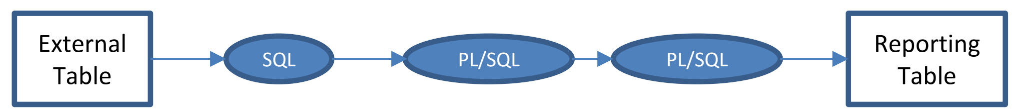
Transformation

Methods

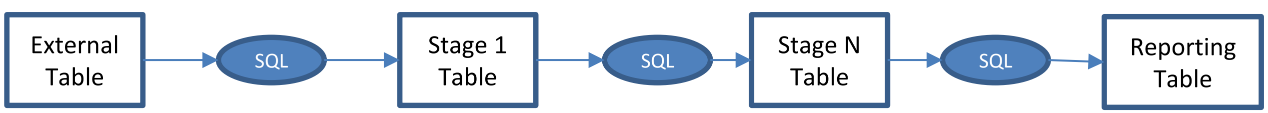
- Use single SQL if possible ★★★

- Pipelined PL/SQL functions ★
  - no staging or only global temporary table used as staging

```
INSERT /*+ APPEND PARALLEL */ ... SELECT ...
  FROM TABLE(fn_trans1_pipe(fn_trans2_pipe(CURSOR(SELECT ... FROM data_xt ...))));
```



- Multiple staging ★
  - Without PL/SQL



- More storage space used and more IO operations
- Time consuming and difficult to control parallelism
- Restartable from “not success” stage (not from beginning)

  - With PL/SQL - last resort for most complicated situations

Topic Agenda

Data Compression

- Overview
- Tables Compression
  - Basic (Direct Load)
  - OLTP (Advanced)
  - HCC (Exadata)
  - SecureFiles LOB
- Index Compression
  - Bitmap Index
  - B-Tree Index
- External Table Compression
- Saving Storage Space
  - Uniform Size Extents

Overview

Positive effects of compression

- Less space used for the same data
  - measured using compression ratio = uncompressed/compressed data size
- Reduced storage cost
  - $ amount spent to store 1 GB of application data
- SQL statements use less number of blocks
  - reduced disk cost in SQL
- Reduced storage I/O load
  - blocks per second

Overview

Compression challenges and costs

- Additional CPU load (CPU time)
  - needed to transform data into compressed form (and vice versa)
- Choosing object - not all data should be compressed
  - best are: huge archival not modified data
- Gain better compression ratio effort
  - sorting loaded data
  - choosing best sorting key columns
  - use bigger block size (e.g. table and B-tree index compression)
- Sometimes misused and to cause rather than solve problems
- Compression can restrict
  - database features (e.g. IOT has only PK columns compression similar to B-tree index compression)
  - commands usage (e.g. ALTER TABLE statement add and drop columns)

Table Compression
Basic – Direct Load

- This compression method is selected during table creation or altering

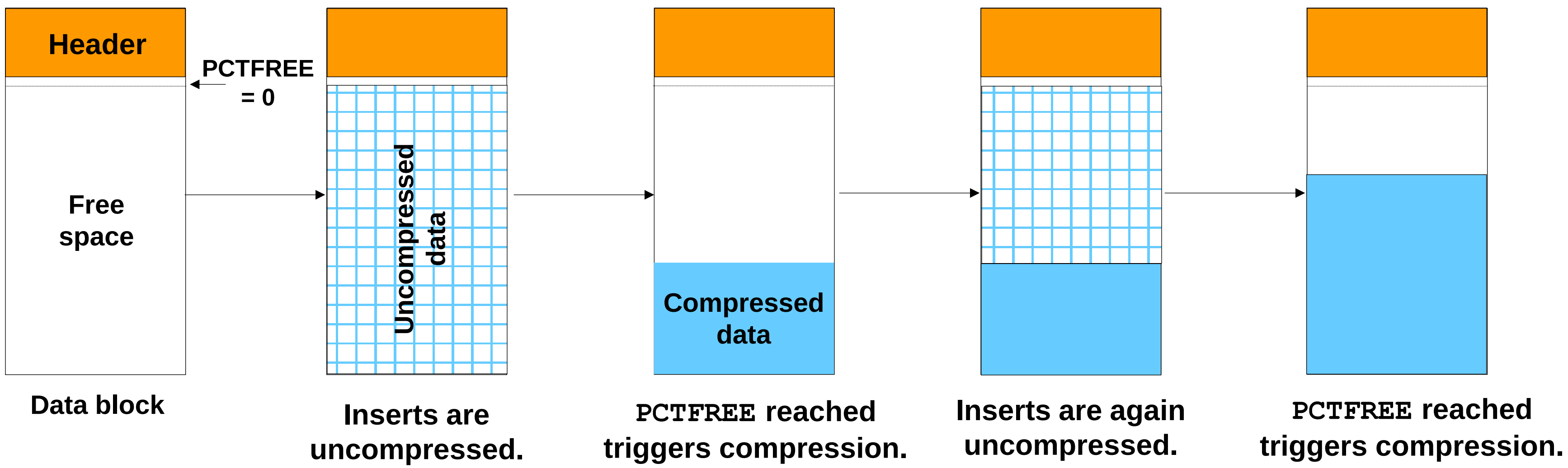
```
CREATE/ALTER TABLE … COMPRESS { [ BASIC | DIRECT_LOAD OPERATIONS ] };
```

- Used only during bulk load operations (Direct Load, CTAS)

```
sqlldr   … DIRECT=TRUE …
INSERT /*+ APPEND */ INTO … SELECT …;
CREATE  TABLE … AS SELECT …;
```

- Data modified using conventional DML not compressed
- Improved performance for queries accessing large amounts of data
  - Fewer I/O s
  - Buffer Cache efficiency
- Data is compressed at the database block level (block size important)
- High compression ratio - up to 10x
- Expensive data modification – modification need decomression
- Good in DW
- As all other table compression methods:
  - enabled at either the table or partition level
  - Completely transparent to applications
  - COMPRESS option modification effect only on future modified blocks

Table Compression
Basic – Direct Load



- Compression:
  - Oracle examines full blocks for any duplicates
  - Creates a symbol for duplicated block content
  - Rewrites the block substituting the symbol for the values it represents
- Limitations
  - Maximum 255 columns in table (BASIC AND OLTP compression)
  - columns can't be dropped if a table is BASIC compressed

Table Compression

Advanced – OLTP

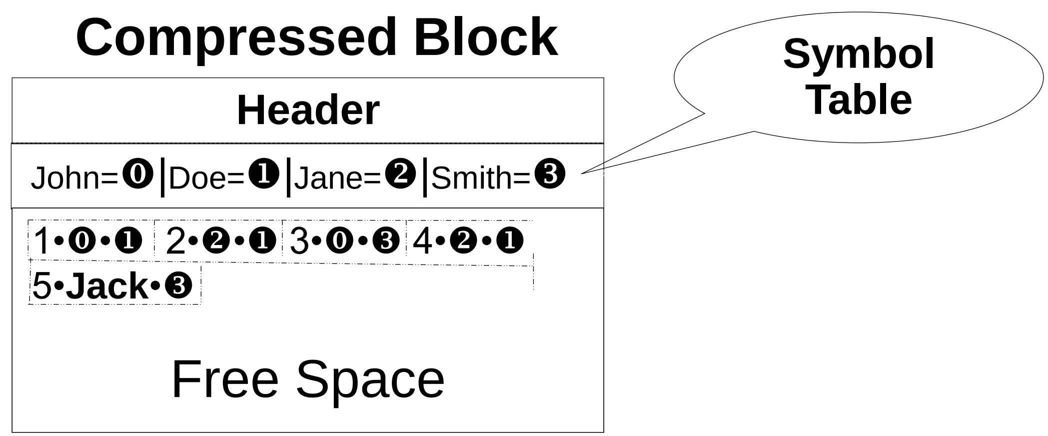
- This compression method is selected during table creation or altering

  `CREATE/ALTER TABLE … COMPRESS FOR [ OLTP | ALL OPERATIONS ];`

- Works during all DML SQL statements
- Lower cost during data modification – good for OLTP applications
- Only compression method where column adding or dropped is possible
- Internals:

Employee Table

ID	FIRST_NAME	LAST_NAME
1	John	Doe
2	Jane	Doe
3	John	Smith
4	Jane	Doe
5	**Jack**	Smith



- Compression:
  - Oracle find duplicated fields values in block
  - Stores duplicates in symbol table
  - Replace field value to symbol

Table Compression

HCC - Hybrid Columnar Compression

- This compression method is selected during table creation or altering

```
CREATE/ALTER TABLE … COMPRESS FOR QUERY HIGH/LOW;
                     COMPRESS FOR ARCHIVE HIGH/LOW;
```

- FOR QUERY - used on frequently queried tables (compression ratio over 10)
- FOR ARCHIVE - slower query time and better compression (ratio up to 70)
- LOW – lower compression ratio but faster load
- HIGH – higher compression ratio but slower load

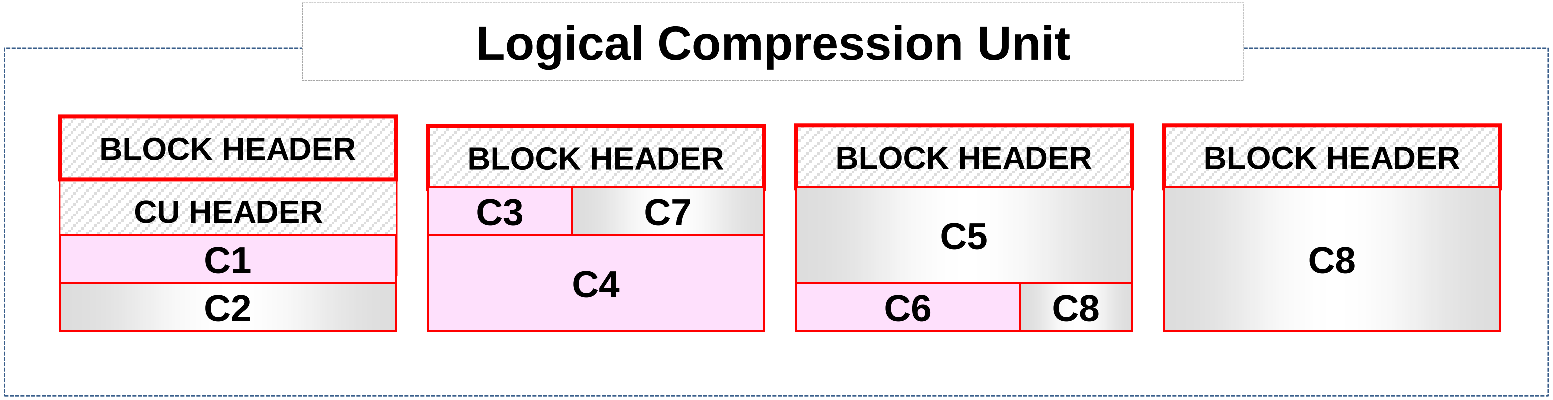
- As basic compression HCC is used only during direct Load and CTAS
- Conventional INSERT results in OLTP Compression
- Updated rows automatically migrate to OLTP Compression
- Traditional compression (BASIC and OLTP) use DB server CPU
  - trade-off between CPU and Disk I/O
- HCC uses storage server hardware (e.g. Exadata) to compress
- Very low DB server CPU load during decompression
- Data remain compressed in buffer cache
- Only columns needed in query are decompressed

more info:
http://www.oracle.com/technetwork/es/articles/database-performance/exadata-hybrid-columnar-compression-2098797-esa.html
http://www.oracle.com/technetwork/issue-archive/2010/10-jan/o10compression-082302.html
http://www.oracle.com/technetwork/database/exadata/ehcc-twp-131254.pdf

Table Compression

HCC Internals

- HCC combines the best of columnar and row organization
- Done by organizing data into Logical Compression Unit – LCU
  - Efficient entire row operations (INSERT, DELETE, SELECT)
  - Minimal I/O when query needs small part of columns (column is stored in small number off blocks)



- LCU
  - spanning multiple database blocks
  - transformation into columnar organization is done during data load
  - each column compressed separately
  - column organization brings similar values close together
  - typical size 32K (4 blocks x block size, block size 8K)

Table Compression

Improve Compression Ratio

- Use bigger block size (for BASIC and OLTP compression)

```
CREATE TABLESPACE … BLOCKSIZE 32k;
```

- Sort data before load by wide and low cardinality columns

```
INSERT /*+ APPEND */ INTO … SELECT … ORDER BY cntry_name, cust_city;
```

- To choose best compression way for existing uncompressed table use package:
  - DBMS_COMP_ADVISOR (from 9r2 to 11r1 - only BASIC method)
  - DBMS_COMPRESSION (from 11r2 - all methods) - recommends various strategies for compression

```
declare
  v_cmp_ratio     number;
  …
begin  dbms_compression.get_compression_ratio(
  tabname     =>  table_name
  comptype   => dbms_compression.comp_for_query_high,
  cmp_ratio   => v_cmp_ratio
  … );
  dbms_output.put_line('Compression ratio is: '||to_char(v_cmp_ratio));
end;
/
```

- Picks the right compression algorithm for a particular data set
- Guide sorts on a particular column for increasing the compression ratio
- Presents tradeoffs between different compression algorithms

Table Compression

Method Comparison

method	size [MB]	load time	query time
NO COMPRESS	3809	00:58	00:42
BASIC	2500	01:32	00:28
OLTP	2997	04:27	00:29
QUERY HIGH	512	02:16	00:09
QUERY LOW	856	01:05	00:07
ARCHIVE HIGH	424	12:03	00:23
ARCHIVE LOW	488	03:06	00:21

Table Compression
Workshop

- Create basic compressed version of fact table.
  - Use sales_sh as source
- Check compression ratio before and after compression and with sort compressed
  - You can calculate row length and number of rows from dba_tables view
  - This need statistic calculation on tables
  - You can get segment size from dba_segments view

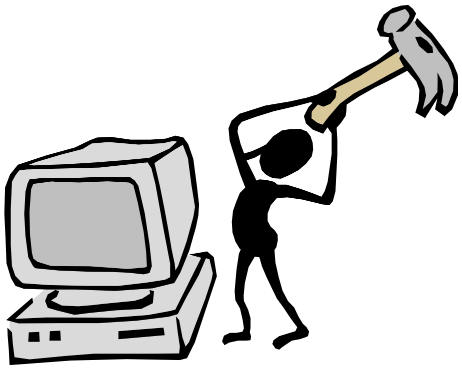


Table Compression

Secure Files LOB Compression

```
CREATE TABLE profiles_prc
(
        id          NUMBER,
        first_name  VARCHAR2 (40),
        last_name   VARCHAR2 (80),
        info        CLOB,
        video       BLOB
)
LOB(video)  STORE AS BASICFILE
LOB(info)   STORE AS SECUREFILE ( KEEP_DUPLICATES NOCOMPRESS/COMPRESS HIGH ) ;
```

