

The Educational Conference For Oracle Technology Users

ODTUG  
Kscope23  
aurora, co      june 25-29

Welcome

DRIVEN BY EXCELLENCE

# Make Your Oracle Data Warehouse Fast: **Comparing Performance Features**

Dani Schnider



# About me



## Dani Schneider

- Working for Callista
- Oracle ACE Director
- Member of Symposium 42
- Hobby: Craft Beer Brewing



[@dani\\_schnider](https://twitter.com/dani_schnider)

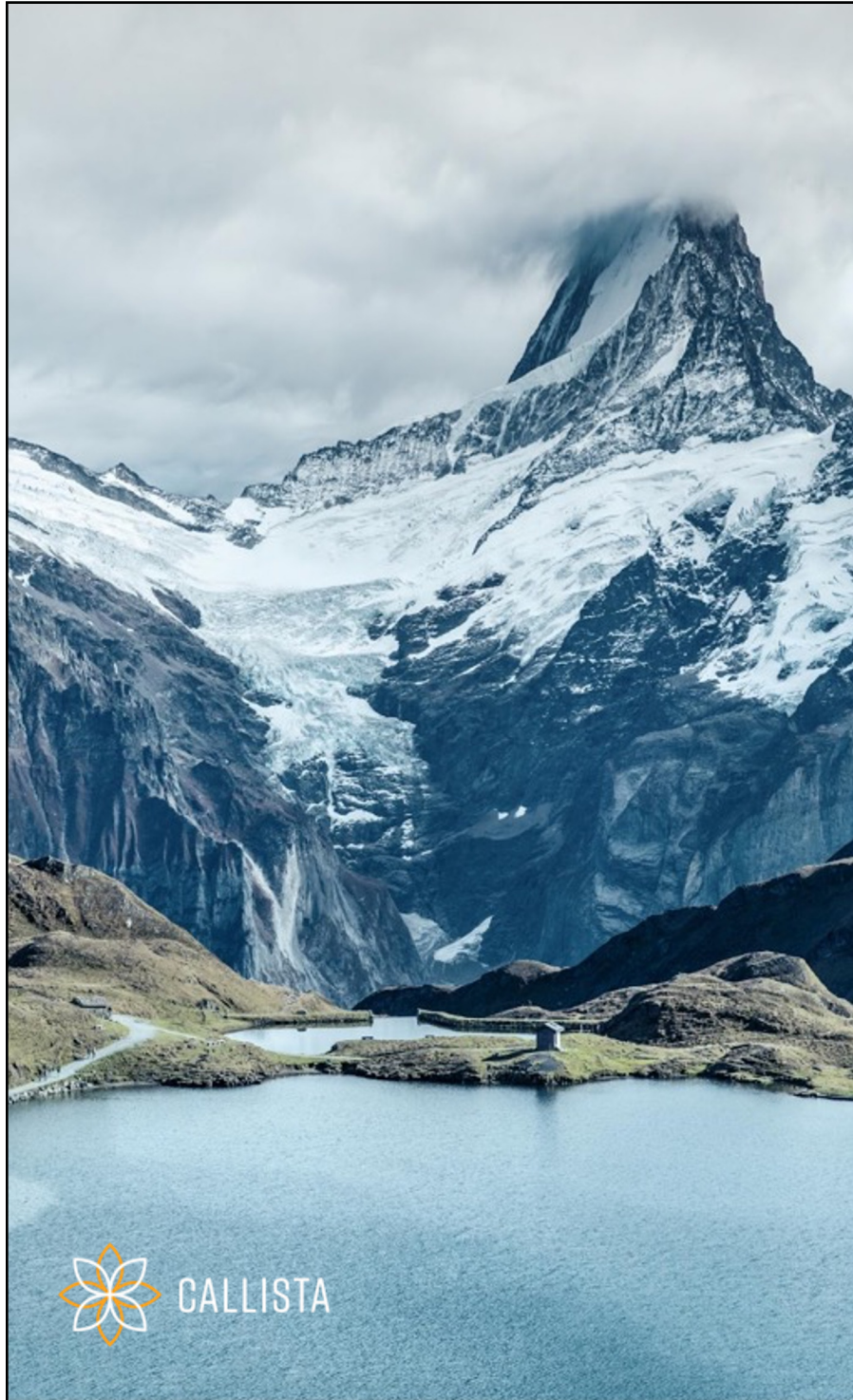


[danischnider.wordpress.com](https://danischnider.wordpress.com)



[www.linkedin.com/in/danischnider/](https://www.linkedin.com/in/danischnider/)



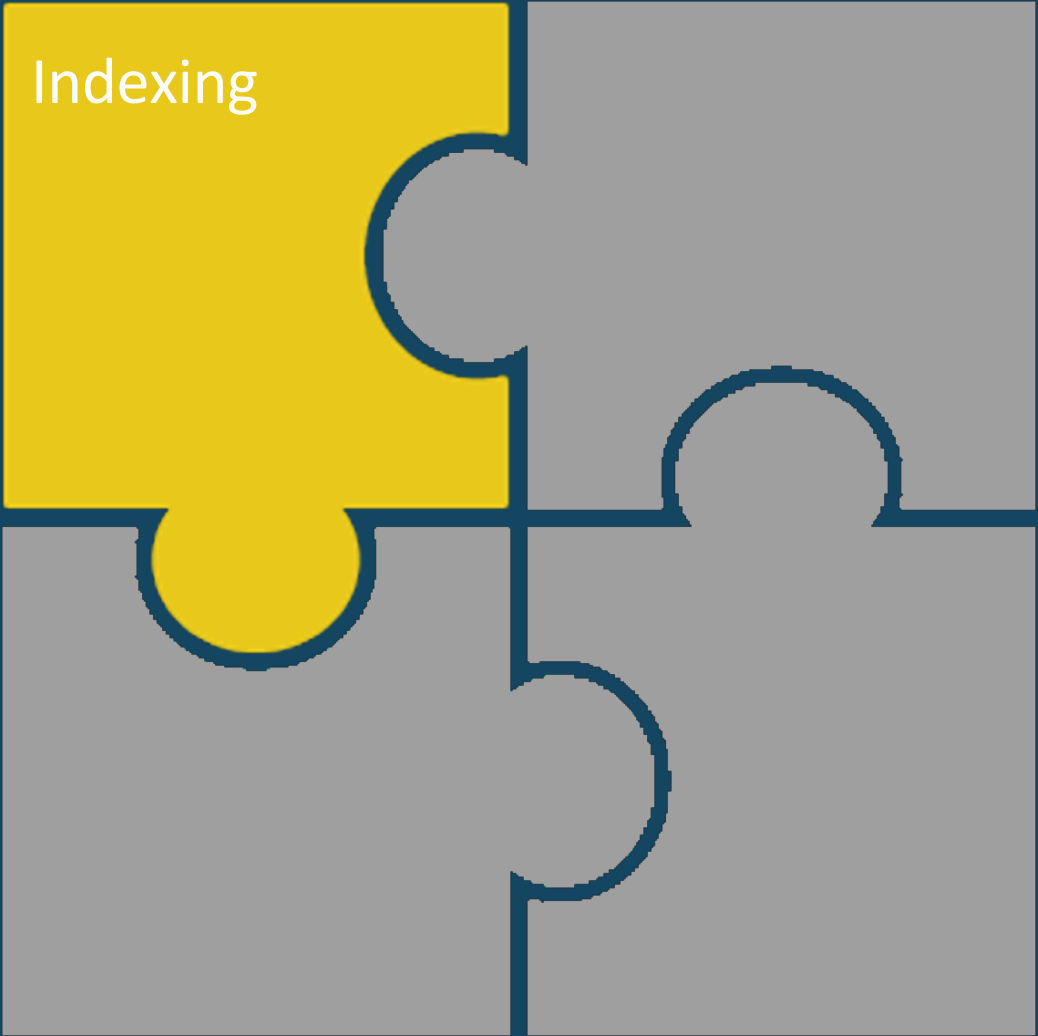
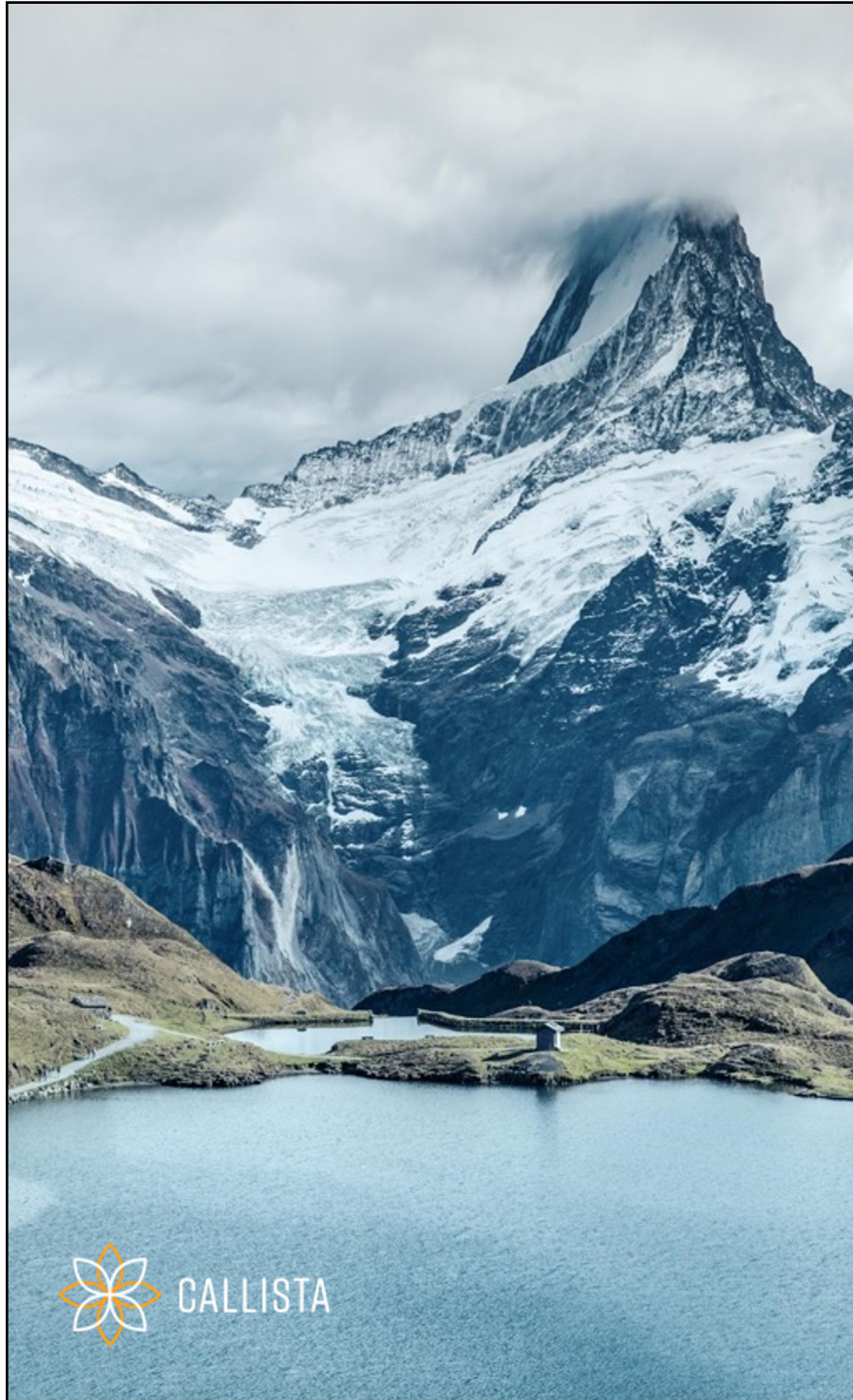


