

# Data Warehouse Optimizing Data Loading and Data Querying

Presented by:  
Andreas Katsaris  
Arisant, LLC

- Warehouse definition and terminology
- Warehouse design aspects and considerations
- Oracle specific technologies
  - ETL (Extraction, Transformation, Loading)
  - Querying
  - Index, constraint and statistics management

# Data Warehouse Mission Arisant

- Gather, integrate and reconcile operational, decision support, and external data
- Provide meaningful, accessible, consistent, and easy to understand business information to enterprise users
- Act as a single integrated source of data for processing information

# What is a Data Warehouse? Arisant

- a relational database designed for query and analysis
- contains historical data derived from transaction data, but it can include data from other sources
- enables an organization to consolidate data from several sources

# What is a Data Warehouse? Arisant

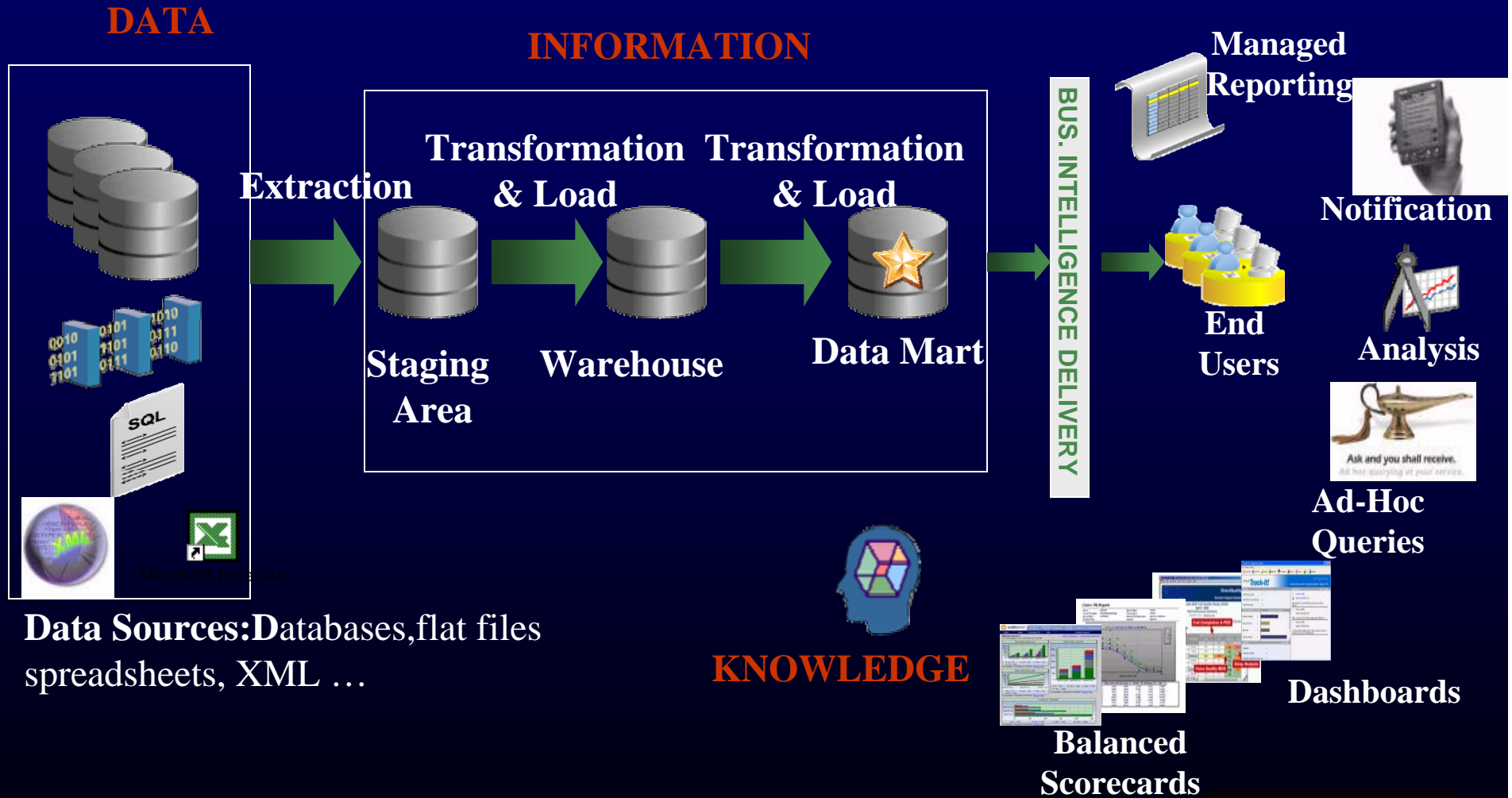
- Components of a warehouse environment
  - extraction, transformation, and loading (ETL) engine
  - online analytical processing (OLAP) solution
  - client analysis tools
  - other applications that manage the delivery of data to enterprise users

## Warehouse design aspects and considerations

- Data Warehouse Architecture (Basic)
- Data Warehouse Architecture (with Staging Area)
- Data Warehouse Architecture (with Staging Area and Data Marts)