- options for the compression:
  - COMPRESS HIGH: Provides the best compression but incurs the most work
  - COMPRESS MEDIUM: Is the default value
  - NOCOMPRESS: Disables compression

- options for the deduplication:
  - DEDUPLICATE: Is the default value
  - KEEP_DUPLICATES: duplicated LOBs uses separate storage

Bitmap Index

Internals

<table>
<tr><td rowspan="5">Bitmap Index</td><td>Key Value</td><td colspan="3">Piece</td><td colspan="3">Piece</td></tr>
<tr><td>Key Value</td><td colspan="2">Piece</td><td colspan="3">Piece</td><td colspan="3">Piece</td></tr>
<tr><td rowspan="3">Key Value</td><td colspan="6">Piece</td></tr>
<tr><td colspan="3" rowspan="2">Value & Rowids</td><td colspan="3">Bitmap</td></tr>
<tr><td>Group</td><td>Group</td><td>Group</td></tr>
</table>

- Each indexed key column value may have one or more bitmap pieces
- Bitmap Piece – describe contiguous set of rows (DML locks are set on piece)
  - key value
  - starting and ending ROWID (rounded to nearest byte boundary)
  - bitmap with compressed zeros

- Bitmap compression:
  - only zeros are compressed, ones are not compressed
  - bitmap is divided into bitmap groups
  - bitmap group is described by control byte
  - first 2 bits in control byte are not "11" - only one control byte

    `containing number of "0" (until next "1" bit)  - maximum 191 zeros`
  - first 2 bits in control byte are "11" - one ore more bytes

    `containing number of "0" to next "1" bit`

    `and number of unchanged bits (if "111" than overflows to next byte)`

    `and bytes after control bytes`

control bits		num of zeros bits			num of unchanched bytes		
1	1	0/1	0/1	0/1	0/1	0/1	0/1

Bitmap Index

Bitmap Compression Example

Example bitmap for one key value

uncompressed

00000000 00000000 00000000 00000000 00000000 00000100 00000000 00000000 00000000 01000000 00000000 00101100 11000000 00000000 00000000 00000001

46 zeros | 27 zeros | 16 zeros | 1 byte | 27 zeros

1 zero

where ___ indicate 1 bitmap group

compressed

00101110	00011011	00010000	11001001	11001100	00011011	- binary
0x2E	0x1B	0x10	0xC9	0xCC	0x1B	- hexadecimal
46	27	16	201	204	27	- decimal

more info:
https://www.juliandyke.com/Presentations/BitmapIndexInternals.ppt
http://www.freepatentsonline.com/5907297.html

Bitmap Index
Compression Optimization

- Bitmap compression has no parameters and can't be turned off
- You can make bitmap smaller (faster) by:
  - use it in columns with low cardinality (small number of distinct values)
  - clustering data in table by bitmap index column

    makes longer zero bit sequences and better bitmap compression ratio

unclustered	US	CA	US	CA	CA	US	CA
clustered	US	US	US	CA	CA	CA	CA

  - tune Håkan Factor using command:

```
ALTER TABLE table_name MINIMIZE RECORDS_PER_BLOCK;
```

```
reduces index size by optimize the mapping of bitmaps to rowids
factor is determined during table creation from metadata (data types and NULL/NOT NULL settings)
not actualized automatically after e.g. column addition
ALTER … MINIMIZE … calculates exact value using full table scan (data needed)
change can effect on next INSERT operations
factor informs how many rows could possibly be in a data block
for bitmap index means -  number of bits allocated for each table data block
table uses the same value in all bitmap indexes -
   - drop bitmap indexes before change
```

100000 rows	8kb blocks	
distinct Keys	B*Tree	Bitmap
1	237	3
2	237	5
5	237	13
10	237	25
100	237	50
1000	237	48
10000	237	87
50000	237	210
100000	237	363

B-tree Index Compression

- In DW only for multicolumn PK and UQ in bigger dimension and metadata tables
- Non unique columns use bitmap indexes in DW (smaller)
- Each leaf row is split into a prefix and a suffix
- Number of columns in prefix (to compress) you can use after COMPRESS keyword

```
CREATE INDEX i1 ON prcss_log_prc (prcss_name, tbl_name, start_time) COMPRESS 1;
```

prcss_log_prc table

prcss_name	tbl_name	start_time	msgs
differential load	sales_fct	<tstamp>	
full load	mkt_dim	<tstamp>	
full load	cust_dim	<tstamp>	
differential load	prod_dim	<tstamp>	
full load	sales_fct	<tstamp>	
full load	cust_dim	<tstamp>	

normal index entries

differential load	prod_dim	<tstamp>	rowid
differential load	sales_fct	<tstamp>	rowid
full load	cust_dim	<tstamp>	rowid
full load	cust_dim	<tstamp>	rowid
full load	mkt_dim	<tstamp>	rowid
full load	sales_fct	<tstamp>	rowid

compressed index entries

prefix	suffix		
differential load	prod_dim	<tstamp>	rowid
	sales_fct	<tstamp>	rowid
full load	cust_dim	<tstamp>	rowid
	cust_dim	<tstamp>	rowid
	mkt_dim	<tstamp>	rowid
	sales_fct	<tstamp>	rowid

- Save storage, fewer IO, query faster, cheaper execution plan option for CBO
- Modification cost not very higher, no locking problem
- Fewer entries in the prefix leads to better compression ratio

Compressed SqlLoader External Table

- Create external table with preprocessor
- Preprocessor is used below example to decompress CSV file

```
CREATE TABLE CUSTOMER_ADDRESS
(   "CA_ADDRESS_ID"              CHAR(16)
  ,"CA_STREET_NAME"             VARCHAR2(60)
  ,"CA_CITY"                    VARCHAR2(60)
  ,"CA_ZIP"                     CHAR(10)
) ORGANIZATION EXTERNAL
(  TYPE ORACLE_LOADER DEFAULT DIRECTORY load_dir
   ACCESS PARAMETERS
   (  RECORDS DELIMITED BY NEWLINE
      PREPROCESSOR exec_dir:'gunzip' OPTIONS '-c'
      BADFILE log_dir: 'CUSTOMER_ADDRESS.bad'
      LOGFILE log_dir: 'CUSTOMER_ADDRESS.log'
      FIELDS TERMINATED BY '|' MISSING FIELD VALUES ARE NULL
      (
          "CA_ADDRESS_ID"
         ,"CA_STREET_NAME"
         ,"CA_CITY"
         ,"CA_ZIP"
      )
   ) LOCATION ('customer_address.csv.gz')
) REJECT LIMIT UNLIMITED;

SELECT * FROM CUSTOMER_ADDRESS;
```

Compressed DataPump External Table

- Data can be unloaded into compressed dump file from source database

```
CREATE TABLE sales_compressed_xt
 ORGANIZATION EXTERNAL
 (
 TYPE ORACLE_DATAPUMP
 DEFAULT DIRECTORY xt_dir
 ACCESS PARAMETERS (COMPRESSION ENABLED)
 LOCATION ( 'sales_compressed_xt.dmp' )
 ) AS
 SELECT *
 FROM fl_sales_fct
 WHERE cntrt_id = 'CSCHRUS';
```

- Then can be loaded into destination database

```
CREATE TABLE sales_compressed_xt ( columns definition )
 ORGANIZATION EXTERNAL
 (
 TYPE ORACLE_DATAPUMP
 DEFAULT DIRECTORY xt_dir
 ACCESS PARAMETERS (COMPRESSION ENABLED)
 LOCATION ( 'sales_compressed_xt.dmp' )
 );

INSERT INTO destination_table
  SELECT * FROM sales_compressed_xt;
```

Saving Space

Uniformed Size Extents

- Defined on tablespace level

```
CRATE TABLESPACE prjts01 UNIFORM SIZE 1MB;
```

- All extents in tablespace have the same size
  - if size to small – metadata overhead – a lot of small extents
  - If to high – waste space if number of rows in segments is small

- Partitions has minimum size 8MB in none US tablespace
  - Please use US TS with 1MB or 512KB extend size to reduce storage cost

- Please place more then 10K rows in one segment
  - More than 100K rows if segment is compressed (good ratio)
  - Choose correct partitioning keys
    - Avoid almost empty partitions or subpartitions

Topic Agenda

Materialized View

- Materialization Overview
- MV Purpose
- Creating
- Refresh
- Query Rewrite
- Explain MV & QR
- SQL Access Advisor

Materialization

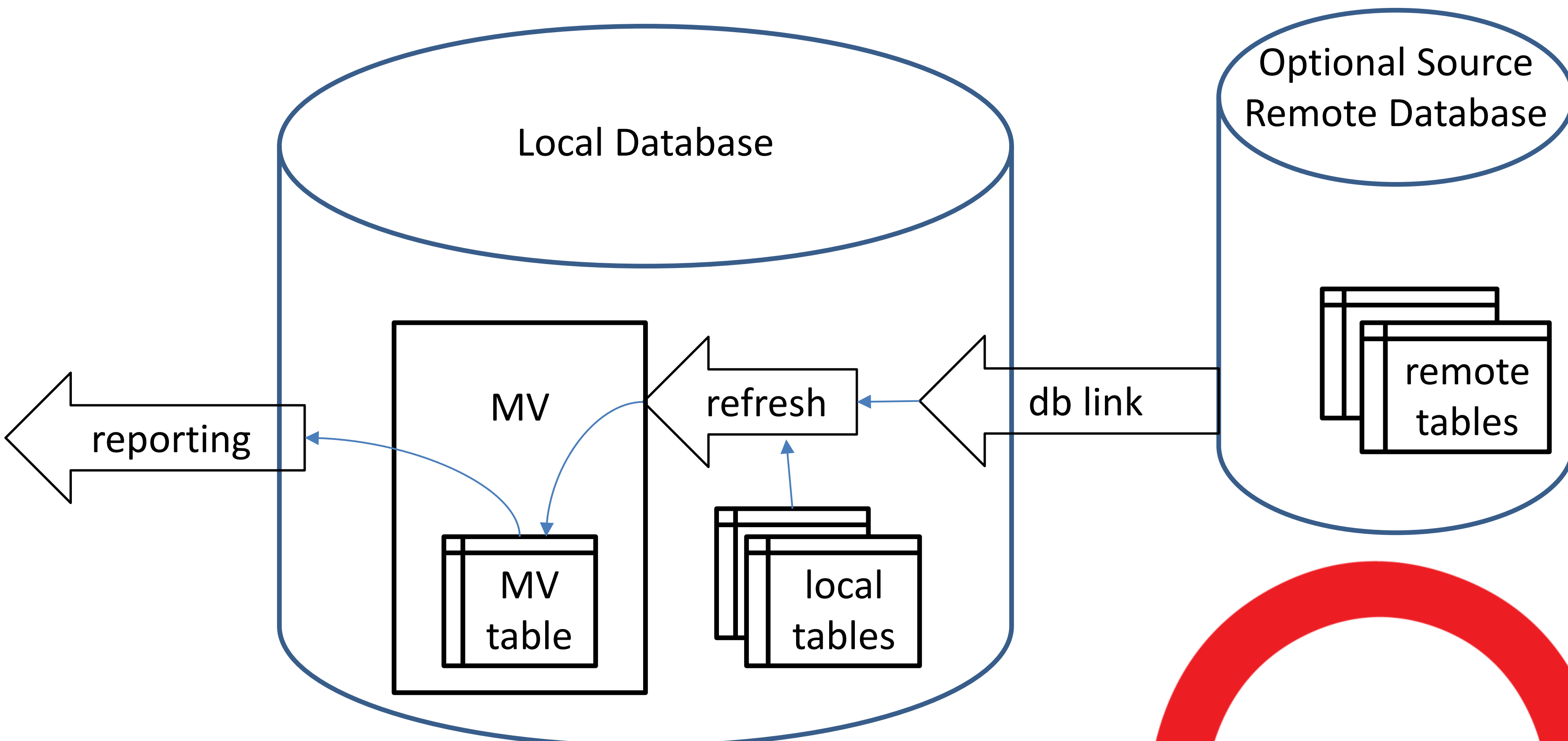
Introduction

- Storing intermediate ETL or reporting results set (typically on disk)
- Beneficial only if result set is used many times
  - Result set costly production is done only ones – improve performance
- If requirement is known on data model creation phase use:
  - if results sets produced and used on single session - Global Temporary Table
  - Otherwise - permanent DFCT or IFCT fact tables
  - If result set is very small and used many times
    - On single session – object type table collection in PL/SQL variable
    - On multiple sessions - Query Result Cache – RESULT_CACHE hint
- If don’t wont to modify data model
  - Server can create and use GTT automatically (SQL tuning training)
    - If undocumented hint /*+ MATERIALIZE */ is used in WHILE clause query
    - During Star Transformation if large dimension table is used many times
  - Otherwise use MV

Materialized View

Overview

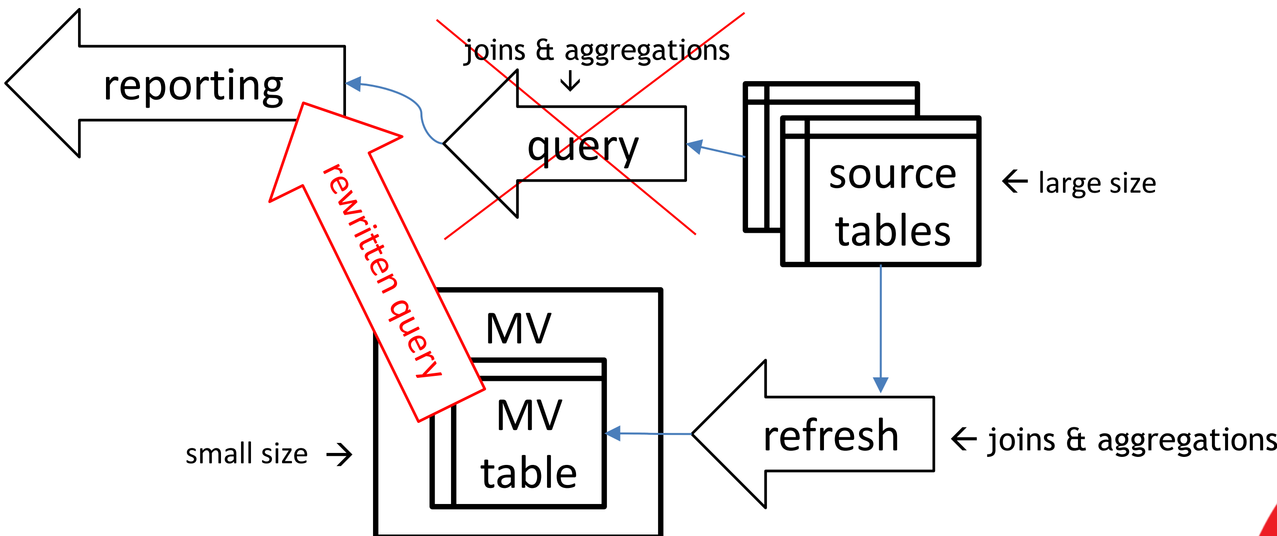
- MV is similar to normal view but internally uses table
- MV table stores MV query resulting rows



