Vertica Live Aggregate Projections
Modern Materialized Views for Big Data

Nga Tran - HPE Vertica - Nga.Tran@hpe.com
November 2015
Outline

• What is Big Data?
• How Vertica provides Big Data Solutions?
• What are Materialized Views (MV)s?
• What are the Roles of MVs in Big Data?
• How does Vertica implement their MVs?
  • Why Vertica names their MVs Live Aggregate Projections (LAPs)?
  • What are the differences between LAPS & MVs?
What is Big Data?
What is Big Data?

What you have and what you want?
What is Big Data?

What you have and what you want?

- You have a lot of data.
What is Big Data?

What you have and what you want?

- You have a lot of data.
- You want to mine your data for “treasures”
What is Big Data?

What you have and what you want?

- You have a lot of data.
- You want to mine your data for “treasures”
- You want sub-second mining processes. Why?
What is Big Data?

What you have and what you want?

- You have a lot of data.
- You want to mine your data for “treasures”
- You want sub-second mining processes. Why?
  - “Treasures” no more after that time.
What is Big Data?

What you have and what you want?

- You have a lot of data.
- You want to mine your data for “treasures”
- You want sub-second mining processes. Why?
  - “Treasures” no more after that time.
  - “Treasures” help with urgent decisions.
What is Big Data?

Recap: What you have and what you want?
What is Big Data?

Recap: What you have and what you want?

- **HAVE**: a lot of data.
- **WANT**: dig them for treasures in sub-second
What is Big Data?

How you make it?
What is Big Data?

How you make it?

• **Places to store and access your data safely & efficiently.**
What is Big Data?

How you make it?

• **Places to store and access** your data safely & efficiently.

• **Tools to analyze** them quickly.
What is Big Data?

How you make it?

• Places to store and access your data safely & efficiently.

• Tools to analyze them quickly.

• [Columnar] Database

• Analytic Tools
What is Big Data?

How you make it?

- **Places to store and access your data safely & efficiently.**
  - [Columnar] Database

- **Tools to analyze them quickly.**
  - Analytic Tools
    - [Vertica] Analytic Databases
Outline

• What is Big Data?
• How Vertica provides Big Data Solutions?
• What are Materialized Views (MVs)?
• What are the Roles of MVs in Big Data?
• How does Vertica implement their MVs?
  • Why Vertica names their MVs Live Aggregate Projections (LAPs)?
  • What are the differences between LAPS & MVs?
How Vertica provides Big Data Solutions?
How Vertica provides Big Data Solutions?

- Same answers of “What is Vertica Analytic Database?”
What is Vertica Analytic Database?
Columnar and MPP RDBMS and More
Columnar and MPP RDBMS and More

Column Store

Conventional databases store a “heap file” with all records in row order

Column stores store each column separately, so you only read what you need

```
SELECT id, game2_score FROM scores;
```

<table>
<thead>
<tr>
<th>id</th>
<th>name</th>
<th>game1_score</th>
<th>game2_score</th>
<th>profile_txt</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Michael</td>
<td>1120</td>
<td>42</td>
<td>&lt;long string&gt;</td>
</tr>
<tr>
<td>2</td>
<td>Christopher</td>
<td>9001</td>
<td>17</td>
<td>&lt;long string&gt;</td>
</tr>
<tr>
<td>3</td>
<td>Matthew</td>
<td>4041</td>
<td>17</td>
<td>&lt;long string&gt;</td>
</tr>
<tr>
<td>4</td>
<td>Jessica</td>
<td>7218</td>
<td>42</td>
<td>&lt;long string&gt;</td>
</tr>
<tr>
<td>5</td>
<td>Ashley</td>
<td>3364</td>
<td>42</td>
<td>&lt;long string&gt;</td>
</tr>
</tbody>
</table>

Sorted Column Store

Want the highest values? Trivial if it’s sorted.

```
SELECT id, game2_score
FROM scores
ORDER BY game2_score DESC
LIMIT 3;
```

Joining two tables? If they’re sorted on the join key, just line up the rows.

```
SELECT scores.name, 
    MAX(sessions.last_login_time)
FROM scores, sessions
WHERE scores.id = sessions.user_id
GROUP BY scores.id, scores.name;
```
Sorted fields often have lots of duplicate values. Let’s de-duplicate!

