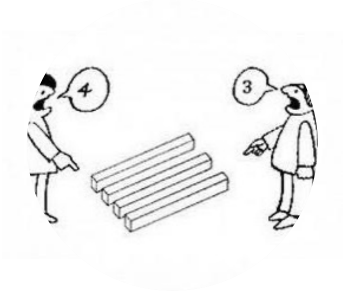
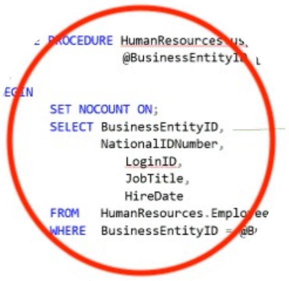
Database Applications
Lecture 5: Views, Triggers, Stored Procedures

Santha Sumanasekara

August 2019

RMIT UNIVERSITY





```
PROCEDURE HumanResources.us
          @BusinessEntityI
EGIN
    SET NOCOUNT ON;
    SELECT BusinessEntityID,
           NationalIDNumber,
             LoginID,
             JobTitle,
             HireDate
    FROM   HumanResources.Employee
    WHERE  BusinessEntityID = @B
```

Views

Views

Materialized Views

Triggers

Before Triggers

After Triggers

INSTEAD OF Triggers

Stored Procedures

PL/SQL

Anonymous Blocks

Stored Procedures

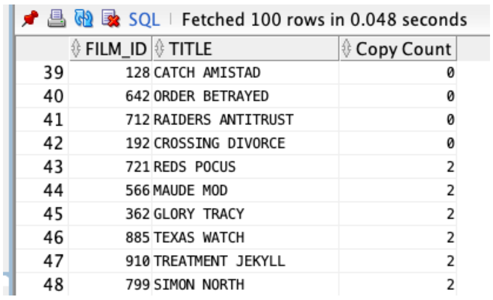
RMIT UNIVERSITY

“Simple” Views– why?

- Views are kind of virtual tables, allow users to do the following:
  - Structure data in a way that users find natural or intuitive.
  - Restrict access to the data such that a user can only see limited data instead of complete table.
  - Summarise data from various tables which can be used to generate reports.
- Views are defined using an SQL statement. So, you may consider a view as a stored query, assigned with a name (and also a virtual schema derived from the underlying SQL statement).

Views – Create a view

- On our movie rental database, we can summarise data, say for each film, we can count the number copies in the inventory, and store along with the movie title:

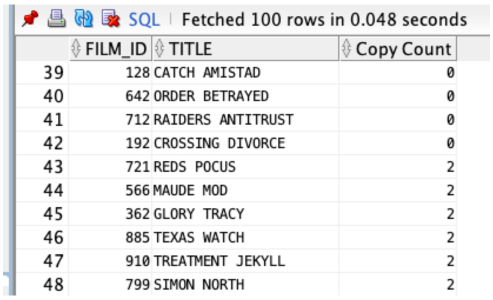


Fetched 100 rows in 0.048 seconds

	FILM_ID	TITLE	Copy Count
39	128	CATCH AMISTAD	0
40	642	ORDER BETRAYED	0
41	712	RAIDERS ANTITRUST	0
42	192	CROSSING DIVORCE	0
43	721	REDS POCUS	2
44	566	MAUDE MOD	2
45	362	GLORY TRACY	2
46	885	TEXAS WATCH	2
47	910	TREATMENT JEKYLL	2
48	799	SIMON NORTH	2

Views – Create a view

```
SELECT f.film_id, title, COUNT(i.film_id) AS "Copy Count"
    FROM film f LEFT OUTER JOIN inventory i ON f.film_id = i.film_id
    GROUP BY f.film_id, title
    ORDER BY COUNT(i.film_id);
```



```
Fetched 100 rows in 0.048 seconds

      FILM_ID  TITLE                Copy Count
39        128  CATCH AMISTAD                 0
40        642  ORDER BETRAYED                0
41        712  RAIDERS ANTITRUST             0
42        192  CROSSING DIVORCE              0
43        721  REDS POCUS                    2
44        566  MAUDE MOD                     2
45        362  GLORY TRACY                   2
46        885  TEXAS WATCH                   2
47        910  TREATMENT JEKYLL              2
48        799  SIMON NORTH                   2
```