Indexing

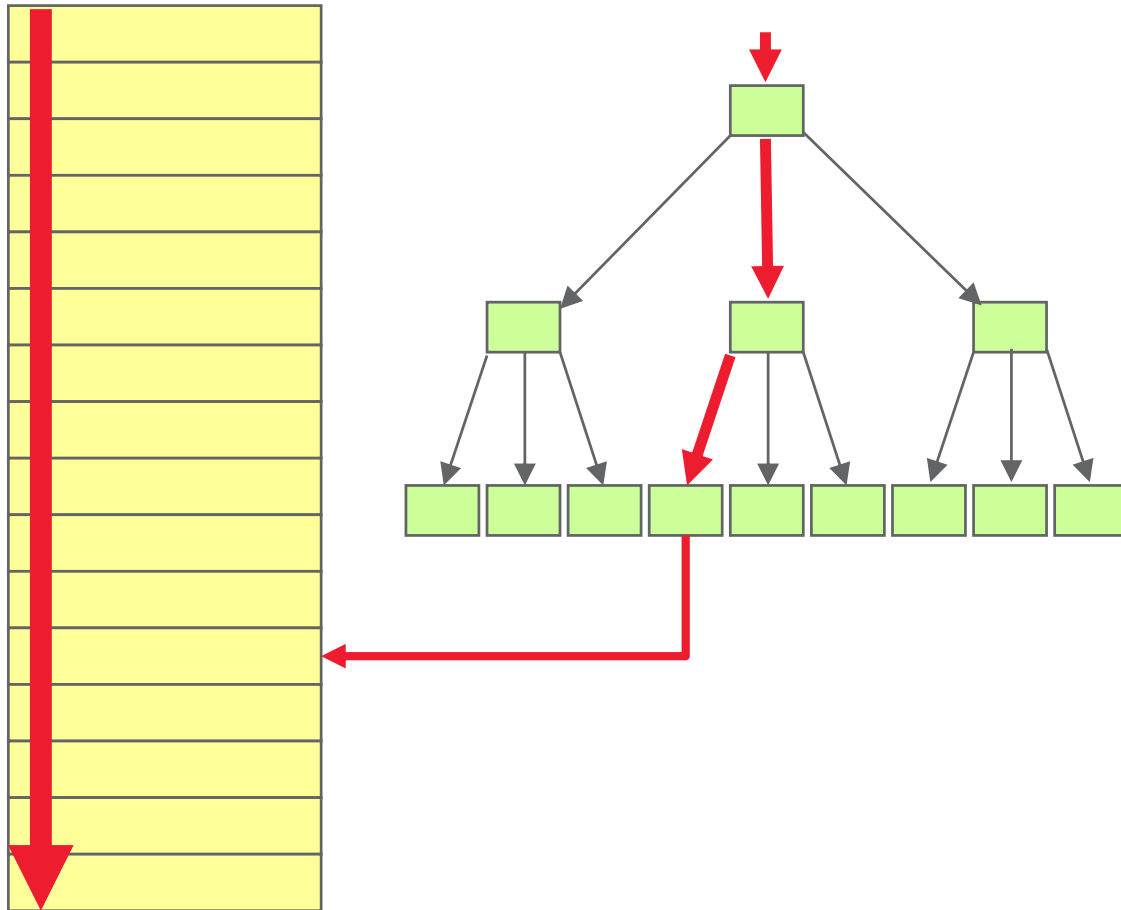
Materialized Views

Partitioning

In-Memory



# Full Table Scan vs. Index Scan



## Full Table Scan

- Efficient for weak selectivity
- High percentage of data

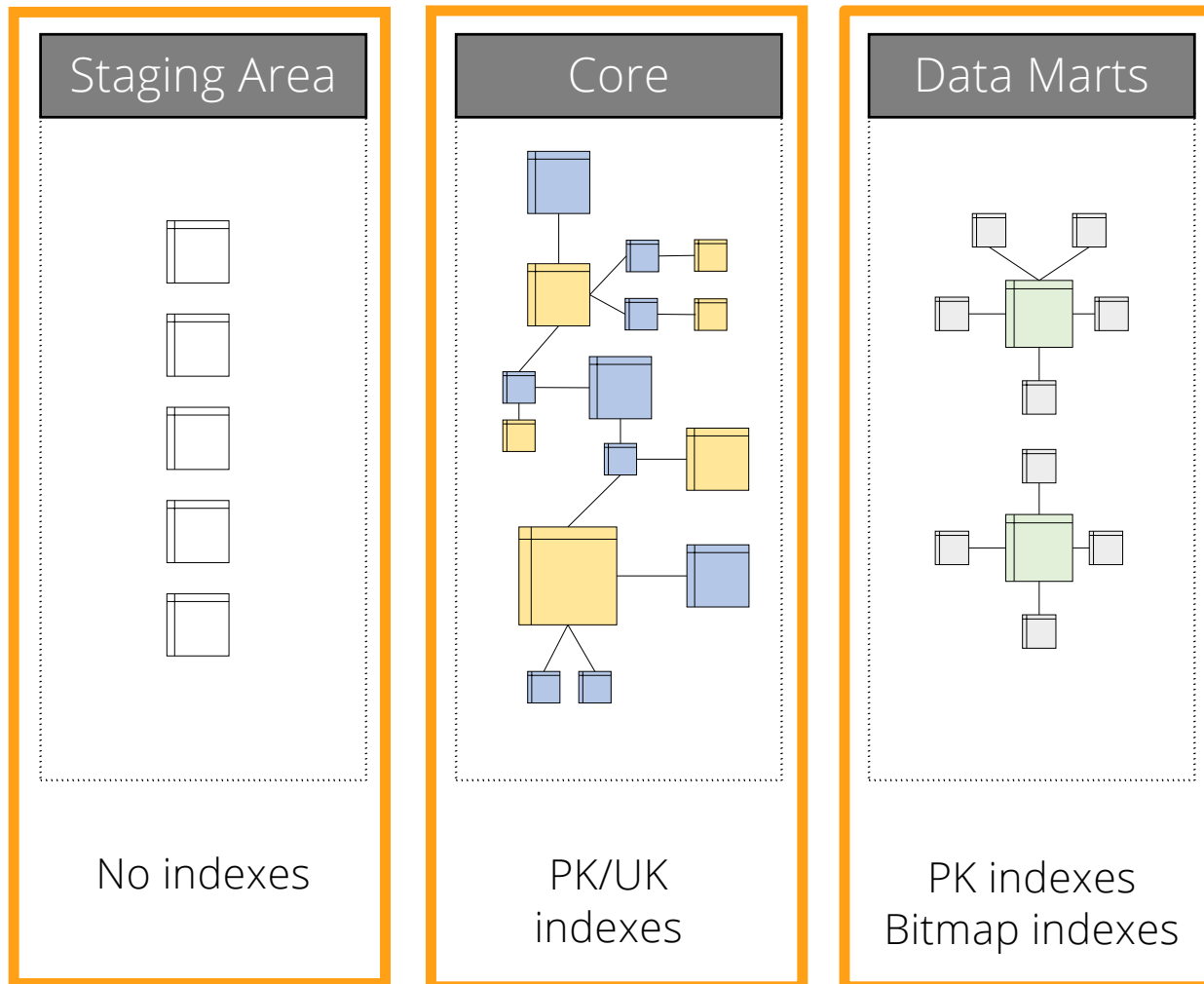
## Index Scan

- Efficient for strong selectivity
- Small percentage of data

## Typical for Data Warehouse

- High percentage (often 100%) of data in ETL jobs
- Aggregations on high volumes of data, few filter restrictions

# Indexing Data Warehouse Layers



## Dimension tables

- Primary key (unique index)
- Bitmap indexes on filter columns (optional)

## Fact tables

- Bitmap indexes on dimension keys
- Typically no primary key

SH.TIMES	
P * TIME_ID	DATE
* DAY_NAME	VARCHAR2 (9 BYTE)
* DAY_NUMBER_IN_WEEK	NUMBER (1)
* DAY_NUMBER_IN_MONTH	NUMBER (2)
* CALENDAR_WEEK_NUMBER	NUMBER (2)
* FISCAL_WEEK_NUMBER	NUMBER (2)
* WEEK_ENDING_DAY	DATE
* WEEK_ENDING_DAY_ID	NUMBER
* CALENDAR_MONTH_NUMBER	NUMBER (2)
* FISCAL_MONTH_NUMBER	NUMBER (2)
* CALENDAR_MONTH_DESC	VARCHAR2 (8 BYTE)
* CALENDAR_MONTH_ID	NUMBER
* FISCAL_MONTH_DESC	VARCHAR2 (8 BYTE)
* FISCAL_MONTH_ID	NUMBER
* DAYS_IN_CAL_MONTH	NUMBER
* DAYS_IN_FIS_MONTH	NUMBER
* END_OF_CAL_MONTH	DATE
* END_OF_FIS_MONTH	DATE
* CALENDAR_MONTH_NAME	VARCHAR2 (9 BYTE)
* FISCAL_MONTH_NAME	VARCHAR2 (9 BYTE)
* CALENDAR_QUARTER_DESC	CHAR (7 BYTE)
* CALENDAR_QUARTER_ID	NUMBER
* FISCAL_QUARTER_DESC	CHAR (7 BYTE)
* FISCAL_QUARTER_ID	NUMBER
* DAYS_IN_CAL_QUARTER	NUMBER
* DAYS_IN_FIS_QUARTER	NUMBER
* END_OF_CAL_QUARTER	DATE
* END_OF_FIS_QUARTER	DATE
* CALENDAR_QUARTER_NUMBER	NUMBER (1)
* FISCAL_QUARTER_NUMBER	NUMBER (1)
* CALENDAR_YEAR	NUMBER (4)
* CALENDAR_YEAR_ID	NUMBER
* FISCAL_YEAR	NUMBER (4)
* FISCAL_YEAR_ID	NUMBER
* DAYS_IN_CAL_YEAR	NUMBER
* DAYS_IN_FIS_YEAR	NUMBER
* END_OF_CAL_YEAR	DATE
* END_OF_FIS_YEAR	DATE
◆ TIMES_PK	

SH.CHANNELS	
P * CHANNEL_ID	NUMBER
* CHANNEL_DESC	VARCHAR2 (20 BYTE)
* CHANNEL_CLASS	VARCHAR2 (20 BYTE)
* CHANNEL_CLASS_ID	NUMBER
* CHANNEL_TOTAL	VARCHAR2 (13 BYTE)
* CHANNEL_TOTAL_ID	NUMBER
◆ CHANNELS_PK	

SH.SALES	
F * PROD_ID	NUMBER (6)
F * CUST_ID	NUMBER
F * TIME_ID	DATE
F * CHANNEL_ID	NUMBER
F * PROMO_ID	NUMBER (6)
* QUANTITY_SOLD	NUMBER (10,2)
* AMOUNT_SOLD	NUMBER (10,2)
◆ SALES_CHANNEL_BIX	
◆ SALES_CUST_BIX	
◆ SALES_PROD_BIX	
◆ SALES_PROMO_BIX	
◆ SALES_TIME_BIX	

SH.CUSTOMERS	
P * CUST_ID	NUMBER
* CUST_FIRST_NAME	VARCHAR2 (20 BYTE)
* CUST_LAST_NAME	VARCHAR2 (40 BYTE)
* CUST_GENDER	CHAR (1 BYTE)
* CUST_YEAR_OF_BIRTH	NUMBER (4)
* CUST_MARITAL_STATUS	VARCHAR2 (20 BYTE)
* CUST_STREET_ADDRESS	VARCHAR2 (40 BYTE)
* CUST_POSTAL_CODE	VARCHAR2 (10 BYTE)
* CUST_CITY	VARCHAR2 (30 BYTE)
* CUST_CITY_ID	NUMBER
* CUST_STATE_PROVINCE	VARCHAR2 (40 BYTE)
* CUST_STATE_PROVINCE_ID	NUMBER
F * COUNTRY_ID	NUMBER
* CUST_MAIN_PHONE_NUMBER	VARCHAR2 (25 BYTE)
* CUST_INCOME_LEVEL	VARCHAR2 (30 BYTE)
* CUST_CREDIT_LIMIT	NUMBER
* CUST_EMAIL	VARCHAR2 (50 BYTE)
* CUST_TOTAL	VARCHAR2 (14 BYTE)
* CUST_TOTAL_ID	NUMBER
* CUST_SRC_ID	NUMBER
* CUST_EFF_FROM	DATE
* CUST_EFF_TO	DATE
* CUST_VALID	VARCHAR2 (1 BYTE)
◆ CUSTOMERS_GENDER_BIX	
◆ CUSTOMERS_MARITAL_BIX	
◆ CUSTOMERS_PK	
◆ CUSTOMERS_YOB_BIX	

SH.COUNTRIES	
P * COUNTRY_ID	NUMBER
* COUNTRY_ISO_CODE	CHAR (2 BYTE)
* COUNTRY_NAME	VARCHAR2 (40 BYTE)
* COUNTRY_SUBREGION	VARCHAR2 (30 BYTE)
* COUNTRY_SUBREGION_ID	NUMBER
* COUNTRY_REGION	VARCHAR2 (20 BYTE)
* COUNTRY_REGION_ID	NUMBER
* COUNTRY_TOTAL	VARCHAR2 (11 BYTE)
* COUNTRY_TOTAL_ID	NUMBER
* COUNTRY_TOTAL_HIST	VARCHAR2 (40 BYTE)
◆ COUNTRIES_PK	

SH.PRODUCTS	
P * PROD_ID	NUMBER (6)
* PROD_NAME	VARCHAR2 (50 BYTE)
* PROD_DESC	VARCHAR2 (4000 BYTE)
* PROD_SUBCATEGORY	VARCHAR2 (50 BYTE)
* PROD_SUBCATEGORY_ID	NUMBER
* PROD_SUBCATEGORY_DESC	VARCHAR2 (2000 BYTE)
* PROD_CATEGORY	VARCHAR2 (50 BYTE)
* PROD_CATEGORY_ID	NUMBER
* PROD_CATEGORY_DESC	VARCHAR2 (2000 BYTE)
* PROD_WEIGHT_CLASS	NUMBER (3)
* PROD_UNIT_OF_MEASURE	VARCHAR2 (20 BYTE)
* PROD_PACK_SIZE	VARCHAR2 (30 BYTE)
* SUPPLIER_ID	NUMBER (6)
* PROD_STATUS	VARCHAR2 (20 BYTE)
* PROD_LIST_PRICE	NUMBER (8,2)
* PROD_MIN_PRICE	NUMBER (8,2)
* PROD_TOTAL	VARCHAR2 (13 BYTE)
* PROD_TOTAL_ID	NUMBER
* PROD_SRC_ID	NUMBER
* PROD_EFF_FROM	DATE
* PROD_EFF_TO	DATE
* PROD_VALID	VARCHAR2 (1 BYTE)
◆ PRODUCTS_PK	
◆ PRODUCTS_PROD_CAT_IX	
◆ PRODUCTS_PROD_STATUS_BIX	
◆ PRODUCTS_PROD_SUBCAT_IX	