- Saves lots of disk space, disk IO
- Values don’t need to be identical
- They will be similar; many compression schemes work well
- Can compute directly on compressed data for more performance
  - \( \text{COUNT(column)} \) secretly rewritten as \( \text{SUM(run_count)} \)
  - \( \text{SUM(column)} \) secretly rewritten as \( \text{SUM(run_count} \times \text{value}) \)
## Columnar and MPP RDBMS and More

**Distributed** **Compressed** **Sorted**  **Column Store**

- We’re already lining up join keys
- Let’s segment the data across machines
  - Enforce that matching values are always on the same node
  - Means we can JOIN in parallel

### Example Table

<table>
<thead>
<tr>
<th>name</th>
<th>id</th>
<th>user_id</th>
<th>last_login_time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Michael</td>
<td>1</td>
<td>1 x3</td>
<td>7:32 PM, 5:18 PM, 4:59 PM</td>
</tr>
<tr>
<td>Christopher</td>
<td>2</td>
<td>2 x2</td>
<td>3:02 PM, 1:03 PM</td>
</tr>
</tbody>
</table>

### SQL Query

```
SELECT * FROM scores, sessions;
```
The world is moving away from straight Map/Reduce; back towards transactional systems

- Google (“F1: A Distributed SQL Database That Scales”; VLDB 2013)
- Yahoo (“Peta-Scale Data Warehousing at Yahoo!”; Sigmod 2009)

Consistency is really hard!
If you leave it to each application developer, rather than guaranteeing it always, you get bugs.
Columnar and MPP RDBMS and More

ACID-Compliant Distributed Compressed Sorted

The problem: Data being loaded and queried on all nodes by many users all at once

- Must provide a consistent view of the data
- Either you see the whole change (on all machines) or none of it
- Must scale linearly (more nodes + more data == same performance)

Node_01

<table>
<thead>
<tr>
<th>name</th>
<th>id</th>
<th>user_id</th>
<th>last_login_time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Michael</td>
<td>1</td>
<td>1 x3</td>
<td>7:32 PM</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>5:18 PM</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>4:59 PM</td>
</tr>
</tbody>
</table>

Node_02

<table>
<thead>
<tr>
<th>name</th>
<th>id</th>
<th>user_id</th>
<th>last_login_time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Christopher</td>
<td>2</td>
<td>2 x2</td>
<td>3:02 PM</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1:03 PM</td>
</tr>
</tbody>
</table>
Columnar and MPP RDBMS and More

**ACID-Compliant**  **Distributed**  **Compressed**  **Sorted**

The solution: Distributed Agreement
- “Spread” (open-source library)

- Send carefully-designed control messages around a ring of nodes
- Ring enforces order
- Arbitrary/unordered processing between messages

Is the above diagram really accurate? No. If you want to know more, come interview with us!
Columnar and MPP RDBMS and More

UPDATE
SELECT
INSERT

a = true
WHEN b > 10

Query Optimizer

Node_01

name | id | user_id | last_login_time
--- | --- | --- | ---
Michael | 1 | 1 x3 | 7:32 PM
 | | | 5:18 PM
 | | | 4:59 PM

Christopher | 2 | 2 x2 | 3:02 PM
 | | | 1:03 PM

Node_02
Columnar and MPP RDBMS and More

Query Optimizer

Generating Optimal Query Plan

SQL Query → Query Optimizer

Statistics & Histogram

Query Plan
Database Designer

Designing Optimal Projection Set for an available Row-Stored Schema

Scenario: Workload + Space Budget

Tables & Constraints
Data Set
Queries

DBD

Projection Set
Query Performance
Storage Footprint
Fault Tolerance
Recovery
Columnar and MPP RDBMS and More

Tuple Mover

Combining small storage containers into larger ones
Columnar and MPP RDBMS and More

Analytic Queries and Functions

Built-in Analytic Queries and Functions

• Time Series: evaluate the values of a given set of variables over time and group those values into a window

```sql
SELECT symbol, slice_time, TS_FIRST_VALUE(bid1) AS first_bid
FROM Tickstore
WHERE symbol IN ('MSFT', 'IBM')
TIMESERIES slice_time AS '5 seconds' OVER (PARTITION BY symbol ORDER BY ts)
```

• Event Series Pattern Matching

```sql
SELECT uid, sid, ts, refurl, pageurl, action, event_name(), pattern_id(), match_id()
FROM clickstream_log
MATCH (PARTITION BY uid, sid ORDER BY ts
DEFINE Entry AS RefURL NOT ILIKE '%website2.com%' AND PageURL ILIKE '%website2.com%',
    Onsite AS PageURL ILIKE '%website2.com%' AND Action='V',
    Purchase AS PageURL ILIKE '%website2.com%' AND Action = 'P'
PATTERN P AS (Entry Onsite* Purchase)
ROWS MATCH FIRST EVENT);
```
Vertica can run R, C++ and Java extensions internally.

