Database Applications
Lecture 4: Database Partitioning

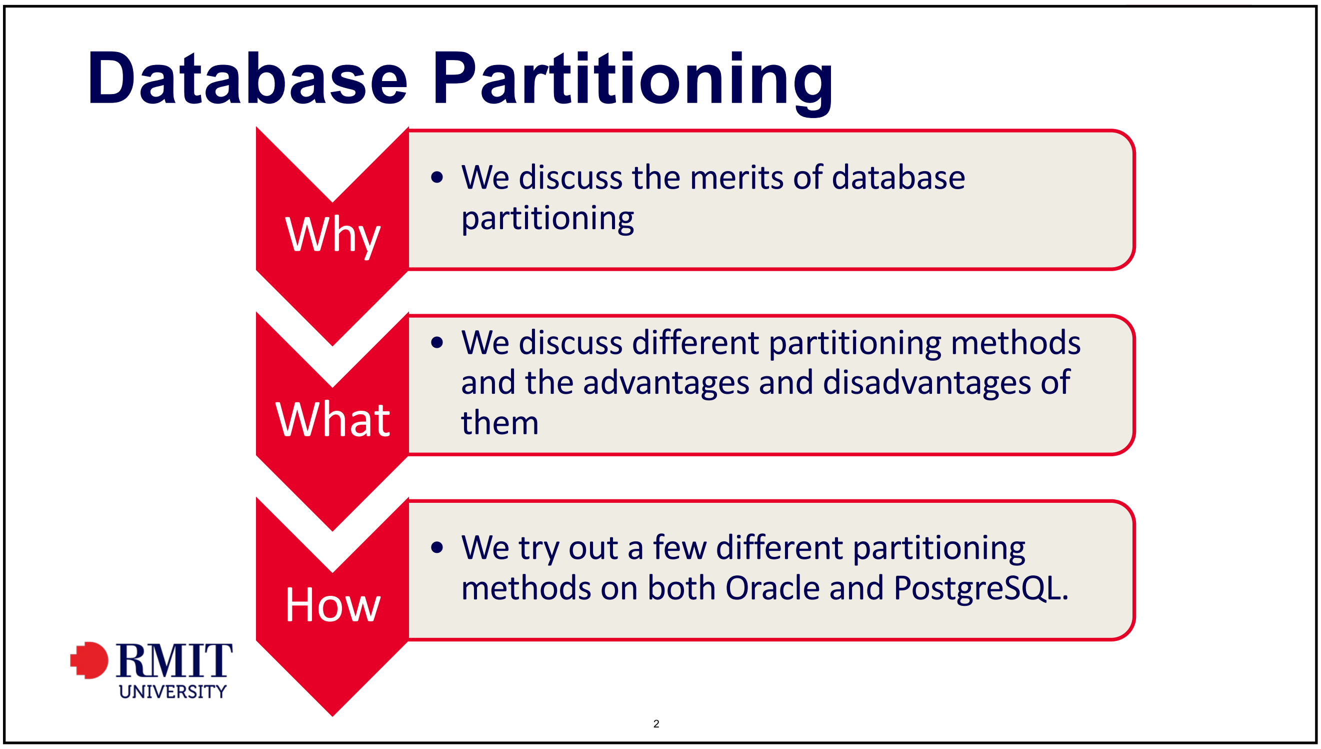
Santha Sumanasekara

August 2022

RMIT UNIVERSITY

1

1



Database Partitioning

Why
- We discuss the merits of database partitioning

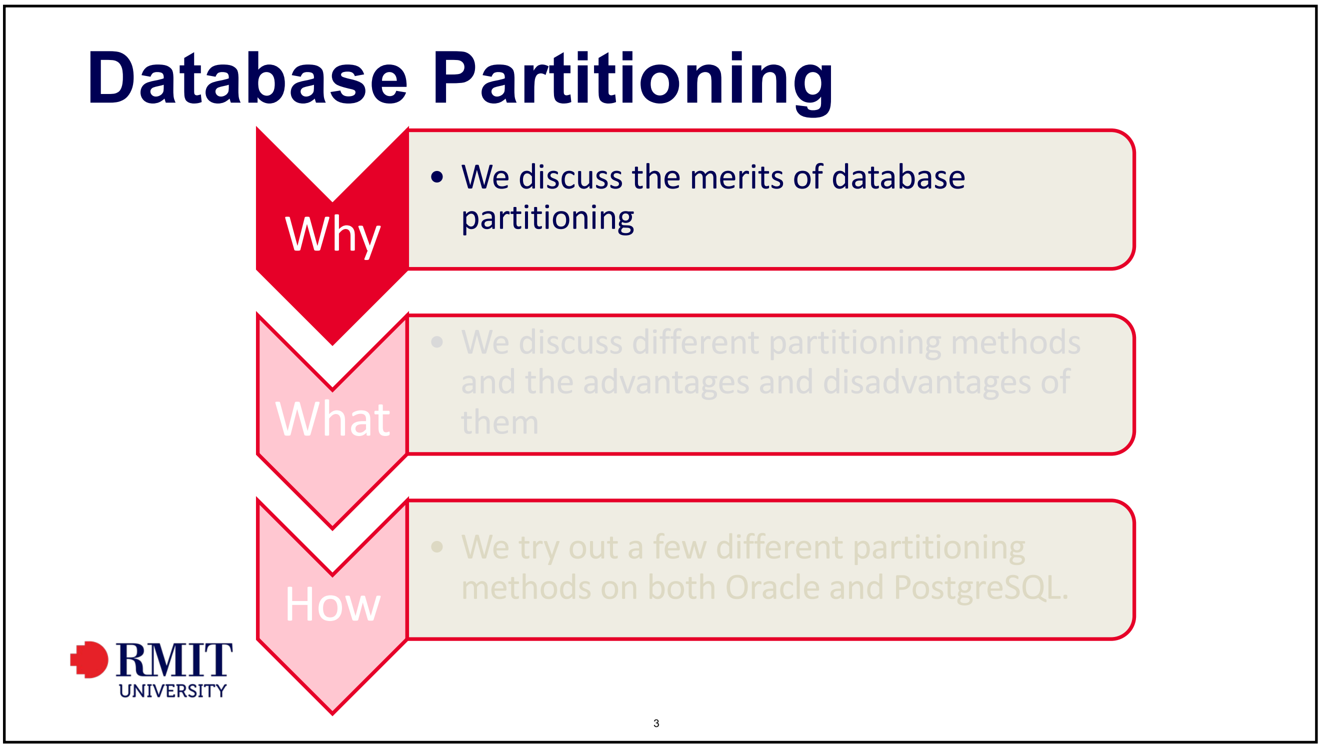
What
- We discuss different partitioning methods and the advantages and disadvantages of them

How
- We try out a few different partitioning methods on both Oracle and PostgreSQL.

RMIT UNIVERSITY

2

2



Database Partitioning

Why
- We discuss the merits of database partitioning

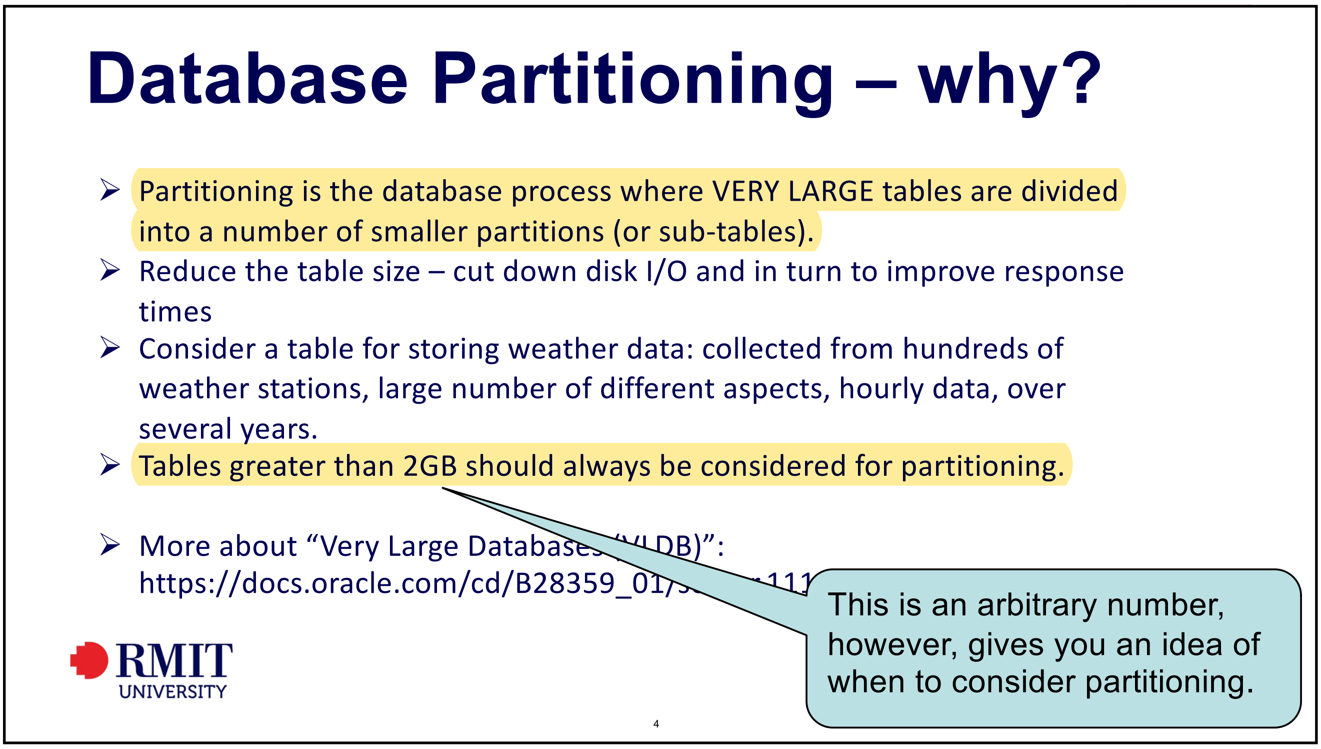
What
- We discuss different partitioning methods and the advantages and disadvantages of them

How
- We try out a few different partitioning methods on both Oracle and PostgreSQL.

RMIT UNIVERSITY

3

3



Database Partitioning – why?

- Partitioning is the database process where VERY LARGE tables are divided into a number of smaller partitions (or sub-tables).
- Reduce the table size – cut down disk I/O and in turn to improve response times
- Consider a table for storing weather data: collected from hundreds of weather stations, large number of different aspects, hourly data, over several years.
- Tables greater than 2GB should always be considered for partitioning.

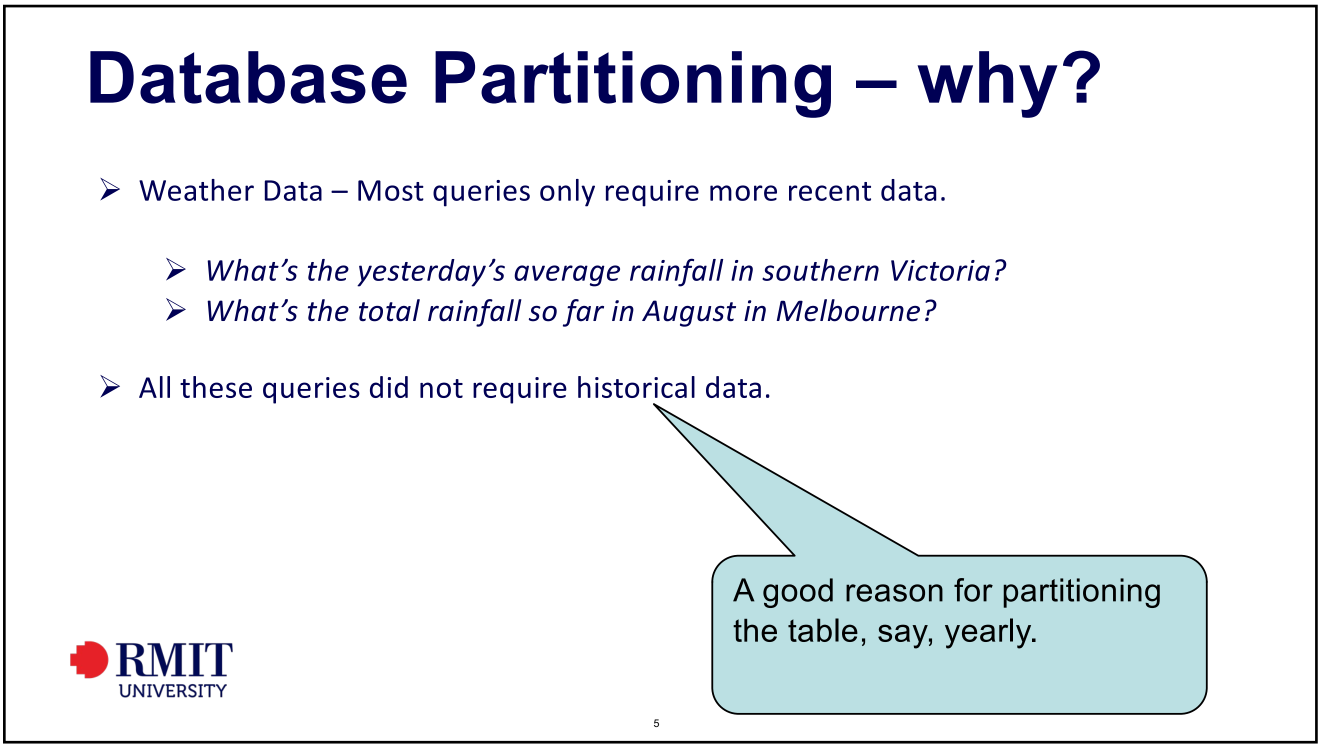
- More about “Very Large Databases (VLDB)”: https://docs.oracle.com/cd/B28359_01/se [illegible] 111

This is an arbitrary number, however, gives you an idea of when to consider partitioning.

RMIT UNIVERSITY

4

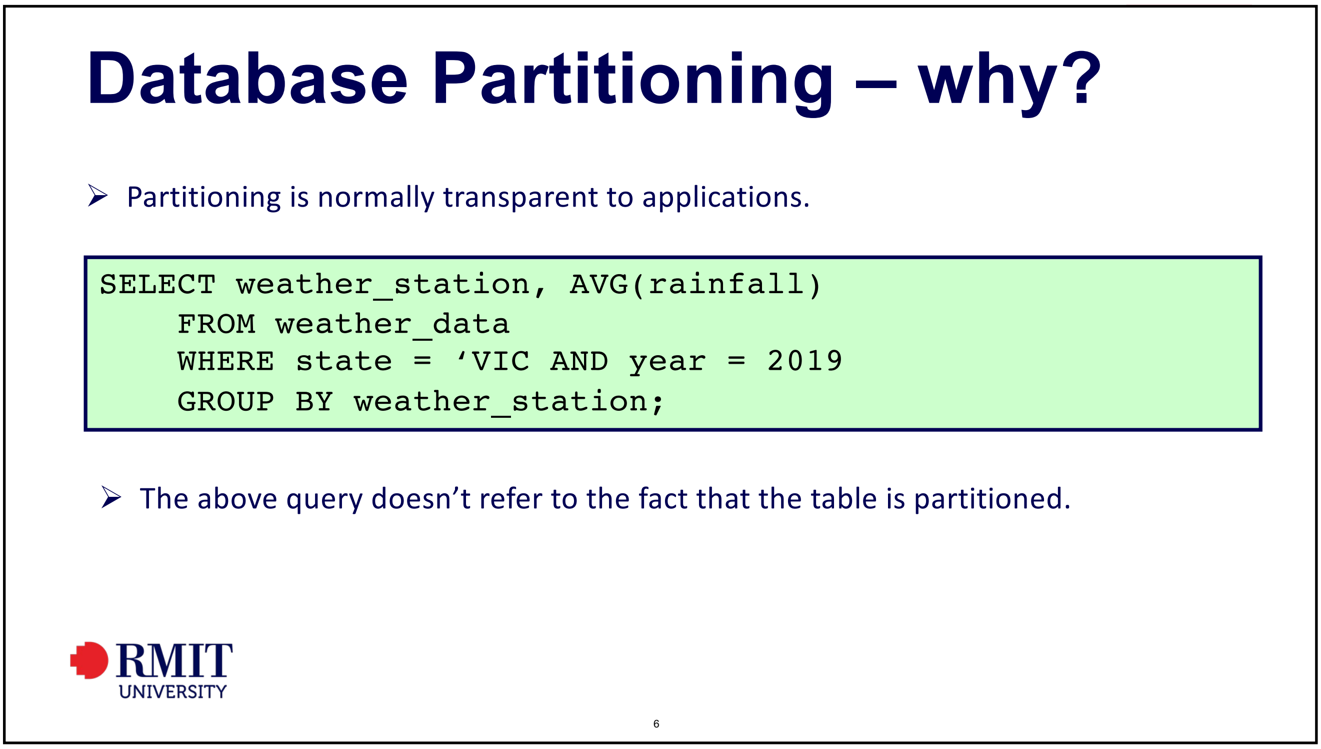
4



Database Partitioning – why?