RMIT
UNIVERSITY

Views – Create a view

```
SELECT f.film_id, title, COUNT(i.film_id) AS "Copy Count"
    FROM film f LEFT OUTER JOIN inventory i ON f.film_id = i.film_id
    GROUP BY f.film_id, title
    ORDER BY COUNT(i.film_id);
```

- Use the above SQL query as the basis of the view definition.

```
CREATE VIEW film_copy (film_id, title, copy_count)
AS
      SELECT f.film_id, title, COUNT(i.film_id) AS "Copy Count"
          FROM film f LEFT OUTER JOIN inventory i ON f.film_id = i.film_id
          GROUP BY f.film_id, title
          ORDER BY COUNT(i.film_id);
```

RMIT
UNIVERSITY

Views – Using a view

- A view can be use exactly the same way as a table.

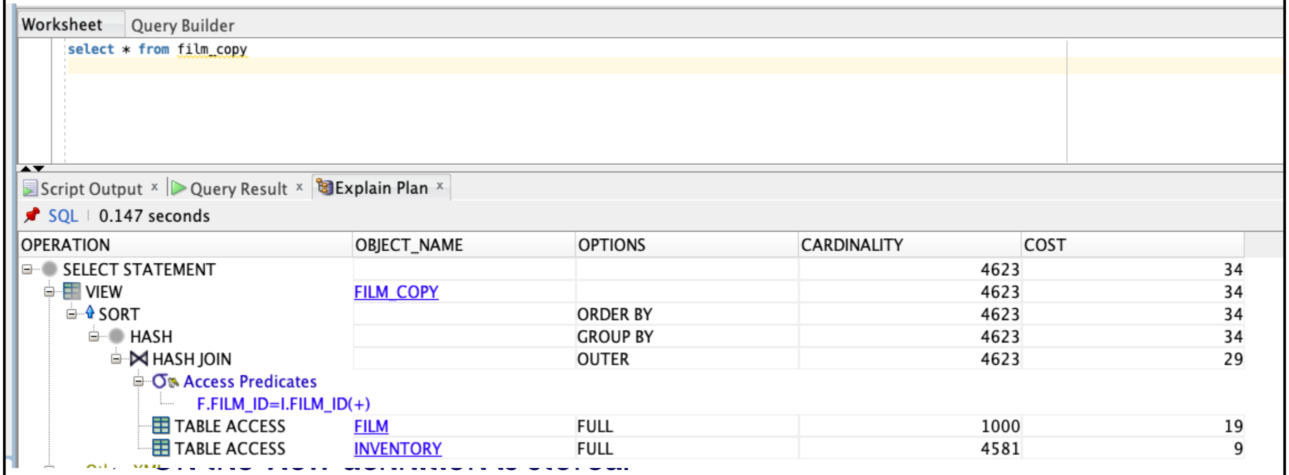
```
SELECT *
    FROM film_copy;
```

- When you run this query, data are extracted from base tables.
- Data are not physically stored on the view.
- On the view definition is stored.

Film_copy is the view we created in the previous step.