SH.PROMOTIONS	
P * PROMO_ID	NUMBER (6)
* PROMO_NAME	VARCHAR2 (30 BYTE)
* PROMO_SUBCATEGORY	VARCHAR2 (30 BYTE)
* PROMO_SUBCATEGORY_ID	NUMBER
* PROMO_CATEGORY	VARCHAR2 (30 BYTE)
* PROMO_CATEGORY_ID	NUMBER
* PROMO_COST	NUMBER (10,2)
* PROMO_BEGIN_DATE	DATE
* PROMO_END_DATE	DATE
* PROMO_TOTAL	VARCHAR2 (15 BYTE)
* PROMO_TOTAL_ID	NUMBER
◆ PROMO_PK	

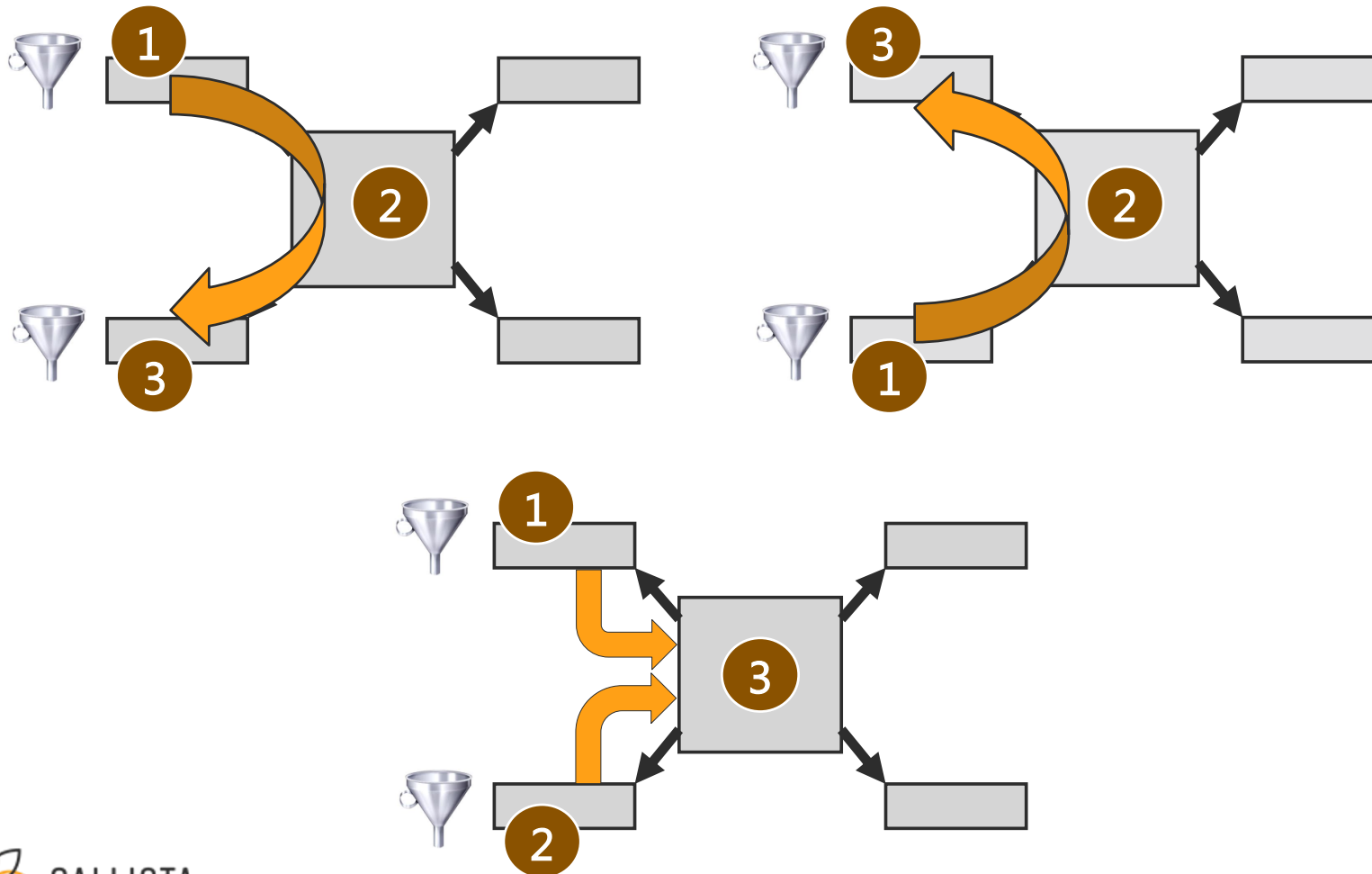
Oracle Sample  
Schema **SH**  
(Sales History)



Download: <https://github.com/oracle-samples/db-sample-schemas/releases>



# Joins in Star Schema



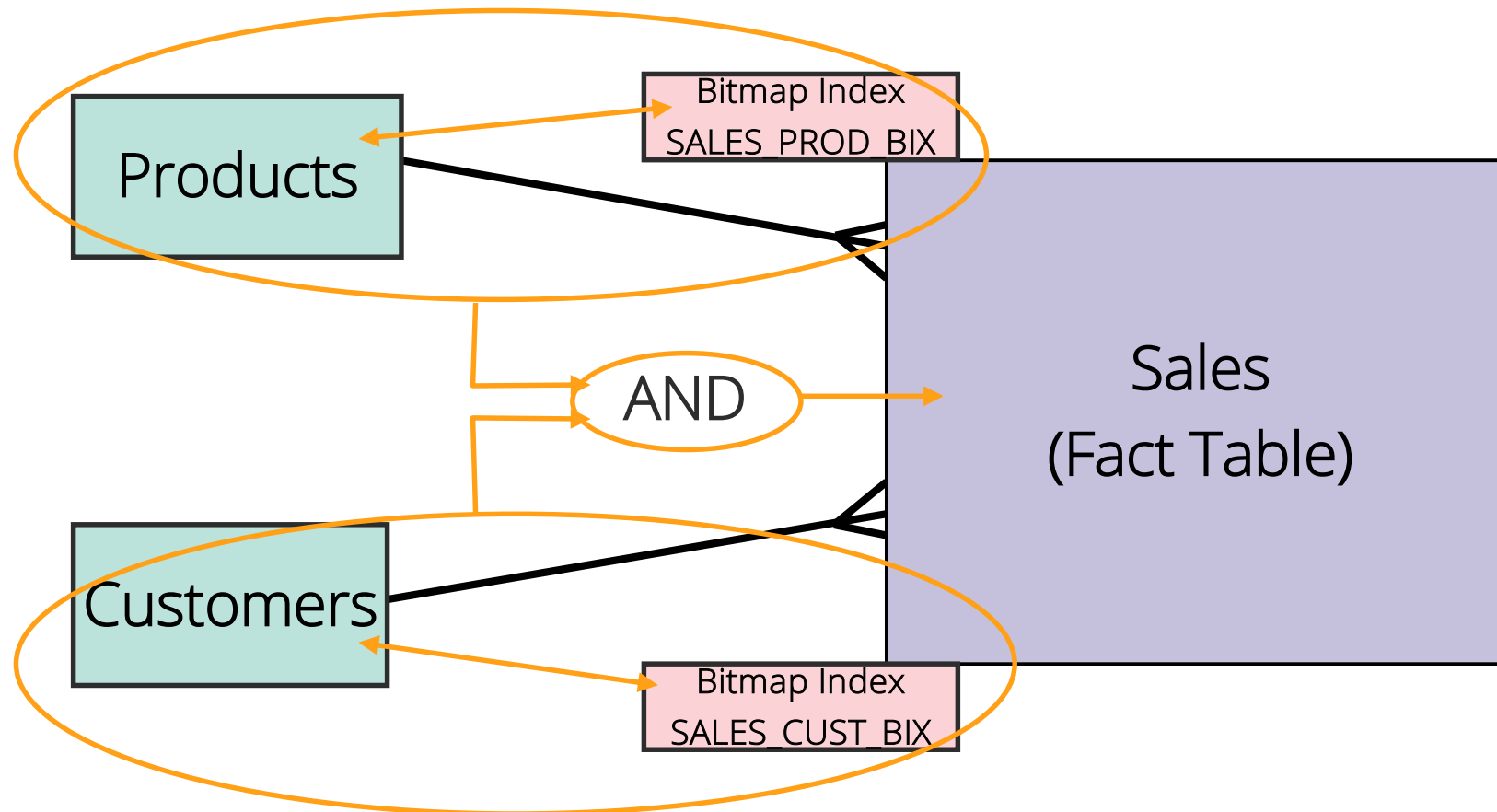
## Typical queries on star schema

- Filter on multiple dimensions
- Facts are joined with all required dimensions

## Challenge for optimizer

- Dimensions (with filters) should be read first
- No relationships between dimensions => fact table is joined too early