- Weather Data – Most queries only require more recent data.
  - *What’s the yesterday’s average rainfall in southern Victoria?*
  - *What’s the total rainfall so far in August in Melbourne?*
- All these queries did not require historical data.

A good reason for partitioning the table, say, yearly.



Database Partitioning – why?

- Partitioning is normally transparent to applications.

```
SELECT weather_station, AVG(rainfall)
    FROM weather_data
    WHERE state = ‘VIC AND year = 2019
    GROUP BY weather_station;
```

- The above query doesn’t refer to the fact that the table is partitioned.

Database Partitioning

This query is aware of the partitioning and refers to the 2019 partition -- `weather_data_2019`.

- Partitioning is normally transparent to applications.

```
SELECT weather_station, AVG(rainfall)
    FROM weather_data
    WHERE state = ‘VIC’ AND year = 2
    GROUP BY weather_station;
```

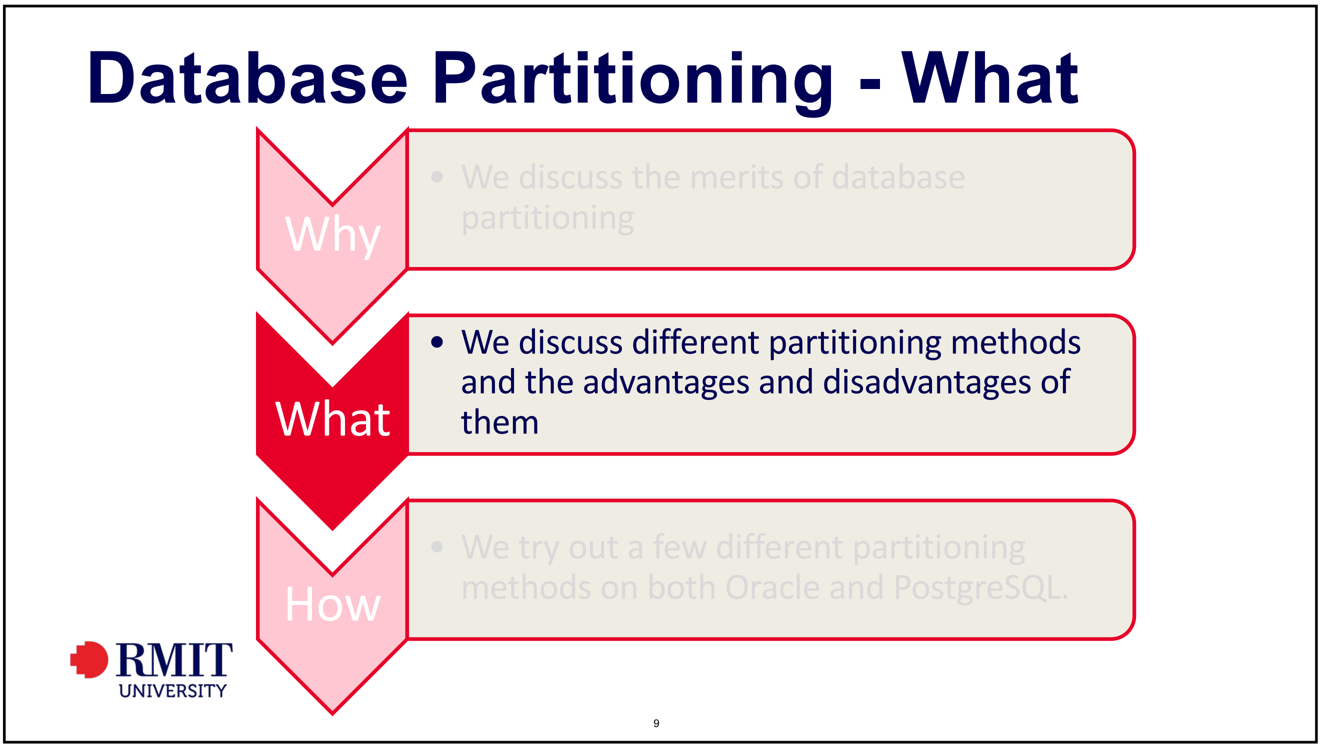
```
SELECT weather_station, AVG(rainfall)
    FROM weather_data PARTITION (weather_data_2019)
    WHERE state = ‘VIC’
    GROUP BY weather_station;
```

RMIT UNIVERSITY

Database Partitioning – why?

- Partitioning enables data management operations such bulk uploads, index creation and rebuilding, and backup/recovery at the partition level, rather than on the entire table.
- Partitioning improves query performance. In many cases, the results of a query can be achieved by accessing a subset of partitions, rather than the entire table.
- Partitioning can significantly reduce the impact of scheduled downtime for maintenance operations.

Consider a query that require a full-table scan on a 2GB table!



Database Partitioning - What

Why

- We discuss the merits of database partitioning

What

- We discuss different partitioning methods and the advantages and disadvantages of them

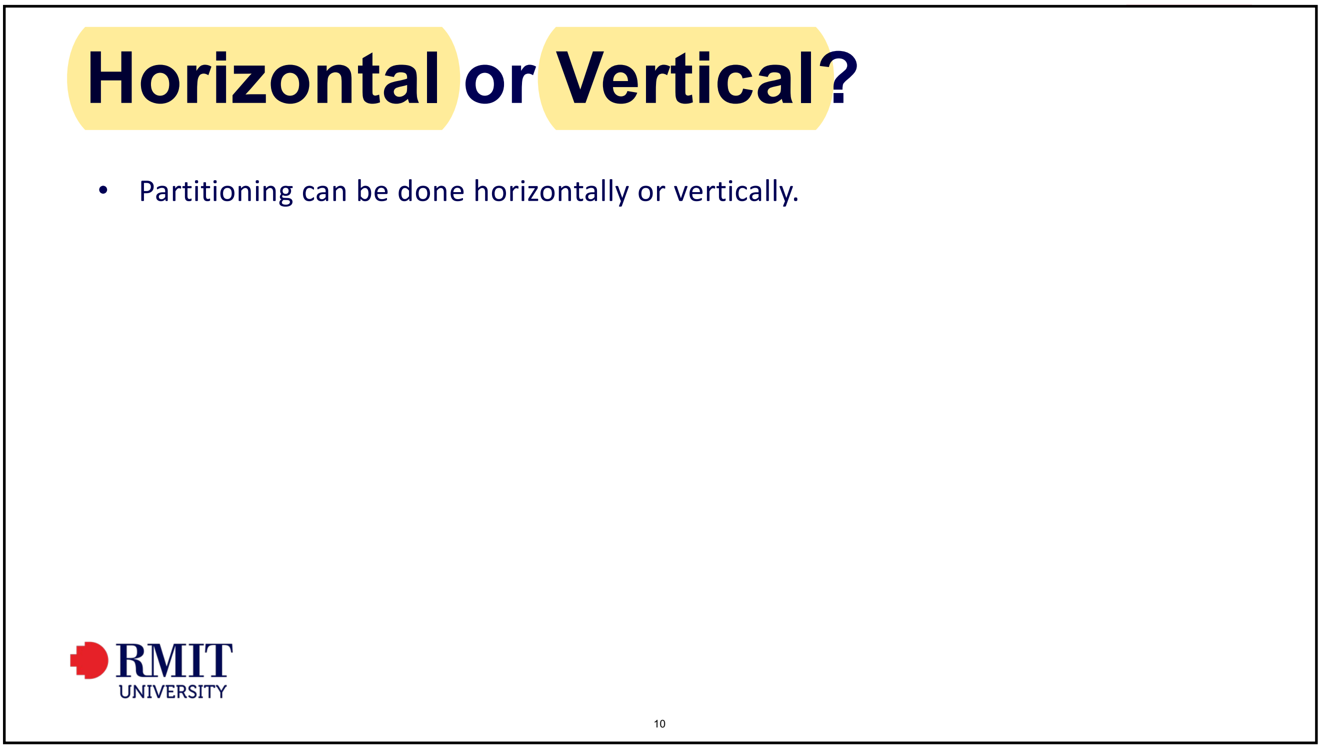
How

- We try out a few different partitioning methods on both Oracle and PostgreSQL.

RMIT UNIVERSITY

9

9



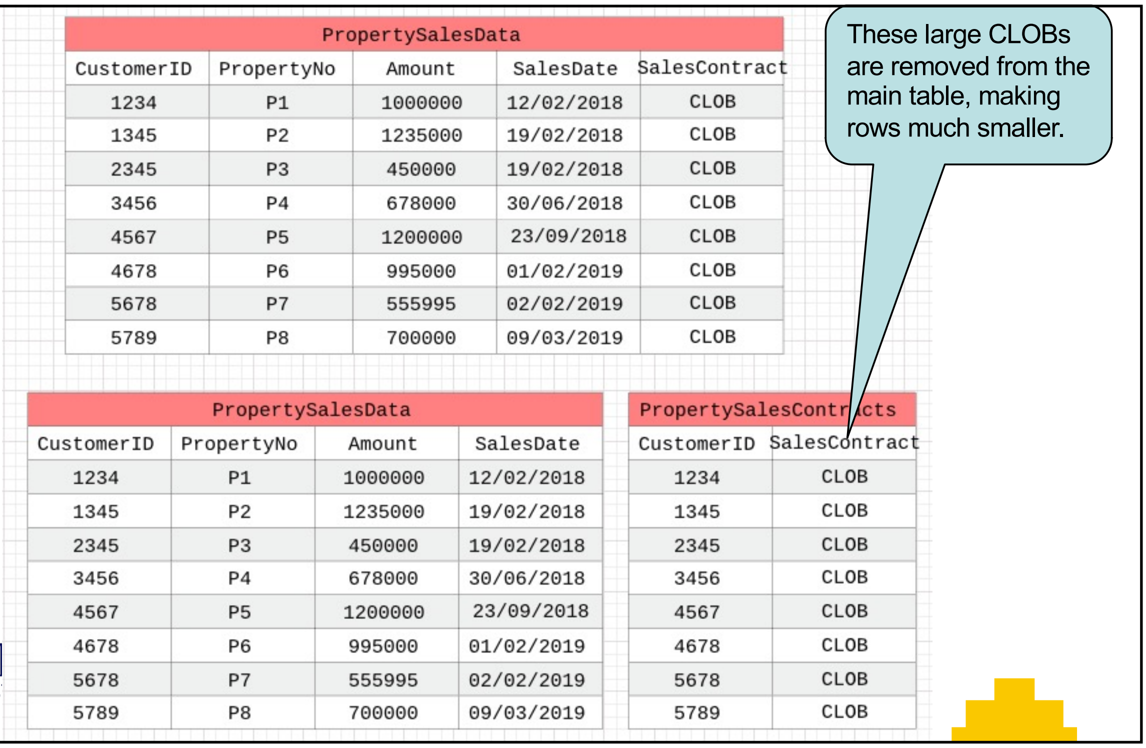
Horizontal or Vertical?

- Partitioning can be done horizontally or vertically.

RMIT UNIVERSITY

10

10



PropertySalesData

CustomerID	PropertyNo	Amount	SalesDate	SalesContract
1234	P1	1000000	12/02/2018	CLOB
1345	P2	1235000	19/02/2018	CLOB
2345	P3	450000	19/02/2018	CLOB
3456	P4	678000	30/06/2018	CLOB
4567	P5	1200000	23/09/2018	CLOB
4678	P6	995000	01/02/2019	CLOB
5678	P7	555995	02/02/2019	CLOB
5789	P8	700000	09/03/2019	CLOB

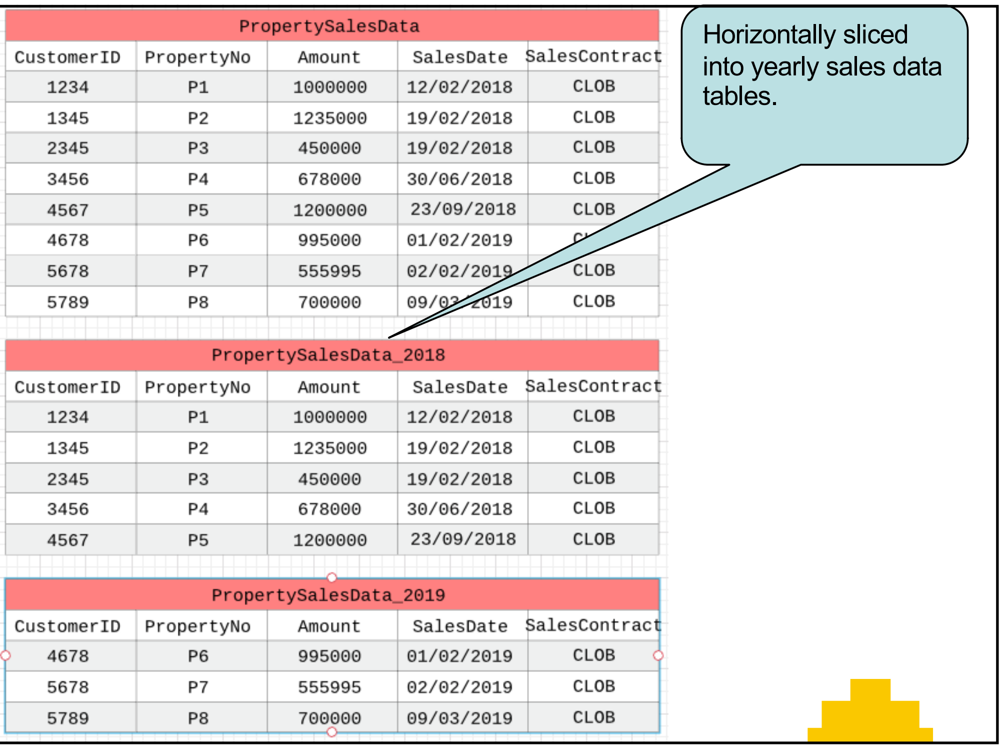
PropertySalesData

CustomerID	PropertyNo	Amount	SalesDate
1234	P1	1000000	12/02/2018
1345	P2	1235000	19/02/2018
2345	P3	450000	19/02/2018
3456	P4	678000	30/06/2018
4567	P5	1200000	23/09/2018
4678	P6	995000	01/02/2019
5678	P7	555995	02/02/2019
5789	P8	700000	09/03/2019

PropertySalesContracts

CustomerID	SalesContract
1234	CLOB
1345	CLOB
2345	CLOB
3456	CLOB
4567	CLOB
4678	CLOB
5678	CLOB
5789	CLOB

These large CLOBs are removed from the main table, making rows much smaller.