We’ve even enabled parallel/distributed execution within R itself

• “distributedR”
• Extensive collaboration with HP Labs and various universities
• “Presto: Distributed Machine Learning and Graph Processing with Sparse Matrices” (Eurosys 2013)
• Can use Vertica as a data store and preprocessor
Columnar and MPP RDBMS and More

Example data:

<table>
<thead>
<tr>
<th>name</th>
<th>id</th>
<th>user_id</th>
<th>last_login_time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Michael</td>
<td>1</td>
<td>1 x3</td>
<td>7:32 PM 5:18 PM 4:59 PM</td>
</tr>
<tr>
<td>Christopher</td>
<td>2</td>
<td>2 x2</td>
<td>3:02 PM 1:03 PM</td>
</tr>
</tbody>
</table>

Diagram:

- UPDATE
- SELECT
- INSERT
- Query Optimizer
- HDFS
- DBD
- UDX
- TM
- R
- Node_01
- Node_02
- LAPs
- Text Search
- BI Tools
- Web App
- Flex (Unstructured)
This talk is about LAPs implementation

- UPDATE
- SELECT
- INSERT

- R
- TM
- HDFS
- UDx
- DBD

Node_01

- a = true
- WHEN b > 10

Query Optimizer

Node_02

- LAPs
- Text Search
- Flex (Unstructured)
- BI Tools
- Web App

- A = true
- WHEN b > 10

<table>
<thead>
<tr>
<th>name</th>
<th>id</th>
<th>user_id</th>
<th>last_login_time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Michael</td>
<td>1</td>
<td>1 x3</td>
<td>7:32 PM</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>5:18 PM</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>4:59 PM</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>name</th>
<th>id</th>
<th>user_id</th>
<th>last_login_time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Christopher</td>
<td>2</td>
<td>2 x2</td>
<td>3:02 PM</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1:03 PM</td>
</tr>
</tbody>
</table>
What are LAPs?

- LAPs = Live Aggregate Projections
- ~ Materialized Views in other Databases
Outline

• What is Big Data?
• How Vertica provides Big Data Solutions?
• What are Materialized Views (MV)s?
• What are the Roles of MVs in Big Data?
• How does Vertica implement their MVs?
  • Why Vertica names their MVs Live Aggregate Projections (LAPs)?
  • What are the differences between LAPS & MVs?
What are Materialized Views (MVs)?

Redundant Data Storages for Performance Purposes
What are Materialized Views (MVs)?
Redundant Data Storages for Performance Purposes

**Example 1: Avoid Expensive Joins**

**SalesData**

<table>
<thead>
<tr>
<th>Emp</th>
<th>DateOfSale</th>
<th>Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nga</td>
<td>2013-12-12</td>
<td>15K</td>
</tr>
<tr>
<td>Natalya</td>
<td>2013-12-30</td>
<td>7K</td>
</tr>
<tr>
<td>Jaimin</td>
<td>2014-01-10</td>
<td>3K</td>
</tr>
<tr>
<td>Nga</td>
<td>2014-01-20</td>
<td>11K</td>
</tr>
<tr>
<td>Jaimin</td>
<td>2014-01-30</td>
<td>10K</td>
</tr>
</tbody>
</table>

**Employee**

<table>
<thead>
<tr>
<th>EmpID</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jaimin</td>
<td>Boston</td>
</tr>
<tr>
<td>Natalya</td>
<td>New York</td>
</tr>
<tr>
<td>Nga</td>
<td>Chicago</td>
</tr>
</tbody>
</table>

```sql
SELECT Emp, DateOfSale, Amount, Location
FROM SalesData, Employee
WHERE Emp = EmpID;
```
What are Materialized Views (MVs)?  
Redundant Data Storages for Performance Purposes