Materialized View

Purpose

- Save network if view is used to query remote tables
- Query Rewrite – kind of Oracle Optimizer q-y transformation
  - Server transparently change data source from source tables to MV table
  - Reporting query don't need to reference MV – only MV source tables
  - No need to modify SQL source code to use newly created MV
  - <u>Improve performance</u> without source code query modification



Materialized View

Creating

```
CREATE MATERIALIZED VIEW cust_sales_mv
  PCTFREE 0  TABLESPACE tbs_name
  STORAGE …
  BUILD DEFERRED
  REFRESH COMPLETE
  ENABLE QUERY REWRITE
  AS SELECT c.cust_id, s.chanl_id,
            SUM(sold_amt)
     FROM   sales_fct s, cust_dim c
     WHERE  s.cust_id = c.cust_id
     GROUP BY c.cust_id, s.chanl_id
     ORDER BY c.cust_id, s.chanl_id;
```

Name

Storage options

|IMMEDIATE

When to build it

|FAST|FORCE

|ON COMMIT|ON DEMAND

How to refresh the data

Use this for query rewrite

Source query

Source tables

- SQL Access Advisor is used to generate MV creation recommendations based on current workload

Materialized View

Naming Standard ★★★

- Example name

SHF_M_898C4_505P6_1MV

- SHF – first letters from source fact table name words – SHPMT_HIST_FCT
- M – aggregation level – Monthly
- 898, 505 – hierarchy number
- C, P - dimension – Customer, Product
- 4, 6 – level number of this aggregation in hierarchy

MV

Refresh

- ON COMMIT
  - Automatic refresh after transaction is finished on source tables
  - MV is always fresh
  - Problematic if large number of small transaction is used on source tables
- ON DEMAND
  - Manual refresh using DBMS_MV package procedures
  - Manual refresh using **MV Loader** – PG building block
  - MVL can drop before and create MV after MV table load
  - Prebuild Table is needed under MV to use MVL
  - MVL can be used to load any partitioned tables
  - For none partitioned tables load use Dynamite PG building block
- DBMS_MV
  - COMPLETE – truncate and load all data to MV table
  - FAST – propagate only modifications from source tables

    not recommended for large volume modifications (e.g fact tables)

    MV logs needed on source tables

    For partitioned source tables uses Partition Change Tracking
  - FORCE – FAST if possible otherwise COMPLETE

MV

Prebuild Table

- Existing table can be converted to materialized view
  - When table is already populated with data
  - When want that table to accept query rewrite
- Table must have the same name as MV

```
CREATE TABLE cust_sales_mv ... ;
CREATE MATERIALIZED VIEW cust_sales_mv ... ENABLE_QUERY_REWRITE ... ;
...
DROP MATERIALIZED VIEW cust_sales_mv;
INSERT /* +APPEND */ INTO cust_sales_mv ... ;
CREATE MATERIALIZED VIEW cust_sales_mv ... ENABLE_QUERY_REWRITE ... ;
```

- Using prebuild table is recommended ★ ★ ★
- PG MVL automatically create MV on prebuild table after has finished load

QR

Requirements

- Create MV with ENABLE QUERY REWRITE
- Set **query_rewrite_enabled** parameter to TRUE (default) or FORCE
  - TRUE - Oracle Optimizer decide based on costs estimation, FORCE - deterministic QR use
  - Or use **QUERY_REWRITE hint** in query
- If query text is the same in report and MV then its enough
- Otherwise MV (1 or many) need to have data needed by query
- Data in MV not need to be completed – can be partially processed
- Use dbms_mv.**explain_rewrite** to see what to change to enable QR
- **e.g. query_rewrite_integrity** parameter if is set to
  - ENFORCED

    requires enabled FK constraints on join key columns
  - TRUSTED

    RELY DISABLED FK and Dimension objects are sufficient for QR
  - STALE_TOLERATED

    MV not need to be fresh – QR lead to old data in query results
- If **REWRITE_OR_ERROR** hint is used
  - Query returns error if can't use QR

QR

TRUSTED QR integrity

- Best choice for data warehouse
- Used if MV query use join and have no the same sql text
- **FK** state can be **DISABLE** but with **RELY flag**
  - It means that constraint not check data loaded to fact table - performance reason
  - Integrity should be additionally checked by software (mainly after load)
  - RELY flag means that optimizer trust that data are consistent and can do QR

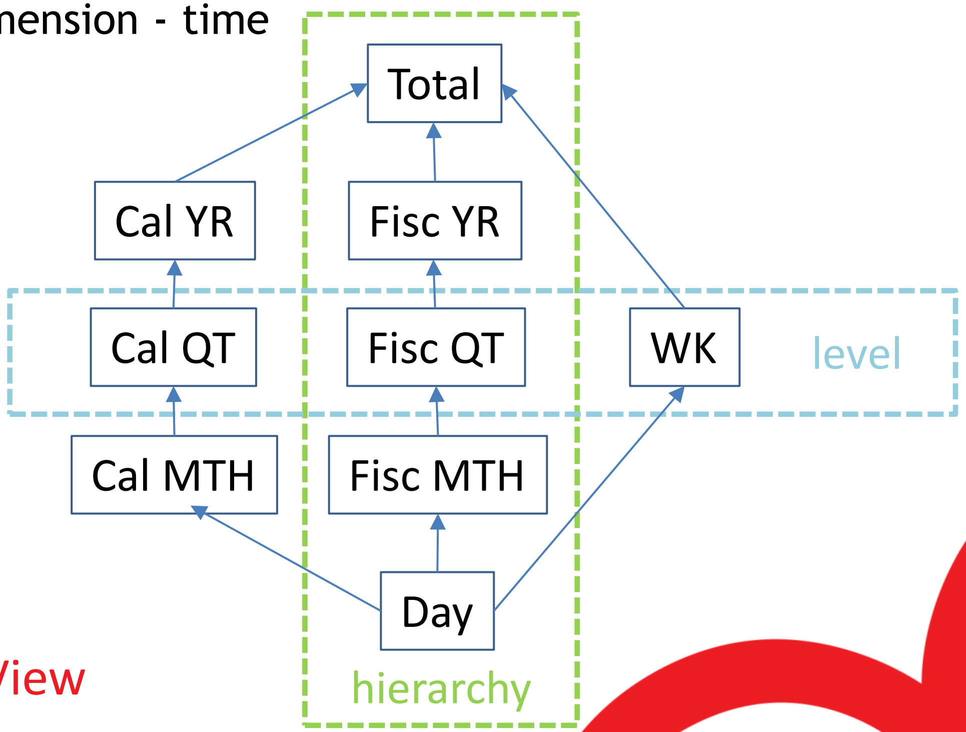
```
ALTER TABLE sales ADD CONSTRAINT sales_time_fk
  FOREIGN KEY (time_id) REFERENCES times (time_id) RELY DISABLE NOVALIDATE;
```

- If data in MV are aggregated on lower level than needed
  - QR is possible but **Oracle Dimension** object should be created
  - Dimension and hierarchy levels show how query need to aggregate between levels

Oracle Dimension

Overview

- Object created in database schema
- **Contains metadata used to describe hierarchies, levels and attributes**
- Build on one or many dimension tables
- Defines parent-child relationship between pairs of column sets (level)
  - Similar to FK constraints but are always in disable state
  - Optimizer uses this relationship with MV **to perform query rewrite**
  - SQL Access Advisor uses these relationships to recommend creation of specific materialized views
- Example Dimension - time



Oracle Dimension

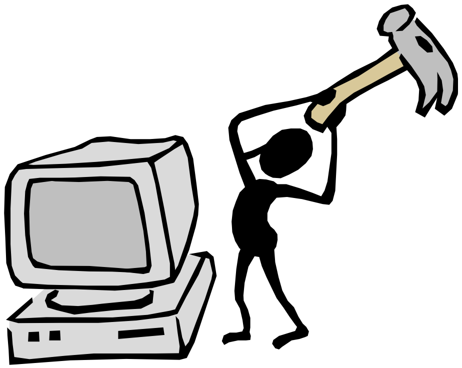
Creating

```
CREATE DIMENSION prod_odim
  LEVEL product            IS (prod_dim.prod_id)
  LEVEL subcategory        IS (prod_dim.prod_sub_categ_name)
  LEVEL category           IS (prod_dim.prod_categ_name)
  HIERARCHY prod_rollup (
    product         CHILD OF
    subcategory     CHILD OF
    category
  )
  ATTRIBUTE product DETERMINES
    (prod_dim.prod_name, prod_dim.prod_desc,
     prod_wght_class_code, prod_uom_name,
     prod_pack_name, prod_sttus_code, prod_list_price, prod_min_price)
  ATTRIBUTE subcategory DETERMINES
    (prod_sub_categ_name, prod_sub_categ_desc)
  ATTRIBUTE category DETERMINES
    (prod_categ_name, prod_categ_desc);
```

Materialized View

Workshop

- Create 1 MV for following 2 reports:
  - Create and populate prebuild table
    Use sh_sales_fct and sh_prod_dim as source
  - Create Oracle Dimension
  - Create MV
  - Check if QR is used for both reports



```
1.
SELECT
       p.prod_categ_name AS categ,
       SUM(s.sold_amt)   AS sales
  FROM sales_fct s
  JOIN prod_dim p ON (s.prod_id = p.prod_id)
  GROUP BY p.prod_categ_name;
2.
SELECT
       p.prod_sub_categ_name AS sub_categ,
       SUM(s.sold_amt)       AS sales
  FROM sales_fct s
  JOIN prod_dim p ON (s.prod_id = p.prod_id)
  WHERE p.prod_categ_name <> 'Smartfones'
GROUP BY p.prod_sub_categ_name;
```

Topic Agenda

Partitioning

- Introduction
- Table Partitioning Methods
  - List
  - Range
  - Interval
  - Hash
- Table Partitioning Types
  - Composite
  - Referential
  - Virtual Column-Based
  - System
- Index Partitioning
  - Local Partitioned
  - Global Partitioned
- Partition DDLs
  - Add, Drop, Merge, Split, Move, Exchange, Truncate
- Recomendations

Partitioning
Introduction

- Dividing large tables and indexes into smaller pieces called:
  - **Partitions** – one partitioning key and method
  - **Subpartitions** – for composite partitioning - two keys and methods
- Value in **partition key** (or subpartition key) column(s) is used to determine which partition (or subpartition) used to store a row
- Advantage
  - Performance – partition pruning, partition-wise join (on SQL tuning training)
  - Manageability – Load and DDL on partitions and moving window
  - Flexibility – names and physical attributes can be set on (sub)partition level
  - Availability – after failure or during lock
- Transparent for application
  - Partitioned object can be used as one object
  - No change in source code after partitioning
  - Referencing partition names in DML - possible but not mandatory
- Very useful for large fact tables in data warehouse

Partitioned Tables

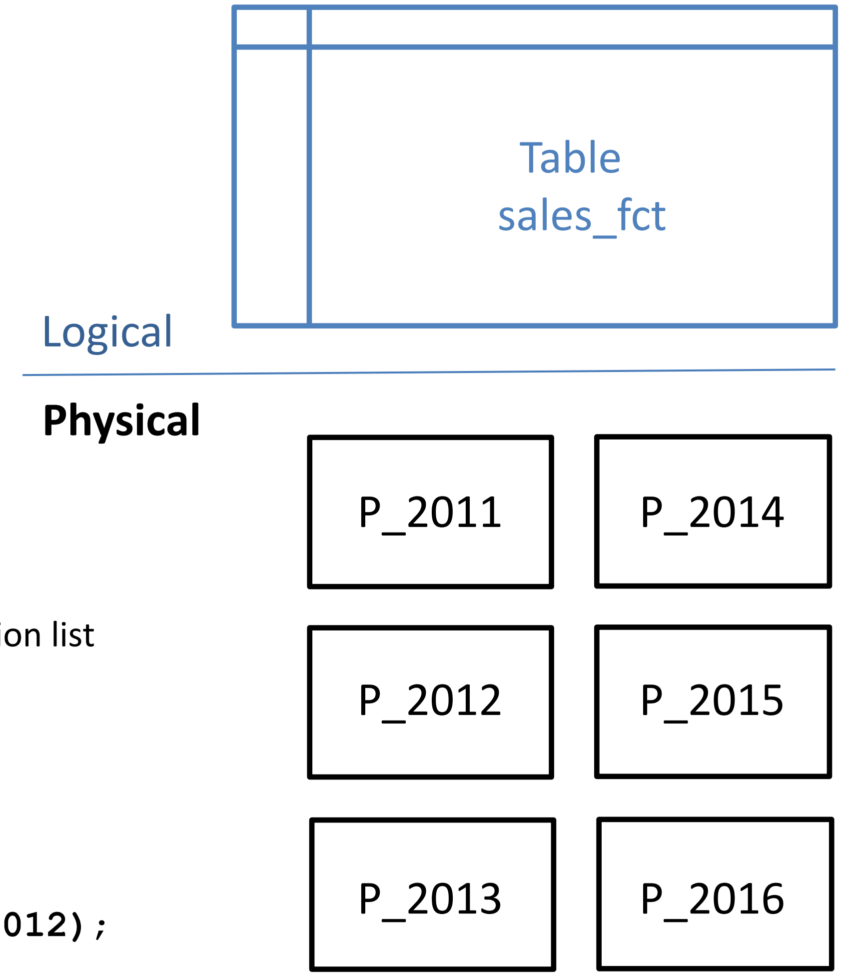
Introduction

– Create partitioned table

```
CREATE TABLE sales_fct
  ...
 PARTITION BY RANGE|LIST|HASH|REFERENCE      ← methods
   (column,[column,...])     ← key
   (PARTITION name attributes,
    PARTITION P_2012 COMPRESS,     ← partition list
    ...)
```

– Referencing

```
SELECT ...
  FROM sales_fct PARTITION (p_2011, p_2012);

INSERT INTO sales_fct PARTITION (p_2014) ...;
```



Partitioning Methods
LIST

- – Full row placement control
  - Each key value is explicitly assigned to partition
- – Problematic when too many key values
  - – All key value need to be assigned to partition
- – No moving window operations
- – PW Join works; **Pruning only for equality predicate**
- – Example

```
PARTITION BY LIST (store_id)
(PARTITION pNorth VALUES IN (3,5,6,9,17),
 PARTITION pEast  VALUES IN (1,2,19,20),
 PARTITION pWest  VALUES IN (4,12,13,14,18),
 PARTITION pCntrl VALUES IN (7,8,15,16,NULL)
 PARTITION pOther VALUES IN (DEFAULT)
)
```