# Star Transformation

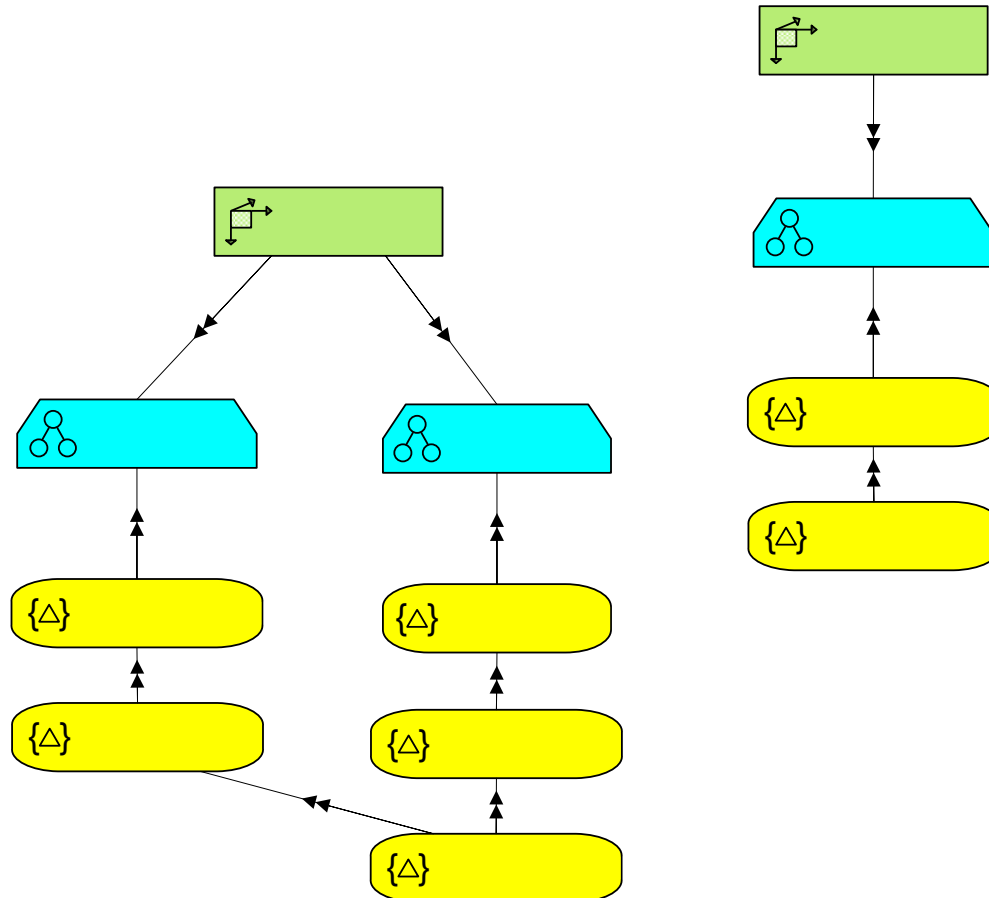




# Live Demo

## Star Transformation

# Drill-Up on Star Schemas

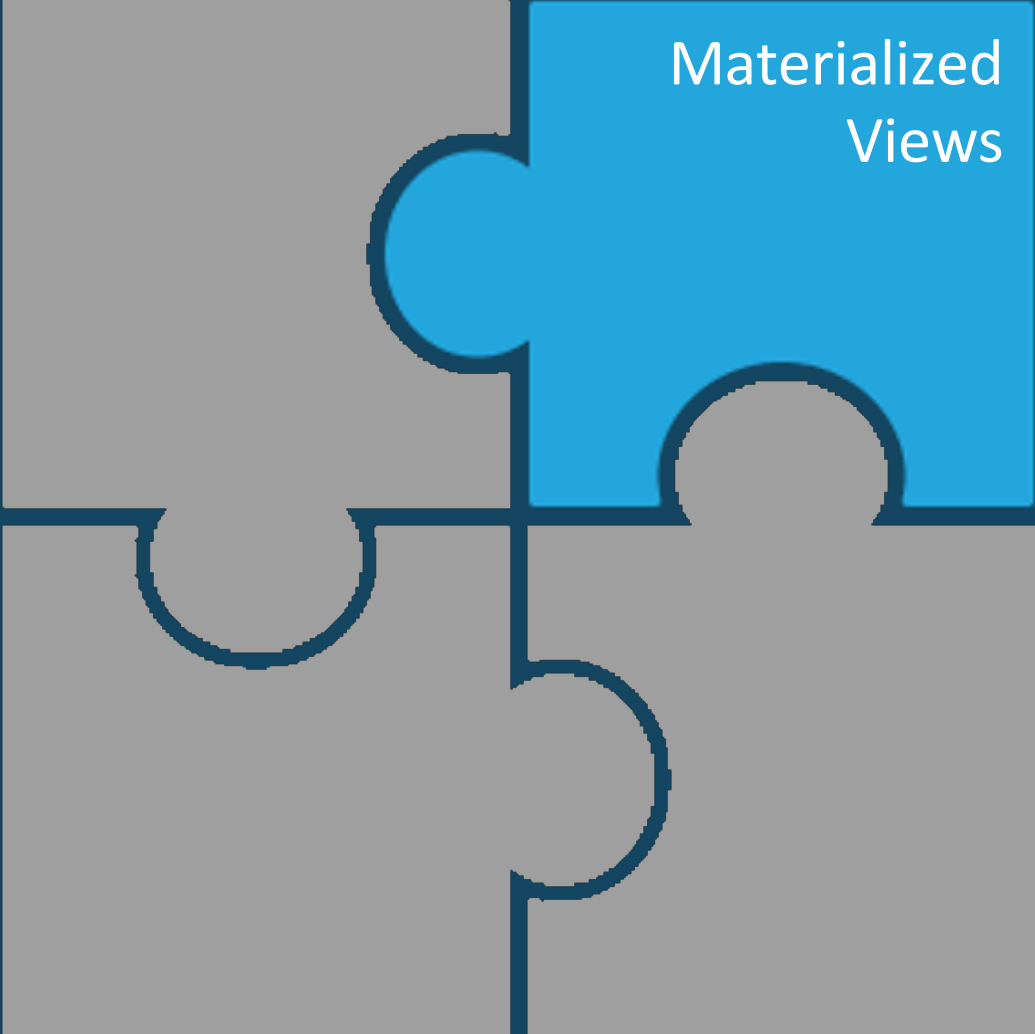
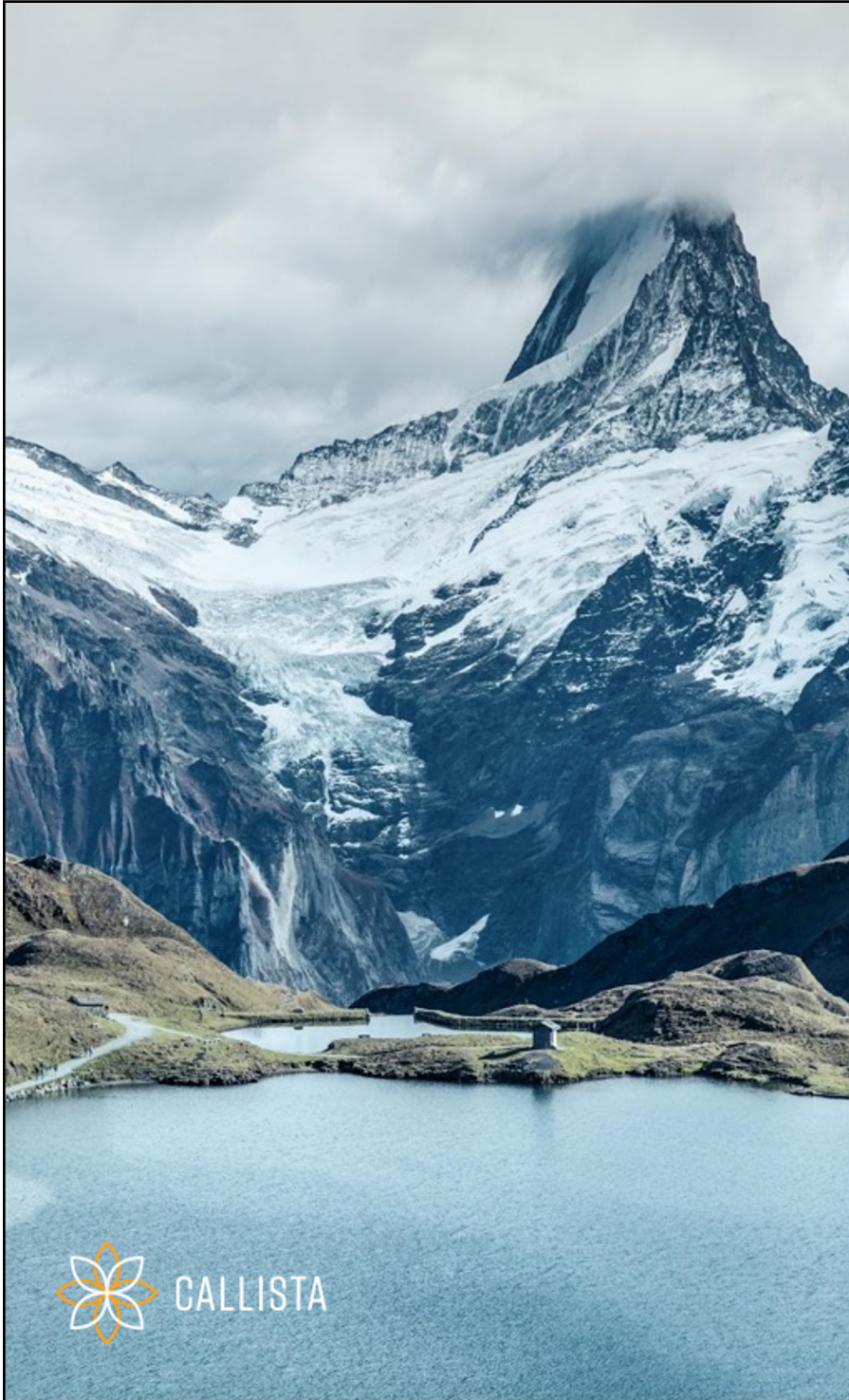


## Drill-up on dimensions

- Grouping on higher hierarchy levels of dimension
- Aggregation of facts along dimensions

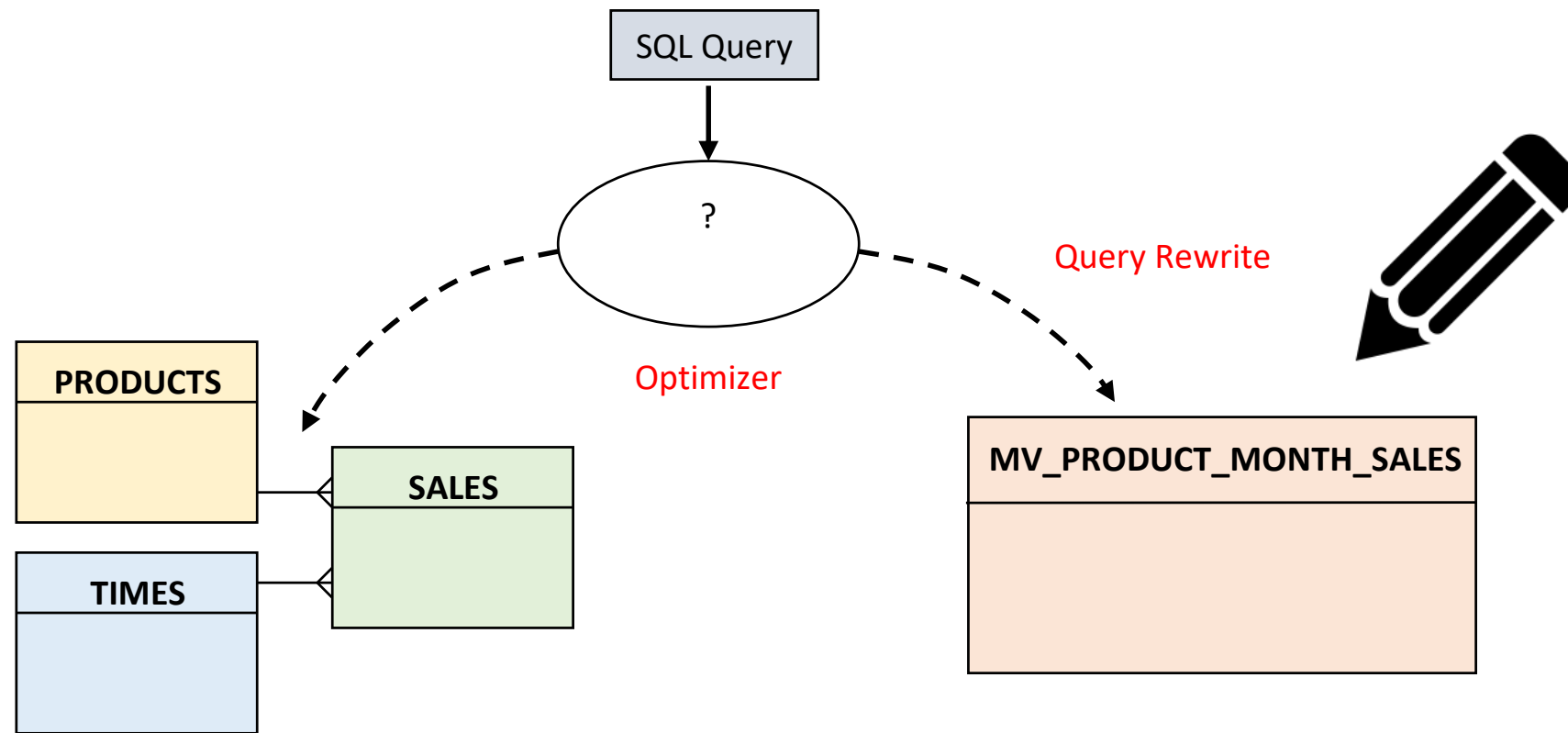
## Challenge for optimizer

- For aggregation, many (or all) rows in fact table must be read
- How can a full table scan on entire fact table be avoided?
- Indexing does not help here



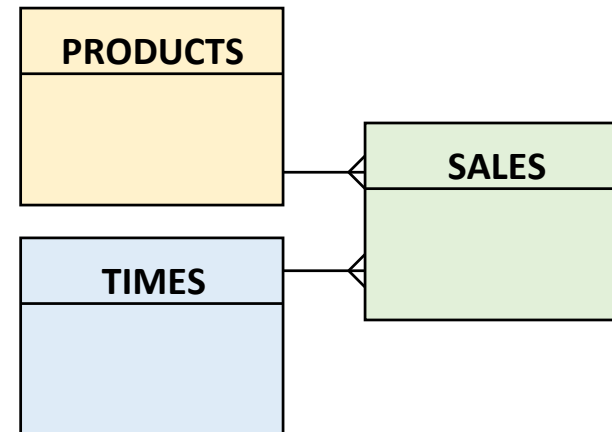
Materialized  
Views

# Materialized View and Query Rewrite



# Create Materialized View

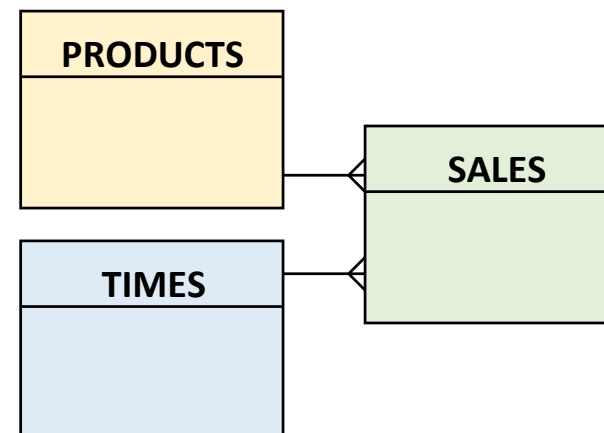
```
CREATE MATERIALIZED VIEW mv_product_month_sales
ENABLE QUERY REWRITE
AS
SELECT t.calendar_month_desc
       , p.prod_name
       , SUM(s.amount_sold)
FROM   sales s
       , times t
       , products p
WHERE  t.time_id = s.time_id
       AND p.prod_id = s.prod_id
GROUP BY t.calendar_month_desc, p.prod_name
```



# Query Rewrite

## Query Rewrite with Full Text Match

```
SELECT t.calendar_month_desc
       , p.prod_name
       , SUM(s.amount_sold)
FROM sales s
       , times t
       , products p
WHERE t.time_id = s.time_id
      AND p.prod_id = s.prod_id
GROUP BY t.calendar_month_desc, p.prod_name
```



Id	Operation	Name
0	SELECT STATEMENT	
1	MAT_VIEW REWRITE ACCESS FULL	MV_PRODUCT_MONTH_SALES

MV\_PRODUCT\_MONTH\_SALES





# Materialized Views in DWH

## Materialized Views on Star Schema

- Joins between fact table and dimension tables
- Pre-aggregated facts on often used hierarchy levels
- Query rewrite on higher hierarchy levels
- Refresh at the end of ETL job