RMIT UNIVERSITY

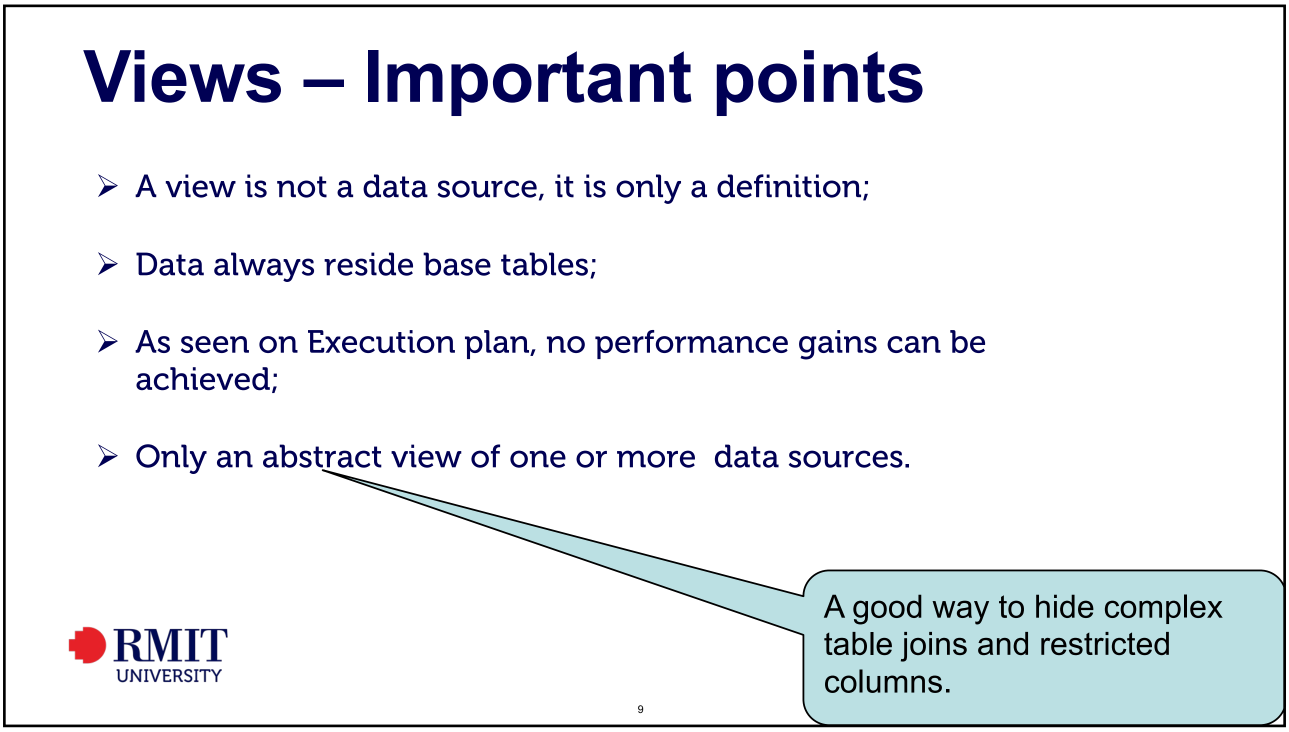
Views – Using a view



```
select * from film_copy
```

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			4623	34
VIEW	FILM_COPY		4623	34
SORT		ORDER BY	4623	34
HASH		GROUP BY	4623	34
HASH JOIN		OUTER	4623	29
Access Predicates				
F.FILM_ID=I.FILM_ID(+)				
TABLE ACCESS	FILM	FULL	1000	19
TABLE ACCESS	INVENTORY	FULL	4581	9

RMIT UNIVERSITY



Views – Important points

- A view is not a data source, it is only a definition;
- Data always reside base tables;
- As seen on Execution plan, no performance gains can be achieved;
- Only an abstract view of one or more data sources.

A good way to hide complex table joins and restricted columns.