- – Creation of DEFAULT partition is recommended ★ ★

	store_id
pNorth	3,5,6,9,17
pEast	1,2,19,20
pWest	4,12,13,14,18
pCntrl	7,8,15,16,null
pOther	0,10,11,21

Partitioning Methods

RANGE

- – Good for many key values
  - – But values should increase monotonically
- – PW Join works; Pruning for **all** predicates
- – Moving window works
- – Example

ALTER SESSION SET nls_date_format='YYYYMMDD'

```
PARTITION BY RANGE (day_date)
(PARTITION hist  VALUES LESS THEN ('20140101'),
 PARTITION y14h1 VALUES LESS THEN ('20140601'),
 PARTITION y14h2 VALUES LESS THEN ('20150101'),
 PARTITION y15h1 VALUES LESS THEN ('20150601'),
 PARTITION feature VALUES IN (MAXVALUE)
)
```

day_date

hist - **131231**

y14h1 **140101** - **140531**

y14h2 **140601** - **141231**

y15h1 **150101** - **150531**

feature

Partitioning Methods

INTERVAL for RANGE

- Automatically adds partitions for new ranges
- Recursive adds before load into not existing partition

- Example

ALTER SESSION SET nls_date_format='YYYYMMDD'

```
PARTITION BY RANGE (day_date)
INTERVAL(NUMTOYMINTERVAL(6, 'MONTH'))
(PARTITION hist  VALUES LESS THEN ('20140101'),
 PARTITION y14h1 VALUES LESS THEN ('20140601'),
 PARTITION y14h2 VALUES LESS THEN ('20150101'),
 PARTITION y15h1 VALUES LESS THEN ('20150601')
);

INSERT INTO sales_fct (…,day_date)
               VALUES (…'20150808');
```

day_date

hist	- 131231
y14h1	140101 - 140531
y14h2	140601 - 141231
y15h1	150101 - 150531
SYS_Pn	150601 - 151231

Partitioning Methods

HASH

- Easy - no partition list needed
- Only cardinality needed - power of 2 strongly recommended ★★
  - Modify cardinality to add or remove partitions
- Good row distribution -
  - Needed for skewed high cardinality key
- Good performance (load balance)
  - PW Join works; **Pruning only for equality predicate**
- Example

```
PARTITION BY HASH (prod_sub_categ_id)
 PARTITIONS 8
```

1 .. 6000

hash function

SYS_P4322

SYS_P4323

...

SYS_P4329

- No row placement control
  - Data has no logical groupings
  - Hash function used
  - No rolling window
  - No DDLs based on key values ★★
- Exchange don't work - **not recommended** on ETL fact tables
- Server generated names

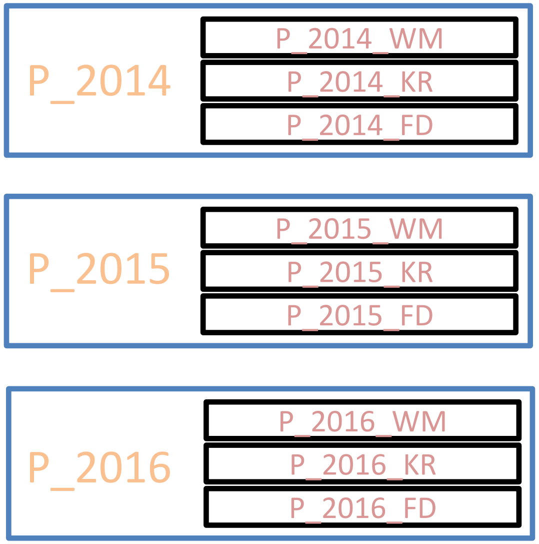
Partitioning Types

Composite

- Enables two methods and keys in one table

RANGE-HASH
RANGE-LIST
RANGE-RANGE
LIST-RANGE
LIST-HASH
LIST-LIST
INTERVAL-RANGE
INTERVAL-LIST
INTERVAL-HASH
HASH-HASH

```
CREATE TABLE sales_fct
  ...
 PARTITION    BY RANGE|LIST|HASH (yr_num)
 SUBPARTITION BY RANGE|LIST|HASH (cust_id)
 PARTITIONS 8 SUBPARTITIONS 4      ← HASH

 SUBPARTITION TEMPLETE             ← LIST
 ( SUBPARTITION WM VALES (WM),
   SUBPARTITION KR VALES (KR), ... [(DEFAULT)]
 )

 ( PARTITION P_2014                ← RANGE
     VALUES LESS THEN (2014),
   PARTITION P_2015
     VALUES LESS THEN (2015), ... [(MAXVALUE)]
 )
```



Partitioning Types

Reference

- Equi-partitioning for FK-PK related tables
- DDL operations on the parent table automatically cascade to the child table.

```
CREATE TABLE parent_emp
  ...
  PARTITION BY LIST (JOB)
  (PARTITION p_job_dba VALUES ('DBA'),
   PARTITION p_job_mgr VALUES ('MGR'),
   PARTITION p_job_vp  VALUES ('VP')
  );

CREATE TABLE reference_emp
  (...
    CONSTRAINT fk_empno FOREIGN KEY(empno)
      REFERENCES parent_emp(empno)
   )
   PARTITION BY REFERENCE (fk_empno)
```

Partitioning Types
Virtual Column - Based

- Virtual column is based on expression using other columns
  - Stored only as metadata - **save storage**
- Virtual column can be used as partitioning key
- Example

```
CREATE TABLE cust_dim
( cust_id NUMBER,
  ...
  region_code AS substr(account_name,1,1)
)
PARTITION BY LIST (region_code)
...
```

Partitioning Types

System

- Not using partitioning key and method
  - Partitioning possible even not suitable column exists for key
- Target partition name have to be specified during DML statement
- Example

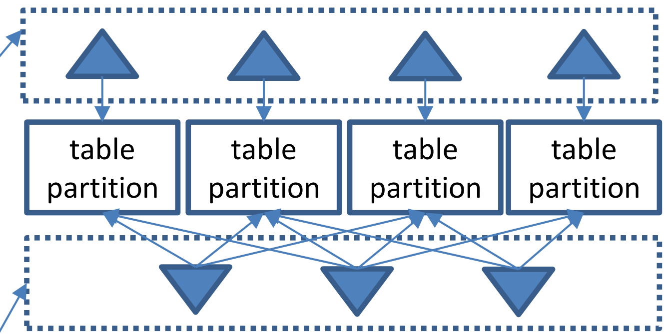
```
CREATE TABLE table_name ...
  PARTITION BY SYSTEM
  (PARTITION p1
   PARTITION p2
   ...
  );

INSERT INTO table_name PARTITION p2 VALUES ...
```

Index Partitioning

Methods

- Global non-partitioned index
- Local partitioned
  - Index is partitioned same way as base table
  - Example

```
CREATE BITMAP INDEX sales_fct_cust_id_bx1
  ON sales_fct (cust_id) LOCAL;
```



- Global partitioned
  - Example

```
CREATE TABLE sales_fct ...
  PARTITION BY RANGE(day_date);
CREATE BITMAP INDEX sales_fct_prod_id_bx2
  ON sales_fct (prod_id) PARTITION BY HASH PARTITIONS 16;
```

  - Better performance - adjusted to reports requirements
  - Bigger size than local partitioned
  - Whole index “UNUSABLE” after DDL on table

```
ALTER table ... UPDATE GLOBAL INDEXES  <- needed
```

Table Partitioning

Example DDLs on Partitions

- Add

```
ALTER TABLE sales_fct ADD PARTITION y15h2 VALUES LESS THEN
('20160101');
```

- Drop

```
ALTER TABLE sales_fct DROP PARTITION hist;
```

- Merge

```
ALTER TABLE sales_fct MERGE PARTITIONS y15h1, y15h2 INTO y15;
```

- Split

```
ALTER TABLE sales_fct SPLIT PARTITION y14h1 INTO
  (PARTITION y14q1 VALUES LESS THAN ('20140401'), PARTITION y14q2);
```

- Move

```
ALTER TABLE sales_fct MOVE PARTITION hist TABLESPACE slow_dsk_ts;
```

- Exchange

```
ALTER TABLE sales_fct EXCHANGE PARTITION y15h2 WITH TABLE sales_tfct;
```

- Truncate

```
ALTER TABLE sales_fct TRUNCATE PARTITION hist;
```

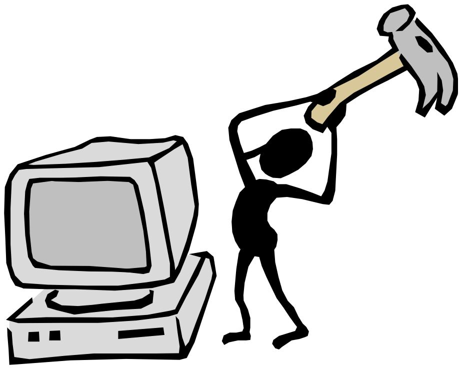
Table Partitioning

Recommendations ★★★

- Number of partitions should be below 10k – meta size – performance
  - Try to keep below 1k
- Number of rows per partition – at least 100k
- To minimize impact during load use staging table and exchange
  - Not partitioned staging (temporary) table have the same structure as partitioned table
- Partitioning design should balance
  - reporting needs – to pruning be possible
  - ease of loading data via partition exchanges during ETL
- Try to avoid load data into not finale partition before split it
- Using global partitioned indexes is not recommended when
  - ETL do split, drop, exchange on base table
- All partition management operations should be performed
  - Automatically by application
  - Not manually by DBA or support
  - e.g. – creating new partitions, purging old data etc.

Table Partitioning
Workshop

- Check partitioning in sales_fct table
- Try insert rows above last partition
- Add new partition and try inset again



Topic Agenda

SQL Parallel Execution

- Introduction
- Architecture
- Using
- PQ DOP - Parallel Query Degree Of Parallelism
  - Manual DOP
  - Automatic DOP & Statement Queuing
  - Resource Manager DOP Limit
- PDML & PDDL

RAC

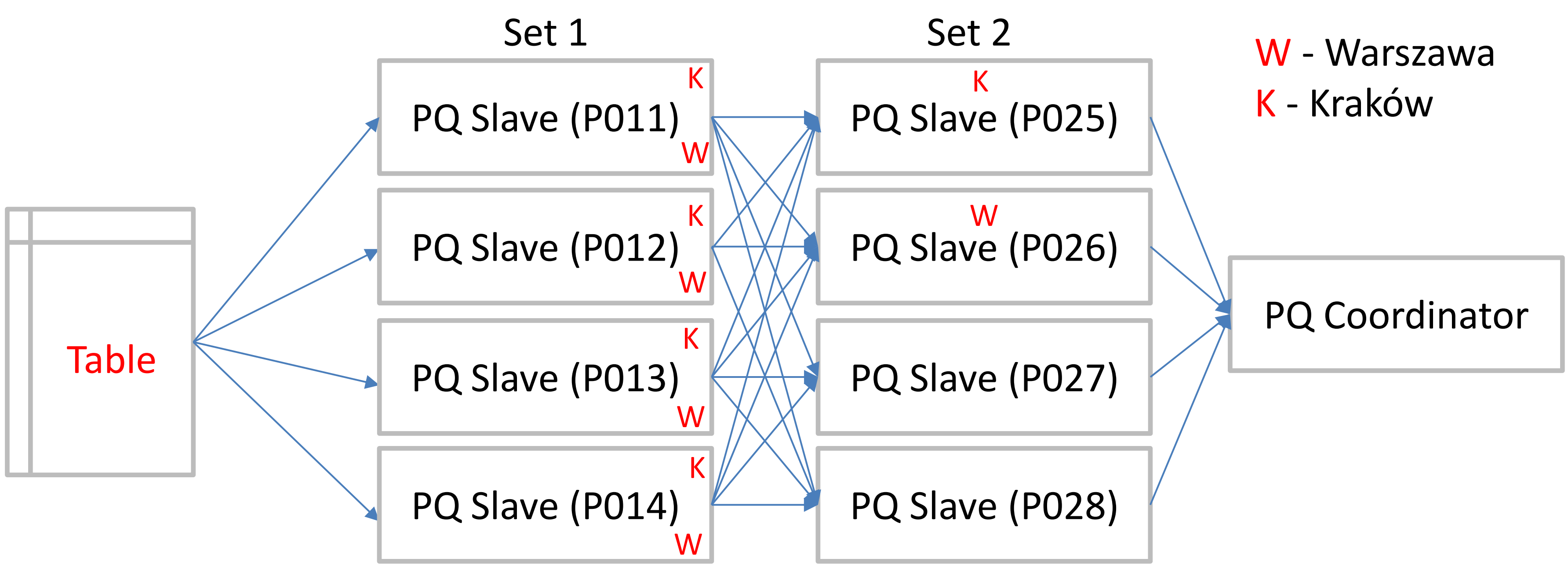
- Architecture

SQL Parallel Execution
Introduction

- PQ - Parallel Query, PDML, PDDL
  - SQL statements where execution is distributed on more then one server process (PX Slave Processes)
- Advantages
  - Execution time is reduced
- Disadvantages
  - Summary execution cost is higher
  - Pitfall with CPU or IO resources saturation
- QC – Query Coordinator
  - Process used to distribute work between Slaves and to consolidate results
- DOP – Degree Of Parallelism
  - Number of threads per SQL execution
- Default DOP = CPU_COUNT * PARALLEL_THREADS_PER_CPU
  - Used if DOP is no explicitly specified
- Slave Set
  - Group of slave processes which do the same work
- One SQL statement execution uses 1 or 2 sets
  - (depending on distribution method – SQL tuning training)

Parallel Query
Architecture

e.g. SELECT /*+ PARALLEL(4) */ SUM(sales_amt) … GROUP BY city_name



DOP = 4 2 slave sets = 8 slaves

SQL Parallel Execution
Using

- Check status on the your session

```
SELECT pdml_status, pddl_status, pq_status
  FROM v$session
  WHERE audsid = USERENV('SESSIONID');
```

PDML_STATUS	PDDL_STATUS	PQ_STATUS
DISABLED	ENABLED	ENABLED

- If enabled server auto decide if to use parallel execution or not
  - Based on estimated execution time and number of available slave processes
- It is possible do turn on/off on session or on statement (using hints/clause)

```
ALTER SESSION ENABLE/DISABLE/FORCE PARALLEL QUERY/DML/DDL PARALLEL(DOP);
```
← session

```
SELECT /*+ PARALLEL */ ...
SELECT /*+ PARALLEL(DOP) */ ...
SELECT /*+ PARALLEL(alias, DOP) */ ...
```

```
SELECT /*+ NOPARALLEL */ ...
```
← query

```
INSERT /*+ APPEND PARALLEL(f, DOP) */
  INTO sales_fct f ...
```
← DML

```
CREATE TABLE sales_fct ...
  PARALLEL(DOP) AS SELECT ...
```
← DDL