## Further information:

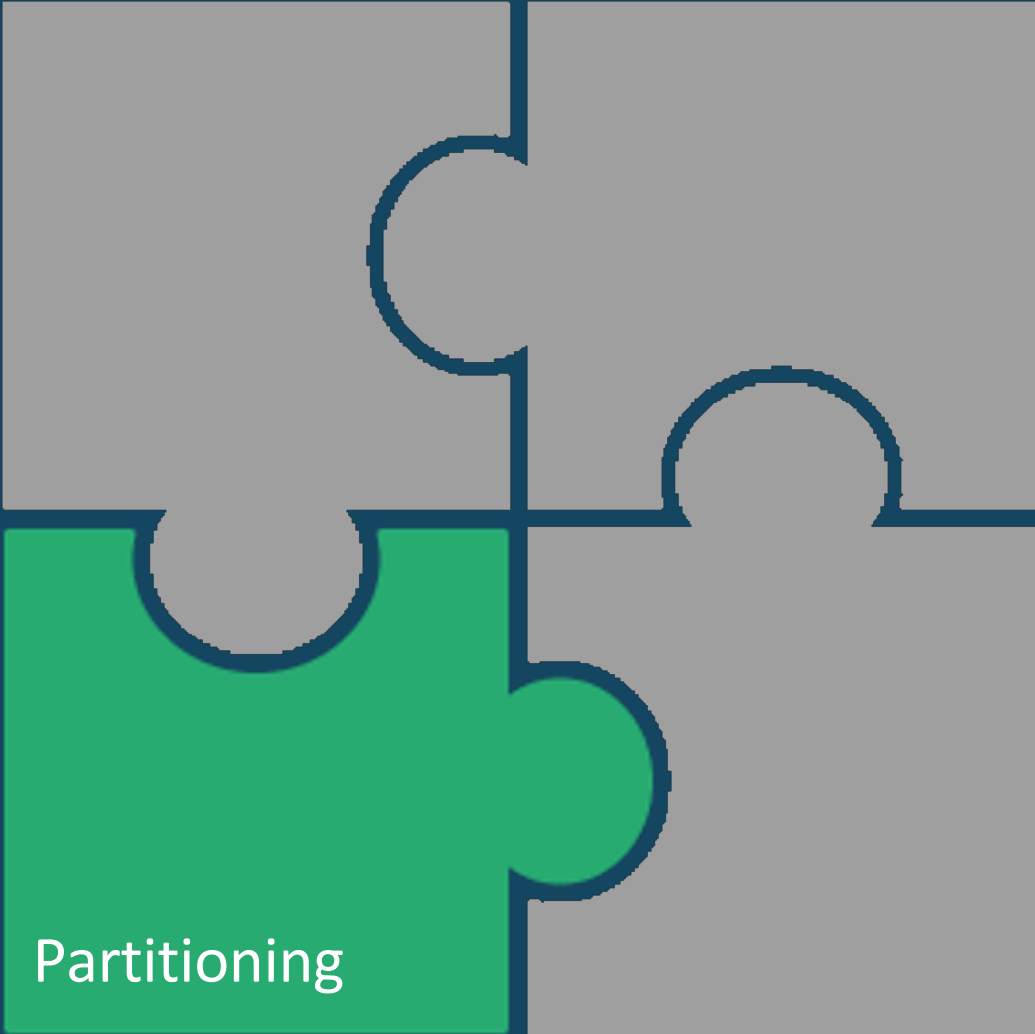
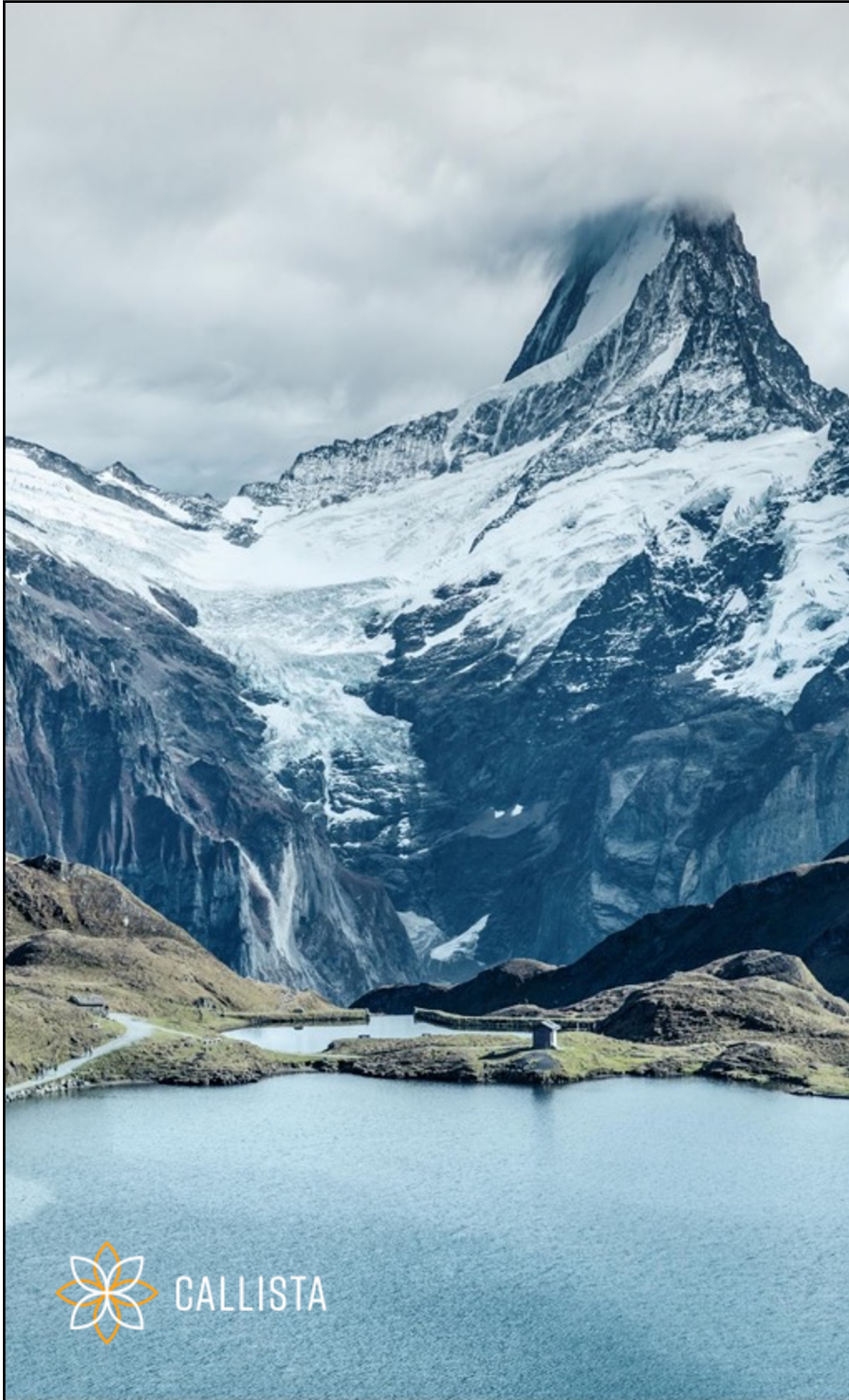
- [Query Rewrite: The Supreme League of Materialized Views](#)
- [Design Tips for Query Rewrite](#)





# Live Demo

## Query Rewrite



Partitioning

# Advantages of Partitioning

- Query Performance
- ETL Performance
- Information Lifecycle Management
- Database Administration

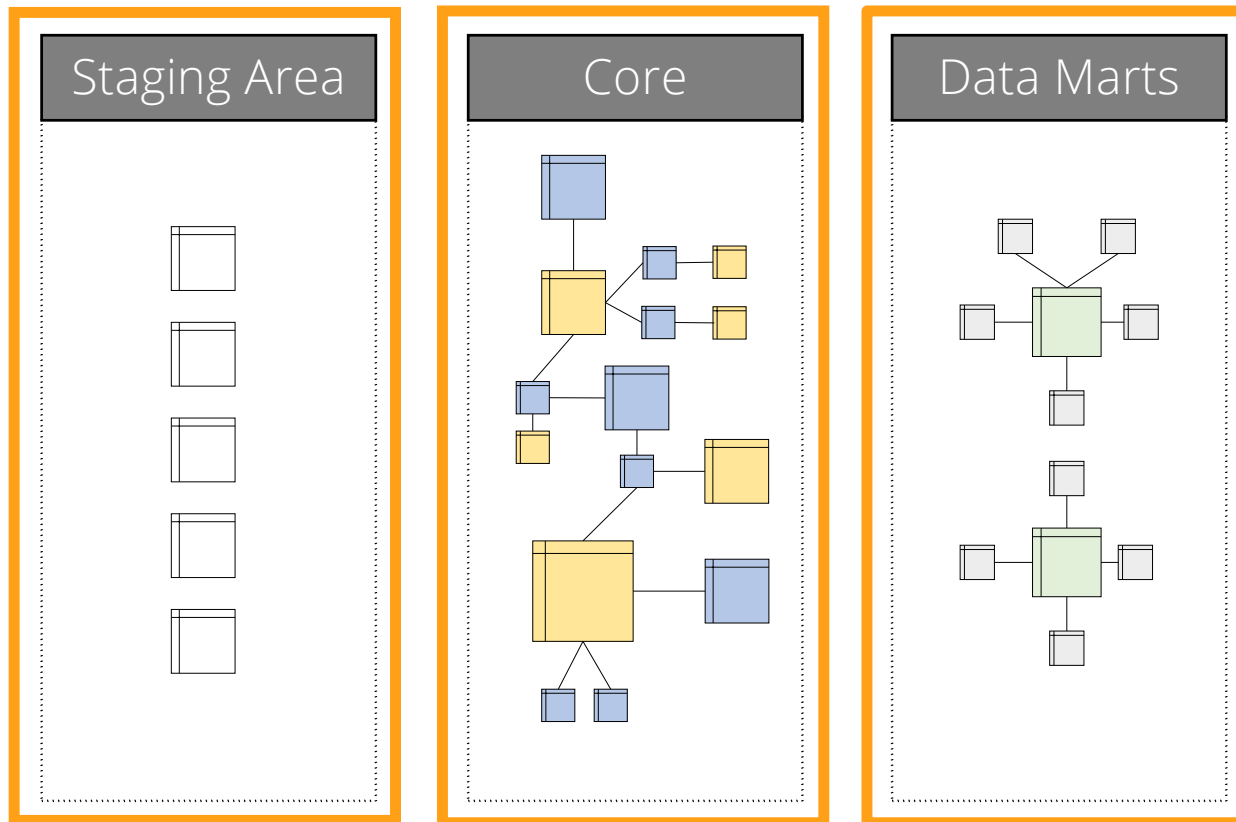
- Partition pruning
- Partition-wise join

- Parallelization of load jobs
- Partition exchange loading

- Moving time windows
- Delete / archive outdated data

- Backups current partitions only
- Compress historical partitions

# Partitioning in Data Warehouse



## Staging Area

- Partitioning can be useful if multiple versions of loads must be archived

## Core

- Different partitioning strategies, mainly to improve ETL performance

## Data Marts

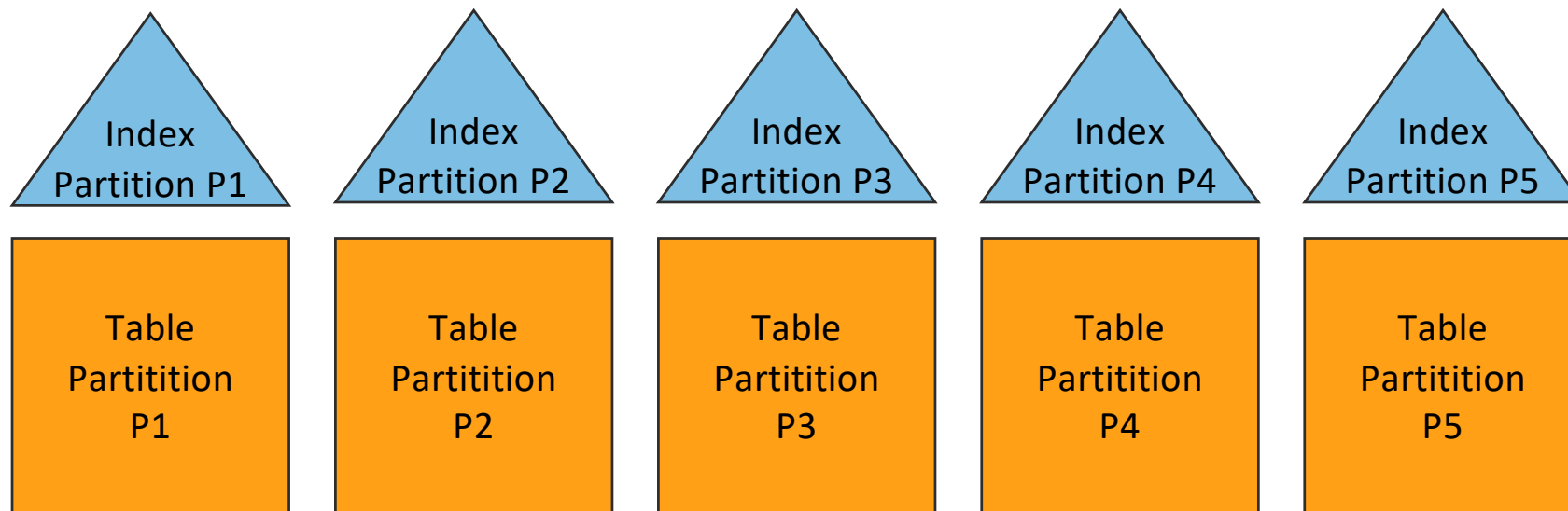
- Partitioning fact tables on time dimension key

# Information Lifecycle Management

- RANGE or INTERVAL partitioning by date
- Outdated data can be archived/deleted easily