RMIT UNIVERSITY

9

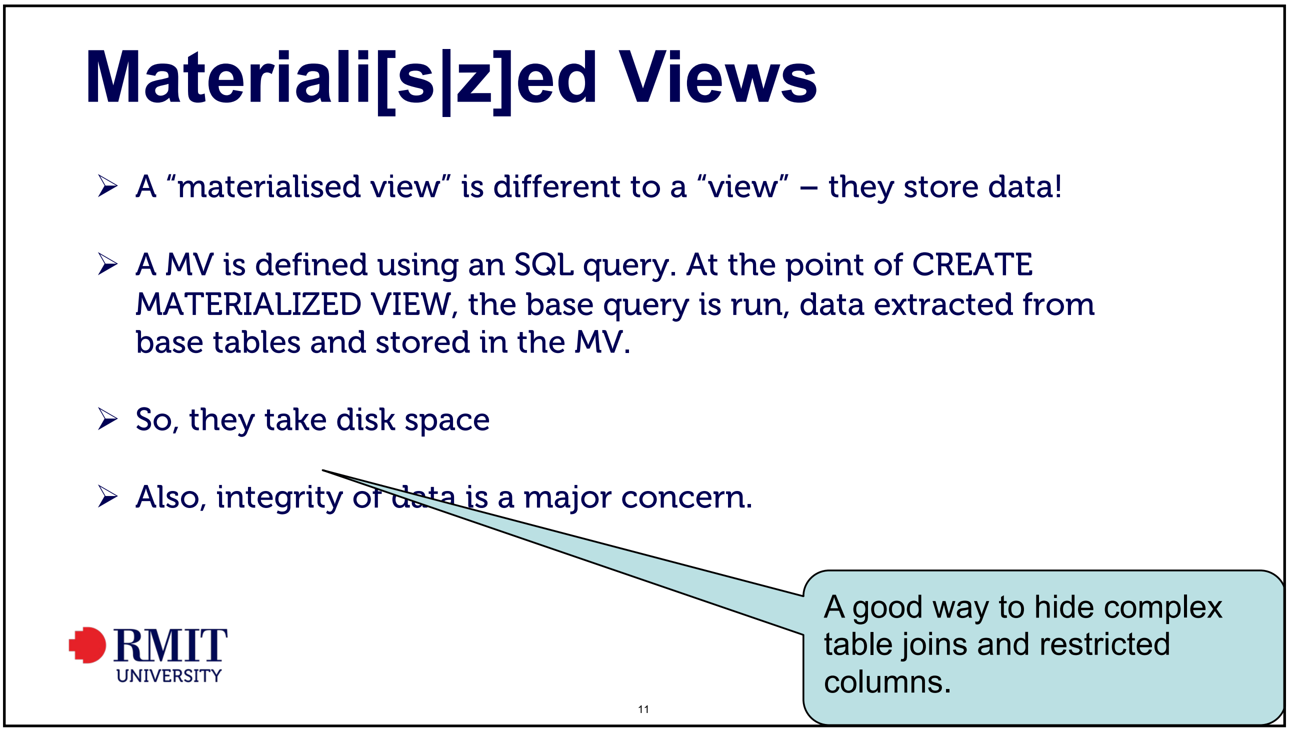
9

Materiali[s|z]ed Views

Mother of all query optimisations!