PropertySalesData

CustomerID	PropertyNo	Amount	SalesDate	SalesContract
1234	P1	1000000	12/02/2018	CLOB
1345	P2	1235000	19/02/2018	CLOB
2345	P3	450000	19/02/2018	CLOB
3456	P4	678000	30/06/2018	CLOB
4567	P5	1200000	23/09/2018	CLOB
4678	P6	995000	01/02/2019	CLOB
5678	P7	555995	02/02/2019	CLOB
5789	P8	700000	09/03/2019	CLOB

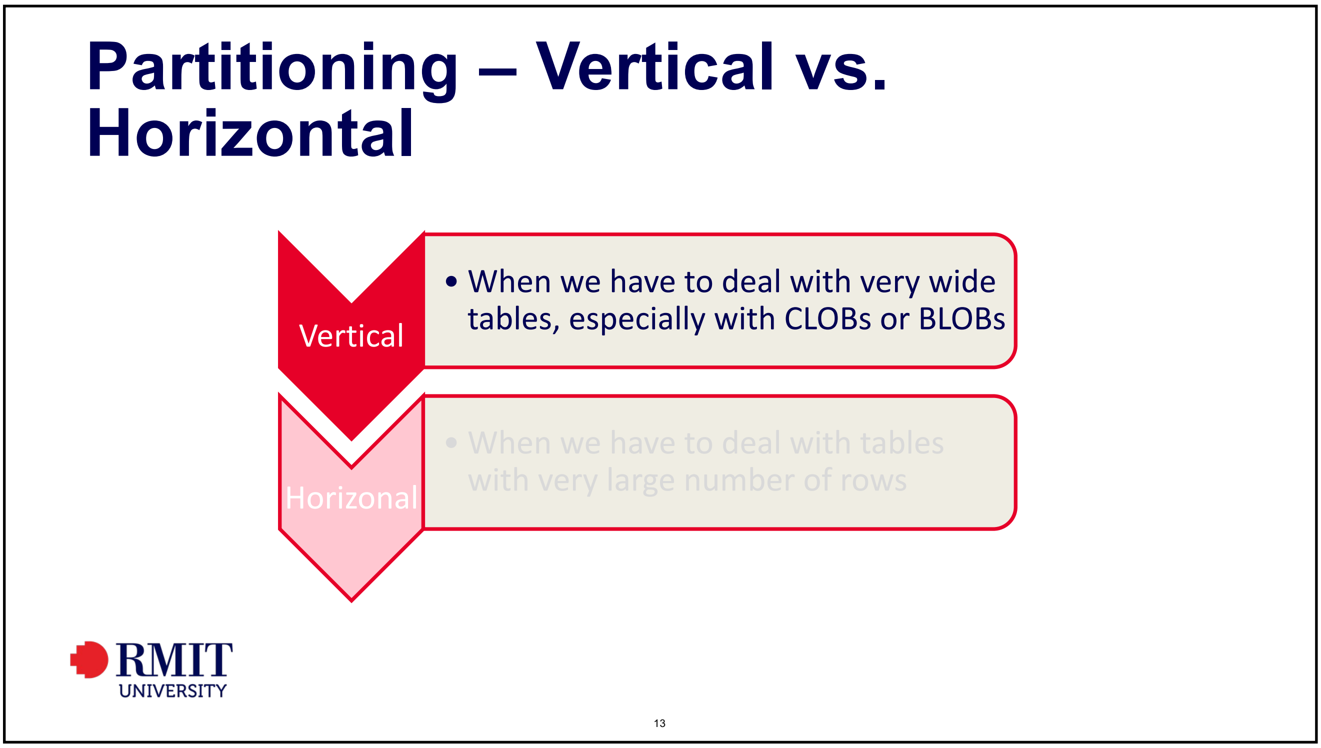
PropertySalesData_2018

CustomerID	PropertyNo	Amount	SalesDate	SalesContract
1234	P1	1000000	12/02/2018	CLOB
1345	P2	1235000	19/02/2018	CLOB
2345	P3	450000	19/02/2018	CLOB
3456	P4	678000	30/06/2018	CLOB
4567	P5	1200000	23/09/2018	CLOB

PropertySalesData_2019

CustomerID	PropertyNo	Amount	SalesDate	SalesContract
4678	P6	995000	01/02/2019	CLOB
5678	P7	555995	02/02/2019	CLOB
5789	P8	700000	09/03/2019	CLOB

Horizontally sliced into yearly sales data tables.



Partitioning – Vertical vs. Horizontal

Vertical

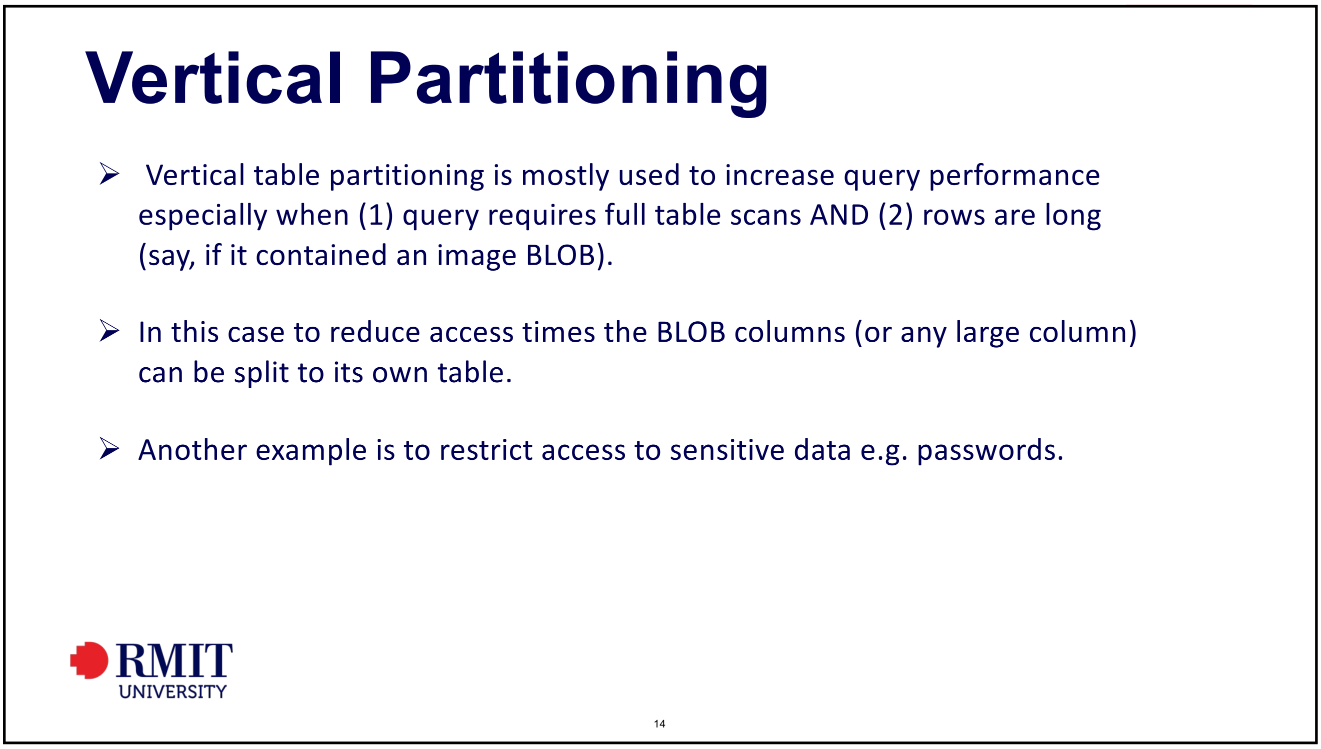
- When we have to deal with very wide tables, especially with CLOBs or BLOBs

Horizonal

- When we have to deal with tables with very large number of rows

13

13

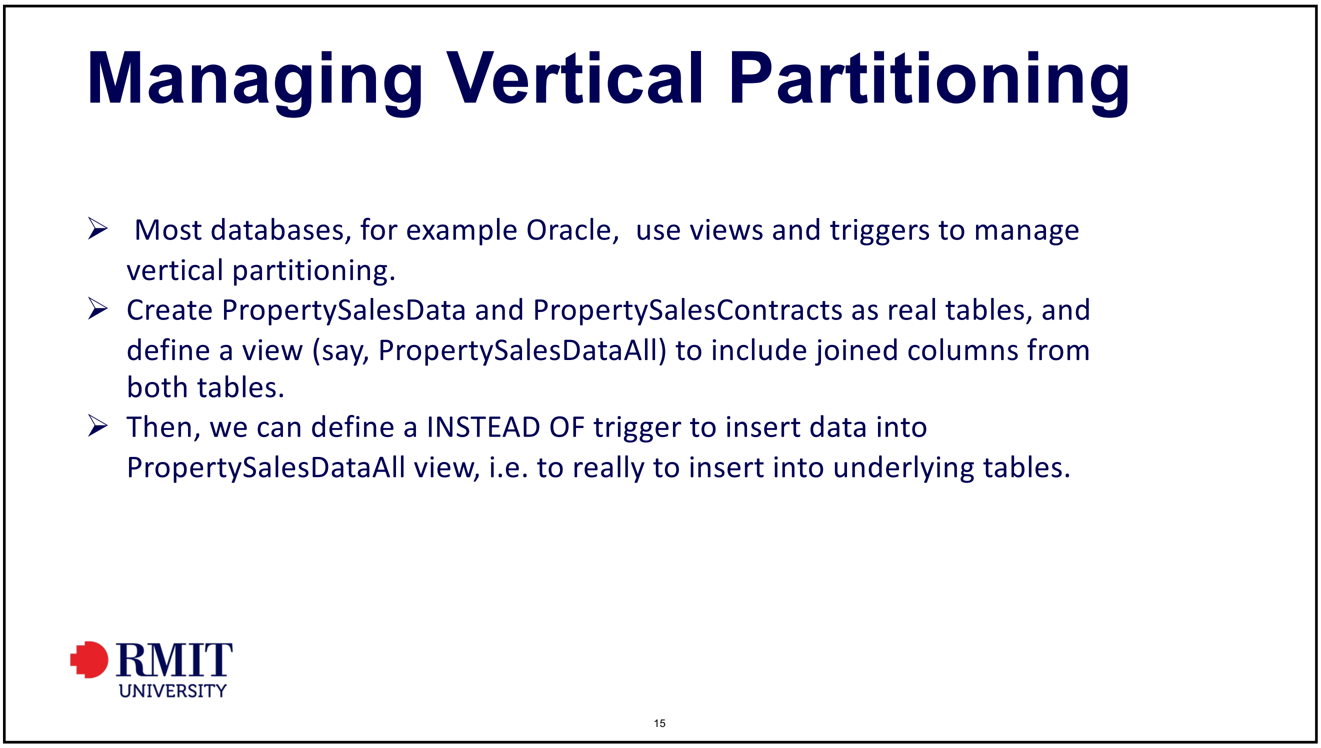


Vertical Partitioning

- Vertical table partitioning is mostly used to increase query performance especially when (1) query requires full table scans AND (2) rows are long (say, if it contained an image BLOB).
- In this case to reduce access times the BLOB columns (or any large column) can be split to its own table.
- Another example is to restrict access to sensitive data e.g. passwords.

14

14



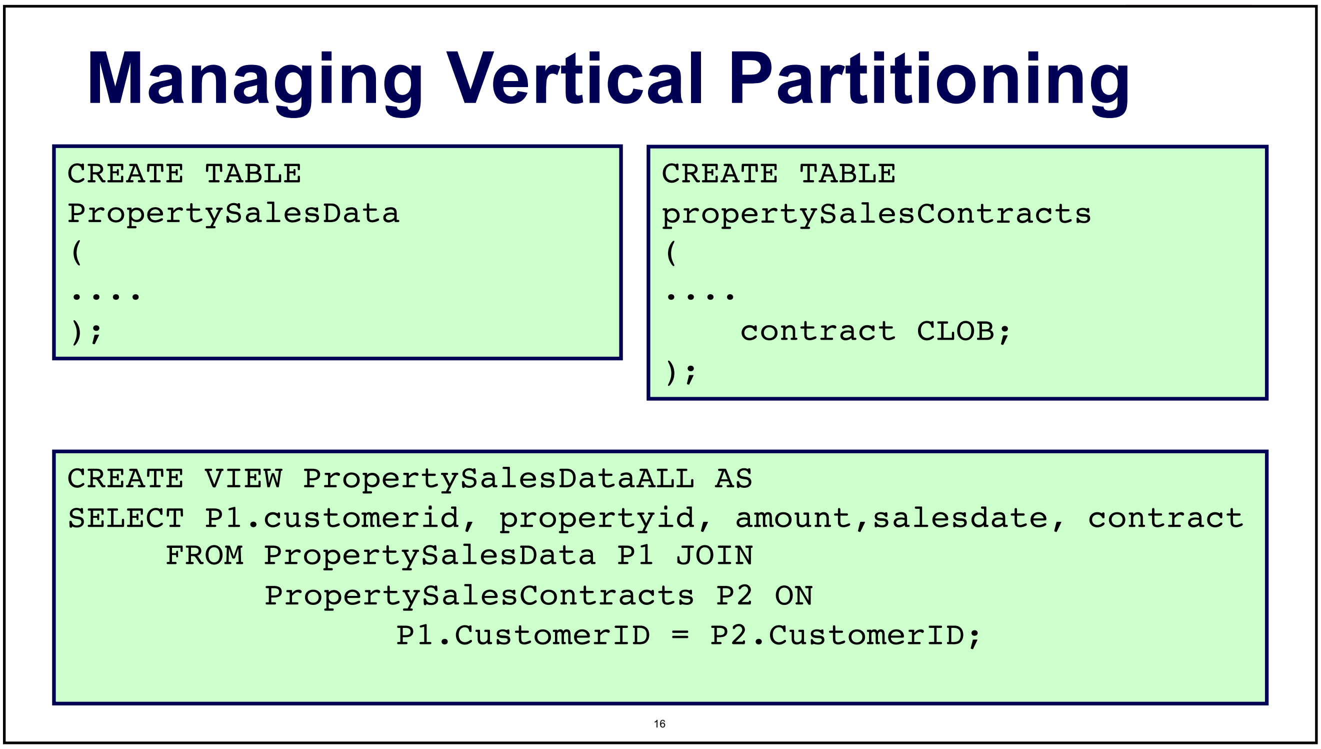
Managing Vertical Partitioning

- Most databases, for example Oracle, use views and triggers to manage vertical partitioning.
- Create PropertySalesData and PropertySalesContracts as real tables, and define a view (say, PropertySalesDataAll) to include joined columns from both tables.
- Then, we can define a INSTEAD OF trigger to insert data into PropertySalesDataAll view, i.e. to really to insert into underlying tables.

RMIT UNIVERSITY

15

15



Managing Vertical Partitioning

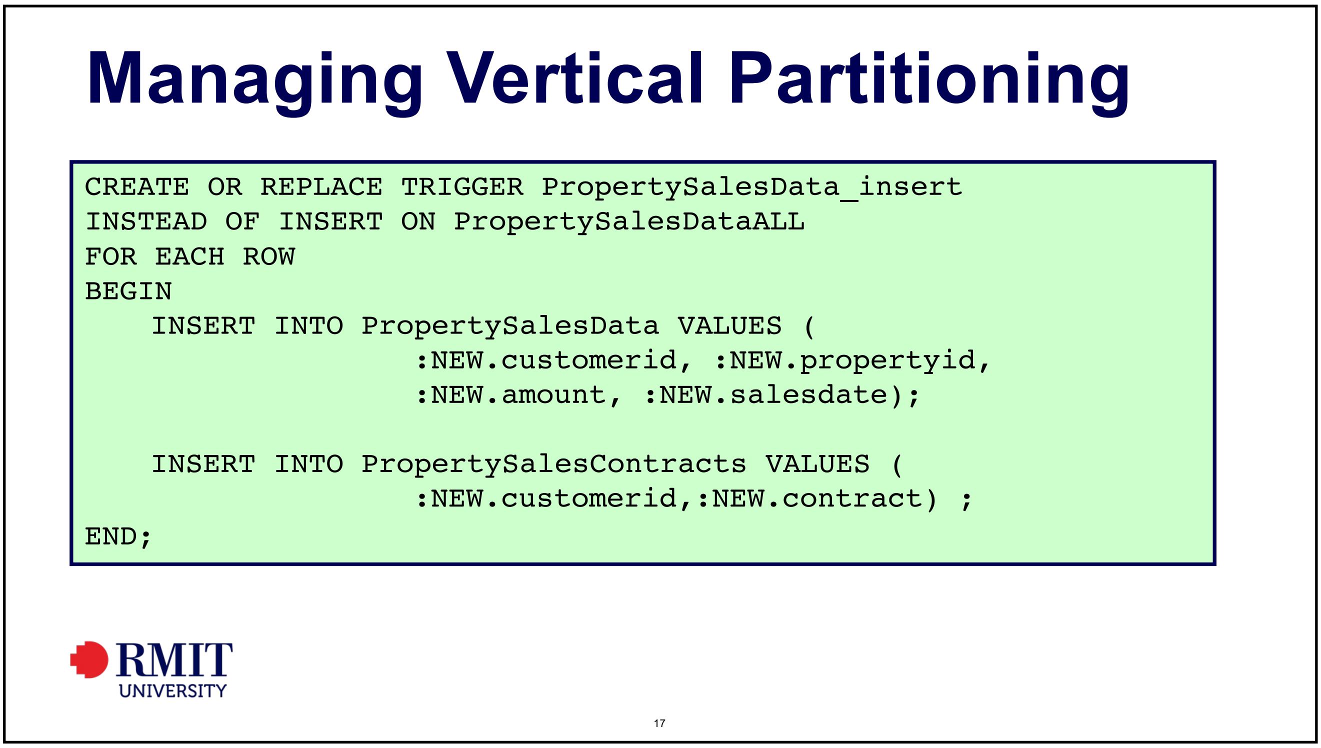
```
CREATE TABLE
PropertySalesData
(
....
);
```

```
CREATE TABLE
propertySalesContracts
(
....
    contract CLOB;
);
```

```
CREATE VIEW PropertySalesDataALL AS
SELECT P1.customerid, propertyid, amount,salesdate, contract
     FROM PropertySalesData P1 JOIN
          PropertySalesContracts P2 ON
                P1.CustomerID = P2.CustomerID;
```