Example 1: Avoid Expensive Joins

SalesData

<table>
<thead>
<tr>
<th>Emp</th>
<th>DateOfSale</th>
<th>Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nga</td>
<td>2013-12-12</td>
<td>15K</td>
</tr>
<tr>
<td>Natalya</td>
<td>2013-12-30</td>
<td>7K</td>
</tr>
<tr>
<td>Jaimin</td>
<td>2014-01-10</td>
<td>3K</td>
</tr>
<tr>
<td>Nga</td>
<td>2014-01-20</td>
<td>11K</td>
</tr>
<tr>
<td>Jaimin</td>
<td>2014-01-30</td>
<td>10K</td>
</tr>
</tbody>
</table>

Employee

<table>
<thead>
<tr>
<th>EmpID</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jaimin</td>
<td>Boston</td>
</tr>
<tr>
<td>Natalya</td>
<td>New York</td>
</tr>
<tr>
<td>Nga</td>
<td>Chicago</td>
</tr>
</tbody>
</table>

SELECT Emp, DateOfSale, Amount, Location  
FROM SalesData, Employee  
WHERE Emp = EmpID;

Normal Way: Run above query and get the results.
What are Materialized Views (MVs)?

Redundant Data Storages for Performance Purposes

Example 1: Avoid Expensive Joins

SalesData

<table>
<thead>
<tr>
<th>Emp</th>
<th>DateOfSale</th>
<th>Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nga</td>
<td>2013-12-12</td>
<td>15K</td>
</tr>
<tr>
<td>Natalya</td>
<td>2013-12-30</td>
<td>7K</td>
</tr>
<tr>
<td>Jaimin</td>
<td>2014-01-10</td>
<td>3K</td>
</tr>
<tr>
<td>Nga</td>
<td>2014-01-20</td>
<td>11K</td>
</tr>
<tr>
<td>Jaimin</td>
<td>2014-01-30</td>
<td>10K</td>
</tr>
</tbody>
</table>

Employee

<table>
<thead>
<tr>
<th>EmplID</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jaimin</td>
<td>Boston</td>
</tr>
<tr>
<td>Natalya</td>
<td>New York</td>
</tr>
<tr>
<td>Nga</td>
<td>Chicago</td>
</tr>
</tbody>
</table>

SELECT Emp, DateOfSale, Amount, Location
FROM SalesData, Employee
WHERE Emp = EmpID;

Normal Way: Run above query and get the results.

If many joins and a lot of data, the query can be slow
What are Materialized Views (MVs)?

Redundant Data Storages for Performance Purposes

Example 1: Avoid Expensive Joins

SalesData

<table>
<thead>
<tr>
<th>Emp</th>
<th>DateOfSale</th>
<th>Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nga</td>
<td>2013-12-12</td>
<td>15K</td>
</tr>
<tr>
<td>Natalya</td>
<td>2013-12-30</td>
<td>7K</td>
</tr>
<tr>
<td>Jaimin</td>
<td>2014-01-10</td>
<td>3K</td>
</tr>
<tr>
<td>Nga</td>
<td>2014-01-20</td>
<td>11K</td>
</tr>
<tr>
<td>Jaimin</td>
<td>2014-01-30</td>
<td>10K</td>
</tr>
</tbody>
</table>

Employee

<table>
<thead>
<tr>
<th>EmpID</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jaimin</td>
<td>Boston</td>
</tr>
<tr>
<td>Natalya</td>
<td>New York</td>
</tr>
<tr>
<td>Nga</td>
<td>Chicago</td>
</tr>
</tbody>
</table>

SELECT Emp, DateOfSale, Amount, Location
FROM   SalesData, Employee
WHERE  Emp = EmpID;

Normal Way: Run above query and get the results.

If many joins and a lot of data, the query can be slow
-> For better performance,
  Pre-join data and insert them into its own table, Sales
What are Materialized Views (MVs)?

Redundant Data Storages for Performance Purposes

Example 1: Avoid Expensive Joins

### SalesData

<table>
<thead>
<tr>
<th>Emp</th>
<th>DateOfSale</th>
<th>Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nga</td>
<td>2013-12-12</td>
<td>15K</td>
</tr>
<tr>
<td>Natalya</td>
<td>2013-12-30</td>
<td>7K</td>
</tr>
<tr>
<td>Jaimin</td>
<td>2014-01-10</td>
<td>3K</td>
</tr>
<tr>
<td>Nga</td>
<td>2014-01-20</td>
<td>11K</td>
</tr>
<tr>
<td>Jaimin</td>
<td>2014-01-30</td>
<td>10K</td>
</tr>
</tbody>
</table>