RMIT UNIVERSITY

10

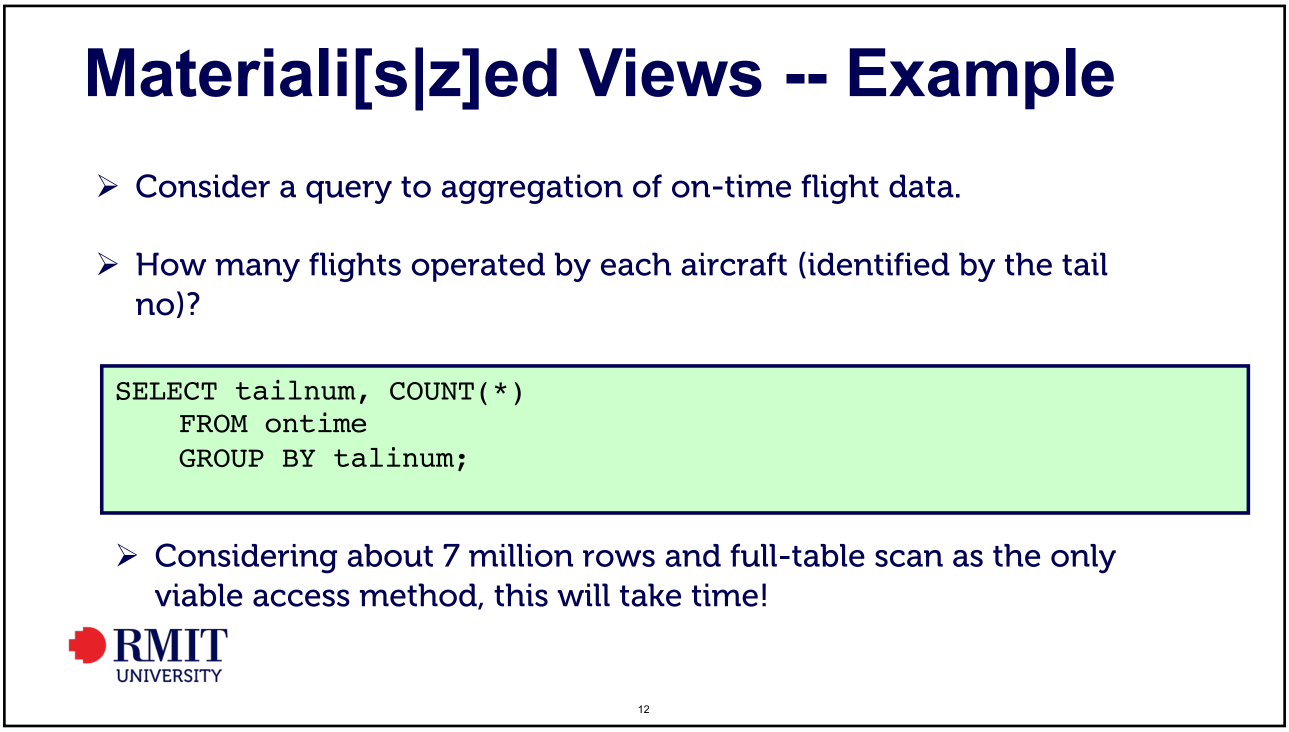


Materiali[s|z]ed Views

- A "materialised view" is different to a "view" – they store data!
- A MV is defined using an SQL query. At the point of CREATE MATERIALIZED VIEW, the base query is run, data extracted from base tables and stored in the MV.
- So, they take disk space
- Also, integrity of data is a major concern.

A good way to hide complex table joins and restricted columns.

RMIT UNIVERSITY



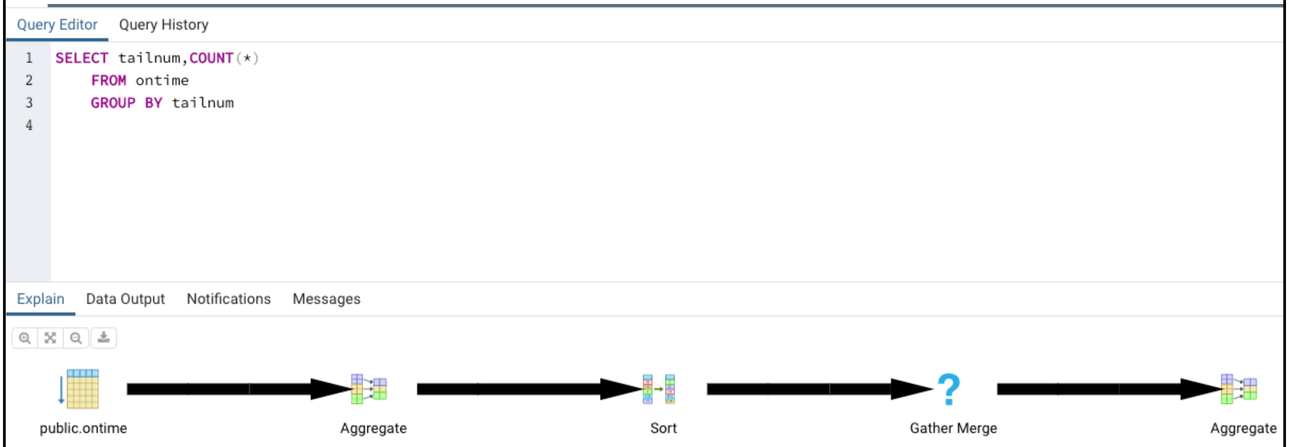
Materiali[s|z]ed Views -- Example

- Consider a query to aggregation of on-time flight data.
- How many flights operated by each aircraft (identified by the tail no)?