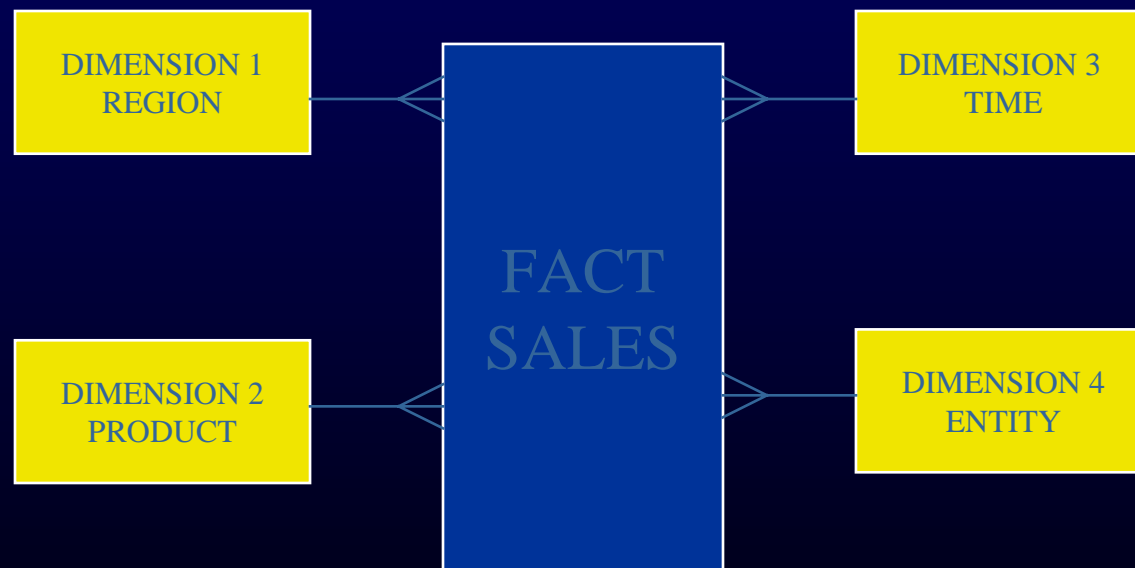
# Data Warehouse Architecture (with Staging Area and Data Marts)

# Arisant



- Evaluate candidate technologies for the data warehouse and front-end tools
- Evolutionary/iterative process, avoid waterfall – ‘big bang’ approach
- Start with one subject area and one target user group
- Identify data sources
- Perform Data Profiling

- Intuitive to end-users
- Mirrors the way people think about the the business



- FACT table
  - ‘skinny’ and large
  - Contains numeric measurements and FKs to dimensions
  - FK columns – bitmap indexes
  - Consider partitioning and archiving aspects
- Dimensions
  - Denormalized – small and ‘wide’
  - Contain business attributes relating to the measurements
- Avoid snowflake if possible

- Oracle specific technologies for ETL and querying optimization
- Index Management
- Constraint Management
- Statistics Management

- 3<sup>rd</sup> party Tools
  - Informatica - SAS
  - Ab Initio - DataStage
- Oracle Specific
  - PL/SQL
  - Replication
  - Transportable Tablespaces
  - External Tables (access to flat files)
  - SqlLoader (access to flat files)
  - CDC (Change Data Capture)
  - Streams
  - export/import/data pump
  - Warehouse Builder

- Easy to write, flexible, fast deployment
- Extraction
  - CTAS operations with NOLOGGING
  - INSERT ... SELECT with APPEND and PARALLEL hints
  - Ideal for incremental extractions (e.g. timestamp based)
- Transformations/Loading
  - Pipelined functions, MERGE, Multitable INSERT, INSERT FIRST
  - Parallel processing
    - Reference
      - [http://asktom.oracle.com/pls/asktom/f?p=100:11:0::::P11\\_QUESTION\\_ID:10498431232211](http://asktom.oracle.com/pls/asktom/f?p=100:11:0::::P11_QUESTION_ID:10498431232211)

- BULK operations
  - BULK COLLECT
    - bulk binds with SELECT statements
    - fetch into user defined array or PL/SQL table
  - FORALL
    - bulk binds with INSERT, UPDATE, and DELETE statements
  - RETURNING INTO
    - Retrieving DML Results into a Collection

- Snapshot logs on source system
- Snapshot technology can be extended to produce/preserve exact row changes
- Resistance from source system owners
  - usually complain about performance impact
  - space requirements for snapshot logs

- copying of datafiles (tablespaces in read only mode)
- move both table and index data avoids index rebuilds
- Cross platform support
- When to consider:
  - Lots of source system changes
  - Required tables can be placed in specific tablespace(s)
  - No timestamps so incremental extraction not possible
  - Other options not allowed (streams, replication, CDC)

- routes published information to subscribed destinations
- only changes to desired objects are captured
- Architecture
  - Capture
  - Staging/Propagation
  - Consumption
  - Parallel processing of log files
- Non-intrusive
  - No triggers or snapshot logs
  - uses redo log/archive log
- Custom Transformations possible
- Access via Oracle-supplied PL/SQL packages or Enterprise Manager Console

# Change Data Capture

Arisant

- simplifies the process of identifying changed data since the last extraction (incremental extraction)
- architecture is based on publisher-subscriber model
- 9i – synchronous (part of transaction)
  - change data is captured via triggers and stored inside the database in change tables
- 10g – may be setup as asynchronous (not part of transaction)
  - lightweight Oracle Streams application
  - Changes extracted from the log files
- Captured data made available to the target systems in a controlled manner (subscription window), using database views