16

16



Managing Vertical Partitioning

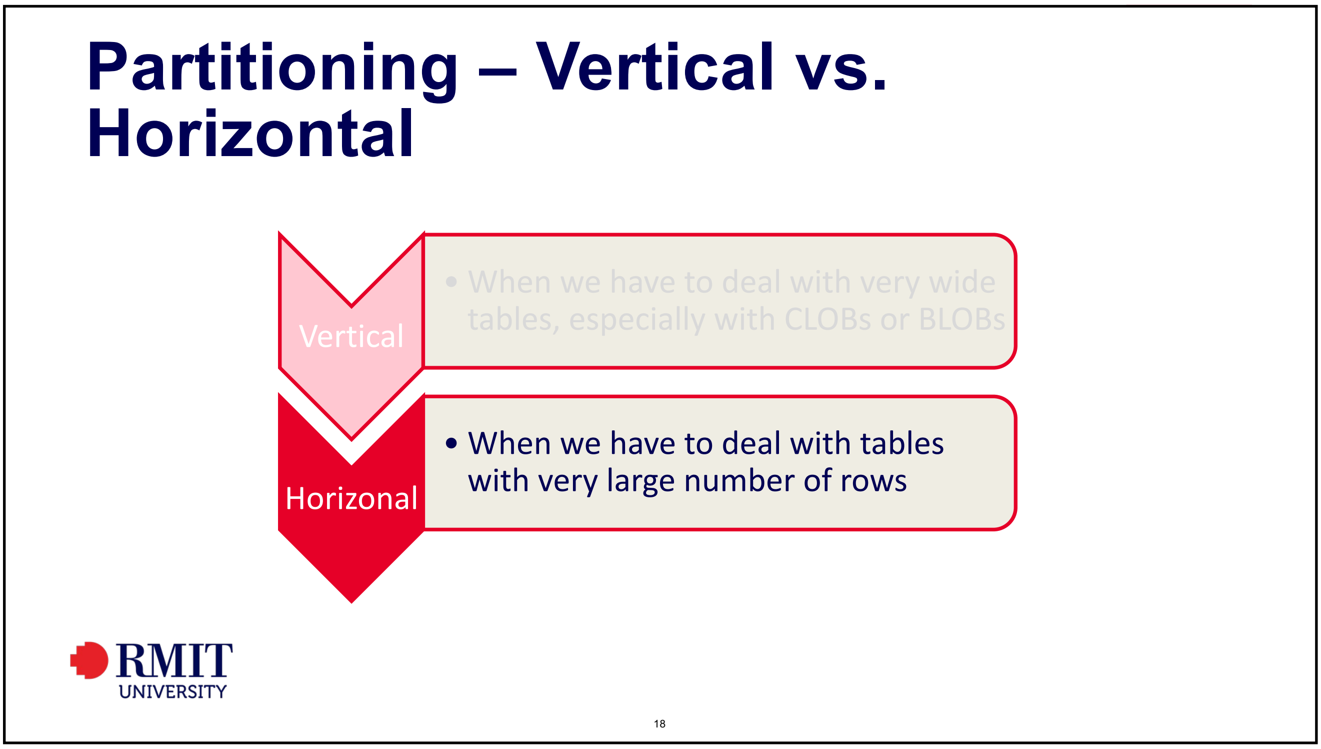
```
CREATE OR REPLACE TRIGGER PropertySalesData_insert
INSTEAD OF INSERT ON PropertySalesDataALL
FOR EACH ROW
BEGIN
    INSERT INTO PropertySalesData VALUES (
                    :NEW.customerid, :NEW.propertyid,
                    :NEW.amount, :NEW.salesdate);

    INSERT INTO PropertySalesContracts VALUES (
                    :NEW.customerid,:NEW.contract) ;
END;
```

RMIT UNIVERSITY

17

17



Partitioning – Vertical vs. Horizontal

- When we have to deal with very wide tables, especially with CLOBs or BLOBs

- When we have to deal with tables with very large number of rows

RMIT UNIVERSITY

18

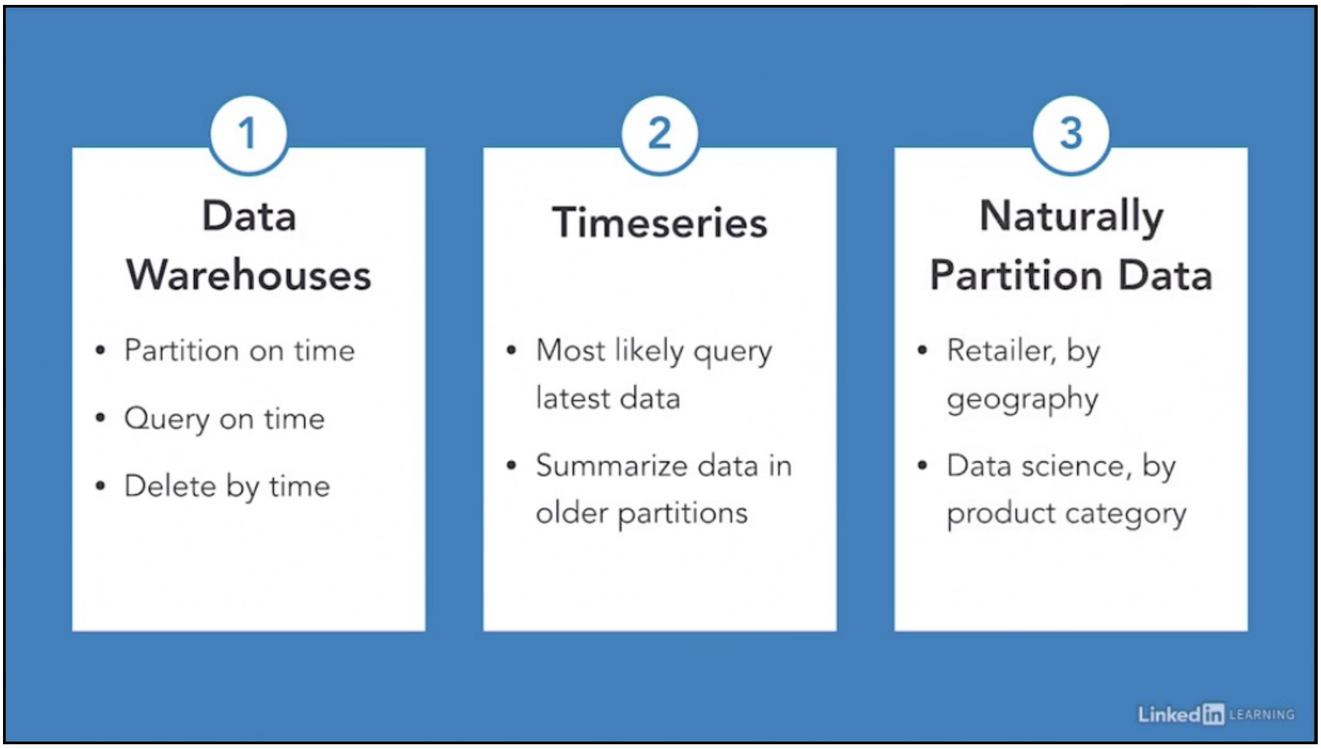
18

Horizontal Partitioning

- By far the most common partition method is horizontal partitioning
- Each partition are like sub-tables – they all have the same columns, but each will have a subset of rows.
- Full table scans can/ may be limited to full partition scans, which may require far less disk I/O
- Localised indexes can be built on each partition, so, even index scans can be more efficient

RMIT UNIVERSITY

19



1 Data Warehouses

- Partition on time
- Query on time
- Delete by time

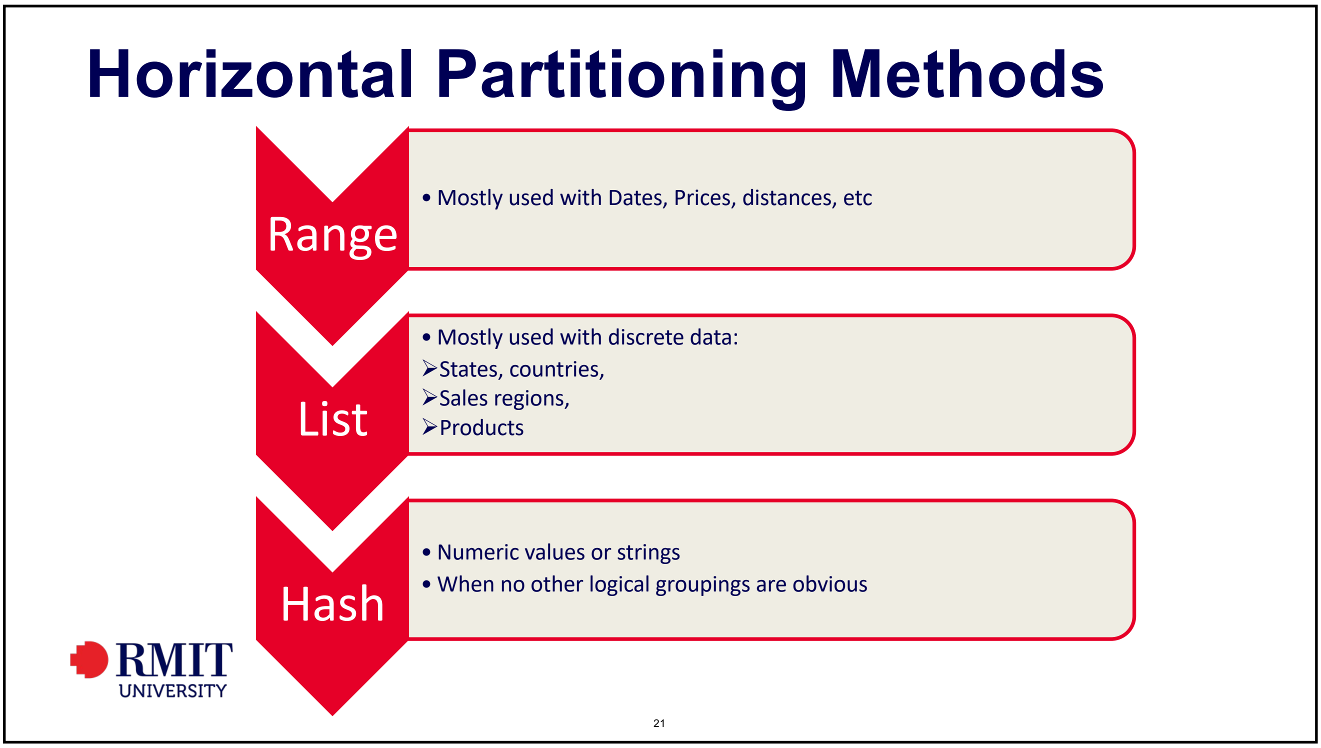
2 Timeseries

- Most likely query latest data
- Summarize data in older partitions

3 Naturally Partition Data

- Retailer, by geography
- Data science, by product category

20



Horizontal Partitioning Methods

Range

- Mostly used with Dates, Prices, distances, etc

List

- Mostly used with discrete data:
  - States, countries,
  - Sales regions,
  - Products