### Employee

<table>
<thead>
<tr>
<th>EmpID</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jaimin</td>
<td>Boston</td>
</tr>
<tr>
<td>Natalya</td>
<td>New York</td>
</tr>
<tr>
<td>Nga</td>
<td>Chicago</td>
</tr>
</tbody>
</table>

### Sales

<table>
<thead>
<tr>
<th>Emp</th>
<th>DateOfSale</th>
<th>Amount</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nga</td>
<td>2013-12-12</td>
<td>15K</td>
<td>Chicago</td>
</tr>
<tr>
<td>Natalya</td>
<td>2013-12-30</td>
<td>7K</td>
<td>New York</td>
</tr>
<tr>
<td>Jaimin</td>
<td>2014-01-10</td>
<td>3K</td>
<td>Boston</td>
</tr>
<tr>
<td>Nga</td>
<td>2014-01-20</td>
<td>11K</td>
<td>Chicago</td>
</tr>
<tr>
<td>Jaimin</td>
<td>2014-01-30</td>
<td>10K</td>
<td>Boston</td>
</tr>
</tbody>
</table>

The query before becomes simpler and run faster

This simplified query is generated implicitly

```sql
SELECT * from Sales;
```
What are Materialized Views (MVs)?
Redundant Data Storages for Performance Purposes

**Example 1: Avoid Expensive Joins**

<table>
<thead>
<tr>
<th>SalesData</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Emp</td>
<td>DateOfSale</td>
<td>Amount</td>
<td></td>
</tr>
<tr>
<td>Nga</td>
<td>2013-12-12</td>
<td>15K</td>
<td></td>
</tr>
<tr>
<td>Natalya</td>
<td>2013-12-30</td>
<td>7K</td>
<td></td>
</tr>
<tr>
<td>Jaimin</td>
<td>2014-01-10</td>
<td>3K</td>
<td></td>
</tr>
<tr>
<td>Nga</td>
<td>2014-01-20</td>
<td>11K</td>
<td></td>
</tr>
<tr>
<td>Jaimin</td>
<td>2014-01-30</td>
<td>10K</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Employee</th>
<th>Location</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>EmpID</td>
<td>Location</td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>Jaimin</td>
<td>Boston</td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>Natalya</td>
<td>New York</td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>Nga</td>
<td>Chicago</td>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Sales</th>
<th>DateOfSale</th>
<th>Amount</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nga</td>
<td>2013-12-12</td>
<td>15K</td>
<td>Chicago</td>
</tr>
<tr>
<td>Natalya</td>
<td>2013-12-30</td>
<td>7K</td>
<td>New York</td>
</tr>
<tr>
<td>Jaimin</td>
<td>2014-01-10</td>
<td>3K</td>
<td>Boston</td>
</tr>
<tr>
<td>Nga</td>
<td>2014-01-20</td>
<td>11K</td>
<td>Chicago</td>
</tr>
<tr>
<td>Jaimin</td>
<td>2014-01-30</td>
<td>10K</td>
<td>Boston</td>
</tr>
</tbody>
</table>

The query before becomes simpler and run faster
This simplified query is generated implicitly

```
SELECT * from Sales;
```

Sales is a Materialized View
What are Materialized Views (MVs)?

Redundant Data Storages for Performance Purposes

Example 2: Pre-aggregate Data

SalesData

<table>
<thead>
<tr>
<th>Emp</th>
<th>DateOfSale</th>
<th>Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nga</td>
<td>2013-12-12</td>
<td>15K</td>
</tr>
<tr>
<td>Natalya</td>
<td>2013-12-30</td>
<td>7K</td>
</tr>
<tr>
<td>Jaimin</td>
<td>2014-01-10</td>
<td>3K</td>
</tr>
<tr>
<td>Nga</td>
<td>2014-01-20</td>
<td>11K</td>
</tr>
<tr>
<td>Jaimin</td>
<td>2014-01-30</td>
<td>10K</td>
</tr>
<tr>
<td>Jaimin</td>
<td>2014-02-15</td>
<td>18K</td>
</tr>
<tr>
<td>Nga</td>
<td>2014-02-16</td>
<td>5K</td>
</tr>
<tr>
<td>Nga</td>
<td>2014-02-25</td>
<td>9K</td>
</tr>
</tbody>
</table>

```
SELECT   year(dateOfSale) year, emp,
         sum(amount) sum, max(amount) max
FROM     SalesData
GROUP BY year, emp
ORDER BY year, emp;
```

Results

<table>
<thead>
<tr>
<th>Year</th>
<th>Emp</th>
<th>Sum</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>2013</td>
<td>Natalya</td>
<td>7K</td>
<td>7K</td>
</tr>
<tr>
<td>2013</td>
<td>Nga</td>
<td>15K</td>
<td>15K</td>
</tr>
<tr>
<td>2014</td>
<td>Jaimin</td>
<td>31K</td>
<td>18K</td>
</tr>
<tr>
<td>2014</td>
<td>Nga</td>
<td>25K</td>
<td>11K</td>
</tr>
</tbody>
</table>
What are Materialized Views (MVs)?