- Files stored outside the database
- Read only in 9i – Read/Write in 10g(CTAS)
- DML/index creation not allowed
- Can be read with SQL as if they were tables
  - May be joined with database tables
  - Filtering allowed with WHERE clause
- Can be used to load staging area or act as a staging area themselves

- Next generation export/import tools
- Various interfaces
  - expdp / impdp
  - Web based GUI via Database Control
  - DBMS\_DATAPUMP
- Jobs (exports and loads) are interruptible and resumable
- Parallelism
- Fine-grain object selection
- Allows data movement via db links!
  - Works in a pipelined fashion

## Oracle Warehouse Technologies and Features

- Resumable operations
- Direct Path operations
  - NOLOGGING
  - APPEND hint
- Parallelism
- Partitioning

# Resumable Operations

Arisant

- Instance level
  - init.ora: RESUMABLE\_TIMEOUT = 3600
- Session level
  - ALTER SESSION ENABLE RESUMABLE <TIMEOUT secs>;
  - ALTER SESSION DISABLE RESUMABLE;
- Used for:
  - DDL
    - **CREATE INDEX, CTAS ...**
  - DML
    - **INSERT/UPDATE**
- Detection:
  - Look in DBA\_RESUMABLE where STATUS='SUSPENDED'

- NOLOGGING option, APPEND hint
- Different behavior between database log modes
- No archive log mode
  - Table maybe in LOGGING or NOLOGGING
  - /\*+ append \*/ does not generate redo/undo
- Archive log mode
  - Table in NOLOGGING - /\*+ append \*/ does not generate redo/undo
  - Table in LOGGING - /\*+ append \*/ generates redo but no undo
- CTAS ... NOLOGGING
- INSERT /\*+ append \*/ INTO ... SELECT ...
  - exclusive locks on the table (no parallel streams, use PARALLEL hint instead)
  - No enabled FKs (hint ignored)
  - Index maintenance deferred to end to direct path insert operation
- CREATE INDEX .... NOLOGGING;

- Queries requiring large table scans, joins, or partitioned index scans
- Creation of large indexes
- Creation of large tables (including materialized views)
- Bulk inserts, updates, merges, and deletes
- Understand related init.ora parms and set appropriately

- Types:
  - Parallel Query
  - Parallel DDL
  - Parallel DML
- Implementation:
  - via a hint
    - `/*+ PARALLEL(<table name>,<degree of parallelism>) */`
  - at the object definition level
    - `CREATE/ALTER TABLE finance_trans PARALLEL 4;`
    - `CREATE/ALTER INDEX ... PARALLEL 2;`

- DOP – the number of parallel execution servers associated with a single operation is known as the degree of parallelism.
- Examples

```
INSERT /*+ APPEND PARALLEL (t3,2) */ INTO t3  
SELECT /*+ PARALLEL */ * FROM ...
```

```
DELETE /*+ PARALLEL (t1, 2) */ FROM t1 ...
```

```
SELECT /*+ PARALLEL (t1, 4) */ COUNT(*) FROM  
t1 WHERE ...
```

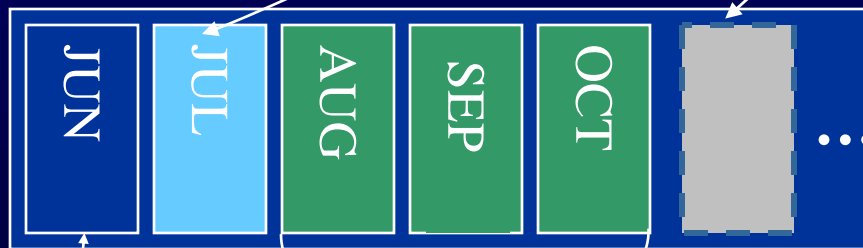
- Types
  - Range Partitioning
  - Hash Partitioning
  - List Partitioning
  - Composite Partitioning

- Partition Pruning (a.k.a partition elimination)
  - evaluate WHERE clause to eliminate unneeded partitions
  - Reduces unnecessary I/O
  - Improved query performance
- Partition operations to facilitate big loads
  - Load multiple partitions in parallel
  - Load stand alone table - EXCHANGE partition
  - Index maintenance at the partition level

# Partitioning

Arisant

Partition key is TRANS\_DT



- Load empty table
- Validate data
- Index maintenance
- EXCHANGE partition

Archive 'old' data  
with EXCHANGE  
partition

```
Select * from  
finance_trans where  
TRANS_DT = july  
  
Select * from  
finance_trans where  
TRANS_DT between  
august and october;
```

## Transformations/Loading

- Pipelined Data Transformation
  - Avoids multiple, serial steps to transform and load data
  - Data transformation becomes more scalable and non-interruptive
- Table Functions
- MERGE
- Multitable INSERT
- INSERT FIRST