SQL Parallel Execution

PQ DOP value

- Is essential for correct parallel execution
  - can be set automatically or manually
- Manual DOP - can be set o 3 levels (in priority order)
  - On statement in hint - best priority

    ```
    SELECT /*+ PARALLEL(8) */ ...
    ```
  - On session

    ```
    ALTER SESSION FORCE PARALLEL QUERY PARALLEL(8);
    ```
  - On table or index in dictionary

    ```
    ALTER TABLE sales_fct PARALLEL 8;
    ```
- Default dictionary DOP is zero
  - means no parallel
  - Default DOP can be also used

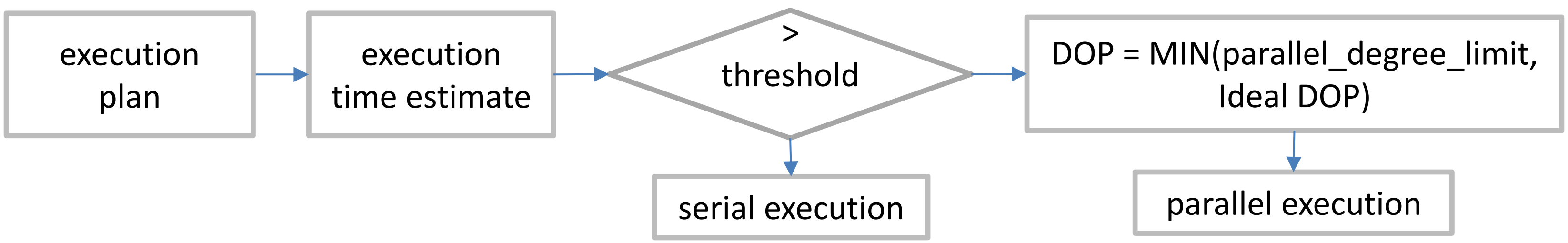
    ```
    ALTER TABLE sales_fct PARALLEL DEFAULT;
    ```
- Base DOP on number of rows in segment

Number of rows	DOP
< 100k	1
100k – 10m	2
10m – 100m	4
100m – 10,000m	6
10,000m – 100,000m	8

SQL Parallel Execution

Automatic DOP

- Determined by server based on execution time estimate




```
ALTER SESSION SET ...
  PARALLEL_MIN_TIME_THRESHOLD = { AUTO | integer }
```

– Integer - minimum estimated execution time in seconds to execute in parallel (with automatic DOP only) default 10 seconds

```
  PARALLEL_DEGREE_POLICY = { MANUAL | LIMITED | AUTO }
```

– LIMITED - for some statements but queuing and in-memory PX is disabled
– AUTO - enables automatic degree, queuing, and in-memory parallel execution (in-memory PQ in SQL tuning training)

```
  PARALLEL_DEGREE_LIMIT  = { CPU | IO | integer }
```

– CPU - DOP = PARALLEL_THREADS_PER_CPU * CPU_COUNT
– IO - DOP = total system throughput / maximum IO bandwidth per process
– Integer - DOP value used for only auto DOP statements

SQL Parallel Execution
Statement Queuing

- Hangs parallel execution until slave processes are available
- Hanged statements waiting in in FIFO queue
- Works if PARALLEL_DEGREE_POLICY = AUTO
- Available Slaves = PARALLEL_SERVERS_TARGET minus slaves currently used
- Servers Target defaults to 4 * CPU_COUNT * PARALLEL_THREADS_PER_CPU * ACTIVE_INSTANCE_COUNT
  - Can be set by DBA
  - To check execute

```
SELECT value FROM v$system_parameter
  WHERE name = 'parallel_servers_target';

SELECT count(*) FROM gv$px_process
  WHERE status = 'IN USE';
```

SQL Parallel Execution
Resource Manager DOP limit

- We can limit DOP differently for different group of users
- Resource Manager directive can limit DOP for particular consumer group
- Active resource plan

```
SELECT name, is_top_plan FROM v$rsrc_plan;
```

- Current session consumer group

```
SELECT resource_consumer_group
  FROM v$session
  WHERE audsid = USERENV('SESSIONID');
```

- Checking DOP limit for consumer_group in current plan

```
SELECT parallel_degree_limit_p1
  FROM dba_rsrc_plan_directives
  WHERE plan='plan'
    AND group_or_subplan='group';
```

SQL Parallel Execution

Parallelism Confirmation

- For current session use

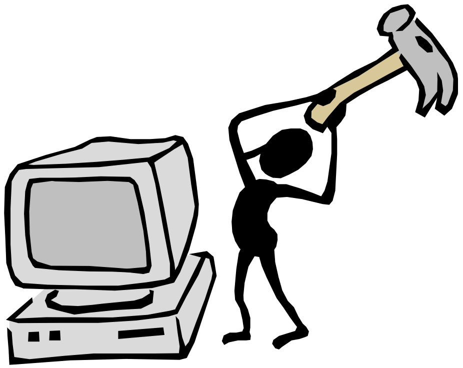
```
SELECT * FROM v$pq_sesstat;
```

STATISTIC	LAST_QUERY	SESSION_TOTAL
Queries Parallelized	0	0
DML Parallelized	0	0
DDL Parallelized	0	0
DFO Trees	0	0
Server Threads	0	0
Allocation Height	0	0
Allocation Width	0	0
Local Msgs Sent	0	0
Distr Msgs Sent	0	0
Local Msgs Recv'd	0	0
Distr Msgs Recv'd	0	0

```
SELECT px_servers_executions,
       executions,
       sql_text
  FROM v$sql
 WHERE sql_id = (
     SELECT prev_sql_id
       FROM v$session
       WHERE audsid = userenv('SESSIONID')
  );
```

SQL Parallel Execution
Workshop

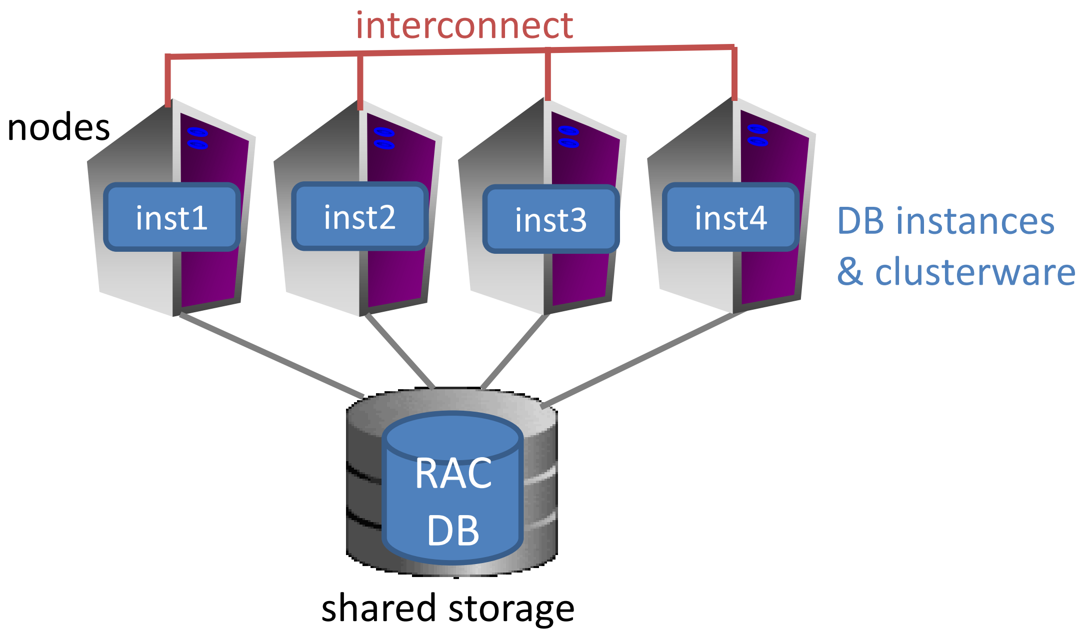
- Load MV prebuild table from MV workshop using PDML
- Confirm if DML is executed in parallel



RAC - Real Application Cluster

Architecture

- Hardware cluster
  - nodes, interconnect, shared storage
- RAC database
  - Opened by many DB instances
  - scalability & ld balance, availability
- Global resources
  - GRD, block, none block, enqueue
- Cache fusion
- GV$ views
  - inst_id column
- Database services
  - name in gv$active_services
- PQ on RAC



Topic Agenda

Warehouse SQL

- Analytical Functions
  - Ranking, Windowing, Syntax, Examples
- Model Clouse
- With Clouse
- Multiple Aggregations
- Multitable Insert
- Merge DML
- DML Error Logging

Analytical Functions
Overview

- Used to compute aggregate value from group-window of rows
- Returns multiple rows per analytical group (not like aggregate functions)
  - if traditional aggregation exists analytical window works on it result and is separated
- For each row sliding window is defined
- Window size can base on number of rows or interval (e.g. time)
- Analytical syntax

```
function_name () OVER (PARTITION BY ...     ← row to window grouping key - optional
                       ORDER BY ...         ← rows order inside window
                       [ROWS|RANGE ...])    ← window size - optional
```

- Results are not additive
- ORDER BY can define NULL placement
  - NULLS { FIRST | LAST }

Analytical Functions

Ranking

- RANK
  - returns rank of a row in an ordered group
  - repeated ordered value returns same rank number
  - gaps in numbers after repeated values
- DENSE_RANK
  - the same but without gaps in numeration
- CUME_DIST
  - cumulative distribution of a value in a group of values
  - number of values less or equal divided by number of rows
  - returns value from 0 to 1
- PERCENT_RANK
  - the same but taking only less and not equal values
  - * 100 = percentile - % of rows with less value

```
SELECT last_name, job_id, salary,

  DENSE_RANK() OVER (PARTITION BY job_id
                     ORDER BY salary DESC)
     AS dense_rank,

  RANK() OVER (PARTITION BY job_id
               ORDER BY salary DESC)
     AS rank

      FROM employees
      WHERE department_id = 50
        AND salary BETWEEN 2500 AND 2700
      ORDER BY job_id, salary DESC
```

LAST_NAME	JOB_ID	SALARY	DENSE_RANK	RANK
OConnell	SH_CLERK	2600	1	1
Grant	SH_CLERK	2600	1	1
Perkins	SH_CLERK	2500	2	3
Sullivan	SH_CLERK	2500	2	3
Mikkilineni	ST_CLERK	2700	1	1
Seo	ST_CLERK	2700	1	1
Matos	ST_CLERK	2600	2	3
Patel	ST_CLERK	2500	3	4
Marlow	ST_CLERK	2500	3	4
Vargas	ST_CLERK	2500	3	4

Analytical Functions

Ranking Example

- With both
  - Traditional grouping - GROUP BY
  - Analytical window grouping - PARTITION BY

```
SELECT job_id, manager_id, sum(salary),
  RANK() OVER (PARTITION BY job_id
               ORDER BY sum(salary) DESC)
    AS rank
  FROM employees
  WHERE salary < 4000
  GROUP BY manager_id, job_id
  ORDER BY job_id, sum(salary) DESC
  ;
```

JOB_ID	MANAGER_ID	SUM(SALARY)	RANK
PU_CLERK	114	13900	1
SH_CLERK	122	12800	1
SH_CLERK	120	11600	2
SH_CLERK	124	11300	3
SH_CLERK	123	9900	4
SH_CLERK	121	6400	5
ST_CLERK	123	12000	1
ST_CLERK	124	11700	2
ST_CLERK	122	10800	3
ST_CLERK	121	10700	4
ST_CLERK	120	10500	5

Analytical Functions
Aggregate Syntax

- Works exactly as traditional aggregate functions
- Uses WITHIN GROUP instead of OVER keyword and no PARTITION BY
- Arguments must match ORDER BY clause in analytical function
- Can use traditional GROUP BY clause
- Available for some functions
  - RANK, DENSE_RANK, PERCENT_RANK, CUME_DIST, LISTAGG
- Example

```
SELECT job_id,
       RANK(15000) WITHIN GROUP
         (ORDER BY salary DESC) AS rank
  FROM employees
  GROUP BY job_id
  ORDER BY 2 DESC
;
```

JOB_ID	RANK
SA_MAN	6
SA_REP	4
AD_VP	3
AD_PRES	2
AC_MGR	2
PU_MAN	2
MK_MAN	2
FI_MGR	2
IT_PROG	1

Analytical Functions

Other Rankings

- ROW_NUMBER - similar to ROWNUM pseudo - column but DO NOT USE ROWNUM in parallel query !
  - similar to RANK but returned values are unique
  - uses different row number for same ORDER BY value

```
SELECT last_name, job_id, salary,

  NTILE(3) OVER (PARTITION BY job_id
                 ORDER BY salary DESC)
     AS ntile,

  ROW_NUMBER() OVER (PARTITION BY job_id
                     ORDER BY salary DESC)
     AS rnum

       FROM employees
       WHERE department_id = 50
         AND salary BETWEEN 2500 AND 2700
       ORDER BY job_id, salary DESC
```

LAST_NAME	JOB_ID	SALARY	NTILE	RNUM
Sullivan	SH_CLERK	2500	1	1
Perkins	SH_CLERK	2500	1	2
OConnell	SH_CLERK	2600	2	3
Grant	SH_CLERK	2600	3	4
Olson	ST_CLERK	2100	1	1
Markle	ST_CLERK	2200	1	2
Philtanker	ST_CLERK	2200	1	3
Landry	ST_CLERK	2400	2	4
Gee	ST_CLERK	2400	2	5
Vargas	ST_CLERK	2500	2	6
Marlow	ST_CLERK	2500	3	7
Patel	ST_CLERK	2500	3	8
Matos	ST_CLERK	2600	3	9

- NTILE(<buckets>)
  - works like histogram generation
  - divides values in group into specified number of buckets

Analytical Functions

WITH_BUCKET(expr, min, max, buckets)

- Construct equi-width histogram
- Range is divided into intervals that have identical size

```
SELECT
    cust_city_name AS city,
    SUM(cust_credt_limit_amt) AS amt,
    WIDTH_BUCKET(SUM(cust_credt_limit_amt), 120000, 1060000, 3)
      AS bckt
  FROM sh_cust_dim
  WHERE cntry_id=52786
  GROUP BY cust_city_name
  ORDER BY 2
;
```

CITY	AMT	BCKT
Lublin	117500	0
Warsaw	201000	1
Gdansk	242500	1
Szczecin	324000	1
Wroclaw	808500	3
Katowice	1052000	3
Krakow	1075000	4

- Below range value = 0
- Above range value = buckets + 1

Analytical Functions

PERCENTILE_COUNT / PERCENTILE_DISC (parameter)

- Inverse (continuous) distribution
- Returns an interpolated value
  - for percentile value (from parameter)
  - respect to the sort specification
  - NULLs are ignored in the calculation
- Can use WITHIN and OVER together
- Both are same if number of rows is odd in group or if 0.5 is used as parameter

```
SELECT last_name, job_id, PERCENTILE_CONT(0.5)
           WITHIN GROUP (ORDER BY salary DESC)
           OVER (PARTITION BY job_id) AS pc
  FROM employees
  WHERE department_id = 50 AND salary < 2700
  ORDER BY job_id;
```