<del>Jan 20</del>	<del>Feb 20</del>	<del>Mar 20</del>	<del>Apr 20</del>	<del>Mai 20</del>	<del>Jun 20</del>	<del>Juli 20</del>	<del>Aug 20</del>	<del>Sep 20</del>	<del>Oct 20</del>	<del>Nov 20</del>	<del>Dec 20</del>
Jan 21	Feb 21	Mar 21	Apr 21	Mai 21	Jun 21	Jul 21	Aug 21	Sep 21	Oct 21	Nov 21	Dec 21
Jan 22	Feb 22	Mar 22	Apr 22	Mai 22	Jun 22	Jul 22	Aug 22	Sep 22	Oct 22	Nov 22	Dec 22
Jan 23	Feb 23	Mar 23	Apr 23	Mai 23	Jun 23	Jul 23	Aug 23	Sep 23	Oct 23	Nov 23	Dec 23

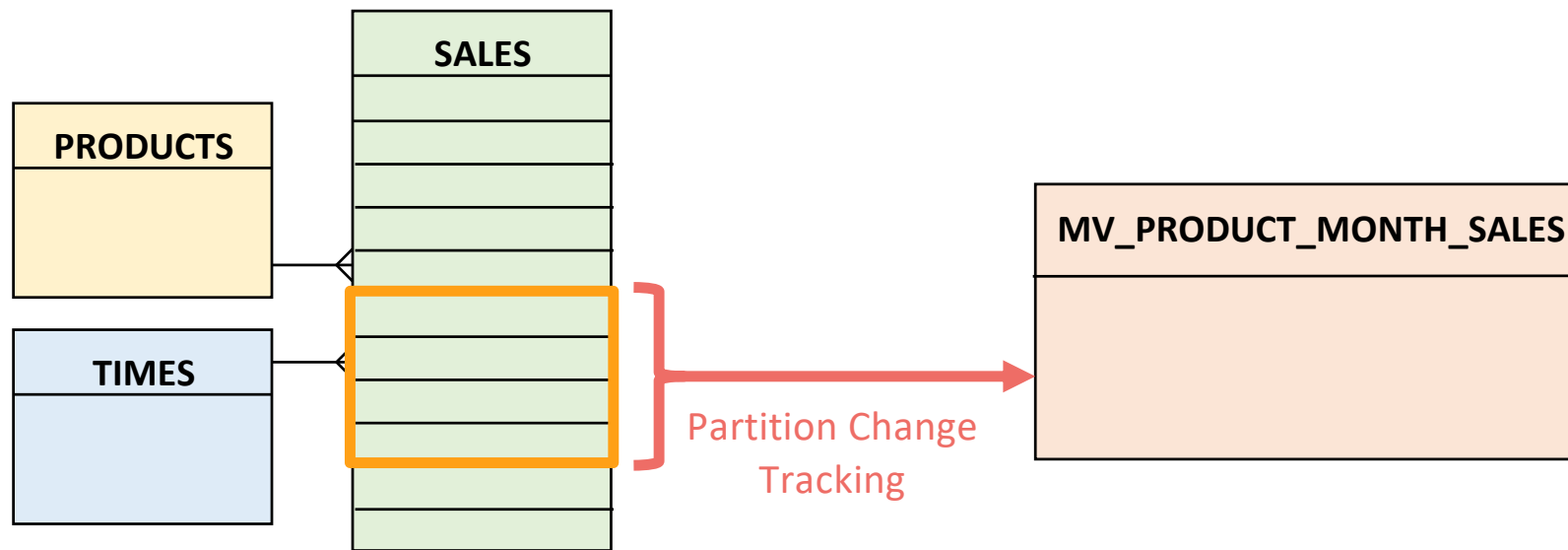
# Partitioning Indexes



## Indexes

- Use local indexes whenever possible
- Bitmap indexes are always local indexes

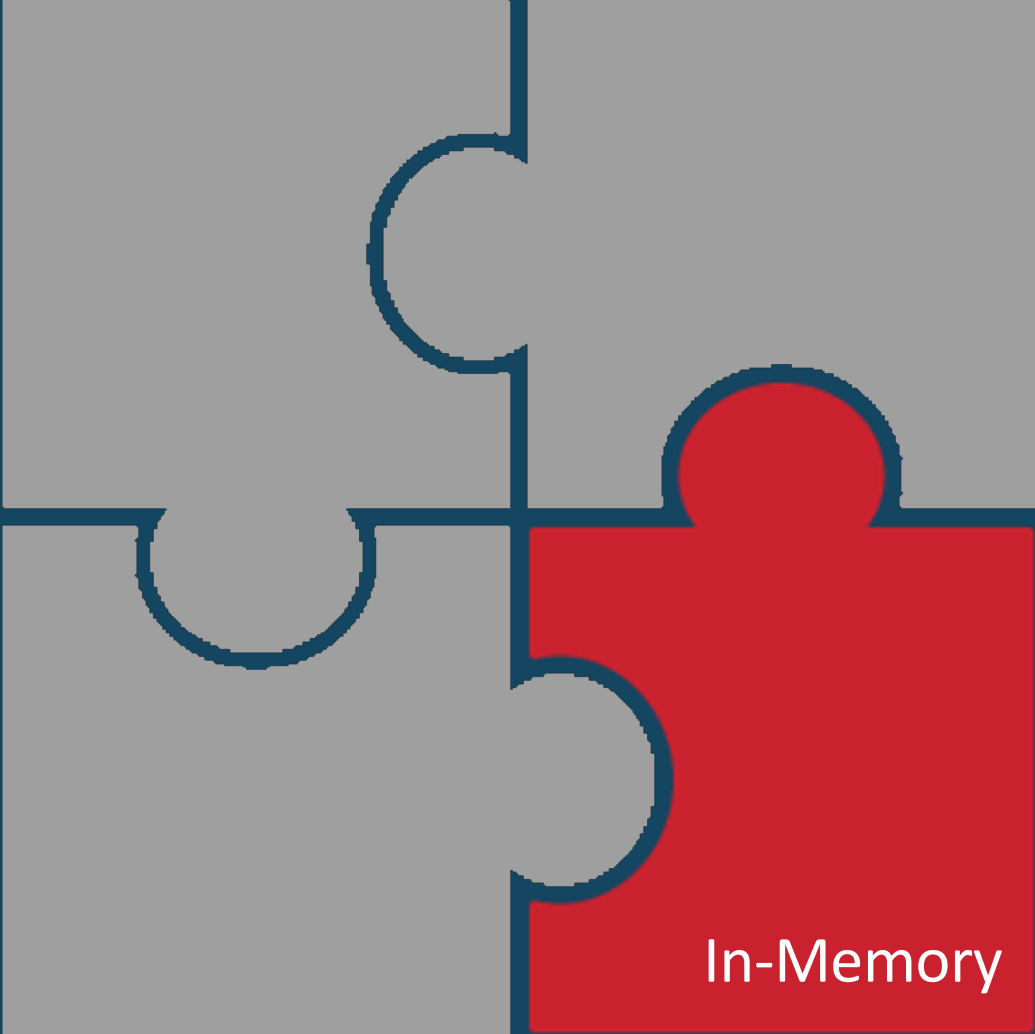
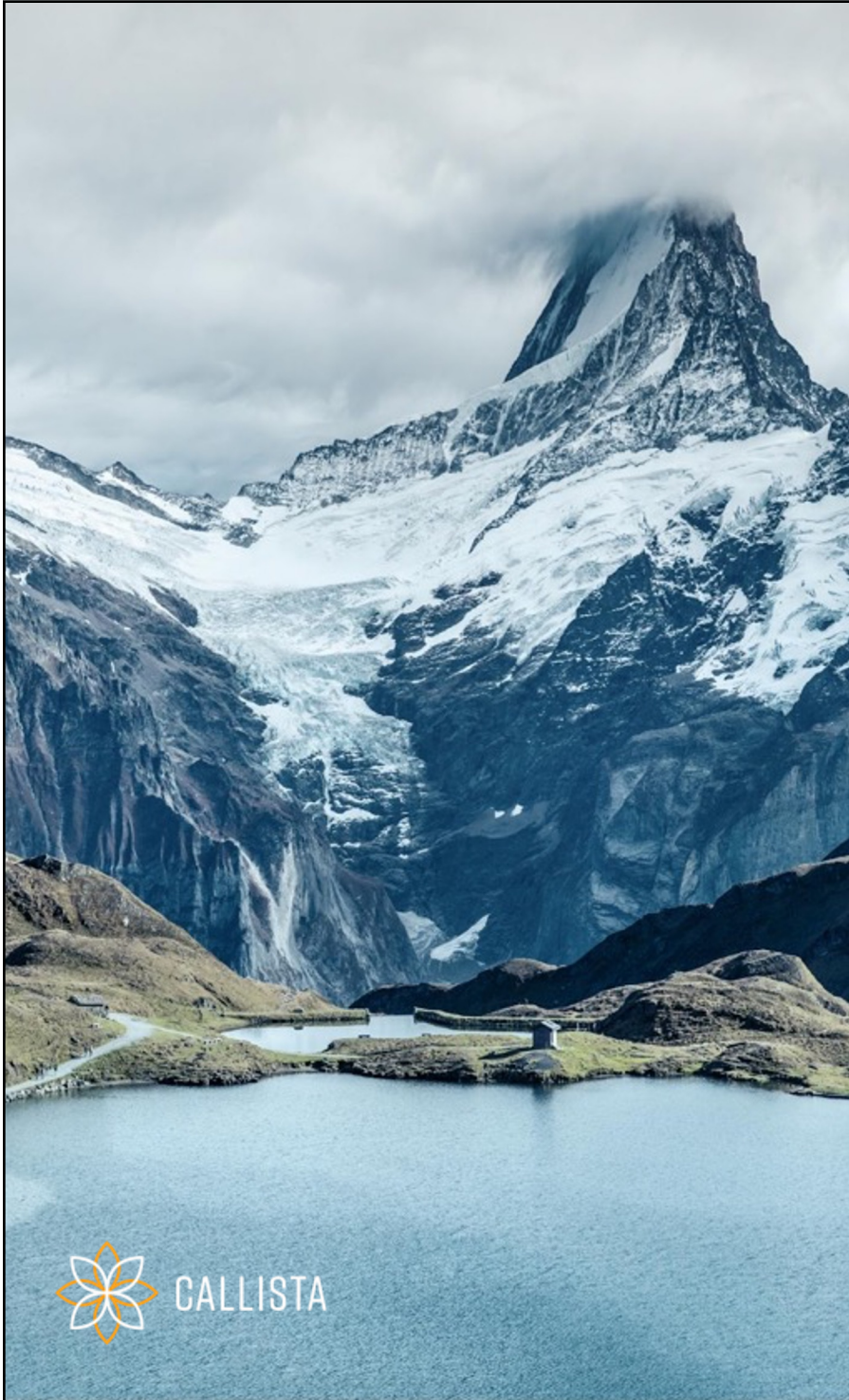
# Partitioning Materialized Views



## Materialized Views

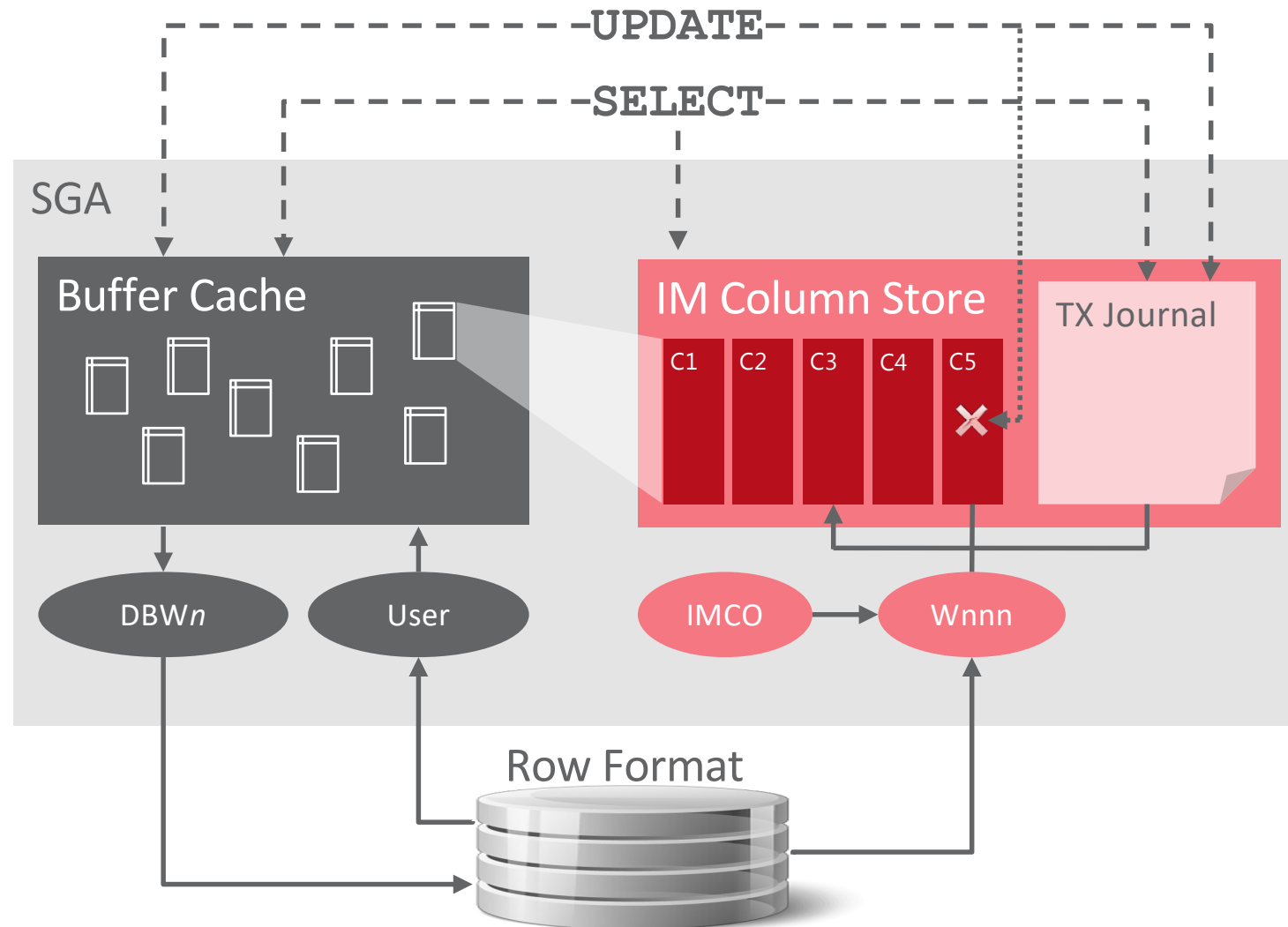
- Materialized views can be partitioned, too
- Partition Change Tracking (PCT) for refresh of materialized views on partitioned tables