Redundant Data Storages for Performance Purposes

Example 2: Pre-aggregate Data

<table>
<thead>
<tr>
<th>SalesData</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Emp</td>
<td>DateOfSale</td>
</tr>
<tr>
<td>Nga</td>
<td>2013-12-12</td>
</tr>
<tr>
<td>Natalya</td>
<td>2013-12-30</td>
</tr>
<tr>
<td>Jaimin</td>
<td>2014-01-10</td>
</tr>
<tr>
<td>Nga</td>
<td>2014-01-20</td>
</tr>
<tr>
<td>Jaimin</td>
<td>2014-01-30</td>
</tr>
<tr>
<td>Jaimin</td>
<td>2014-02-15</td>
</tr>
<tr>
<td>Nga</td>
<td>2014-02-16</td>
</tr>
<tr>
<td>Nga</td>
<td>2014-02-25</td>
</tr>
</tbody>
</table>

For better performance, data is pre-aggregated and inserted to SalesSummary

<table>
<thead>
<tr>
<th>Year</th>
<th>Emp</th>
<th>Sum</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>2013</td>
<td>Natalya</td>
<td>7K</td>
<td>7K</td>
</tr>
<tr>
<td>2013</td>
<td>Nga</td>
<td>15K</td>
<td>15K</td>
</tr>
<tr>
<td>2014</td>
<td>Jaimin</td>
<td>31K</td>
<td>18K</td>
</tr>
<tr>
<td>2014</td>
<td>Nga</td>
<td>25K</td>
<td>11K</td>
</tr>
</tbody>
</table>

The query before becomes simpler and run faster

This simplified query is generated implicitly

```
SELECT * from SalesSummary;
```
What are Materialized Views (MVs)?
Redundant Data Storages for Performance Purposes

Example 2: Pre-aggregate Data

SalesData

<table>
<thead>
<tr>
<th>Emp</th>
<th>DateOfSale</th>
<th>Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nga</td>
<td>2013-12-12</td>
<td>15K</td>
</tr>
<tr>
<td>Natalya</td>
<td>2013-12-30</td>
<td>7K</td>
</tr>
<tr>
<td>Jaimin</td>
<td>2014-01-10</td>
<td>3K</td>
</tr>
<tr>
<td>Nga</td>
<td>2014-01-20</td>
<td>11K</td>
</tr>
<tr>
<td>Jaimin</td>
<td>2014-01-30</td>
<td>10K</td>
</tr>
<tr>
<td>Jaimin</td>
<td>2014-02-15</td>
<td>18K</td>
</tr>
<tr>
<td>Nga</td>
<td>2014-02-16</td>
<td>5K</td>
</tr>
<tr>
<td>Nga</td>
<td>2014-02-25</td>
<td>9K</td>
</tr>
</tbody>
</table>

For better performance, data is pre-aggregated and inserted to SalesSummary

<table>
<thead>
<tr>
<th>Year</th>
<th>Emp</th>
<th>Sum</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>2013</td>
<td>Natalya</td>
<td>7K</td>
<td>7K</td>
</tr>
<tr>
<td>2013</td>
<td>Nga</td>
<td>15K</td>
<td>15K</td>
</tr>
<tr>
<td>2014</td>
<td>Jaimin</td>
<td>31K</td>
<td>18K</td>
</tr>
<tr>
<td>2014</td>
<td>Nga</td>
<td>25K</td>
<td>11K</td>
</tr>
</tbody>
</table>

The query before becomes simpler and run faster
This simplified query is generated implicitly

```
SELECT * from SalesSummary;
```
Outline

• What is Big Data?
• How Vertica provides Big Data Solutions?
• What are Materialized Views (MV)s?
• What are the Roles of MVs in Big Data?
• How does Vertica implement their MVs?
  • Why Vertica names their MVs Live Aggregate Projections (LAPs)?
  • What are the differences between LAPS & MVs?
What are the Roles of MVs in Big Data?
What are the Roles of MVs in Big Data?