LAST_NAME	JOB_ID	PC
Sullivan	SH_CLERK	2550
Perkins	SH_CLERK	2550
OConnell	SH_CLERK	2550
Grant	SH_CLERK	2550
Olson	ST_CLERK	2400
Markle	ST_CLERK	2400
Philtanker	ST_CLERK	2400
Landry	ST_CLERK	2400
Gee	ST_CLERK	2400
Vargas	ST_CLERK	2400
Marlow	ST_CLERK	2400
Patel	ST_CLERK	2400
Matos	ST_CLERK	2400

```
SELECT job_id, PERCENTILE_CONT(0.5)
           WITHIN GROUP (ORDER BY salary DESC) AS pc
  FROM employees
  WHERE department_id = 50 AND salary < 2700
  GROUP BY job_id ORDER BY job_id;
```

JOB_ID	PC
SH_CLERK	2550
ST_CLERK	2400

Analytical Functions
Windowing Aggregate

- Window clause

```
{ ROWS | RANGE }
{ BETWEEN
  { UNBOUNDED PRECEDING | CURRENT ROW | value_expr { PRECEDING | FOLLOWING } }     ← begin
    AND
  { UNBOUNDED FOLLOWING | CURRENT ROW | value_expr { PRECEDING | FOLLOWING } }
  |
  { UNBOUNDED PRECEDING | CURRENT ROW | value_expr PRECEDING }                      ← end
}
```

- Can by used with analytical syntax and following aggregate functions
  - SUM, AVG, MAX, MIN, COUNT, STDEV, STDDEV_POP, STDDEV_SAMP, (not covered here)
  - VARIANCE, VAR_POP, VAR_SAMP, CORR, COVAR_POP, COVAR_SAMP, REGR_*
  - All rank functions and NTH_VALUE (these functions have only analytical syntax)
- To get first or last row in window use
  - FIRST_VALUE, LAST_VALUE

Analytical Functions

Windowing Aggregate – Examples – maximum sales

– from current and preceding two months

```
SELECT c.cust_id, t.cal_mth_code,  SUM(sold_amt) AS SALES,
      MAX(SUM(sold_amt)) OVER (ORDER BY c.cust_id, t.cal_mth_code
                                ROWS 2 PRECEDING)
          AS MOVING_3_MONTH_MAX
FROM sh_sales_fct s, sh_time_dim t, sh_cust_dim c
WHERE s.time_id=t.time_id AND s.cust_id=c.cust_id AND
      t.cal_yr_num=1999 AND c.cust_id IN (6510)
GROUP BY c.cust_id, t.cal_mth_code
ORDER BY c.cust_id, t.cal_mth_code
```

CUST_ID	CAL_MTH_CODE	SALES	MOVING_3_MONTH_MAX
6510	1999-04	124.69	124.69
6510	1999-05	3395.39	3395.39
6510	1999-06	4080.31	4080.31
6510	1999-07	6434.63	6434.63
6510	1999-08	5104.73	6434.63
6510	1999-09	4676.23	6434.63
6510	1999-10	5108.68	5108.68
6510	1999-11	801.64	5108.68

– from preceding, current and follow days

```
SELECT t.time_id, SUM(sold_amt) AS SALES,
      MAX(SUM(sold_amt))
        OVER (ORDER BY t.time_id
              RANGE BETWEEN
                 INTERVAL '1' DAY PRECEDING
               AND
                 INTERVAL '1' DAY FOLLOWING)
        AS CENTERED_3_DAY_AVG
   FROM sh_sales_fct s, sh_time_dim t
  WHERE s.time_id=t.time_id AND t.cal_wk_num IN (51)
    AND cal_yr_num=1999
  GROUP BY t.time_id ORDER BY t.time_id;
```

TIME_ID	SALES	CENTERED_3_DAY_MAX
1999.12.20 00:00	134336.84	134336.84
1999.12.21 00:00	79015.02	134336.84
1999.12.22 00:00	94264.28	94264.28
1999.12.23 00:00	82745.96	102956.68
1999.12.24 00:00	102956.68	102956.68
1999.12.25 00:00	63107.46	102956.68
1999.12.26 00:00	95122.51	95122.51

Analytical Functions
Windowing Aggregate – Example

- Name of the employee with lowest salary in department

```
SELECT department_id, last_name, salary,
       FIRST_VALUE(last_name)
         OVER (PARTITION BY department_id
         ORDER BY salary ASC
         ROWS UNBOUNDED PRECEDING)
         AS lowest_in_dept
 FROM  employees WHERE department_id < 40
 ORDER BY department_id, salary;
```

DEPARTMENT_ID	LAST_NAME	SALARY	LOWEST_IN_DEPT
10	Whalen	4400	Whalen
20	Fay	6000	Fay
20	Hartstein	13000	Fay
30	Colmenares	2500	Colmenares
30	Himuro	2600	Colmenares
30	Tobias	2800	Colmenares
30	Baida	2900	Colmenares
30	Khoo	3100	Colmenares
30	Raphaely	11000	Colmenares

Analytical Functions

Reporting Aggregate - Example

- Function returns same value within group
- Example: find 3 top-selling products for each product subcategory that contributes more than 20% of the sales within its category

```
SELECT SUBSTR(prod_categ_name,1,8) AS categ, prod_sub_categ_name, prod_name, sales
  FROM (SELECT p.prod_categ_name, p.prod_sub_categ_name, p.prod_name,
             SUM(sold_amt) AS SALES,
             SUM(SUM(sold_amt)) OVER (PARTITION BY p.prod_categ_name)
               AS cat_sales,
             SUM(SUM(sold_amt)) OVER (PARTITION BY p.prod_sub_categ_name)
               AS subcat_sales,
             RANK() OVER (PARTITION BY p.prod_sub_categ_name
                          ORDER BY SUM(sold_amt))
               AS RANK_IN_LINE
         FROM sh_sales_fct s, sh_cust_dim c, sh_cntry_lkp co, sh_prod_dim p
        WHERE s.cust_id = c.cust_id AND c.cntry_id = co.cntry_id
          AND s.prod_id = p.prod_id
          AND s.time_id = to_DATE('11-OCT-2000','DD-MON-YYYY')
        GROUP BY p.prod_categ_name, p.prod_sub_categ_name, p.prod_name
        ORDER BY prod_categ_name, prod_sub_categ_name)
 WHERE (SUBCAT_SALES > 0.2*CAT_SALES) AND (RANK_IN_LINE <= 3)
 ORDER BY SALES DESC;
```

Analytical Functions

Reporting Aggregate – Example Results

- Subquery results

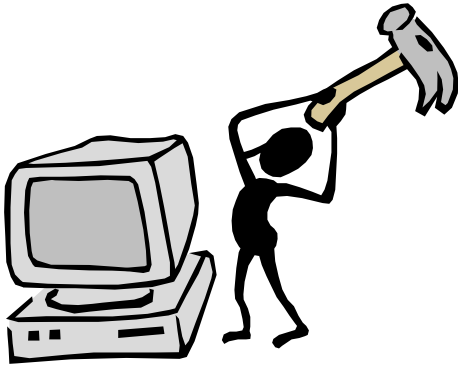
PROD_CATEG_NAME	PROD_SUB_CATEG_NAME	PROD_NAME	SALES	CAT_SALES	SUBCAT_SALES	RANK_IN_LINE
Peripherals and Accessories	Printer Supplies	Model SM26273 Black Ink Cartridge	2104.9	15976.16	15976.16	1
Peripherals and Accessories	Printer Supplies	Model CD13272 Tricolor Ink Cartridge	2264.61	15976.16	15976.16	2
Peripherals and Accessories	Printer Supplies	Model NM500X High Yield Toner Cartridge	11606.65	15976.16	15976.16	3
Software/Other	Bulk Pack Diskettes	3 1/2" Bulk diskettes, Box of 50	~~391.89~~	~~7207.29~~	955.44	1
Software/Other	Bulk Pack Diskettes	3 1/2" Bulk diskettes, Box of 100	~~563.55~~	~~7207.29~~	955.44	2
Software/Other	Recordable CDs	CD-RW, High Speed, Pack of 10	221.27	7207.29	1814.07	1
Software/Other	Recordable CDs	OraMusic CD-R, Pack of 10	239.12	7207.29	1814.07	2
Software/Other	Recordable CDs	CD-R with Jewel Cases, pACK OF 12	246.54	7207.29	1814.07	3
Software/Other	Recordable CDs	CD-R, Professional Grade, Pack of 10	260.28	7207.29	1814.07	~~4~~
Software/Other	Recordable CDs	CD-RW, High Speed Pack of 5	358.96	7207.29	1814.07	~~5~~
Software/Other	Recordable CDs	Music CD-R	487.9	7207.29	1814.07	~~6~~
Software/Other	Recordable DVD Discs	DVD-RW Discs, 4.7GB, Pack of 3	1405.65	7207.29	4437.78	1
Software/Other	Recordable DVD Discs	DVD-R Discs, 4.7GB, Pack of 5	3032.13	7207.29	4437.78	2

- Finale results

CATEG	PROD_SUB_CATEG_NAME	PROD_NAME	SALES
Peripher	Printer Supplies	Model SM26273 Black Ink Cartridge	2104.9
Peripher	Printer Supplies	Model CD13272 Tricolor Ink Cartridge	2264.61
Peripher	Printer Supplies	Model NM500X High Yield Toner Cartridge	11606.65
Software	Recordable CDs	CD-RW, High Speed, Pack of 10	221.27
Software	Recordable CDs	OraMusic CD-R, Pack of 10	239.12
Software	Recordable CDs	CD-R with Jewel Cases, pACK OF 12	246.54
Software	Recordable DVD Discs	DVD-RW Discs, 4.7GB, Pack of 3	1405.65
Software	Recordable DVD Discs	DVD-R Discs, 4.7GB, Pack of 5	3032.13

Analytical Functions
Workshop

- For each product category find the region in which it had max sales
  - Use and modify example from previous slides



Analytical Functions
LAG / LEAD

- Comparing values when the relative positions of rows can be known
  - 2-nd parameter specify the count of rows which separate the target row from the current row
- Can enhance processing speed
  - Provide access to more than one row of a table at the same time without a self-join
- LAG - Access to a row at a given offset **prior** to the current position
- LEAD - Access to a row at a given offset **after** to the current position

- Example

```
SELECT time_id, SUM(sold_amt) AS SALES,
        LAG(SUM(sold_amt),1)  OVER (ORDER BY time_id) AS lag_sales,
        LEAD(SUM(sold_amt),1) OVER (ORDER BY time_id) AS lead_sales
FROM sh_sales_fct
WHERE time_id>=TO_DATE('10-OCT-2000','DD-MON-YYYY')
  AND time_id<=TO_DATE('14-OCT-2000','DD-MON-YYYY')
GROUP BY time_id;
```

TIME_ID	SALES	LAG_SALES	LEAD_SALES
2000.10.10 00:00	238479.49	(null)	23183.45
2000.10.11 00:00	23183.45	238479.49	24616.04
2000.10.12 00:00	24616.04	23183.45	76515.61
2000.10.13 00:00	76515.61	24616.04	29794.78
2000.10.14 00:00	29794.78	76515.61	(null)

Analytical Functions

FIRST / LAST

- Functions can be used for both aggregate and analytic action on the group of logically sorted rows
- Are only functions that deviate from the general syntax of analytic functions
- Do **not** support any <window> clause
- Example - lowest salary in employee department

```
SELECT department_id, last_name, salary,
       MIN(salary) KEEP (DENSE_RANK FIRST
                         ORDER BY salary)
                   OVER (PARTITION BY department_id)
         AS lowest
FROM   employees
ORDER BY  department_id, salary
```

DEPARTMENT_ID	LAST_NAME	SALARY	LOWEST
10	Whalen	4400	4400
20	Fay	6000	6000
20	Hartstein	13000	6000
30	Colmenares	2500	2500
30	Himuro	2600	2500
30	Tobias	2800	2500
30	Baida	2900	2500
30	Khoo	3100	2500
30	Raphaely	11000	2500
40	Mavris	6500	6500

Analytical Functions

Other Useful Functions

DEPARTMENT_ID	JOB_ID	EMPLOYEES
10	AD_ASST	Jennifer
50	SH_CLERK	Nandita,Sarah,Alexis
60	IT_PROG	David,Valli,Diana

- LISTAGG – concatenate strings from group rows into single text

```
SELECT department_id,job_id,
       LISTAGG(first_name, ',') WITHIN GROUP (ORDER BY hire_date) AS employees
  FROM  employees
  WHERE salary BETWEEN 4000 AND 5000
  GROUP  BY department_id,job_id;
```

- RATIO_TO_REPORT - ratio - value to the sum of all values
  - Example - each purchasing clerk's salary to the total of all purchasing clerks' salaries

```
SELECT last_name, salary, RATIO_TO_REPORT(salary) OVER () AS rr
   FROM employees
  WHERE job_id = 'PU_CLERK';
```

LAST_NAME	SALARY	RR
Khoo	3100	0.223
Baida	2900	0.208
Tobias	2800	0.201
Himuro	2600	0.187
Colmenares	2500	0.179

- NTH_VALUE – returns measure value from nth row
  - Example – minimum sold_amt for second channel_id in ascending order for each prod_id

```
NTH_VALUE(MIN(sold_amt), 2)
  OVER (PARTITION BY prod_id
        ORDER BY channel_id
        ROWS BETWEEN UNBOUNDED PRECEDING
                 AND UNBOUNDED FOLLOWING)
```

Analytical Functions

Statistical Functions

See: Oracle® Database
SQL Language Reference
11*g* Release 2 (11.2)

- Descriptive Statistics
  - MEDIAN, STATS_MODE
- Hypothesis Testing - Parametric Tests
  - STATS_T_TEST_ONE, STATS_T_TEST_PAIRED,
  - STATS_T_TEST_INDEP, STATS_T_TEST_INDEPU,
  - STATS_F_TEST, STATS_ONE_WAY_ANOVA
- Crosstab Statistics
  - STATS_CROSSTAB
- Hypothesis Testing - Non-Parametric Tests
  - STATS_BINOMIAL_TEST, STATS_WSR_TEST
  - STATS_MW_TEST, STATS_KS_TEST
- Non-Parametric Correlation
  - CORR_S, CORR_K