- Output is a set of rows that can be queried like a table
- No need to save intermediate results in temporary tables
- Input can be a set of rows and return multiple rows
- Can be parallelized
- Incremental Pipelining
  - Return result sets incrementally vs. waiting for completion of the entire result set
  - producer function uses the PIPELINED keyword
  - each transformed output record is sent to the consumer function using the PIPE ROW keyword as it is completed
  - consumer function or SQL statement uses the TABLE keyword to treat the resulting rows like a regular table

# Table Functions

Arisant

TITLE	AUTHORS
Oracle rman	author1, author2, author3
Oracle Administration	author4, author5
oracle10g comp. ref.	author6, author7

SQL> select b.title, a.\* from books b, table(getauthors(b.title)) a; **TABLE FUNCTION**

TITLE	COLUMN_VALUE
Oracle rman	author1
Oracle rman	author2
Oracle rman	author3
Oracle Administration	author4
Oracle Administration	author5
oracle10g comp. ref.	author6
oracle10g comp. ref.	author7

- Replaces “update record if exists, else insert it” logic
- Simpler SQL statement
- Improved performance (requires fewer scans)
- 10g additions
  - Conditional WHERE to skip insert/update when desired
  - Optional DELETE to cleanse the tables while updating them

- Using PL/SQL

```
-- pseudo code
LOOP through product_delta cursor
  UPDATE product table with values from
    product_delta;
  IF SQL%NOTFOUND THEN
    INSERT INTO product using values
      from product_delta;
  ENDIF;
END LOOP;
```

- Using SQL

```
UPDATE product t
SET <column list> =
(SELECT <column list> from product_delta
s WHERE s.prod_id=t.prod_id);

INSERT INTO products t
SELECT * FROM products_delta s
WHERE s.prod_id NOT IN
(SELECT prod_id FROM products);
```

- Using MERGE

```
MERGE INTO product t
USING product_delta s
ON (t.prod_id=s.prod_id)
WHEN MATCHED THEN
UPDATE SET
t.column1=s.column1,
t.column2=s.column2
WHERE <column_condition> -- conditional update
DELETE WHERE <column_condition> --optional delete
WHEN NOT MATCHED THEN
INSERT
(<column list>) VALUES (s.column1, s.column2, s.column3,...)
WHERE <column_condition>; -- conditional insert
```

- Used when multiple tables are involved as targets
- Avoids processing the same source data  $n$  times and increasing the transformation workload  $n$  times.
- Can be parallelized and used with the direct-load mechanism for faster performance

```
INSERT ALL
  INTO target1 VALUES (<column list>)
  INTO target2 VALUES (<column list>)
  SELECT <source1 column list>, <source2 column
  list>
  FROM source1 s, source2 p
  WHERE s.key = p.key
  AND .... GROUP BY ...;
```

- Optional WHEN clause allows 'conditional' INSERTs

```
INSERT ALL
WHEN column1 IN (SELECT ... FROM source1) THEN
INTO target1 VALUES (<column list>)
INTO target2 VALUES (<column list>)
WHEN column2 > 1000 THEN ...
SELECT ... From source tables...
```

# Pivoting Using Multitable Insert

# Arisant

```
SELECT * FROM sales_source;
```

PROD_ID	CUST_ID	WEEKLY_START_DATE	SALES_SUN	SALES_MON	SALES_TUE	SALES_WED	SALES_THU	SALES_FRI	SALES_SAT
111	222	01-OCT-00	100	200	300	400	500	600	700
222	333	08-OCT-00	200	300	400	500	600	700	800
333	444	15-OCT-00	300	400	500	600	700	800	900

**PIVOTING**



```
select * from sales order by 1,2,4;
```

PROD_ID	CUST_ID	WEEKLY_START_DATE	AMOUNT_SOLD
111	222	01-OCT-00	100
111	222	02-OCT-00	200
111	222	03-OCT-00	300
111	222	04-OCT-00	400
111	222	05-OCT-00	500
111	222	06-OCT-00	600
111	222	07-OCT-00	700
222	333	08-OCT-00	200
222	333	09-OCT-00	300

# Pivoting Using Multitable Insert

Arisant

## pre-9i SQL

```
CREATE table sales NOLOGGING PARALLEL AS
SELECT prod_id,cust_id, weekly_start_date, amount_sold
FROM
(SELECT prod_id,cust_id, weekly_start_date,
sales_sun amount_sold FROM sales_source
UNION ALL
SELECT prod_id,cust_id, weekly_start_date+1,
sales_mon amount_sold FROM sales_source
UNION ALL
SELECT prod_id,cust_id, weekly_start_date+2,
sales_tue amount_sold FROM sales_source
UNION ALL
... <text omitted> ...
UNION ALL
SELECT prod_id,cust_id, weekly_start_date+6,
sales_sat amount_sold FROM sales_source);
```

# Pivoting Using Multitable Insert

Arisant

PLAN\_TABLE\_OUTPUT

Id	Operation	Name	Rows	Bytes	Cost
0	CREATE TABLE STATEMENT		21	1008	7
1	LOAD AS SELECT				
2	VIEW		21	1008	7
3	UNION-ALL				
4	TABLE ACCESS FULL	SALES_SOURCE	3	45	1
5	TABLE ACCESS FULL	SALES_SOURCE	3	45	1
6	TABLE ACCESS FULL	SALES_SOURCE	3	45	1
7	TABLE ACCESS FULL	SALES_SOURCE	3	45	1
8	TABLE ACCESS FULL	SALES_SOURCE	3	45	1
9	TABLE ACCESS FULL	SALES_SOURCE	3	45	1
10	TABLE ACCESS FULL	SALES_SOURCE	3	45	1