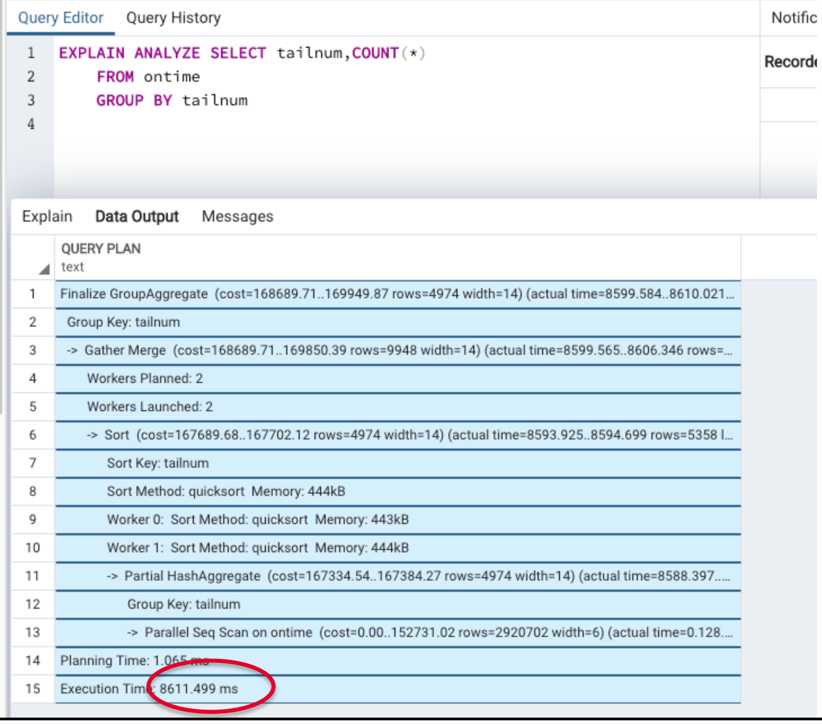
```
SELECT tailnum, COUNT(*)
    FROM ontime
    GROUP BY talinum;
```

- Considering about 7 million rows and full-table scan as the only viable access method, this will take time!

RMIT UNIVERSITY



```sql
SELECT tailnum,COUNT(*)
    FROM ontime
    GROUP BY tailnum
```



```sql
EXPLAIN ANALYZE SELECT tailnum,COUNT(*)
    FROM ontime
    GROUP BY tailnum
```

	QUERY PLAN text
1	Finalize GroupAggregate (cost=168689.71..169949.87 rows=4974 width=14) (actual time=8599.584..8610.021...
2	Group Key: tailnum
3	-> Gather Merge (cost=168689.71..169850.39 rows=9948 width=14) (actual time=8599.565..8606.346 rows=...
4	Workers Planned: 2
5	Workers Launched: 2
6	-> Sort (cost=167689.68..167702.12 rows=4974 width=14) (actual time=8593.925..8594.699 rows=5358 l...
7	Sort Key: tailnum
8	Sort Method: quicksort Memory: 444kB
9	Worker 0: Sort Method: quicksort Memory: 443kB
10	Worker 1: Sort Method: quicksort Memory: 444kB
11	-> Partial HashAggregate (cost=167334.54..167384.27 rows=4974 width=14) (actual time=8588.397.....
12	Group Key: tailnum
13	-> Parallel Seq Scan on ontime (cost=0.00..152731.02 rows=2920702 width=6) (actual time=0.128....
14	Planning Time: 1.065 ms
15	Execution Time: 8611.499 ms

Materiali[s|z]ed Views -- Example

- Let's suppose we already have a pre-built materialized view – `ontime_mv`
- `ontime_mv` was built using the same SQL query.