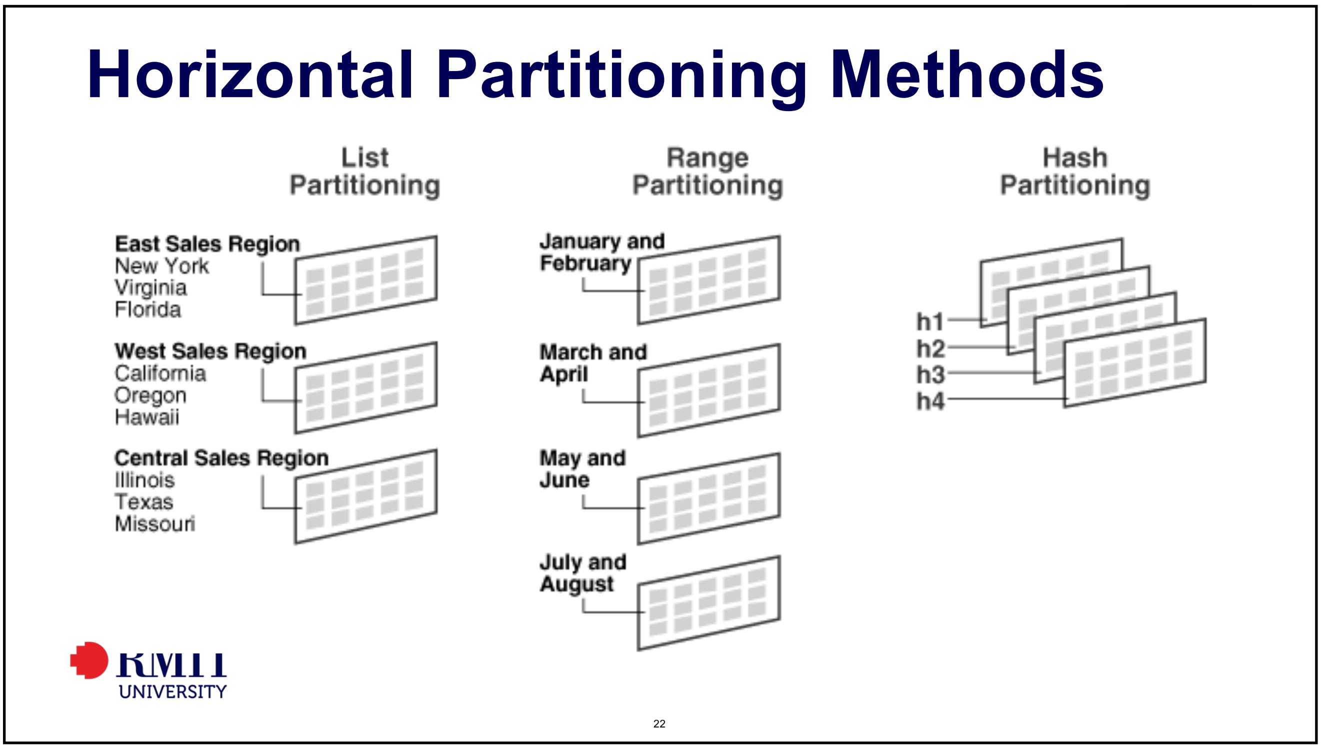
Hash

- Numeric values or strings
- When no other logical groupings are obvious

RMIT UNIVERSITY

21

21



Horizontal Partitioning Methods

List Partitioning

East Sales Region
New York
Virginia
Florida

West Sales Region
California
Oregon
Hawaii

Central Sales Region
Illinois
Texas
Missouri

Range Partitioning

January and February

March and April

May and June

July and August

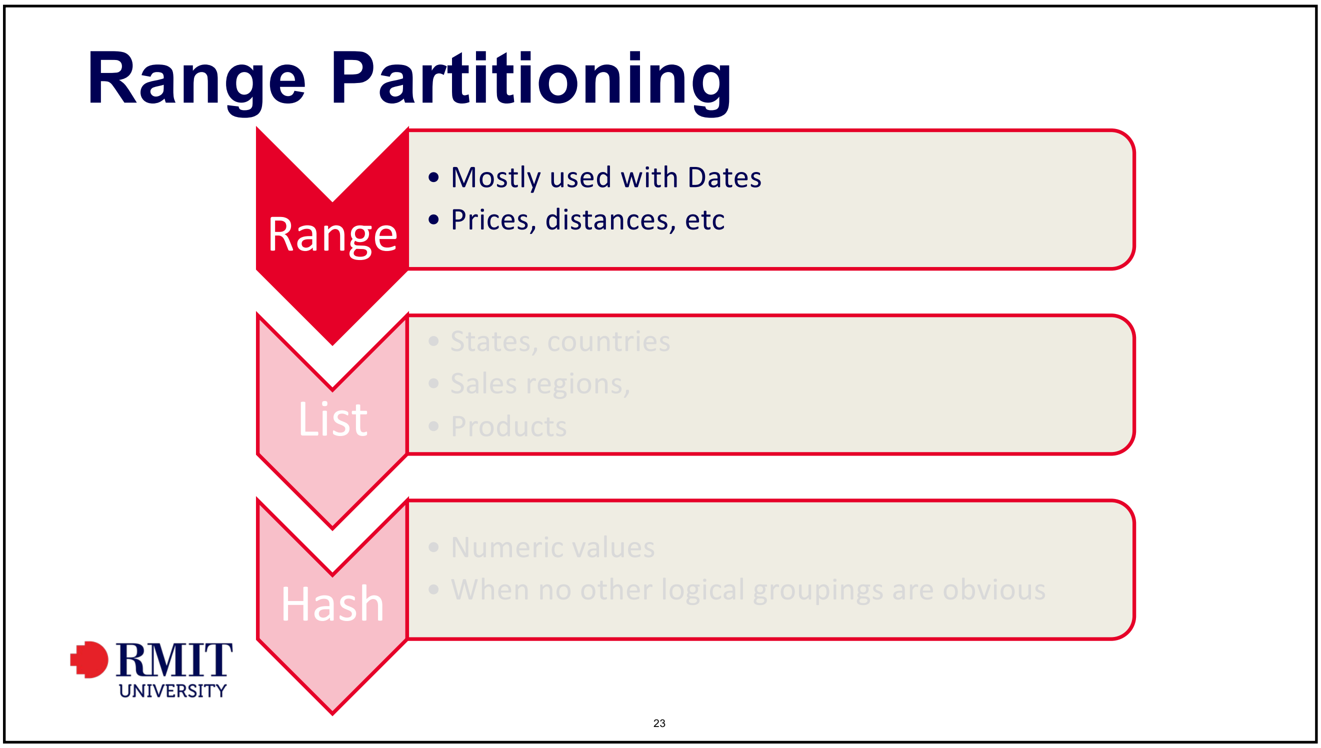
Hash Partitioning

h1
h2
h3
h4

RMIT UNIVERSITY

22

22



Range Partitioning

Range

- Mostly used with Dates
- Prices, distances, etc

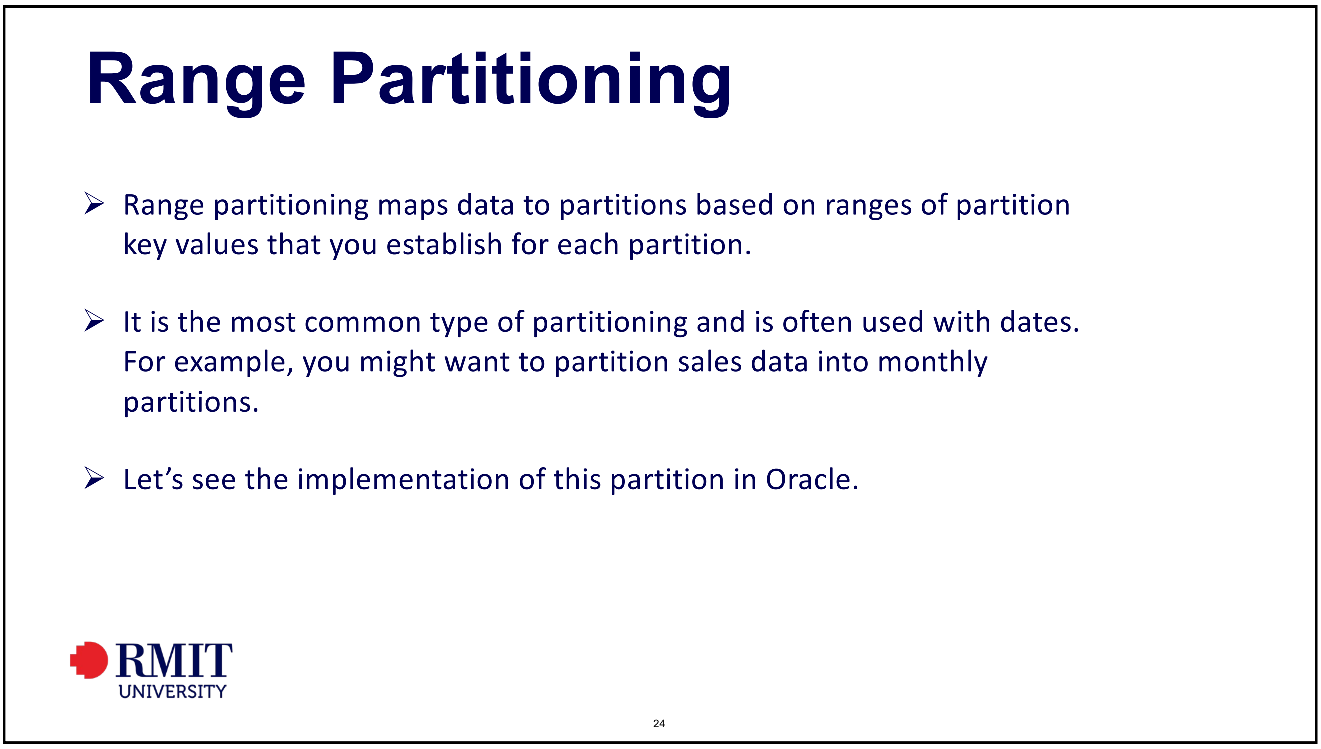
List

- States, countries
- Sales regions,
- Products

Hash

- Numeric values
- When no other logical groupings are obvious

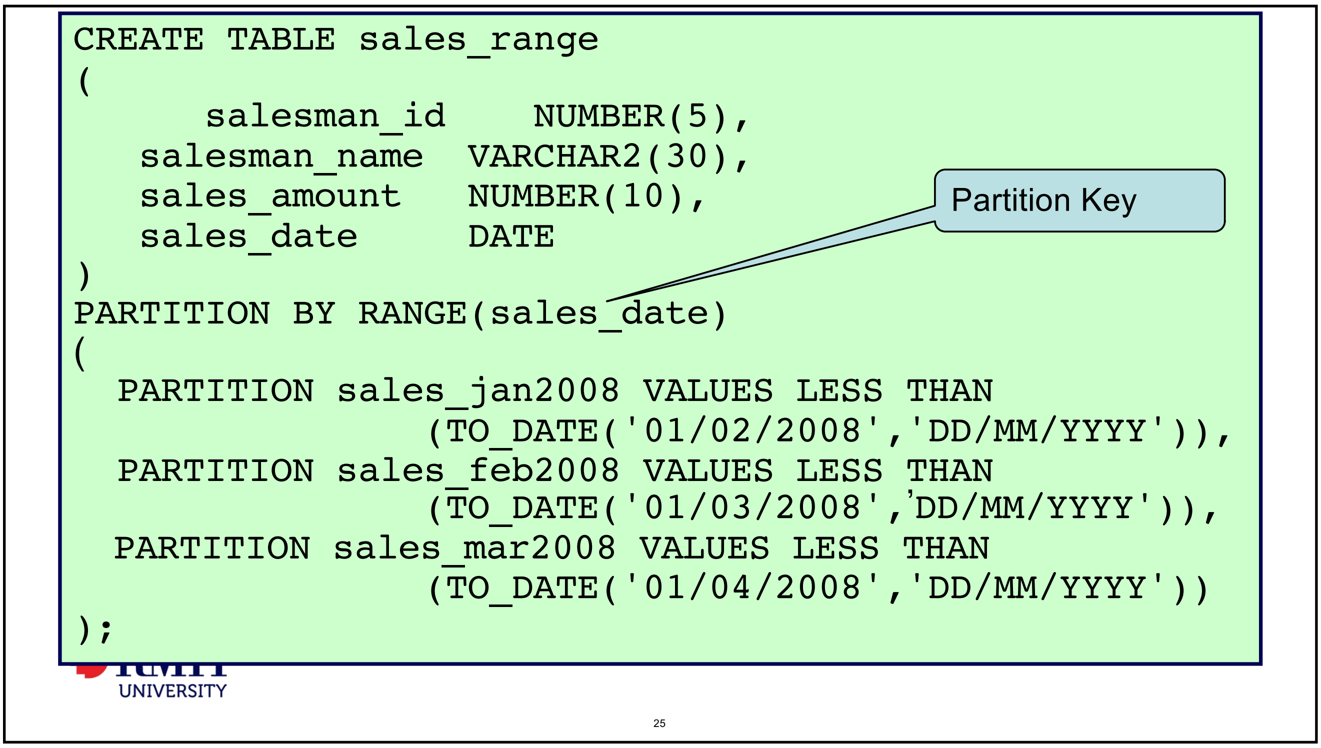
23



Range Partitioning

- ➢ Range partitioning maps data to partitions based on ranges of partition key values that you establish for each partition.
- ➢ It is the most common type of partitioning and is often used with dates. For example, you might want to partition sales data into monthly partitions.
- ➢ Let’s see the implementation of this partition in Oracle.

24

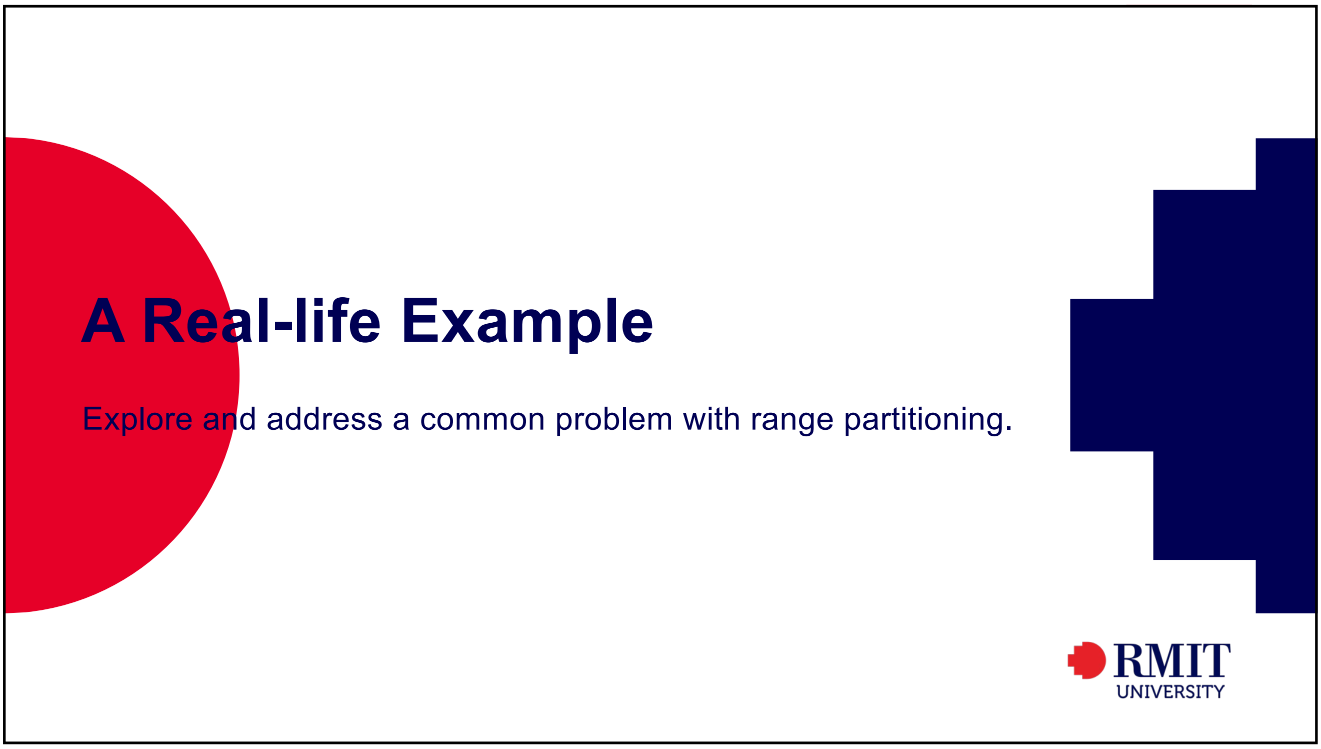


```
CREATE TABLE sales_range
(
      salesman_id    NUMBER(5),
   salesman_name  VARCHAR2(30),
   sales_amount   NUMBER(10),
   sales_date     DATE
)
PARTITION BY RANGE(sales_date)
(
  PARTITION sales_jan2008 VALUES LESS THAN
                (TO_DATE('01/02/2008','DD/MM/YYYY')),
  PARTITION sales_feb2008 VALUES LESS THAN
                (TO_DATE('01/03/2008','DD/MM/YYYY')),
  PARTITION sales_mar2008 VALUES LESS THAN
                (TO_DATE('01/04/2008','DD/MM/YYYY'))
);
```

Partition Key

25

25



A Real-life Example

Explore and address a common problem with range partitioning.

26

Interval (Range) Partitioning

```
SQL> insert into SALES
  2  values ( trunc(sysdate), .... );

insert into SALES
            *
ERROR at line 1:
ORA-14400: inserted partition key does not map to any partition
```

RMIT UNIVERSITY

https://youtu.be/f0tVH6ZylQ0

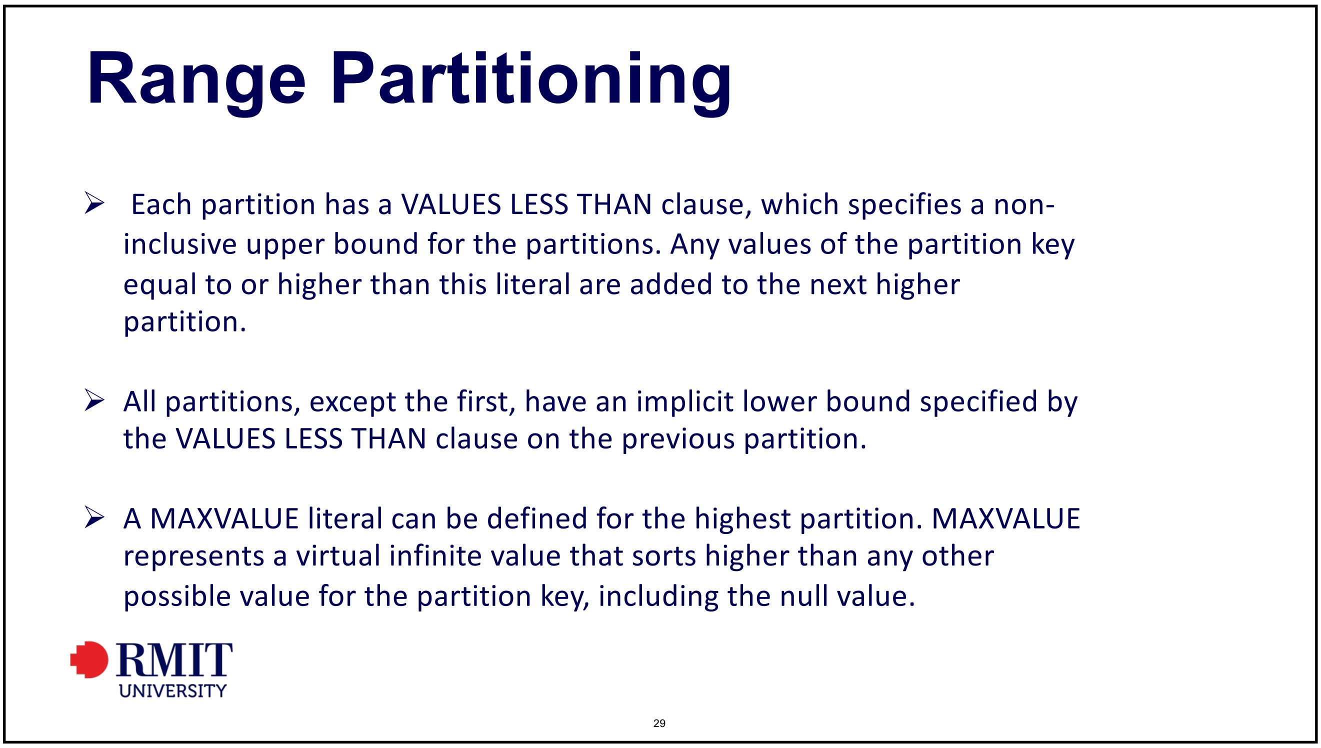
27

27

```
CREATE TABLE sales_range
(
      salesman_id    NUMBER(5),
   salesman_name  VARCHAR2(30),
   sales_amount   NUMBER(10),
   sales_date     DATE
)
PARTITION BY RANGE(sales_date)
(
  PARTITION sales_jan2008 VALUES LESS THAN
                (TO_DATE('01/02/2008','DD/MM/YYYY')),
  PARTITION sales_feb2008 VALUES LESS THAN
                (TO_DATE('01/03/2008','DD/MM/YYYY')),
  PARTITION sales_mar2008 VALUES LESS THAN
                (TO_DATE('01/04/2008','DD/MM/YYYY')),
  PARTITION remaining_sales VALUES LESS THAN
                                       (MAXVALUE)
);
```

Catch-all partition

28



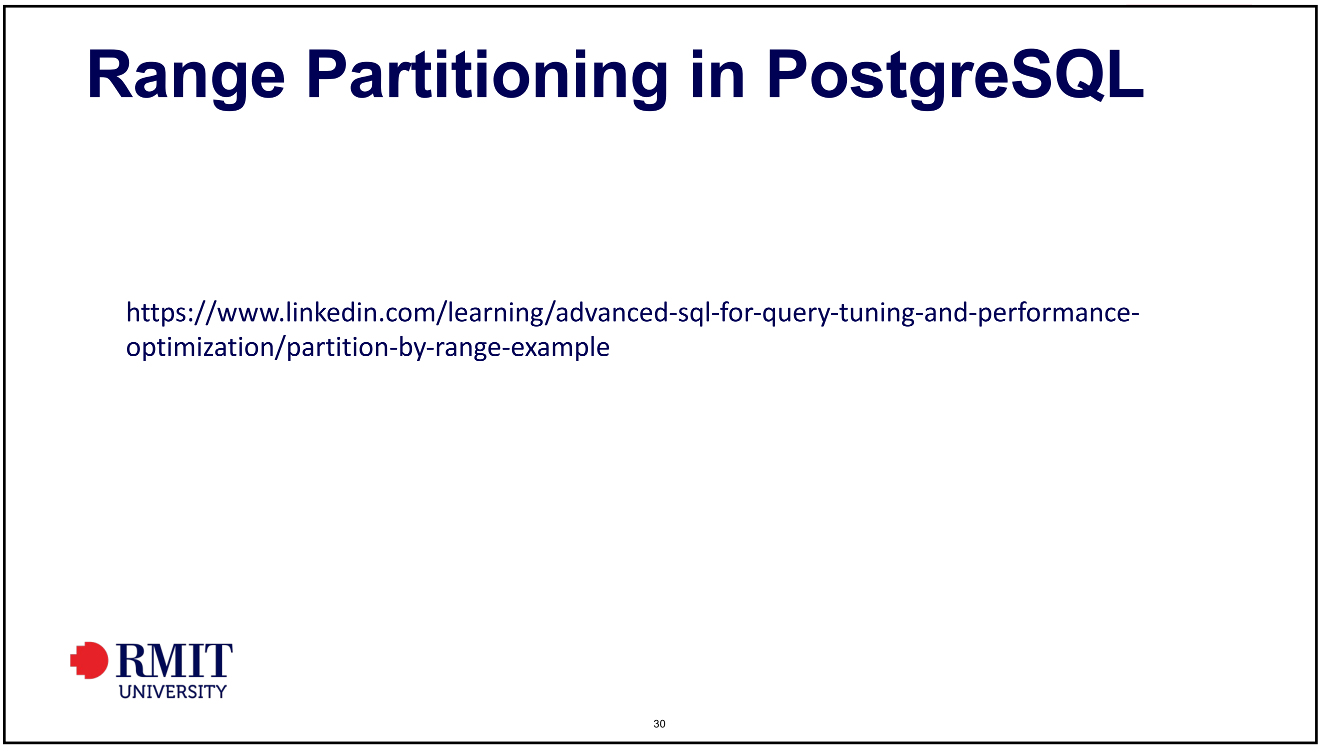
Range Partitioning

- Each partition has a VALUES LESS THAN clause, which specifies a non-inclusive upper bound for the partitions. Any values of the partition key equal to or higher than this literal are added to the next higher partition.