# Pivoting Using Multitable Insert

Arisant

## --With multitable INSERT

```
INSERT ALL
INTO sales (prod_id, cust_id, WEEKLY_START_DATE,amount_sold)
VALUES (prod_id,cust_id, weekly_start_date, sales_sun)
INTO sales (prod_id, cust_id, WEEKLY_START_DATE,amount_sold)
VALUES (prod_id,cust_id, weekly_start_date+1, sales_mon)
INTO sales (prod_id, cust_id, WEEKLY_START_DATE,amount_sold)
VALUES (prod_id,cust_id, weekly_start_date+2, sales_tue)
... <text omitted> ...
VALUES (prod_id,cust_id, weekly_start_date+6, sales_sat)
SELECT prod_id,cust_id, weekly_start_date, sales_sun,
sales_mon, sales_tue, sales_wed, sales_thu, sales_fri, sales_sat
FROM sales_source;
```

# Pivoting Using Multitable Insert

Arisant

PLAN\_TABLE\_OUTPUT

Id	Operation	Name	Rows	Bytes	Cost
0	INSERT STATEMENT		3	81	2
1	MULTI-TABLE INSERT				
2	INTO	SALES			
3	INTO	SALES			
4	INTO	SALES			
5	INTO	SALES			
6	INTO	SALES			
7	INTO	SALES			
8	INTO	SALES			
9	TABLE ACCESS FULL	SALES_SOURCE	3	81	2

- Conditional logic to determine appropriate table to INSERT into

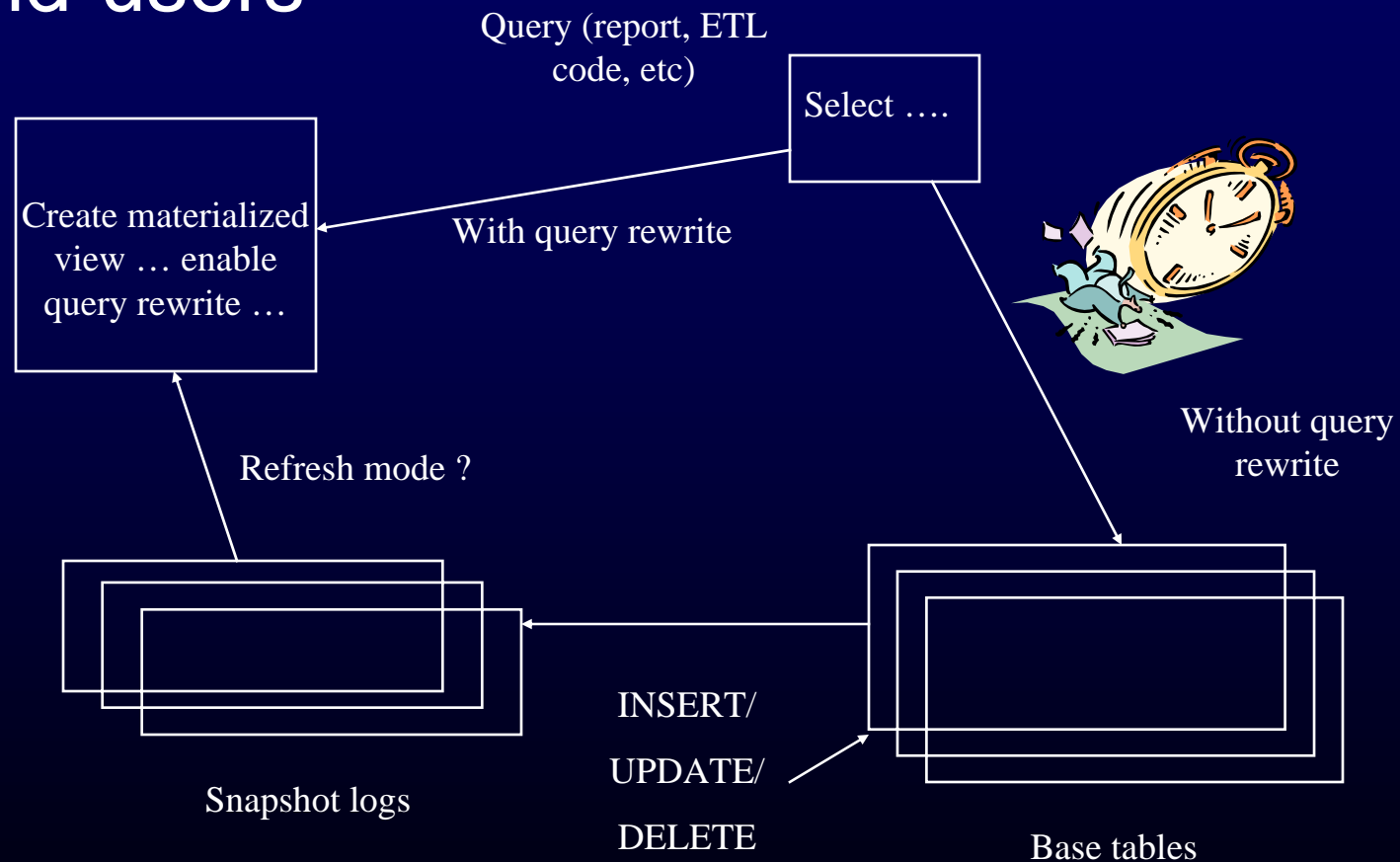
```
INSERT FIRST WHEN <column 1 condition ...> AND <column 2...  
condition> THEN  
INTO table_1 VALUES  
(<values list>)  
WHEN <column 1 ...> AND <column 2...> THEN  
INTO table_2 VALUES  
(<values list>)  
WHEN ...  
ELSE INTO exceptions_table VALUES (values list)  
SELECT <column list>  
FROM source1 s, source2 p  
WHERE s.key = p.key  
AND ... GROUP BY ...;
```