Remember this about Big Data?

- **HAVE**: a lot of data.
- **WANT**: dig them for **treasures in sub-second**
What are the Roles of MVs in Big Data?

Remember this about Big Data?

- **HAVE**: a lot of data.
- **WANT**: dig them for *treasures* in *sub-second*

- MVs helps bring *a lot of data* to a *small* set of data
  → Dig that *small* set of data can happen in *sub-second*
What are the Roles of MVs in Big Data?

Remember this about Big Data?

- **HAVE**: a lot of data.
- **WANT**: dig them for treasures in sub-second

- MVs helps bring **a lot of data** to a **small** set of data
  → Dig that **small** set of data can happen in sub-second

- **Vertica is already fast. MVs will bring it to an even better level**
Outline

• What is Big Data?
• How Vertica provides Big Data Solutions?
• What are Materialized Views (MV)s?
• What are the Roles of MVs in Big Data?
• How does Vertica implement their MVs?
  • Why Vertica names their MVs Live Aggregate Projections (LAPs)?
  • What are the differences between LAPS & MVs?
How Vertica Implements their MVs?

- Answer these 2 questions will answer the above question
  1. Why Vertica names their MVs Live Aggregate Projections (LAPs)?
  2. What are the differences between LAPs and Mvs?
How Vertica Implements their MVs?

• Answer these 2 questions will answer the above question
  1. Why Vertica names their MVs Live Aggregate Projections (LAPs)?
  2. What are the differences between LAPs and MVs?

• The answers of question #2 will light up the answer of question 1.
How Vertica Implements their MVs?

• Answer these 2 questions will answer the above question
  1. Why Vertica names their MVs Live Aggregate Projections (LAPs)?
  2. What are the differences between LAPs and MVs?

• The answers of question 2 will light up the answer of question 1.
  So What are the differences between LAPs and MVs?
Differences between LAPs and MVs?
Differences between LAPs and MVs?

**Fully vs Partially Aggregated**

### MVs

- **Fully** aggregated OR **out-of-date**

<table>
<thead>
<tr>
<th>Year</th>
<th>Emp</th>
<th>Sum</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>2013</td>
<td>Natalya</td>
<td>7K</td>
<td>7K</td>
</tr>
<tr>
<td>2013</td>
<td>Nga</td>
<td>15K</td>
<td>15K</td>
</tr>
<tr>
<td>2014</td>
<td>Jaimin</td>
<td>31K</td>
<td>18K</td>
</tr>
<tr>
<td>2014</td>
<td>Nga</td>
<td>25K</td>
<td>11K</td>
</tr>
</tbody>
</table>

### LAPs

- **Partially** aggregated per **load** on each **partition**.
- **Always** **up-to-date**.

#### Partitioned by MONTH(DateOfSale)

<table>
<thead>
<tr>
<th>Year</th>
<th>Emp</th>
<th>Sum</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>2013-12</td>
<td>2013</td>
<td>Natalya</td>
<td>7K</td>
</tr>
<tr>
<td></td>
<td>2013</td>
<td>Nga</td>
<td>15K</td>
</tr>
<tr>
<td>2014-01</td>
<td>2014</td>
<td>Jaimin</td>
<td>13K</td>
</tr>
<tr>
<td></td>
<td>2014</td>
<td>Nga</td>
<td>11K</td>
</tr>
<tr>
<td>2014-02</td>
<td>2014</td>
<td>Jaimin</td>
<td>18K</td>
</tr>
<tr>
<td></td>
<td>2014</td>
<td>Nga</td>
<td>14K</td>
</tr>
</tbody>
</table>
Differences between LAPs and MVs?

Fully vs Partially Aggregated → Different Insert Strategy

**MV**s

Compute/Recompute MVs data **after** INSERT

**LAP**s

Compute LAP's **during** INSERT

Diagram:
- MVs: Base Table → MVs → Recompute/Maintenance
- LAPs: Base Table → LAPs → GroupBy → Data
Differences between LAPs and MVs?

**Fully vs Partially Aggregated** → **Different Maintenance Cost**

**MV**s

- Compute Delta
- Recompute affected data for all related MVs to bring them up-to-date before they are used

**LAP**s

- Do nothing
Differences between LAPs and MVs?