In-Memory

# Oracle Database In-Memory Architecture

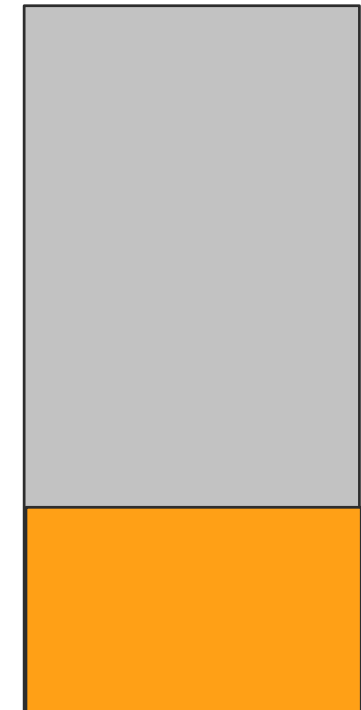
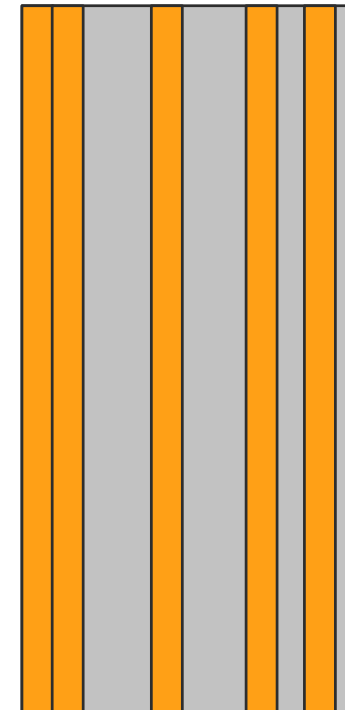
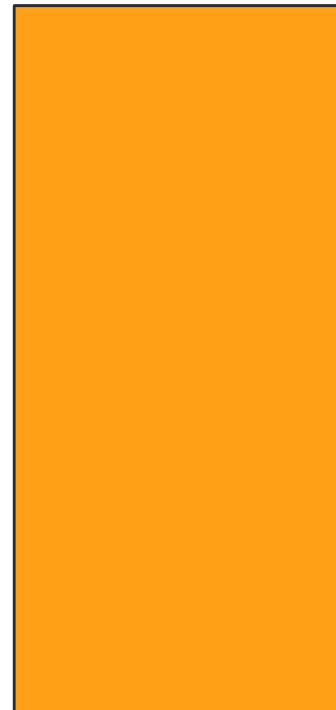


# In-Memory Configuration Options

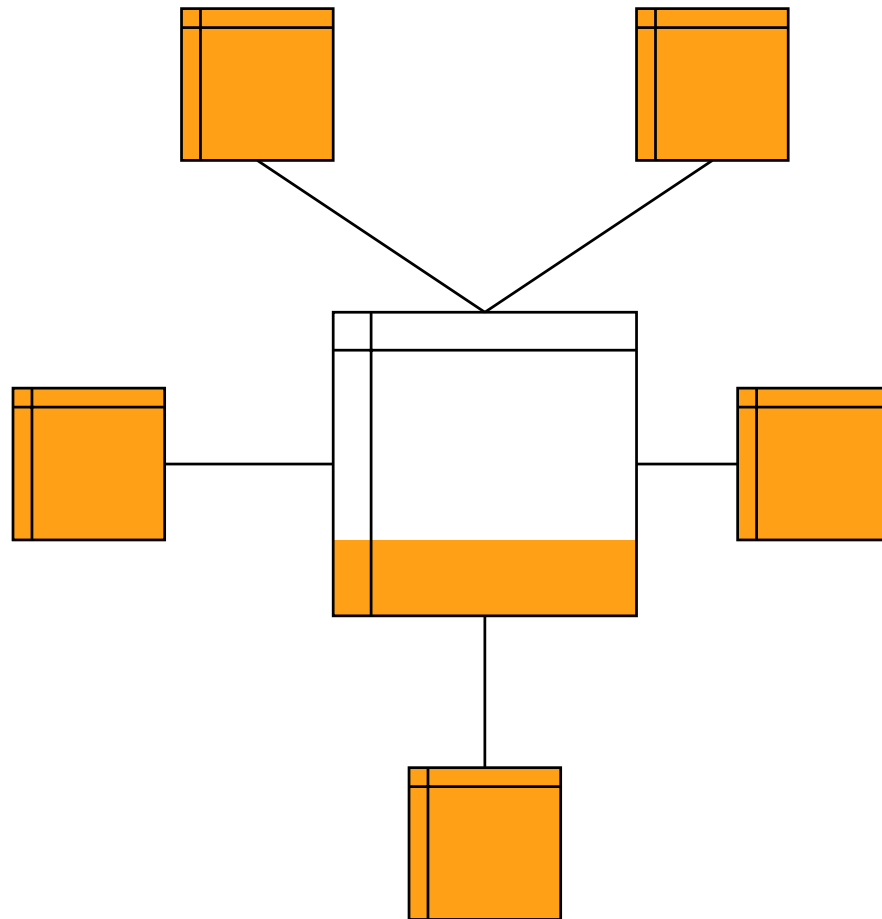
```
ALTER TABLE sales  
INMEMORY
```

```
ALTER TABLE sales  
INMEMORY NO INMEMORY  
(cust_id,time_id)
```

```
ALTER TABLE sales MODIFY  
PARTITION p_2020 INMEMORY
```



# In-Memory for Star Schema



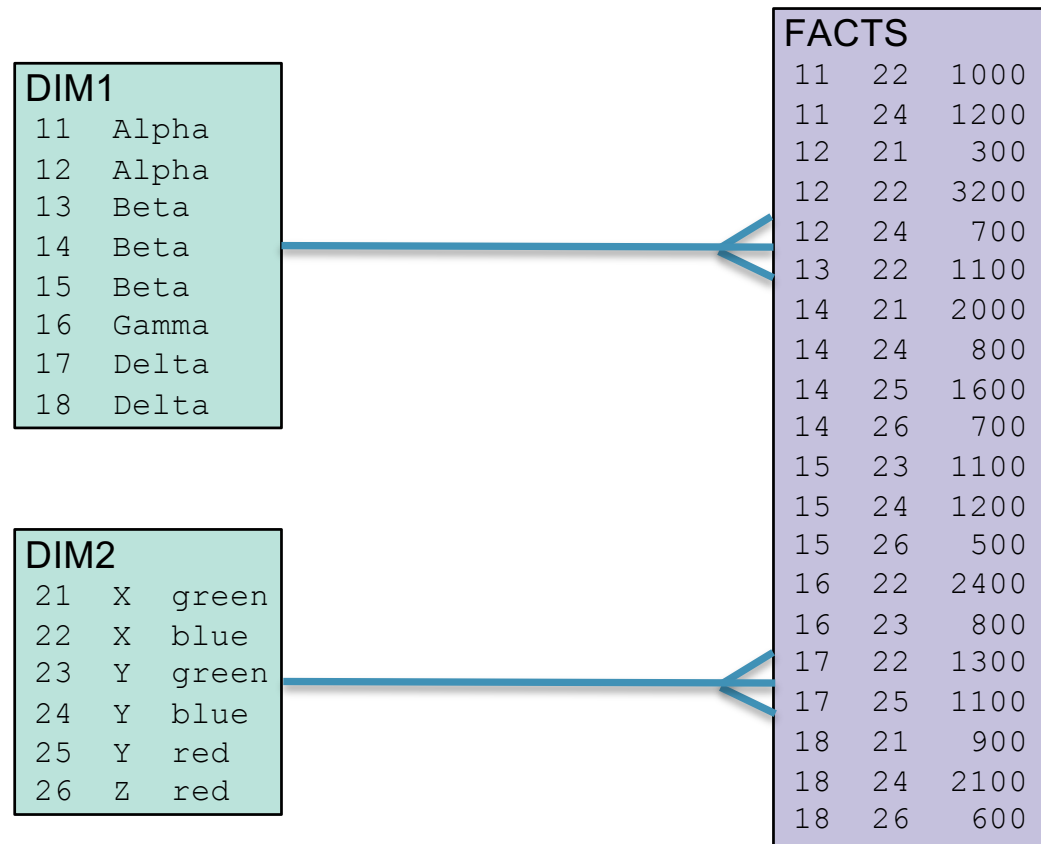
## Dimension tables

- All dimensions are completely populated to In-Memory column store

## Fact tables

- Partitioned by date
- Newest (or often used) partitions populated to In-Memory column store
- Historical partitions not in In-Memory column store

# Vector Transformation



```
SELECT D1, D21, D22
       , SUM(FACTS.F)
FROM FACTS
JOIN DIM1 ON (...)
JOIN DIM2 ON (...)
WHERE D1 IN ('Beta',
            'Gamma')
       AND D21 = 'Y'
GROUP BY D1, D21, D22
```

# Vector Transformation

DIM1		KV1
11	Alpha	0
12	Alpha	0
13	Beta	1
14	Beta	1
15	Beta	1
16	Gamma	2
17	Delta	0
18	Delta	0

TMP1	
1	Beta
2	Gamma

DIM2			KV2
21	X	green	0
22	X	blue	0
23	Y	green	1
24	Y	blue	2
25	Y	red	3
26	Z	red	0

TMP2	
1	Y green
2	Y blue
3	Y red

FACTS			
11	22	1000	0 0
11	24	1200	0 2
12	21	300	0 0
12	22	3200	0 0
12	24	700	0 2
13	22	1100	1 0
14	21	2000	1 0
14	24	800	1 2
14	25	1600	1 3
14	26	700	1 0
15	23	1100	1 1
15	24	1200	1 2
15	26	500	1 0
16	22	2400	2 0
16	23	800	2 1
17	22	1300	0 0
17	25	1100	0 3
18	21	900	0 0
18	24	2100	0 2
18	26	600	0 0