- Materialized views
- SQL WITH
- Index merge
- Bitmap join indexes
- Star Queries

- For critical queries consider benchmarking performance between:
- Automatic PGA memory management
  - pga\_aggregate\_target
  - workarea\_size\_policy
- Manual PGA memory management
  - hash\_area\_size
  - sort\_area\_size
  - bitmap\_merge\_area\_size

- Pre-aggregation of data
- Changes to the underlying tables reflected in mview
  - Fast/Complete refresh
  - Refresh can be on demand or scheduled
- Can be partitioned and indexed like a table
- Query Rewrite
  - `/*+ REWRITE_OR_ERROR */` hint –new in 10g (ORA-30393)
  - `DBMS_MVIEW.EXPLAIN_REWRITE()`
  - `$ORACLE_HOME/rdbms/admin/utlxrw.sql` for `REWRITE_TABLE`
- `DBMS_ADVISOR` package
  - APIs to recommend materialized views and indexes
  - `DBMS_ADVISOR.TUNE_MVIEW()`
    - API to tune existing materialized views and indexes
    - Tuned version in `USER_TUNE_MVIEW`

- Query rewrite makes mviews transparent to end-users



- a.k.a subquery factoring clause
- Enables you to reuse the same query block in a SELECT statement when it occurs more than once within a complex query
- Enhanced query performance
- Eliminates redundant processing

# The SQL WITH Clause

Arisant

```
SQL> select TRANS_KEY,sum(DOLLAR_AMOUNT) from finance_trans, company
2  where finance_trans.company_key=company.company_key
3  group by TRANS_KEY
4  having sum(DOLLAR_AMOUNT) >
5  (select sum(DOLLAR_AMOUNT)*.45 from finance_trans, company
6  where finance_trans.company_key=company.company_key);
```

Elapsed: 00:22:42.33

Id	Operation	Name
0	SELECT STATEMENT	
* 1	FILTER	
2	HASH GROUP BY	
3	NESTED LOOPS	
4	TABLE ACCESS FULL	FINANCE_TRANS
* 5	INDEX UNIQUE SCAN	COMPANY_PK
6	SORT AGGREGATE	
7	NESTED LOOPS	
8	TABLE ACCESS FULL	FINANCE_TRANS
* 9	INDEX UNIQUE SCAN	COMPANY_PK

# The SQL WITH Clause

Arisant

SQL> WITH

**mysum** AS(

select TRANS\_KEY,sum(DOLLAR\_AMOUNT) AS total from finance\_trans, company

where finance\_trans.company\_key=company.company\_key

group by TRANS\_KEY )

select TRANS\_KEY, total from **mysum**

where total >

(select sum(total)\*.45 from **mysum**);

Elapsed: 00:05:13.74

Id	Operation	Name
0	SELECT STATEMENT	
1	TEMP TABLE TRANSFORMATION	
2	LOAD AS SELECT	
3	HASH GROUP BY	
4	NESTED LOOPS	
5	TABLE ACCESS FULL	FINANCE_TRANS
* 6	INDEX UNIQUE SCAN	COMPANY_PK
* 7	VIEW	
8	TABLE ACCESS FULL	SYS_TEMP_0FD9D6601_355006
9	SORT AGGREGATE	
10	VIEW	
11	TABLE ACCESS FULL	SYS_TEMP_0FD9D6601_355006

- Merge two separate indexes
- Avoid creating a new concatenated index
- Implementation via a hint

```
- /*+ index_join(table_name, index1, index2) */
```

```
SQL> select count(*) from FINANCE_TRANS where COMPANY_KEY between 1 and 1000 and CHARGE_CODE = 'BASIC';
```

Elapsed: 00:03:44.03

Execution Plan

```
-----  
0  SELECT STATEMENT Optimizer=CHOOSE (Cost=5147 Card=1 Bytes=11 )  
1  0  SORT (AGGREGATE)  
2  1  TABLE ACCESS (BY INDEX ROWID) OF 'FINANCE_TRANS' (Cost=5147 Card=10004 Bytes=110044)  
3  2  INDEX (RANGE SCAN) OF 'FINANCE_TRANS_IX2' (NON-UNIQUE) (Cost=51 Card=100035)
```

```
SQL> select /*+ index_join(FINANCE_TRANS, FINANCE_TRANS_IX2,TEMP1) */ count(*) from FINANCE_TRANS  
where COMPANY_KEY between 1 and 1000 and CHRGE_CODE='BASIC';
```