- All partitions, except the first, have an implicit lower bound specified by the VALUES LESS THAN clause on the previous partition.

- A MAXVALUE literal can be defined for the highest partition. MAXVALUE represents a virtual infinite value that sorts higher than any other possible value for the partition key, including the null value.

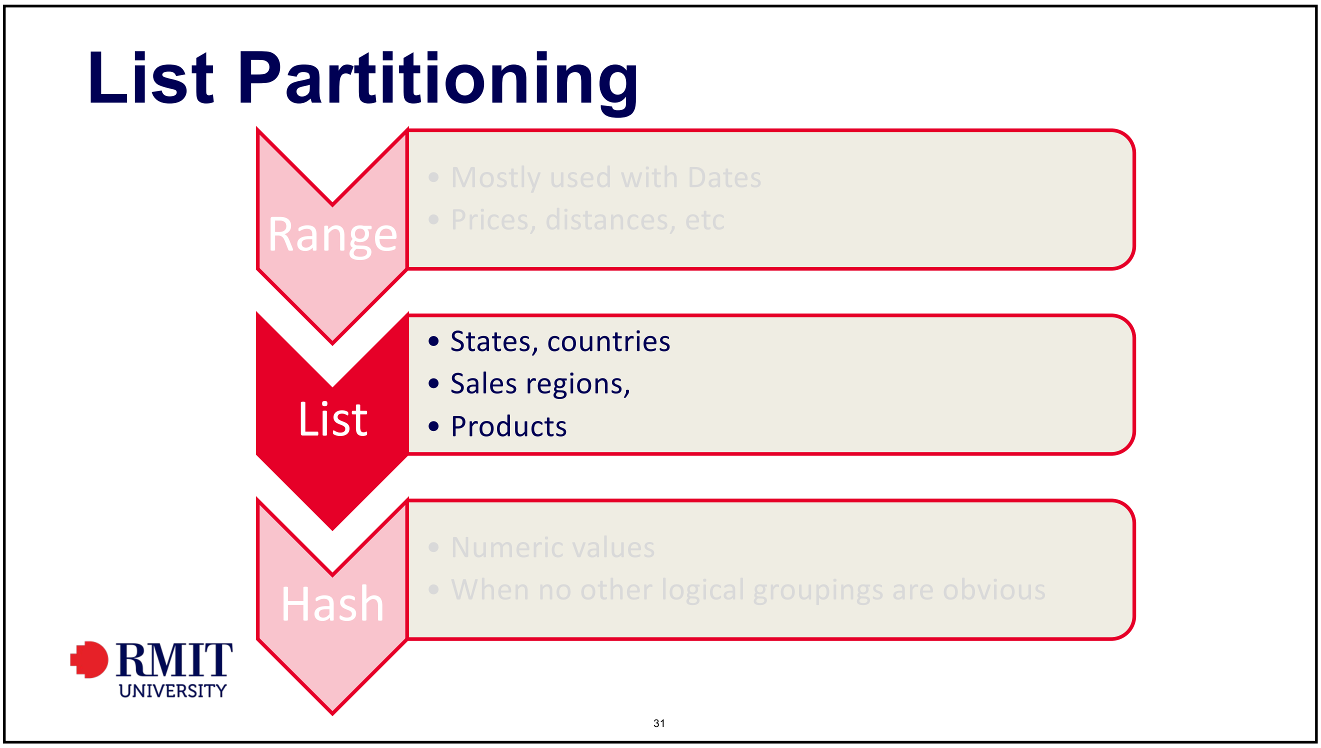
RMIT
UNIVERSITY



Range Partitioning in PostgreSQL

https://www.linkedin.com/learning/advanced-sql-for-query-tuning-and-performance-optimization/partition-by-range-example

RMIT
UNIVERSITY



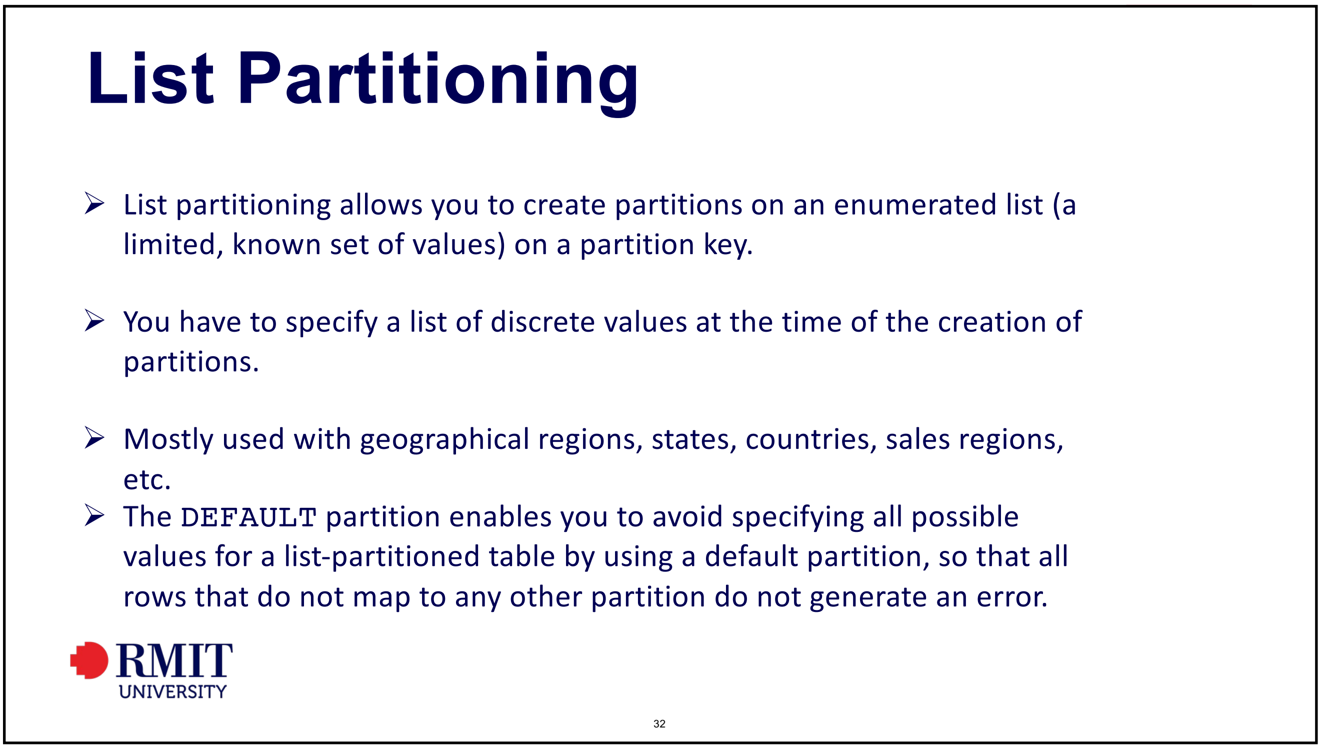
List Partitioning

Range
- Mostly used with Dates
- Prices, distances, etc

List
- States, countries
- Sales regions,
- Products

Hash
- Numeric values
- When no other logical groupings are obvious

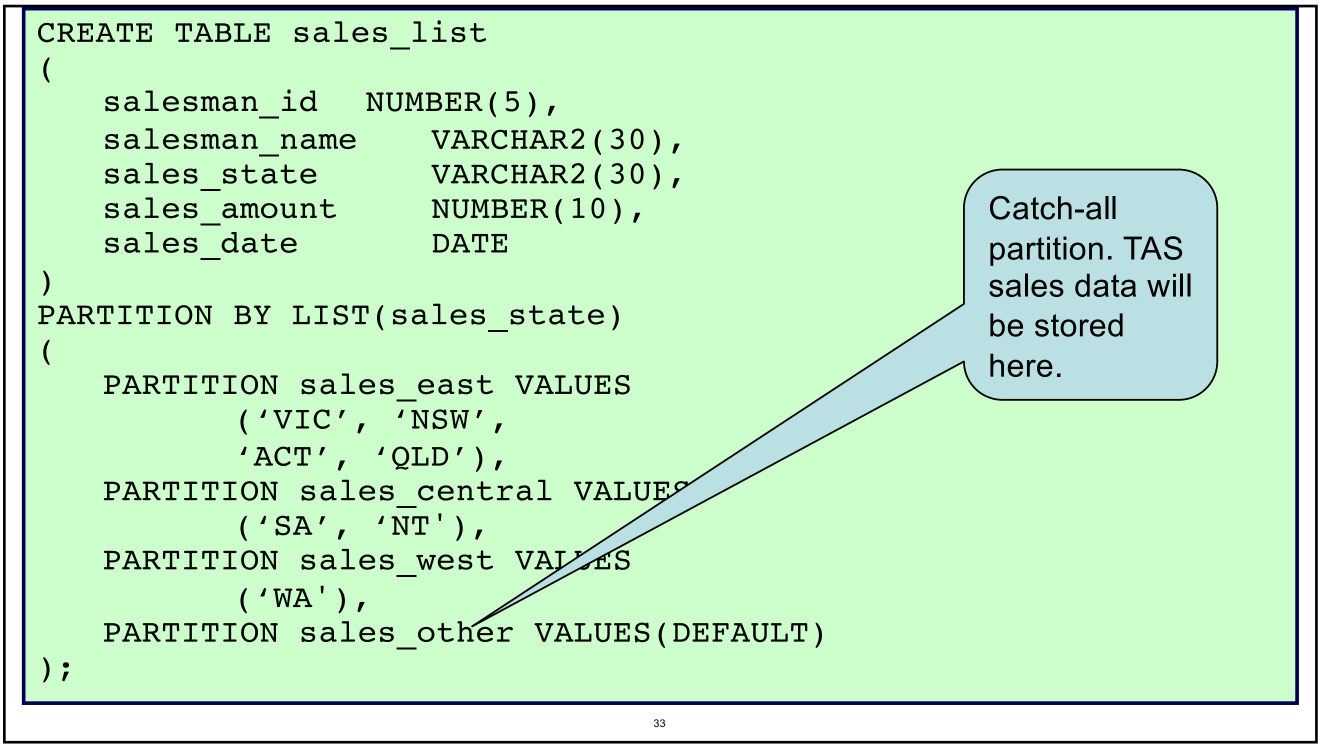
RMIT UNIVERSITY



List Partitioning

- List partitioning allows you to create partitions on an enumerated list (a limited, known set of values) on a partition key.
- You have to specify a list of discrete values at the time of the creation of partitions.
- Mostly used with geographical regions, states, countries, sales regions, etc.
- The `DEFAULT` partition enables you to avoid specifying all possible values for a list-partitioned table by using a default partition, so that all rows that do not map to any other partition do not generate an error.

RMIT UNIVERSITY

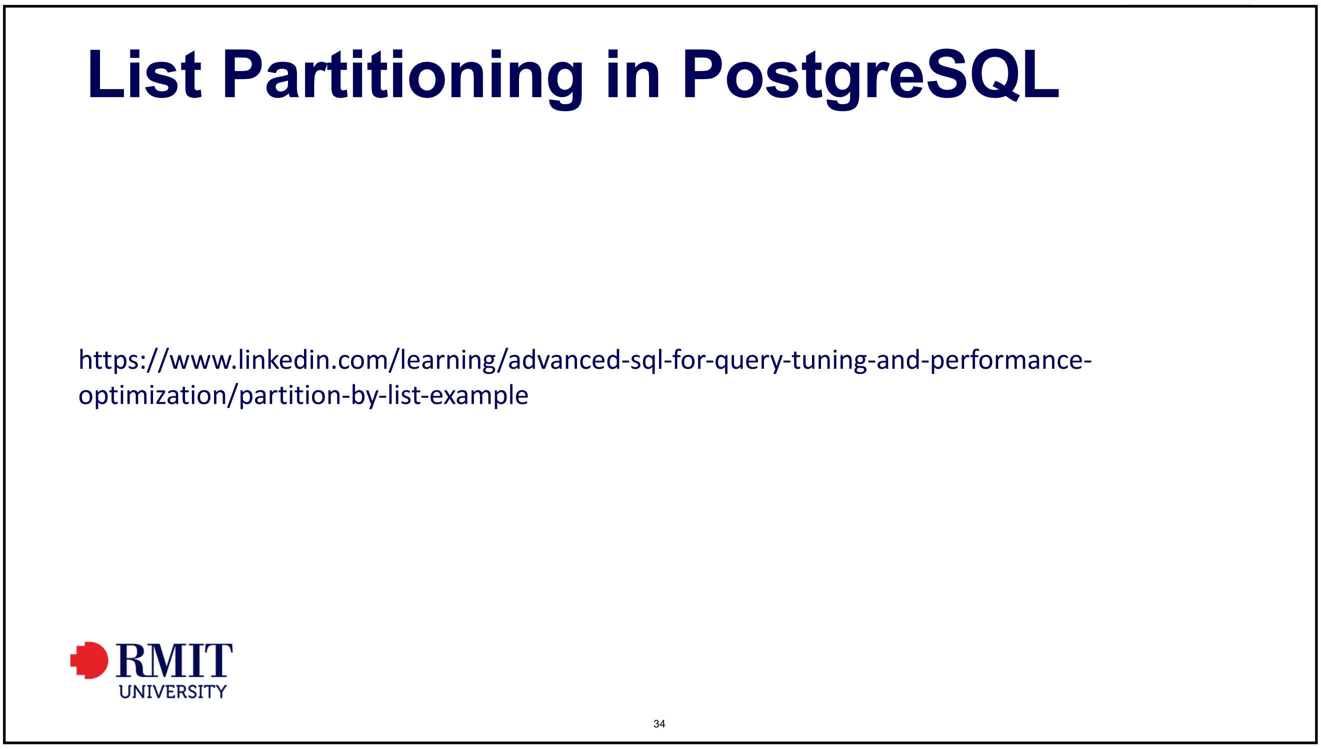


```
CREATE TABLE sales_list
(
   salesman_id  NUMBER(5),
   salesman_name    VARCHAR2(30),
   sales_state      VARCHAR2(30),
   sales_amount     NUMBER(10),
   sales_date       DATE
)
PARTITION BY LIST(sales_state)
(
   PARTITION sales_east VALUES
         (‘VIC’, ‘NSW’,
         ‘ACT’, ‘QLD’),
   PARTITION sales_central VALUES
         (‘SA’, ‘NT'),
   PARTITION sales_west VALUES
         (‘WA'),
   PARTITION sales_other VALUES(DEFAULT)
);
```

Catch-all partition. TAS sales data will be stored here.