Model

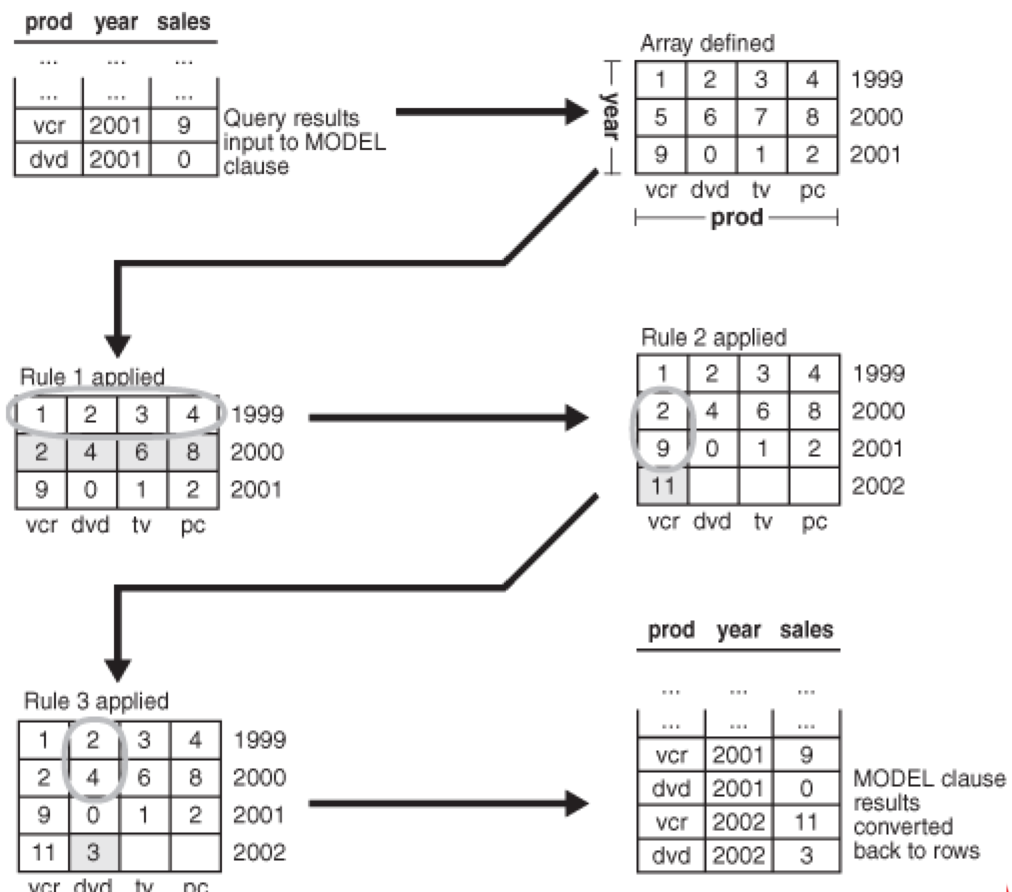
Overview

- Works like spreadsheet
- Create cube from query
- Cube is multitimensional
- Rules do cube transform
- Returns cube as rows

Example

```
SELECT  prod, year, sales
 FROM sales_vw
 MODEL
 DIMENSION BY (prod, year)
 MEASURES (sales s)
 RULES UPSERT
 (s[ANY, 2000]=s[CV(prod), CV(year -1]*2],   --Rule 1
  s[vcr, 2002]=s[vcr, 2001]+s[vcr, 2000],    --Rule 2
  s[dvd, 2002]=AVG(s)[CV(prod), year<2001])  --Rule 3
```

← 2 dimensional cube - array

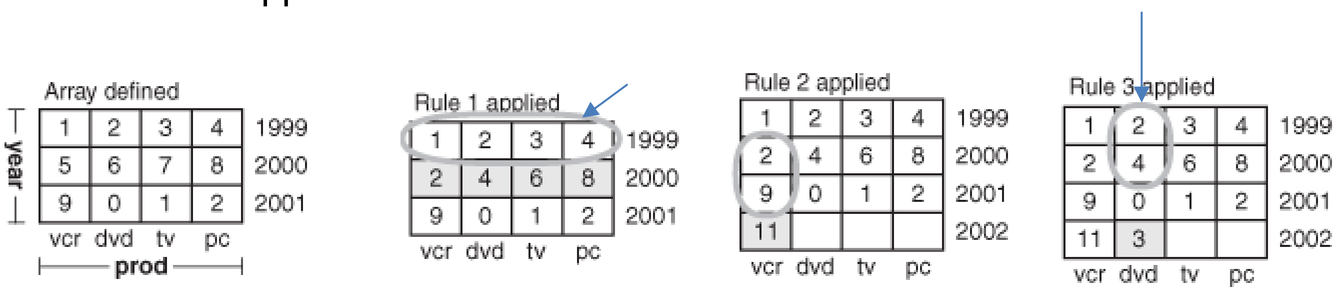


Model

Example Rules Description

```
MODEL
DIMENSION BY (prod, year)
MEASURES (sales s)
RULES UPSERT
s[ANY, 2000] = s[CV(prod), CV(year -1)*2],        --rule 1
s[vcr, 2000] = s[vcr, 2001 + s[vcr, 2000],        --rule 2
s[dvd, 2002] = AVG(s)[CV(prod), year<2001])       --rule 3
```

- rule 1 is applied so the values for 2000 change to vcr=2, dvd=4, tv=6, pc=8
- rule 2 is applied so the value for 2002 vcr = 11
- rule 3 is applied so the value for 2002 dvd = 3



WITH Clouse

Overview

- Predefine named query blocks
- Blocks can be called in main query many times

```
WITH
 <q1 name> AS (SELECT … ),
 <q2_name> AS (SELECT … FROM <q1_name> … ),
 …
SELECT …
  FROM <q2_name> JOIN <q1_name> … JOIN <q1_name> … ;
```

- Improve performance
  - single execution of query block
- Main query is very readable

WITH Clouse

Example – aggregation materialization

```
WITH
  q_sum_sales AS
  (SELECT /*+ MATERIALIZE */
          SUM(quantity) AS all_sales
     FROM sales_fct),
  q_number_stores AS
  (SELECT /*+ MATERIALIZE */
          COUNT(DISTINCT store_id) AS nbr_stores
     FROM sales_fct),
  q_sales_by_store AS
  (SELECT /*+ MATERIALIZE */
          store_name,
          SUM(quantity) AS store_sales
     FROM store_dim NATURAL JOIN sales_fct)
SELECT store_name
  FROM store_dim,
       q_sum_sales,
       q_number_stores,
       q_sales_by_store
 WHERE store_sales > (all_sales / nbr_stores);
```

Star Query Practical Example

```
WITH
  q_costs AS
  ( SELECT
      c.prod_id, t.calendar_year,
      SUM(c.unit_cost) AS costs_amt
    FROM costs_fct c
    INNER JOIN times_dim t
      ON t.time_id=c.time_id
    WHERE c.prod_id IN
        (SELECT prod_id FROM products_dim
           WHERE prod_category = 'Photo')
      AND c.channel_id=2
    GROUP BY c.prod_id,t.calendar_year )
SELECT
  t.calendar_year, p.prod_name,
  SUM(s.quantity_sold * s.amount_sold)
     AS sales_amt,
  SUM(c.costs_amt) AS costs_amt
FROM sales_fct s
  INNER JOIN times_dim t
    ON t.time_id     = s.time_id
  INNER JOIN products_dim p
    ON p.prod_id  = s.prod_id
  INNER JOIN q_costs c
    ON  c.prod_id = s.prod_id
    AND c.calendar_year =
t.calendar_year
WHERE s.prod_id IN
       (SELECT prod_id
          FROM products_dim
          WHERE prod_category = 'Photo')
  AND s.channel_id=2
GROUP BY
  t.calendar_year,
  p.prod_name
HAVING SUM(c.costs_amt) > 0.5 *
  SUM(s.quantity_sold * s.amount_sold)
ORDER BY t.calendar_year
```

WITH Clouse

Recursive WITH Example

- Factorial 5 calculation

```
WITH factorial_tab(val, arg) AS
(
    SELECT 1 val, 1 arg
      FROM dual
  UNION ALL
    SELECT val * (arg + 1), arg + 1
      FROM factorial_tab
     WHERE arg < 10
)
SELECT arg, val fact
  FROM factorial_tab
  JOIN (SELECT 5 AS num FROM dual)
    ON(arg=num);
```

- No WITH factorial 5

```
SELECT ROUND(EXP(SUM(LN(n))))
  FROM
  (
     SELECT LEVEL AS n
       FROM dual
       CONNECT BY LEVEL <= 5
  );
```

Multiple Aggregations

Overview

- Single results from many GROUP BY query without UNION ALL

```
GROUP BY attr1, attr2, attr3                          ← single aggregation

GROUP BY ROLLUP (attr1, attr2, attr3)                 ← 4 aggregations

          … FROM … GROUP BY attr1, attr2, attr3
UNION ALL … FROM … GROUP BY attr1, attr2
UNION ALL … FROM … GROUP BY attr1
UNION ALL … FROM …

GROUP BY CUBE (attr, attr2, attr3)                    ← 6 aggregations

          … FROM … GROUP BY attr1, attr2, attr3
UNION ALL … FROM … GROUP BY attr1, attr2
UNION ALL … FROM … GROUP BY attr1
UNION ALL … FROM …
UNION ALL … FROM … GROUP BY        attr2, attr3
UNION ALL … FROM … GROUP BY               attr3

GROUP BY GROUPING SETS ((a1, a2, a3), (a2, a5), ())   ← customized 3

          … FROM … GROUP BY a1, a2, a3
UNION ALL … FROM … GROUP BY a2, a5
UNION ALL … FROM …
```

Multiple Aggregations
Example

```
SELECT prod_id, chanl_id, promo_id, SUM(sold_amt) AS sold_amt
  FROM sh_sales_fct
  GROUP BY GROUPING SETS ((prod_id, chanl_id), (promo_id), ());
```

Query Result
SQL All Rows Fetched: 233 in 2,286 seconds

	PROD_ID	CHANL_ID	PROMO_ID	SOLD_AMT
223	121	2	(null)	25127.89
224	122	3	(null)	43398.95
225	134	3	(null)	137760.64
226	141	2	(null)	57985.67
227	142	2	(null)	31799.08
228	145	2	(null)	20679.91
229	(null)	(null)	33	277426.26
230	(null)	(null)	350	2199380.9
231	(null)	(null)	351	1224503.26
232	(null)	(null)	999	94504520.79
233	(null)	(null)	(null)	98205831.21

Multiple Aggregations
GROUPING Function

- Function returns 1 if NULL is coming from attribute aggregation

```
SELECT DECODE(GROUPING(cust_marital_sttus_name),1,'ALL',
                       cust_marital_sttus_name) cust_sttus,
       SUM(sold_amt) AS sold_amt
  FROM sh_sales_fct NATURAL JOIN sh_cust_dim
  GROUP BY GROUPING SETS ((cust_marital_sttus_name), ());
```

CUST_STTUS	SOLD_AMT	
Divorc.	47887.28	
Mabsent	5212.59	
Mar-AF	235.95	
Married	161359.36	
NeverM	121361.76	
Separ.	15079.15	
Widowed	15418.87	
divorced	1194829.19	
married	29394927.61	
single	34908947.22	
widow	627160.22	
(null)	31713412.01	← Group identified by NULL status value
ALL	98205831.21	← ALL statuses aggregation

Multitable Insert
Overview

- One statement can populate multiple tables
- Source query is executed only ones
- Row can be loaded to one (FIRST) or many (ALL) tables
    - Syntax

```
INSERT ALL|FIRST
  [WHEN condition THEN] INTO <target_table> [VALUES]
  [WHEN condition THEN] INTO <target_table> [VALUES]
   ...
  [ELSE] INTO <target_table> [VALUES]
  SELECT ...
    FROM <source_table>;
```

Multitable Insert

Examples

- Conditional ALL

```
INSERT ALL
  WHEN (geo_id = 'Poland') THEN
    INTO sales_pl_fct VALUES(…)
  WHEN (chanl_id = 'e-commerce') THEN
    INTO sales_ecmrc_fct VALUES(…)
  SELECT … FROM sales_fct;
```

- Conditional FIRST

```
INSERT FIRST
  WHEN (time_id > ADD_MONTHS(SYSDATE,-6)) THEN
     INTO sales_fct VALUES(…)
  WHEN (time_id > ADD_MONTHS(SYSDATE,-48)) THEN
     INTO sales_hist_fct VALUES(…)
ELSE INTO sales_arch_fct VALUES(…)
  SELECT … FROM sales_ext;
```

Multitable Insert

Examples

- Unconditional

```
INSERT ALL
  INTO sales_fct VALUES(…, sold_amt, sold_qty)
  INTO costs_fct VALUES(…, cost_amt, cost_qty)
  SELECT …, sold_amt, sold_qty,
    FROM finance_sfct;
```

- Unpivoting

```
INSERT ALL
  INTO sales_fct VALUES(…, sold_qrt1)
  INTO sales_fct VALUES(…, sold_qrt2)
  INTO sales_fct VALUES(…, sold_qrt3)
  INTO sales_fct VALUES(…, sold_qrt4)
SELECT …, sold_qrt1, sold_qrt2, sold_qrt3, sold_qrt4
    FROM sales_qtr_fct;
```

Pivot & Unpivot

Examples

PROD_NAME	2_AMT	2_QTY	4_AMT	4_QTY	9_AMT	9_QTY
1.44MB Exter...	60120.52	6455	22167.94	2464	(null)	(null)
128MB Memory...	168783.39	3078	89044.53	1701	(null)	(null)
17" LCD w/bu...	1690316.63	1461	1056793.79	924	(null)	(null)
18" Flat Pan...	1127568.55	1076	1148972.72	1127	204297.73	227

– PIVOT – turn rows to columns

```
SELECT *
  FROM (SELECT prod_name, chanl_id, sold_amt, sold_qty
          FROM sh_sales_fct NATURAL JOIN sh_chanl_dim
                            NATURAL JOIN sh_prod_dim
       )
       PIVOT (SUM(sold_amt) AS amt,
              SUM(sold_qty) AS qty
                FOR chanl_id IN (2, 4, 9)
             )
  ORDER BY prod_name;
```

– UNPIVOT - turn columns to rows

```
SELECT *
  FROM sales_qtr_fct
   UNPIVOT (sold_amt FOR qty IN (sold_qrt1 AS 'Q01',
                                 sold_qrt2 AS 'Q02',
                                 sold_qrt3 AS 'Q03',
                                 sold_qrt4 AS 'Q04'
           )
   UNPIVOT ...;
```

Merge DML

Example

```
MERGE /*+ PARALLEL(f,8) APPEND */
  INTO sales_fct f
  USING TABLE(fn_pl_trans(CURSOR(
          SELECT … FROM sales_extract_ext))) e
     ON (e.prod_id = f.prod_id AND
         e.time_id = f.time_id AND … )
  WHEN MATCHED THEN
    UPDATE SET f.sold_amt = e.sold_amt,
               f.sold_qty = e.sold_qty, …
    DELETE WHERE (f.del_sttus = 'Y')
  WHEN NOT MATCHED THEN
    INSERT (f.prod_id, time_id, … )
    VALUES (e.prod_id, time_id, … );
```