Elapsed: 00:00:01.73

Execution Plan

```
-----  
0  SELECT STATEMENT Optimizer=CHOOSE (Cost=2037 Card=1 Bytes=11)  
1  0  SORT (AGGREGATE)  
2  1  VIEW OF 'index$_join$_001' (Cost=2037 Card=10004 Bytes=110044)  
3  2  HASH JOIN  
4  3  INDEX (RANGE SCAN) OF ' FINANCE_TRANS_IX2' (NON-UNIQUE) (Cost=1811 Card=10004  
                                           Bytes=110044)  
5  3  INDEX (RANGE SCAN) OF 'TEMP1' (NON-UNIQUE) (Cost=1811 Card=10004 Bytes=110044)
```

# Bitmap Join Indexes

Arisant

- An index build on a table using columns from another table(s)!
- Index contains the data to support a join query
- Allows the query to retrieve the data from the index rather than referencing the join tables

```
CREATE BITMAP INDEX temp_bixj1 ON BUDGET_FACT
(LEDDER_DIM.ledger_name)
FROM BUDGET_FACT, LEDDER_DIM
WHERE LEDDER_DIM.ledger_key = BUDGET_FACT.ledger_key
NOLOGGING
TABLESPACE INDEX_1
COMPUTE STATISTICS;
```

- Good for query performance, bad for DML operations
  - Consider drop/recreate

# Bitmap Join Indexes

Arisant

-- Without bitmap join index

```
SQL> SELECT D.ledger_name, SUM(F.base_amount)
FROM BUDGET_FACT F, LEDGER_DIM D
WHERE D.ledger_key = F.ledger_key
AND D.ledger_name = 'Internal Spending'
GROUP BY D.ledger_name;
```

**Elapsed: 00:00:07.58**

Execution Plan

```
0  SELECT STATEMENT Optimizer=CHOOSE (Cost=4667 Card=1 Bytes=29 )
  1  0  SORT (GROUP BY NOSORT) (Cost=4667 Card=1 Bytes=29)
    2  1  HASH JOIN (Cost=4667 Card=1577907 Bytes=45759303)
      3  2  VIEW OF 'index$_join$_002' (Cost=97 Card=6125 Bytes=128625)
        4  3  HASH JOIN
          5  4  INDEX (RANGE SCAN) OF ' LEDGER_DIM_IX4' (NON-UNIQUE ) (Cost=6 Card=6125 Bytes=128625)
            6  4  INDEX (FAST FULL SCAN) OF 'LEDGER_DIM_PK' (UNIQUE)(Cost=6 Card=6125 Bytes=128625)
              7  2  TABLE ACCESS (FULL) OF 'BUDGET_FACT' (Cost=2631 Card=8092591 Bytes=64740728)
```

-- Add bitmap join index TEMP\_BIXJ1

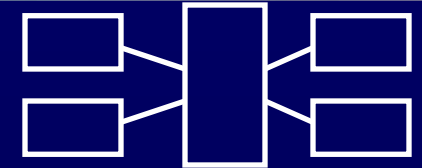
Elapsed: 00:00:00.72

Execution Plan

```
-----  
0  SELECT STATEMENT Optimizer=CHOOSE (Cost=1875 Card=1 Bytes=29 )  
1  0  SORT (GROUP BY NOSORT) (Cost=1875 Card=1 Bytes=29)  
2  1  HASH JOIN (Cost=1875 Card=1577907 Bytes=45759303)  
3  2   VIEW OF 'index$_join$_002' (Cost=97 Card=6125 Bytes=128625)  
4  3    HASH JOIN  
5  4     INDEX (RANGE SCAN) OF 'LEDGER_DIM_IX4' (NON-UNIQUE) (Cost=6 Card=6125 Bytes=128625)  
6  4     INDEX (FAST FULL SCAN) OF 'LEDGER_DIM_PK' (UNIQUE)(Cost=6 Card=6125 Bytes=128625)  
7  2     TABLE ACCESS (BY INDEX ROWID) OF 'BUDGET_FACT' (Cost=1712 Card=8092591 Bytes=64740728)  
8  7      BITMAP CONVERSION (TO ROWIDS)  
9  8       BITMAP INDEX (SINGLE VALUE) OF 'TEMP_BIXJ1'
```

# Star Queries

Arisant



- A join between a fact table and a number of dimension tables
- Require the existence of bitmap indexes on FK columns on the FACT table
- retrieve exactly the necessary rows from the fact table using  $n$  merged bitmap indexes
- join this result set to the dimension tables
- Implementation:
  - `star_transformation_enabled=true|false|temp_disable` (system or session modifiable)
  - hints for 'stubborn' queries
    - `/*+ STAR_TRANSFORMATION */`
    - `/*+ FACT */`

- Why is it a big deal in a warehouse?
- Index maintenance during DML
  - Benchmark loads with/without indexes present
  - Specific attention to bitmap index maintenance
  - Consider invalidate/rebuild or drop/recreate
  - Partition level maintenance provides granular control
  - Provide APIs (stored procs) to ETL application
- Virtual Indexes! – undocumented feature
  - Not intended for standalone usage
  - Part of Tuning Pack's Virtual index Wizard
  - Allows CBO to evaluate a new index without having to create it!
- Index monitoring (watch out from DBMS\_STATS)