Beta	Y	green	1100
Beta	Y	blue	2000
Beta	Y	red	1600
Gamma	Y	green	800



# Live Demo

## Vector Transformation

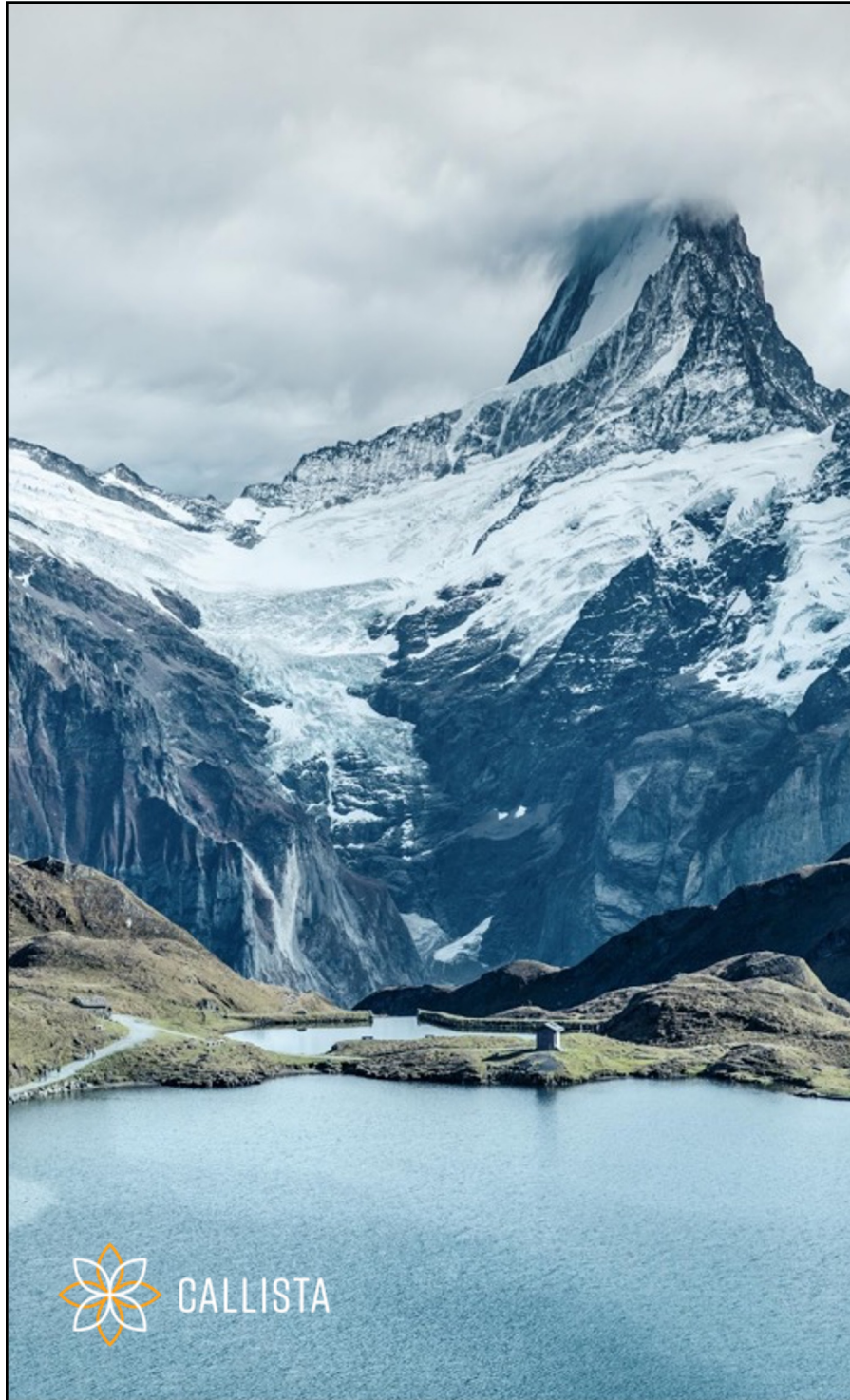
# Combining Performance Features

Indexing

Materialized  
Views

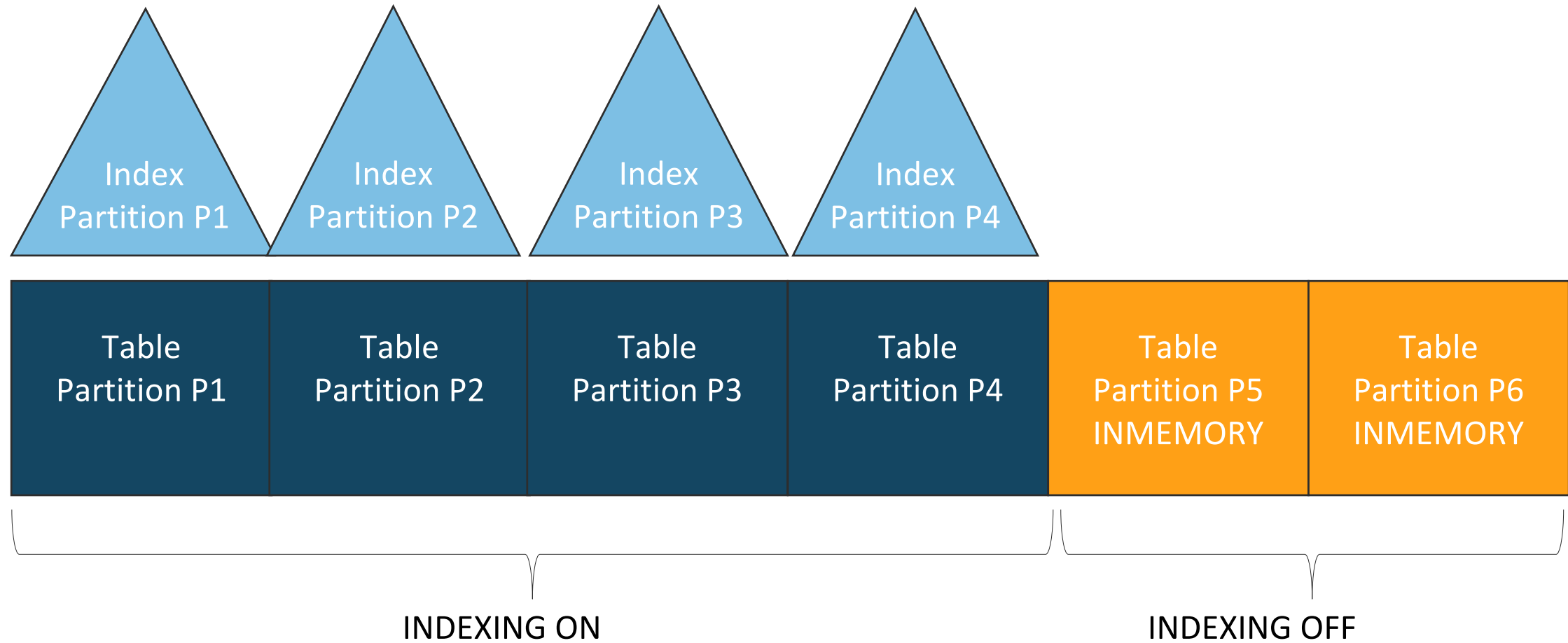
Partitioning

In-Memory





# In-Memory & Partial Indexes



# In-Memory & Partial Indexes

```
SELECT partition_position pos, partition_name, indexing, inmemory
FROM user_tab_partitions WHERE table_name = 'SALES'
ORDER BY partition_position;
```

POS	PARTITION_NAME	INDEXING	INMEMORY
1	SALES_2018	ON	DISABLED
2	SALES_H1_2019	ON	DISABLED
3	SALES_H2_2019	ON	DISABLED
4	SALES_Q1_2020	ON	DISABLED
5	SALES_Q2_2020	ON	DISABLED
6	SALES_Q3_2020	ON	DISABLED
7	SALES_Q4_2020	ON	DISABLED
8	SALES_Q1_2021	ON	DISABLED
9	SALES_Q2_2021	ON	DISABLED
10	SALES_Q3_2021	ON	DISABLED
11	SALES_Q4_2021	ON	DISABLED
12	SALES_Q1_2022	OFF	ENABLED
13	SALES_Q2_2022	OFF	ENABLED
14	SALES_Q3_2022	OFF	ENABLED
15	SALES_Q4_2022	OFF	ENABLED



# Live Demo

## In-Memory & Partial Local Indexes

## Indexing

- B-tree indexes for primary key / unique constraints
- Bitmap indexes on dimension keys of fact tables

## Materialized Views & Query Rewrite

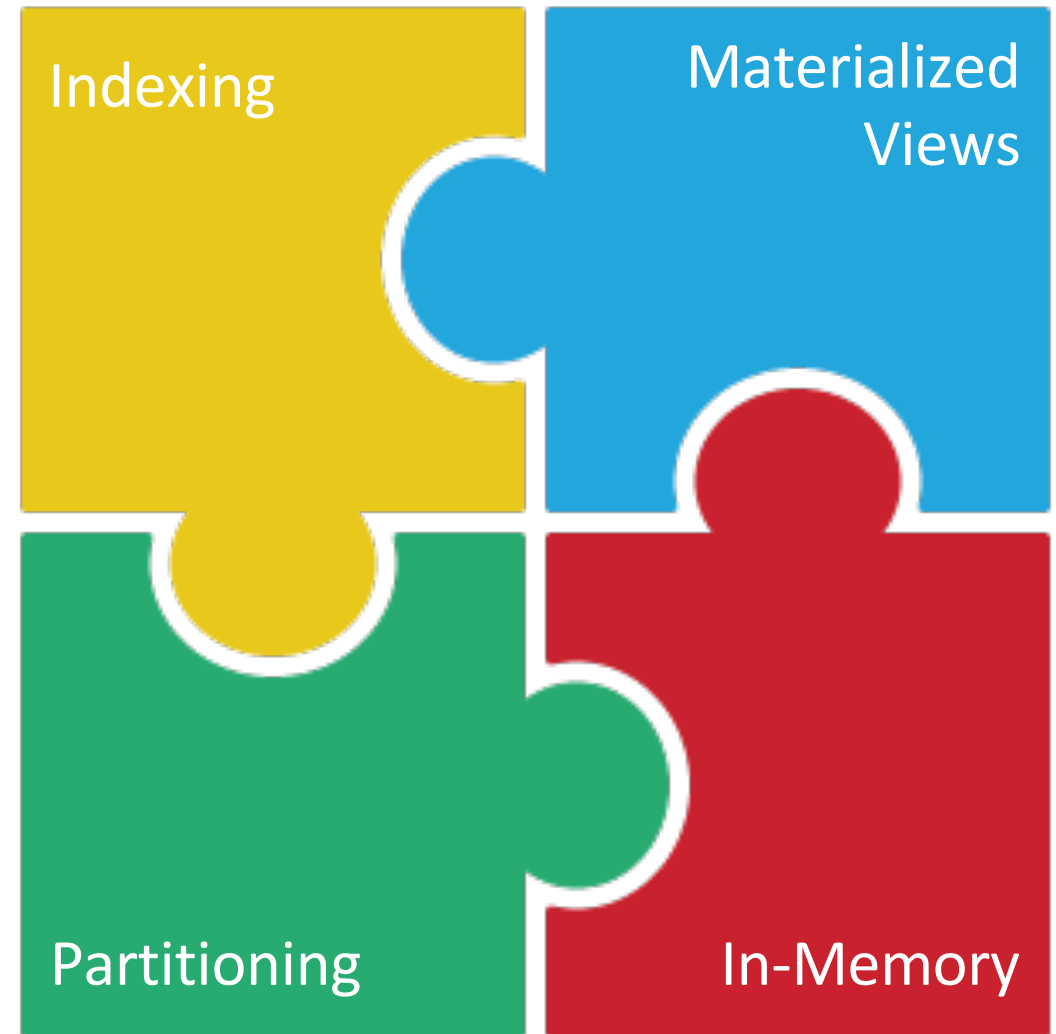
- Useful for drill-up on higher hierarchy levels
- Query Rewrite for queries on star schemas

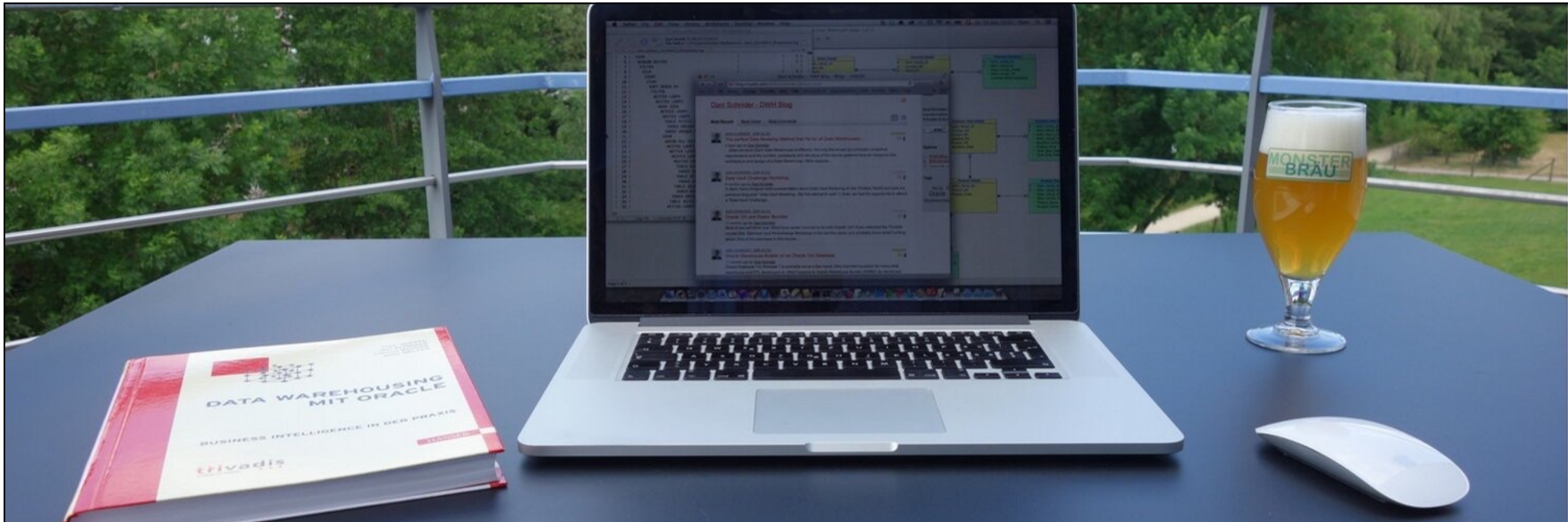
## Partitioning

- RANGE/INTERVAL partitioning on date columns
- Mainly used for fact tables in star schemas

## Database In-Memory

- High performance improvements in star schemas
- No additional indexes (except PK/UK) required anymore





 <https://danischnider.wordpress.com>

 [https://twitter.com/dani\\_schnider](https://twitter.com/dani_schnider)

 <https://www.linkedin.com/in/danischnider/>

# Q&A

The Educational Conference For Oracle Technology Users

ODTUG  
Kscope23  
aurora, co    june 25-29



Interested in volunteering?  
Scan the code to sign up



Don't Forget To  
Fill Out Your Evals

A wide-angle landscape photograph of a mountain range. The foreground features a calm, blue lake with a small structure on the shore. The middle ground shows rolling hills with sparse vegetation. The background is dominated by towering, rugged mountains with significant snow cover and patches of glaciers. The sky is filled with soft, grey clouds.

**DRIVEN BY EXCELLENCE**