33

33



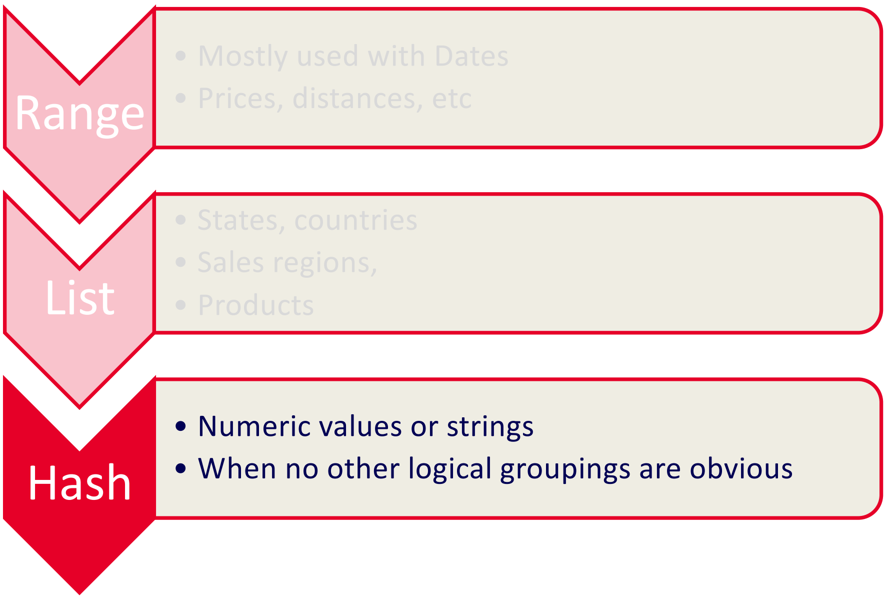
List Partitioning in PostgreSQL

https://www.linkedin.com/learning/advanced-sql-for-query-tuning-and-performance-optimization/partition-by-list-example

34

34

Hash Partitioning



RMIT UNIVERSITY

35

35

Hash Partitioning

- ➢ Let's suppose we store sales data and most queries are based on the customers (say, customer email addresses are used to identify customers.) There is no viable way to define a range with email addresses. We have millions of customers, so, list partitioning is not also possible.
- ➢ This situation leave us with Hash Partitioning as the only option.
- ➢ You are not required to define a hash function for the partitioning -- the DBMS uses its own choice of hash function which attempts to distribute rows as uniformly as possible among partitions.

RMIT UNIVERSITY

36

36

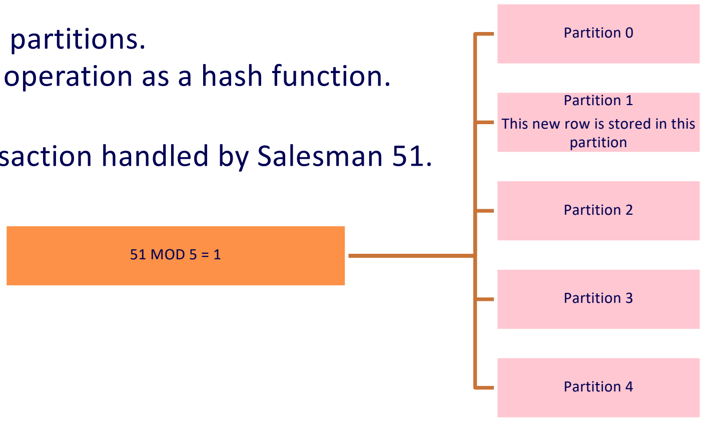
Hash Partitioning

- Hash Partitioning is also used, when:
  - You do not know beforehand how much data maps into a given range
  - Data are not uniformly distributed among range intervals (say, 1000 sales in 2019, 1200 in 2020, then 10,000,000 sales in 2021).
  - Range partitioning would cause the data to be undesirably clustered.
  - The above can happen with List Partitioning, too.

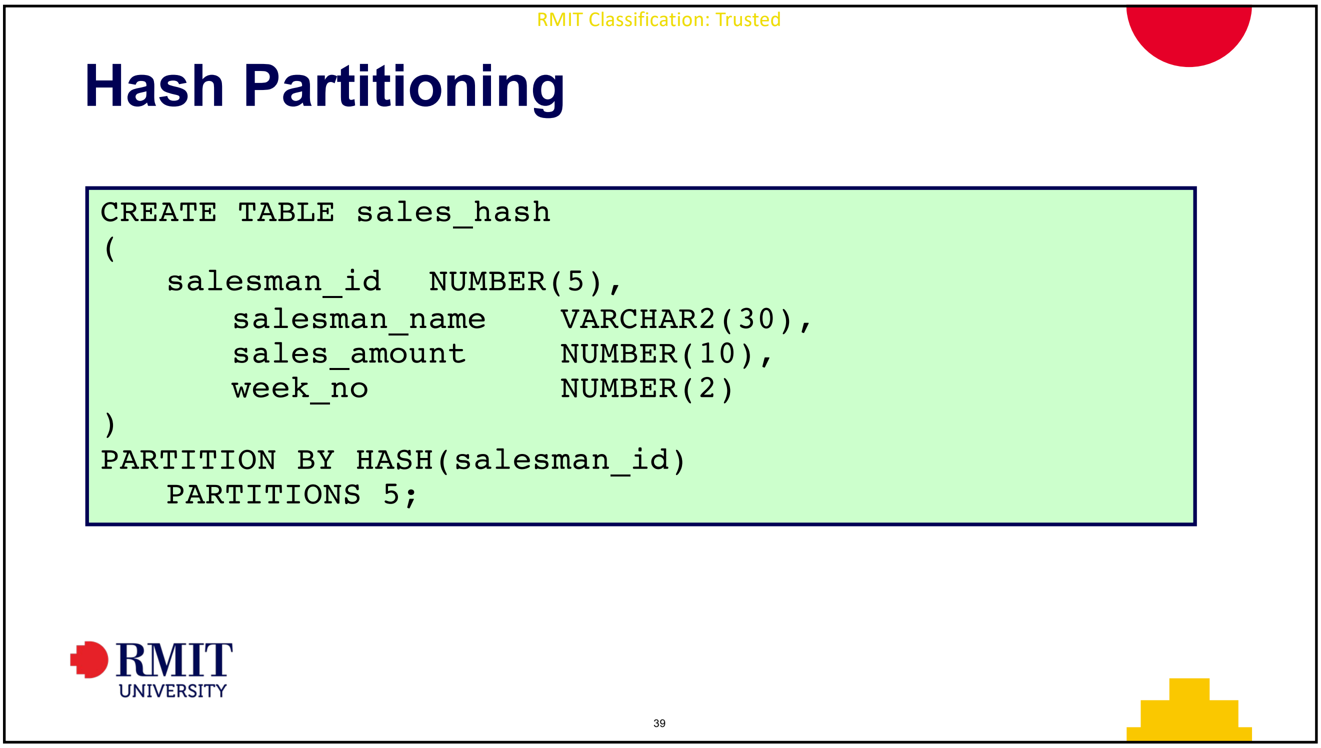
RMIT UNIVERSITY

Hash Partitioning - Example

- Assume we are going to partition `SalesData` table based on `Salesman_ID`.
- Assume we determined to use 5 partitions.
- Most databases uses `Modulus` operation as a hash function.
- E.g. When we insert a sales transaction handled by Salesman 51.

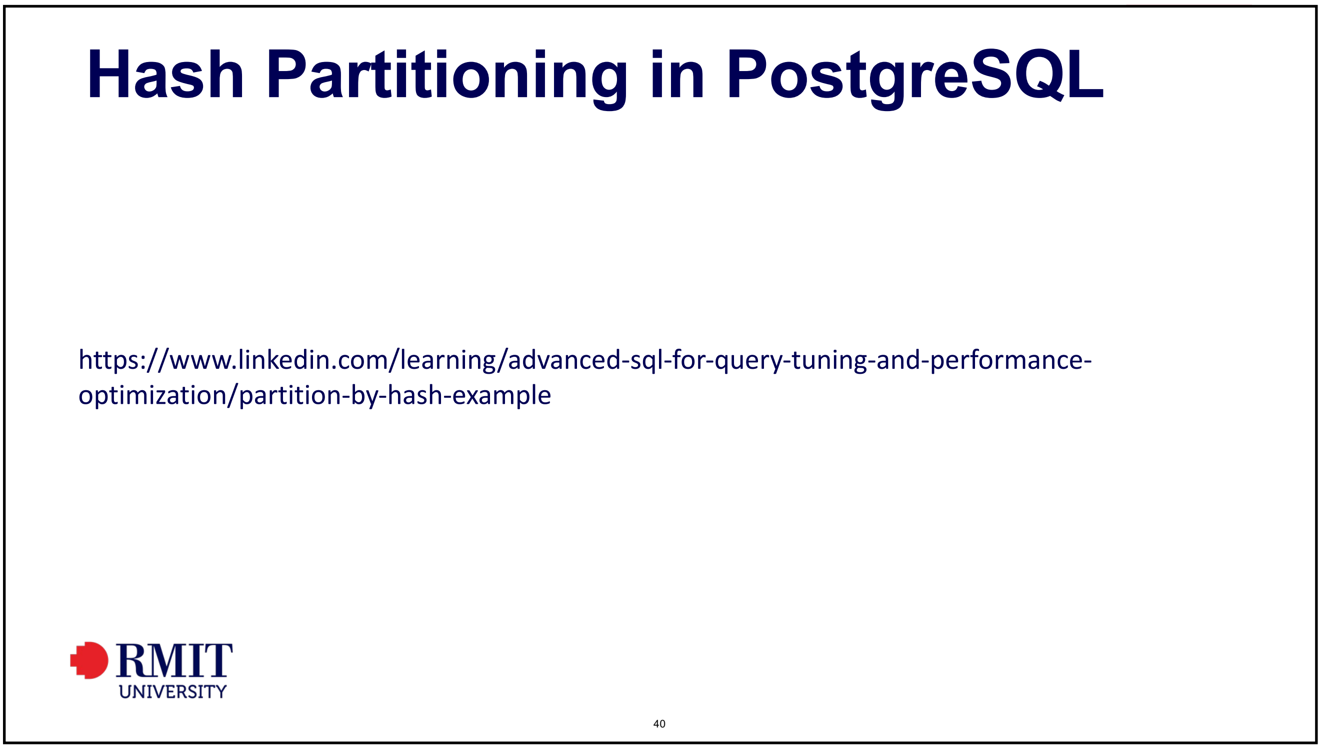


RMIT UNIVERSITY



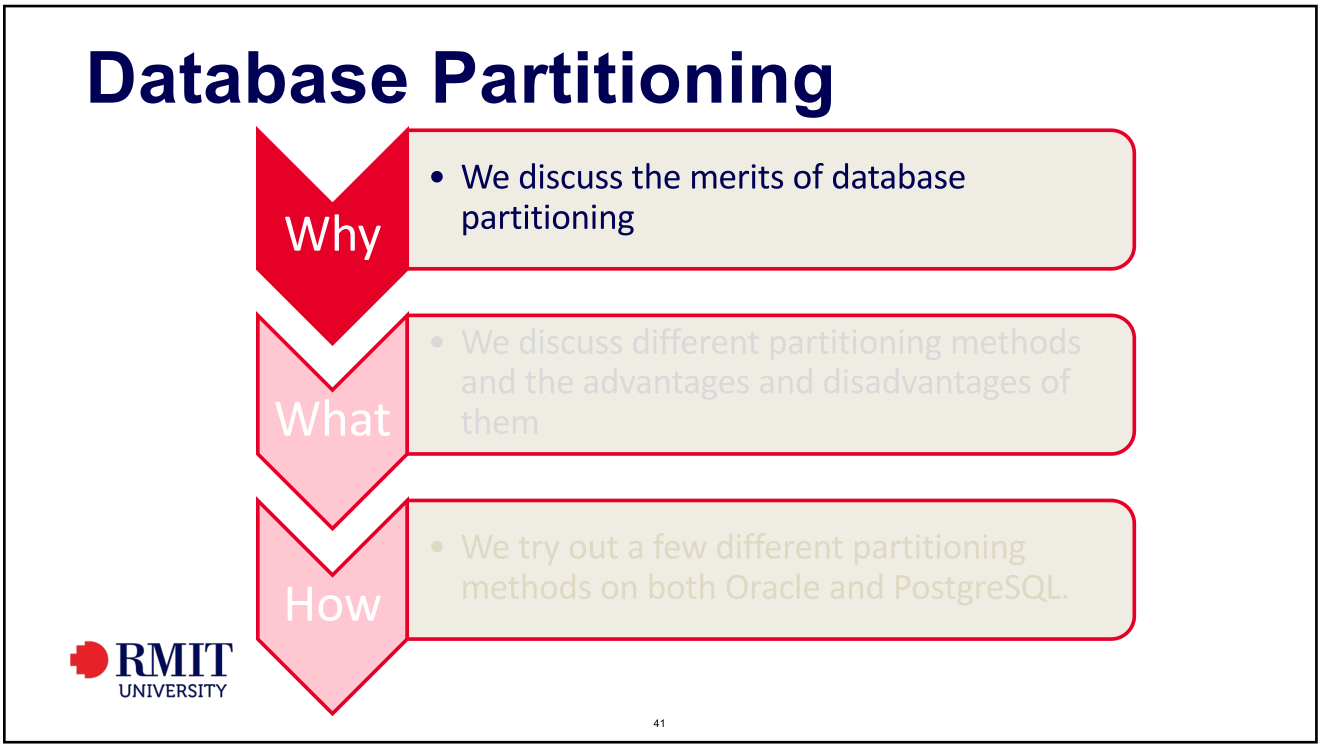
Hash Partitioning

```
CREATE TABLE sales_hash
(
   salesman_id  NUMBER(5),
      salesman_name    VARCHAR2(30),
      sales_amount     NUMBER(10),
      week_no          NUMBER(2)
)
PARTITION BY HASH(salesman_id)
   PARTITIONS 5;
```



Hash Partitioning in PostgreSQL

https://www.linkedin.com/learning/advanced-sql-for-query-tuning-and-performance-optimization/partition-by-hash-example



Database Partitioning

Why

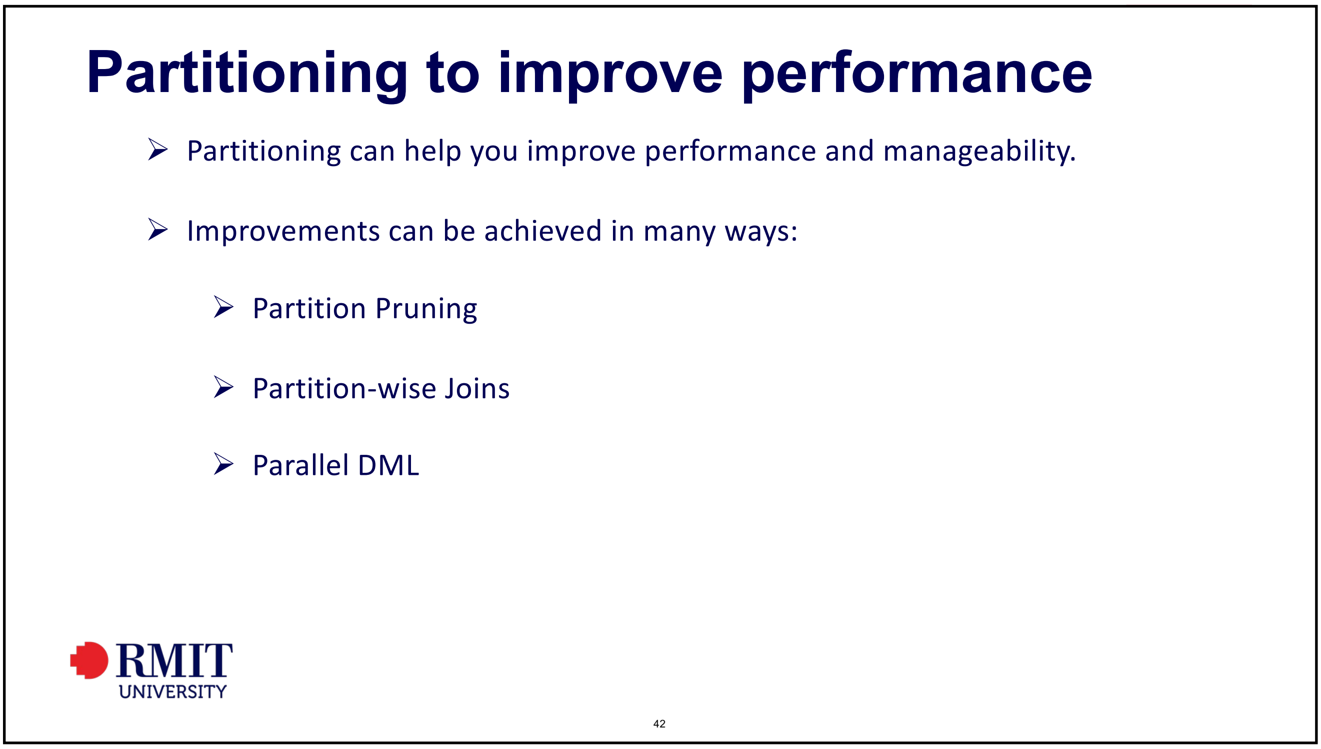
- We discuss the merits of database partitioning

What

- We discuss different partitioning methods and the advantages and disadvantages of them