- Index Maintenance scenario
  - Loading in non partition table
    - 900 million existing records and 3 indexes
    - Load 100 million records
    - Last 30% of records require some lookups on first 70% of loaded records (1 of the indexes is needed)
  - Slow load performance
  - Indexes can be dropped to improve loading
    - Need to be recreated
    - Not available for other queries

# Index Management

Arisant

No partitioning



ETL Load Complete

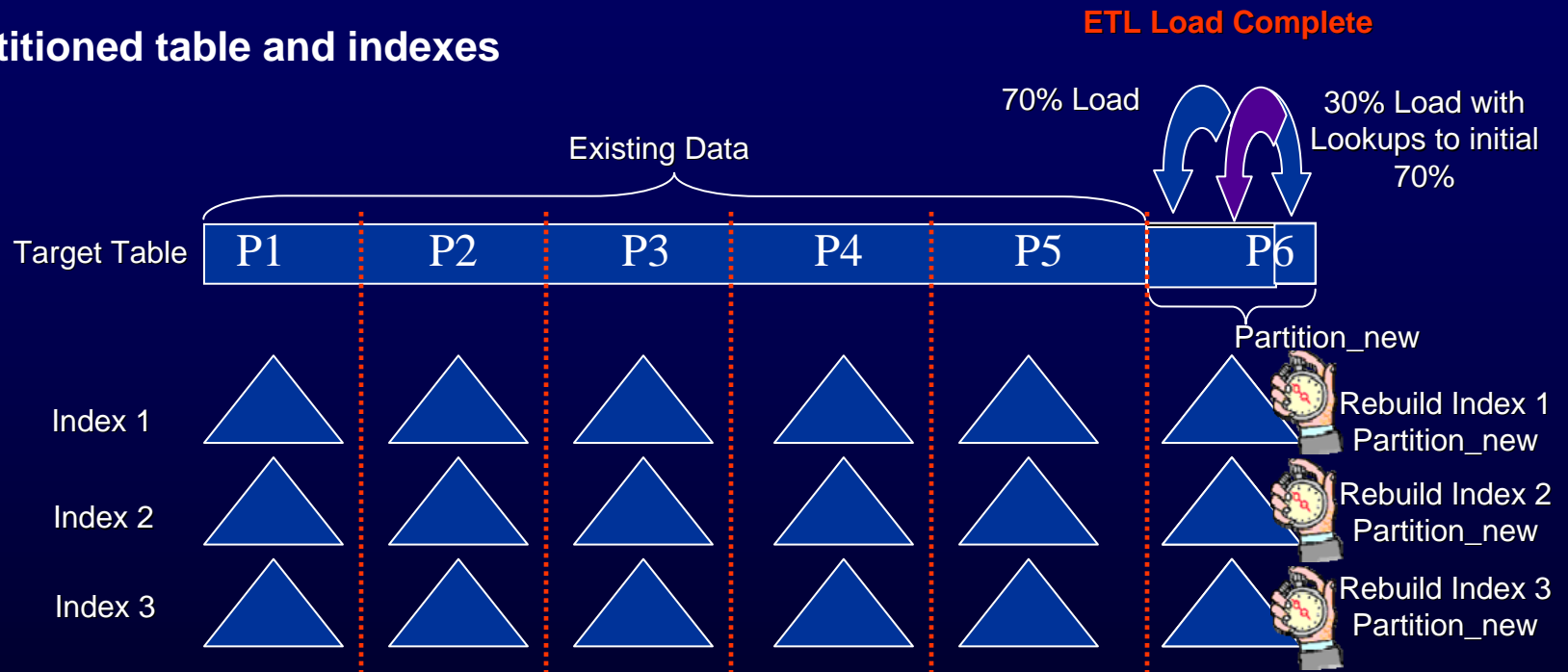


- The solution
  - Partition the table and indexes
  - Invalidate specific index partitions (mark unusable)
    - No inline index maintenance while data is loading
    - Rebuild required index partition to help last part of the load (granular control)
  - Rebuild remaining index partitions
  - Other partitions and related indexes available for querying while loading new data

# Index Management

Arisant

## Partitioned table and indexes



**Unusable partitioned Indexes: No inline index maintenance**



**Usable partitioned Indexes: Rebuilt by the application using stored procedures after a load has been completed**

- Do I need foreign keys in a warehouse?
  - My code takes care of data integrity
  - Our data is clean
- Trace to see what happens while loading
- Consider `/*+ append */` hint behavior
- Consider some of these options
  - Deferred constraints
    - check that constraint is satisfied only at commit time
    - Useful when loading in no particular order
  - Disable/enable, validate/validate
  - Enable in parallel!
    - Metalink Note:124848.1

- Be careful what/when you analyze
- Manage statistics on 'big' tables
  - Do I really need to analyze all the tables?
  - Determine frequency/timing, estimate sample
  - Consider 'faking' statistics
  - Consider providing APIs to ETL application
    - Ability to analyze before/after certain loads
- Table monitoring in 9i and 10g
- Index monitoring
  - Watch out from `dbms_stats.gather_table_stats` and `dbms_stats.gather_index_stats` behavior
    - Procedures mark analyzed indexes as USED!