**Fully vs Partially Aggregated → Different Currency Control**

**MV**s
- Locking needed during MV maintenance
  → Heavy duty system

**LAP**s
- Do nothing since data of new load will be in their own container
Differences between LAPs and MVs?

Fully vs Partially Aggregated → Different Currency Control

**MV**
- Locking needed during MV maintenance
  → Heavy duty system

**LAP**
- Do nothing since data of new load will be in their own container

---

Look back at Insert Plans

```
<table>
<thead>
<tr>
<th>Base Table</th>
<th>MVs</th>
<th>Recompute/Maintenance</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Base Table</td>
<td>LAPs</td>
<td>GroupBy</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Data</td>
</tr>
</tbody>
</table>
```

The recompute/Maintenance step is not cheap and BaseTable has to be locked during that time.
Differences between LAPs and MVs?

Fully vs Partially Aggregated → Different Select Query Plan

**MV**s
Read MV

**LAP**s
Read and Combine partially aggregated data

![Diagram showing the differences between MVs and LAPs in a query plan.](image-url)
Differences between LAPs and MVs?

Fully vs Partially Aggregated → Different Background Maintenance

**MV**
Possible if MVs are not used right after INSERT

**LAP**
Use Tuple Mover (TM) to combine partially aggregated data from different loads
Differences between LAPs and MVs?

**Fully vs Partially Aggregated → Different Delete Handling**

**MV**
- Recompute MVs for affected groups

**LAPs**
- No delete single row but drop a whole partition
  → Do nothing
Differences between LAPs and MVs?

Summary

**MVs**
- Fully aggregated but **not** always up-to-date
- Expensive maintenance process
- Heavy-duty Concurrency Control
- Select data from MVs is **fast**
- Background Maintenance **not** work since MVs must be up-to-date

**LAPs**
- Partially aggregated but always **up-to-date**
- No maintenance needed
- No Concurrency Control needed on LAPs
- Select data from LAPs also fast but need to combine data from different partitions & loads.
- Background Maintenance by TM
Differences between LAPs and MVs?

**Summary**

**MVs**
- Fully aggregated but **not** always up-to-date
- Expensive maintenance process
- Heavy-duty Concurrency Control
- Select data from MVs is **fast**
- Background Maintenance **not** work since MVs must be up-to-date

**LAPs**
- Partially aggregated but always **up-to-date**
- No maintenance needed
- No Concurrency Control needed on LAPs
- Select data from LAPs also fast but need to **combine data** from different partitions & loads.
- Background Maintenance by TM

So why **name** Live Aggregate Projections?
Differences between LAPs and MVs?

Summary

**MVs**
- Fully aggregated but **not** always up-to-date
- Expensive maintenance process
- **Heavy-duty** Concurrency Control
- Select data from MVs is **fast**
- Background Maintenance **not** work since MVs must be up-to-date

**LAPs**
- Partially aggregated but always **up-to-date**
- **No** maintenance needed
- **No** Concurrency Control needed on LAPs
- Select data from LAPs also **fast** but need to **combine data** from different partitions & loads.
- Background Maintenance by TM

So why name **Live** Aggregate Projections?
Differences between LAPs and MVs?

Summary

**MVs**
- **Fully** aggregated but not always up-to-date
- **Expensive** maintenance process
- **Heavy-duty** Concurrency Control
- Select data from MVs is **fast**
- Background Maintenance not work since MVs must be up-to-date

**LAPs**
- **Partially** aggregated but always up-to-date
- **No** maintenance needed
- **No** Concurrency Control needed on LAPs
- Select data from LAPs also fast but need to combine data from different partitions & loads.
- Background Maintenance by TM
- Vertica's physical storages are **Projections**

So why name **Live** Aggregate **Projections**?
Differences between LAPs and MVs?

Summary

**MV**s
- Fully aggregated but **not** always up-to-date
- Expensive maintenance process
- Heavy-duty Concurrency Control
- Select data from MVs is **fast**
- Background Maintenance **not** work since MVs must be up-to-date

**LAP**s
- Partially aggregated but always **up-to-date**
- No maintenance needed
- No Concurrency Control needed on LAPs
- Select data from LAPs also fast but need to **combine data** from different partitions & loads.
- Background Maintenance by TM
- Vertica's physical storages are **Projections**
- Data is **pre-aggregated**

So why name **Live Aggregate Projections**?
Questions?