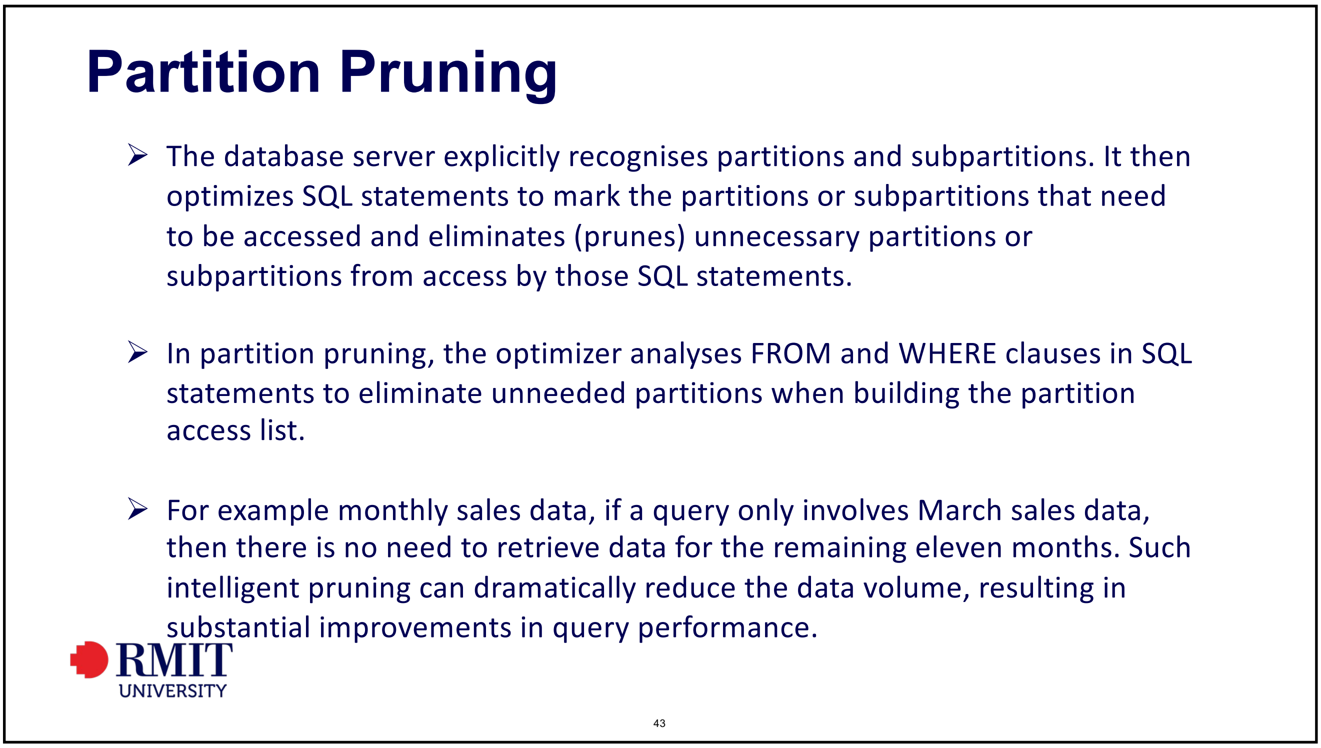
How

- We try out a few different partitioning methods on both Oracle and PostgreSQL.



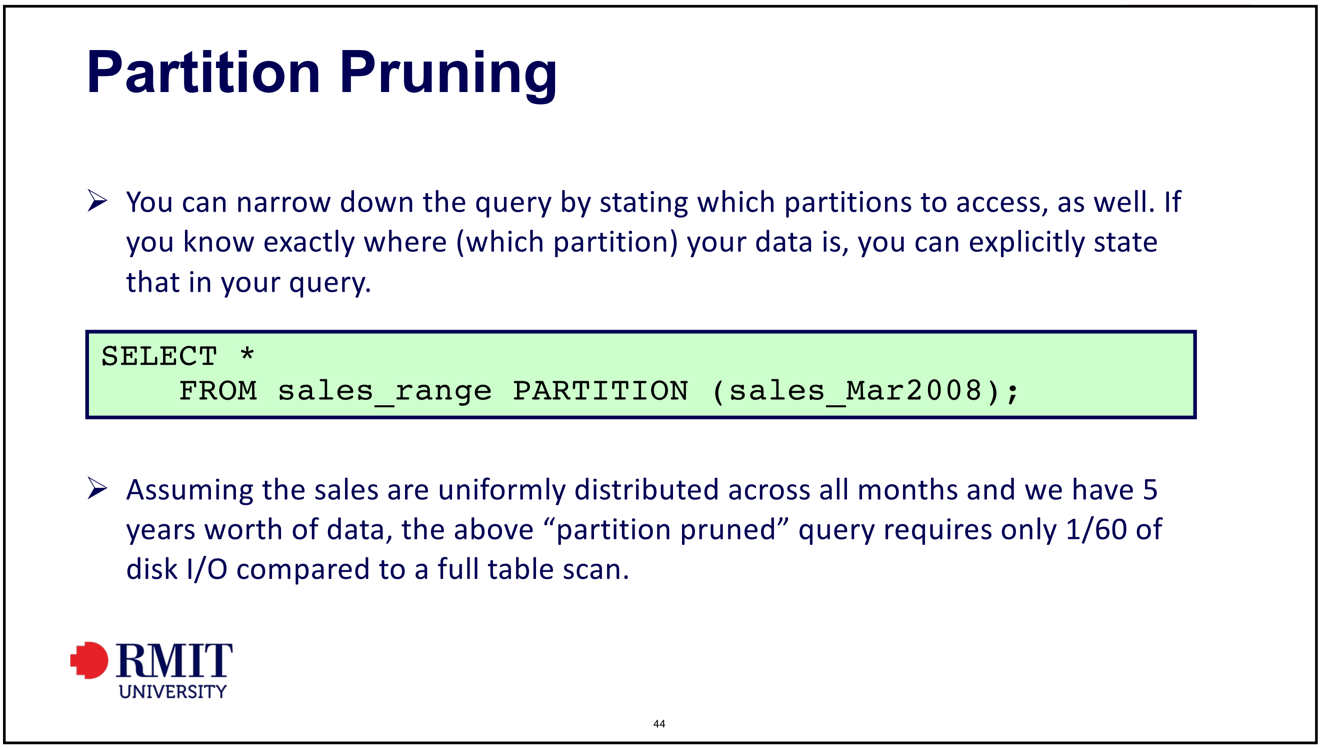
Partitioning to improve performance

- Partitioning can help you improve performance and manageability.
- Improvements can be achieved in many ways:
  - Partition Pruning
  - Partition-wise Joins
  - Parallel DML



Partition Pruning

- The database server explicitly recognises partitions and subpartitions. It then optimizes SQL statements to mark the partitions or subpartitions that need to be accessed and eliminates (prunes) unnecessary partitions or subpartitions from access by those SQL statements.
- In partition pruning, the optimizer analyses FROM and WHERE clauses in SQL statements to eliminate unneeded partitions when building the partition access list.
- For example monthly sales data, if a query only involves March sales data, then there is no need to retrieve data for the remaining eleven months. Such intelligent pruning can dramatically reduce the data volume, resulting in substantial improvements in query performance.



Partition Pruning

- You can narrow down the query by stating which partitions to access, as well. If you know exactly where (which partition) your data is, you can explicitly state that in your query.

```
SELECT *
    FROM sales_range PARTITION (sales_Mar2008);
```

- Assuming the sales are uniformly distributed across all months and we have 5 years worth of data, the above “partition pruned” query requires only 1/60 of disk I/O compared to a full table scan.

Partition-wise Joins

- A partition-wise join is a join optimisation for joining two tables that are both partitioned along the join column(s).
- With partition-wise joins, the join operation is broken into smaller joins that are performed sequentially or in parallel.
- As a result of partitioning, only smaller chunks of records are to be accessed for each join.

Parallel SQL

- Parallel execution dramatically reduces response time for data-intensive operations on large databases typically associated with decision support systems and data warehouses.
- In addition to conventional tables, you can use parallel query and parallel SQL with range- and hash-partitioned tables. By doing so, you can enhance scalability and performance for batch operations.

Summary

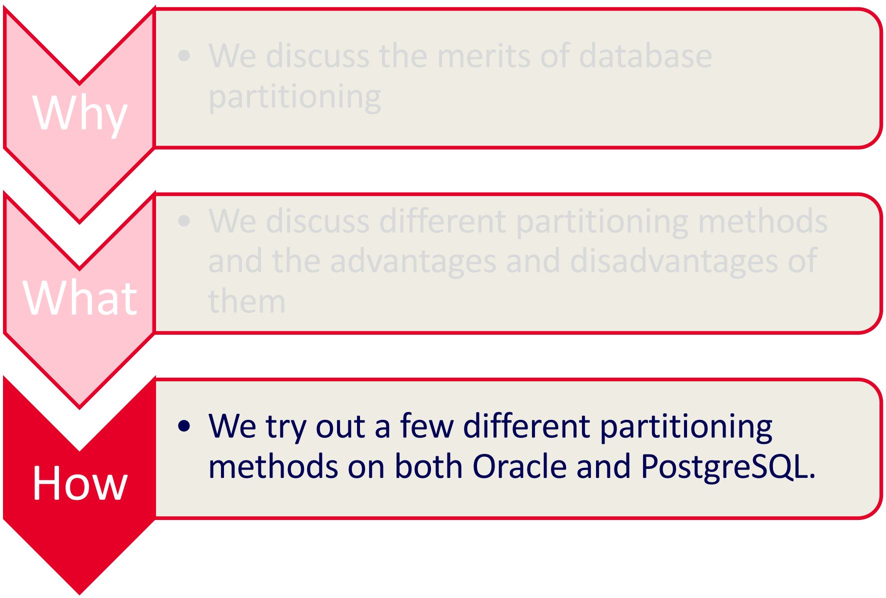
- Discussed two ways of partitioning VERY BIG tables
- Vertical Vs Horizontal
- Discussed three different ways of doing horizontal partitioning – Range, List and Hash
- Some advantages of partitioning.

RMIT UNIVERSITY

47

47

Database Partitioning - Demo


Why
- We discuss the merits of database partitioning

What
- We discuss different partitioning methods and the advantages and disadvantages of them

How
- We try out a few different partitioning methods on both Oracle and PostgreSQL.

RMIT UNIVERSITY

48

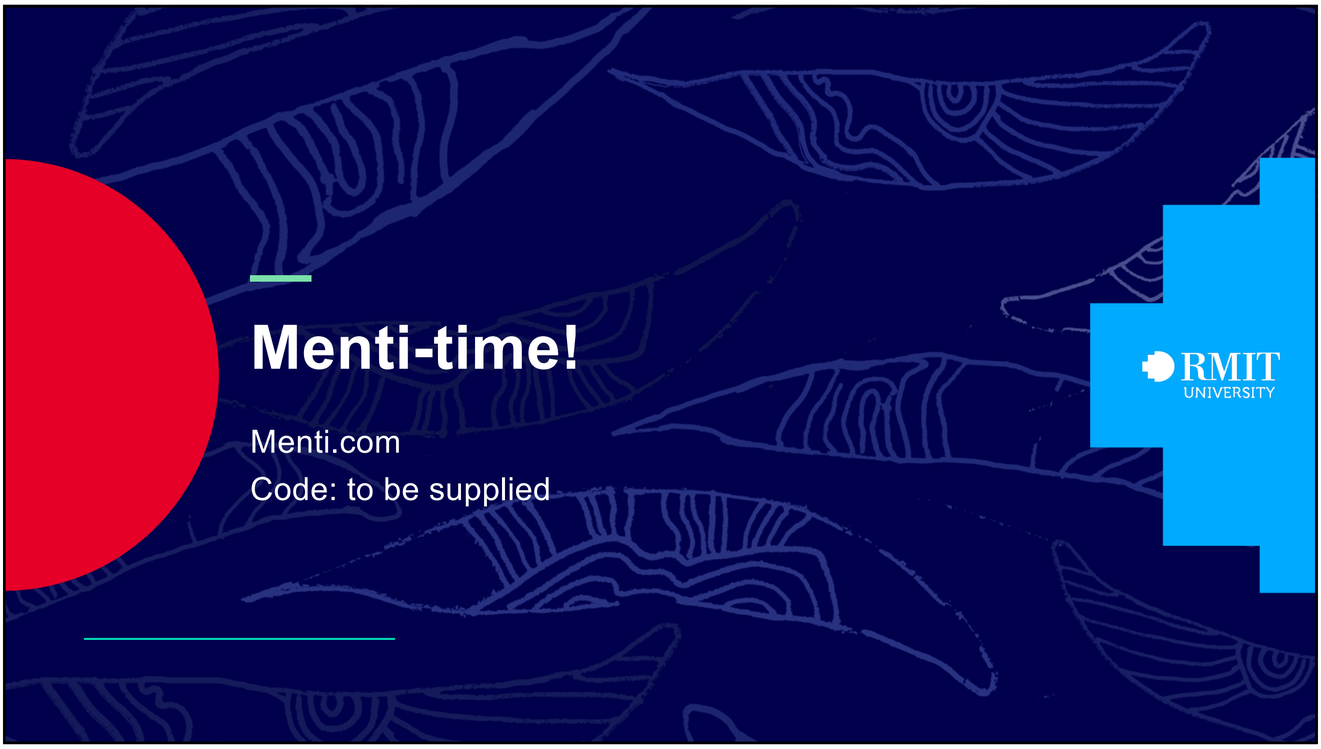
48

Demo Time!

Let’s try out a query.

RMIT UNIVERSITY

49



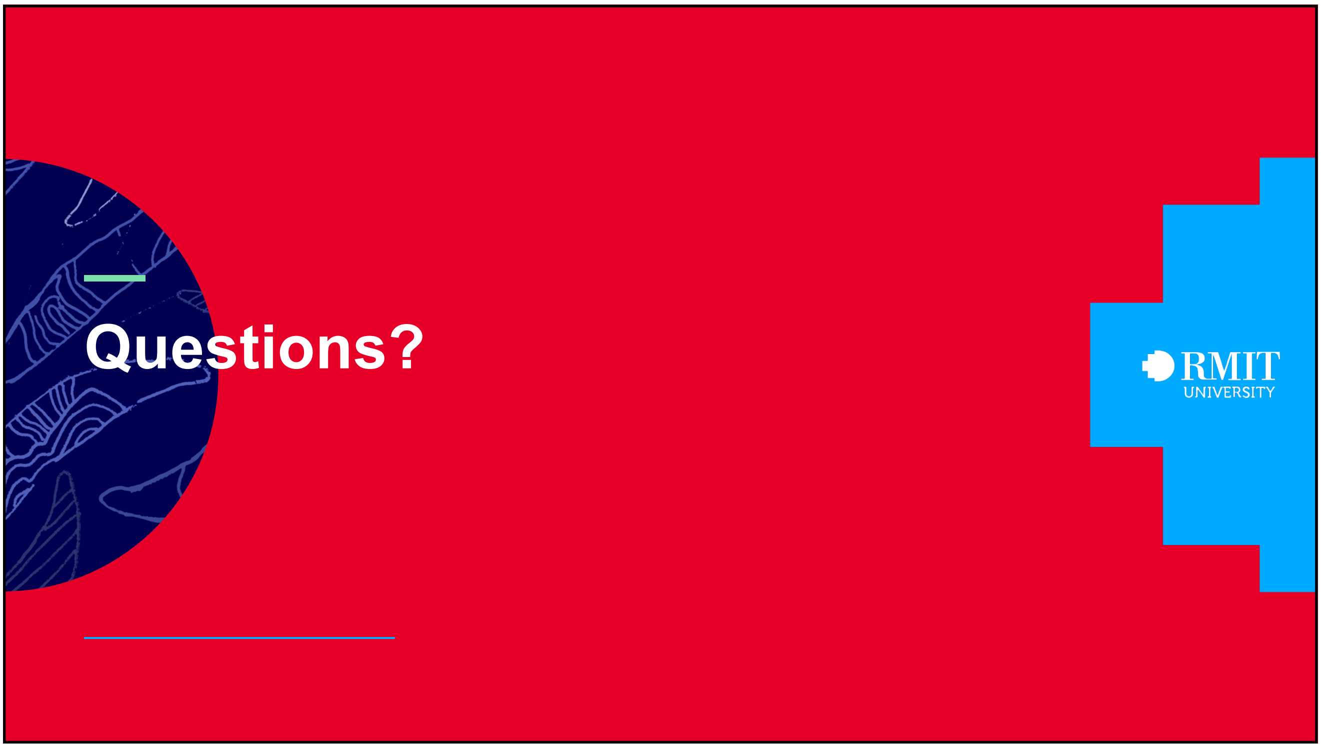
Menti-time!

Menti.com

Code: to be supplied

RMIT UNIVERSITY

50



Questions?

RMIT UNIVERSITY

51