Error Logging in DML

```
desc err$_sales_fct
  ORA_ERR_NUMBER$
  ORA_ERR_MESG$
  ORA_ERR_ROWID$
  ORA_ERR_OPTYP$ - I,U,D
  ORA_ERR_TAG$
  prod_id ...
```

- Logging Table

```
dbms_errorlog.create_error_log(dml_table_name => 'sales_fct');
```

- DML statement can continue on error when error logging is used

```
INSERT INTO dest
   SELECT * FROM source
   LOG ERRORS INTO err$_sales_fct ('INSERT')
   REJECT LIMIT UNLIMITED;
```

- Reexecution not needed (only problematic rows)
- Significantly save time and server resources

Q & A

Oracle Resources

- Oracle Database Documentation Library
  http://docs.oracle.com/cd/E11882_01/index.htm

Workshop Solutions

www.lingaro.com

Bitmap Join Index

Workshop Solution

```
CREATE TABLE sales_fct
  PCTFREE 0 NOLOGGING   COMPRESS   PARTITION BY RANGE ("TIME_ID")
(
 PARTITION "SALES_1995"    VALUES LESS THAN (TO_DATE(' 1996-01-01 00:00:00', 'SYYYY-MM-DD HH24:MI:SS', 'NLS_CALENDAR=GREGORIAN')),
 PARTITION "SALES_1996"    VALUES LESS THAN (TO_DATE(' 1997-01-01 00:00:00', 'SYYYY-MM-DD HH24:MI:SS', 'NLS_CALENDAR=GREGORIAN')),
 PARTITION "SALES_H1_1997" VALUES LESS THAN (TO_DATE(' 1997-07-01 00:00:00', 'SYYYY-MM-DD HH24:MI:SS', 'NLS_CALENDAR=GREGORIAN')),
 PARTITION "SALES_H2_1997" VALUES LESS THAN (TO_DATE(' 1998-01-01 00:00:00', 'SYYYY-MM-DD HH24:MI:SS', 'NLS_CALENDAR=GREGORIAN')),
 PARTITION "SALES_Q1_1998" VALUES LESS THAN (TO_DATE(' 1998-04-01 00:00:00', 'SYYYY-MM-DD HH24:MI:SS', 'NLS_CALENDAR=GREGORIAN')),
 PARTITION "SALES_Q2_1998" VALUES LESS THAN (TO_DATE(' 1998-07-01 00:00:00', 'SYYYY-MM-DD HH24:MI:SS', 'NLS_CALENDAR=GREGORIAN')),
 PARTITION "SALES_Q3_1998" VALUES LESS THAN (TO_DATE(' 1998-10-01 00:00:00', 'SYYYY-MM-DD HH24:MI:SS', 'NLS_CALENDAR=GREGORIAN')),
 PARTITION "SALES_Q4_1998" VALUES LESS THAN (TO_DATE(' 1999-01-01 00:00:00', 'SYYYY-MM-DD HH24:MI:SS', 'NLS_CALENDAR=GREGORIAN')),
 PARTITION "SALES_Q1_1999" VALUES LESS THAN (TO_DATE(' 1999-04-01 00:00:00', 'SYYYY-MM-DD HH24:MI:SS', 'NLS_CALENDAR=GREGORIAN')),
 PARTITION "SALES_Q2_1999" VALUES LESS THAN (TO_DATE(' 1999-07-01 00:00:00', 'SYYYY-MM-DD HH24:MI:SS', 'NLS_CALENDAR=GREGORIAN')),
 PARTITION "SALES_Q3_1999" VALUES LESS THAN (TO_DATE(' 1999-10-01 00:00:00', 'SYYYY-MM-DD HH24:MI:SS', 'NLS_CALENDAR=GREGORIAN')),
 PARTITION "SALES_Q4_1999" VALUES LESS THAN (TO_DATE(' 2000-01-01 00:00:00', 'SYYYY-MM-DD HH24:MI:SS', 'NLS_CALENDAR=GREGORIAN')),
 PARTITION "SALES_Q1_2000" VALUES LESS THAN (TO_DATE(' 2000-04-01 00:00:00', 'SYYYY-MM-DD HH24:MI:SS', 'NLS_CALENDAR=GREGORIAN')),
 PARTITION "SALES_Q2_2000" VALUES LESS THAN (TO_DATE(' 2000-07-01 00:00:00', 'SYYYY-MM-DD HH24:MI:SS', 'NLS_CALENDAR=GREGORIAN')),
 PARTITION "SALES_Q3_2000" VALUES LESS THAN (TO_DATE(' 2000-10-01 00:00:00', 'SYYYY-MM-DD HH24:MI:SS', 'NLS_CALENDAR=GREGORIAN')),
 PARTITION "SALES_Q4_2000" VALUES LESS THAN (TO_DATE(' 2001-01-01 00:00:00', 'SYYYY-MM-DD HH24:MI:SS', 'NLS_CALENDAR=GREGORIAN')),
 PARTITION "SALES_Q1_2001" VALUES LESS THAN (TO_DATE(' 2001-04-01 00:00:00', 'SYYYY-MM-DD HH24:MI:SS', 'NLS_CALENDAR=GREGORIAN')),
 PARTITION "SALES_Q2_2001" VALUES LESS THAN (TO_DATE(' 2001-07-01 00:00:00', 'SYYYY-MM-DD HH24:MI:SS', 'NLS_CALENDAR=GREGORIAN')),
 PARTITION "SALES_Q3_2001" VALUES LESS THAN (TO_DATE(' 2001-10-01 00:00:00', 'SYYYY-MM-DD HH24:MI:SS', 'NLS_CALENDAR=GREGORIAN')),
 PARTITION "SALES_Q4_2001" VALUES LESS THAN (TO_DATE(' 2002-01-01 00:00:00', 'SYYYY-MM-DD HH24:MI:SS', 'NLS_CALENDAR=GREGORIAN')),
 PARTITION "SALES_Q1_2002" VALUES LESS THAN (TO_DATE(' 2002-04-01 00:00:00', 'SYYYY-MM-DD HH24:MI:SS', 'NLS_CALENDAR=GREGORIAN')),
 PARTITION "SALES_Q2_2002" VALUES LESS THAN (TO_DATE(' 2002-07-01 00:00:00', 'SYYYY-MM-DD HH24:MI:SS', 'NLS_CALENDAR=GREGORIAN')),
 PARTITION "SALES_Q3_2002" VALUES LESS THAN (TO_DATE(' 2002-10-01 00:00:00', 'SYYYY-MM-DD HH24:MI:SS', 'NLS_CALENDAR=GREGORIAN')),
 PARTITION "SALES_Q4_2002" VALUES LESS THAN (TO_DATE(' 2003-01-01 00:00:00', 'SYYYY-MM-DD HH24:MI:SS', 'NLS_CALENDAR=GREGORIAN')),
 PARTITION "SALES_Q1_2003" VALUES LESS THAN (TO_DATE(' 2003-04-01 00:00:00', 'SYYYY-MM-DD HH24:MI:SS', 'NLS_CALENDAR=GREGORIAN')),
 PARTITION "SALES_Q2_2003" VALUES LESS THAN (TO_DATE(' 2003-07-01 00:00:00', 'SYYYY-MM-DD HH24:MI:SS', 'NLS_CALENDAR=GREGORIAN')),
 PARTITION "SALES_Q3_2003" VALUES LESS THAN (TO_DATE(' 2003-10-01 00:00:00', 'SYYYY-MM-DD HH24:MI:SS', 'NLS_CALENDAR=GREGORIAN')),
 PARTITION "SALES_Q4_2003" VALUES LESS THAN (TO_DATE(' 2004-01-01 00:00:00', 'SYYYY-MM-DD HH24:MI:SS', 'NLS_CALENDAR=GREGORIAN'))
)
 AS SELECT * FROM sh_sales_fct WHERE chanl_id=4;

CREATE TABLE cust_dim AS SELECT * FROM sh_cust_dim;
ALTER TABLE cust_dim ADD CONSTRAINT cust_dim_pk PRIMARY KEY (cust_id);

CREATE BITMAP INDEX sales_cust_gndr_bx1
  ON sales_fct(c.cust_gendr_code)
  FROM sales_fct s, cust_dim c
  WHERE s.cust_id = c.cust_id
  LOCAL NOLOGGING;

SELECT /*+INDEX(s)*/
    SUM(sold_amt)
  FROM sales_fct s, cust_dim c
  WHERE s.cust_id = c.cust_id
    AND c.cust_gendr_code='F';
```

Table Compression

Workshop Solution

```
CREATE TABLE sales_fct_c
  PCTFREE 0 NOLOGGING   COMPRESS BASIC
  AS SELECT * FROM sales_fct;

CREATE TABLE sales_fct_cs
  PCTFREE 0 NOLOGGING   COMPRESS BASIC
  AS SELECT * FROM sales_fct order by prod_id;

CREATE TABLE sales_fct_nc
  PCTFREE 0 NOLOGGING   NOCOMPRESS
  AS SELECT * FROM sales_fct;

exec dbms_stats.gather_table_stats('USER00','sales_fct_c');
exec dbms_stats.gather_table_stats('USER00','sales_fct_cs');
exec dbms_stats.gather_table_stats('USER00','sales_fct_nc');
exec dbms_stats.gather_table_stats('USER00','sales_fct');

SELECT table_name, ROUND(avg_row_len * num_rows/bytes, 2) AS COMPR_RATIO
  FROM user_tables t
  JOIN ( SELECT segment_name, SUM(bytes) AS bytes
           FROM user_segments
           WHERE segment_name LIKE '%SALES_FCT%'
           GROUP BY segment_name
       ) ON (segment_name = table_name)
  WHERE table_name LIKE '%SALES_FCT%';
```

Materialized View Workshop Solution

```
CREATE TABLE prod_dim
  AS SELECT * FROM sh_prod_dim;
ALTER TABLE prod_dim ADD
  CONSTRAINT prod_dim_pk PRIMARY KEY (prod_id);

CREATE TABLE prod_sales_mv
  PCTFREE 0 COMPRESS BASIC
  AS  SELECT p.prod_categ_name,
             p.prod_sub_categ_name,
             SUM(sold_amt)   sold_amt
  FROM sales_fct s
  JOIN prod_dim p
    ON (s.prod_id = p.prod_id)
  GROUP BY p.prod_categ_name,
           p.prod_sub_categ_name
  ORDER BY prod_categ_name;

CREATE MATERIALIZED VIEW prod_sales_mv
  ON PREBUILT TABLE
  ENABLE QUERY REWRITE
  AS  SELECT p.prod_categ_name,
             p.prod_sub_categ_name,
             SUM(sold_amt)   sold_amt
  FROM sales_fct s
  JOIN prod_dim p
    ON (s.prod_id = p.prod_id)
  GROUP BY p.prod_categ_name,
           p.prod_sub_categ_name
  ORDER BY prod_categ_name;
```

```
CREATE DIMENSION prod_odim
  LEVEL product            IS (prod_dim.prod_id)
  LEVEL subcategory        IS
(prod_dim.prod_sub_categ_name)
  LEVEL category           IS (prod_dim.prod_categ_name)
  HIERARCHY prod_rollup (
    product          CHILD OF
    subcategory      CHILD OF
    category
  )
  ATTRIBUTE product DETERMINES
    (prod_dim.prod_name, prod_dim.prod_desc,
prod_sttus_code)
  ATTRIBUTE subcategory DETERMINES
    (prod_sub_categ_name, prod_sub_categ_desc)
  ATTRIBUTE category DETERMINES
    (prod_categ_name, prod_categ_desc);

show parameter rewrite
alter session set query_rewrite_integrity = trusted;

SELECT
       p.prod_categ_name AS categ,
       SUM(s.sold_amt)   AS sales
  FROM sales_fct s
  JOIN prod_dim p ON (s.prod_id = p.prod_id)
  GROUP BY p.prod_categ_name;

SELECT
       p.prod_sub_categ_name AS sub_categ,
       SUM(s.sold_amt)       AS sales
  FROM sales_fct s
  JOIN prod_dim p ON (s.prod_id = p.prod_id)
  WHERE p.prod_categ_name <> 'Smartfones'
GROUP BY p.prod_sub_categ_name;
```

Table Partitioning

Workshop Solution

```
SELECT partition_name, high_value
  FROM user_tab_partitions
  WHERE table_name = 'SALES_FCT'
  ORDER BY partition_position DESC;

INSERT INTO sales_fct
   (PROD_ID, CUST_ID,
    TIME_ID, CHANL_ID,
    PROMO_ID,
    SOLD_QTY, SOLD_AMT)
  VALUES (15,8586,
          to_date('2004.02.01 00:00',
                  'YYYY.MM.DD HH24:MI'),
          4,999,1,999.99);

ALTER TABLE sales_fct ADD PARTITION
  SALES_Q1_2004  VALUES LESS THAN
    (TO_DATE(' 2004-04-01 00:00:00',
             'SYYYY-MM-DD HH24:MI:SS',
             'NLS_CALENDAR=GREGORIAN'))
;
```

SQL Parallel Execution

Workshop Solution

```
DROP MATERIALIZED VIEW prod_sales_mv;

TRUNCATE TABLE prod_sales_mv;

INSERT /*+APPEND PARALLEL(t, 4)*/ INTO prod_sales_mv t
  SELECT p.prod_categ_name,
         p.prod_sub_categ_name,
         SUM(sold_amt)   sold_amt
  FROM sh_sales_fct s
  JOIN prod_dim p
    ON (s.prod_id = p.prod_id)
  GROUP BY p.prod_categ_name,
           p.prod_sub_categ_name
  ORDER BY prod_categ_name;

  SELECT prev_sql_id
    FROM v$session
    WHERE audsid = userenv('SESSIONID')
;
COMMIT;

SELECT px_servers_executions, executions, sql_text
  FROM v$sql
 WHERE sql_text LIKE 'INSERT /*+APPEND PARALLEL%';

CREATE MATERIALIZED VIEW prod_sales_mv ...;
```

Analytical Functions
Workshop Solution

- For each product category find the region in which it had max sales

```
SELECT prod_categ_name, cntry_regn_name, sold_amt
   FROM (SELECT SUBSTR(p.prod_categ_name,1,8)     AS prod_categ_name,
                 co.cntry_regn_name, SUM(sold_amt) AS sold_amt,
                    MAX(SUM(sold_amt)) OVER (PARTITION BY prod_categ_name)
                      AS MAX_REG_SALES_AMT
            FROM sh_sales_fct s, sh_cust_dim c, sh_cntry_lkp co, sh_prod_dim p
            WHERE s.cust_id = c.cust_id AND c.cntry_id = co.cntry_id
              AND s.prod_id = p.prod_id AND s.time_id = TO_DATE('11-OCT-2001','DD-MON-YYYY')
            GROUP BY prod_categ_name, cntry_regn_name)
   WHERE sold_amt = MAX_REG_SALES_AMT
```