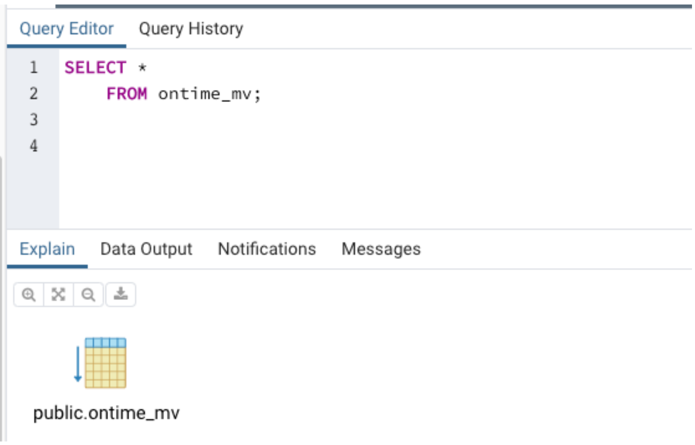
```
SELECT *
    FROM ontime_mv;
```

- Now look at query performance.

RMIT UNIVERSITY

15

15

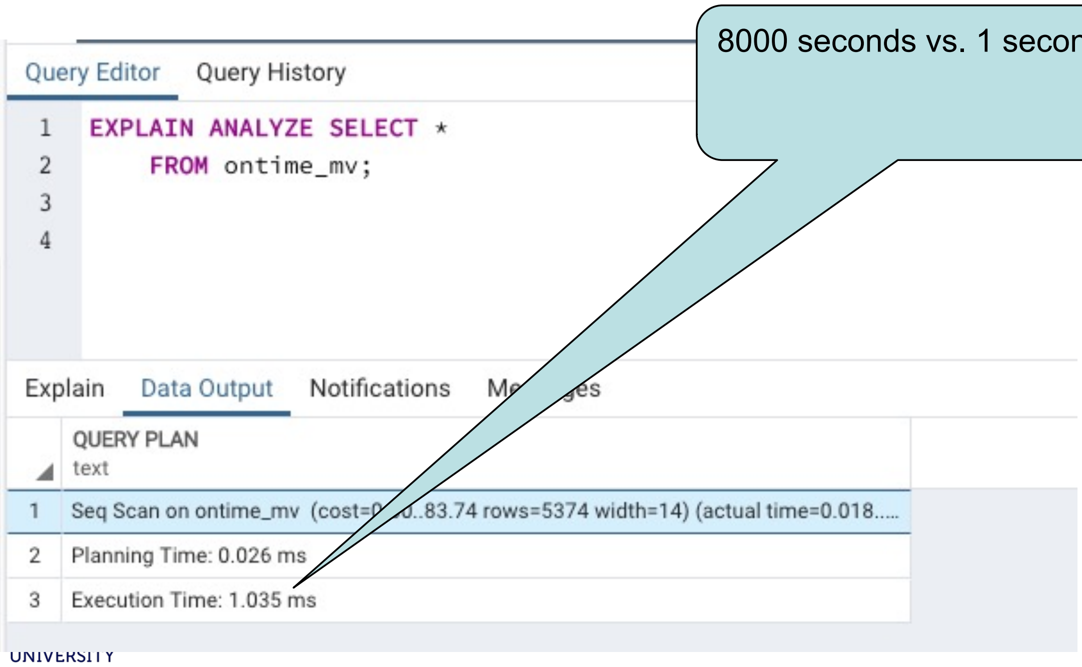


```
SELECT *
    FROM ontime_mv;
```

RMIT UNIVERSITY

16

16



```
EXPLAIN ANALYZE SELECT *
    FROM ontime_mv;
```

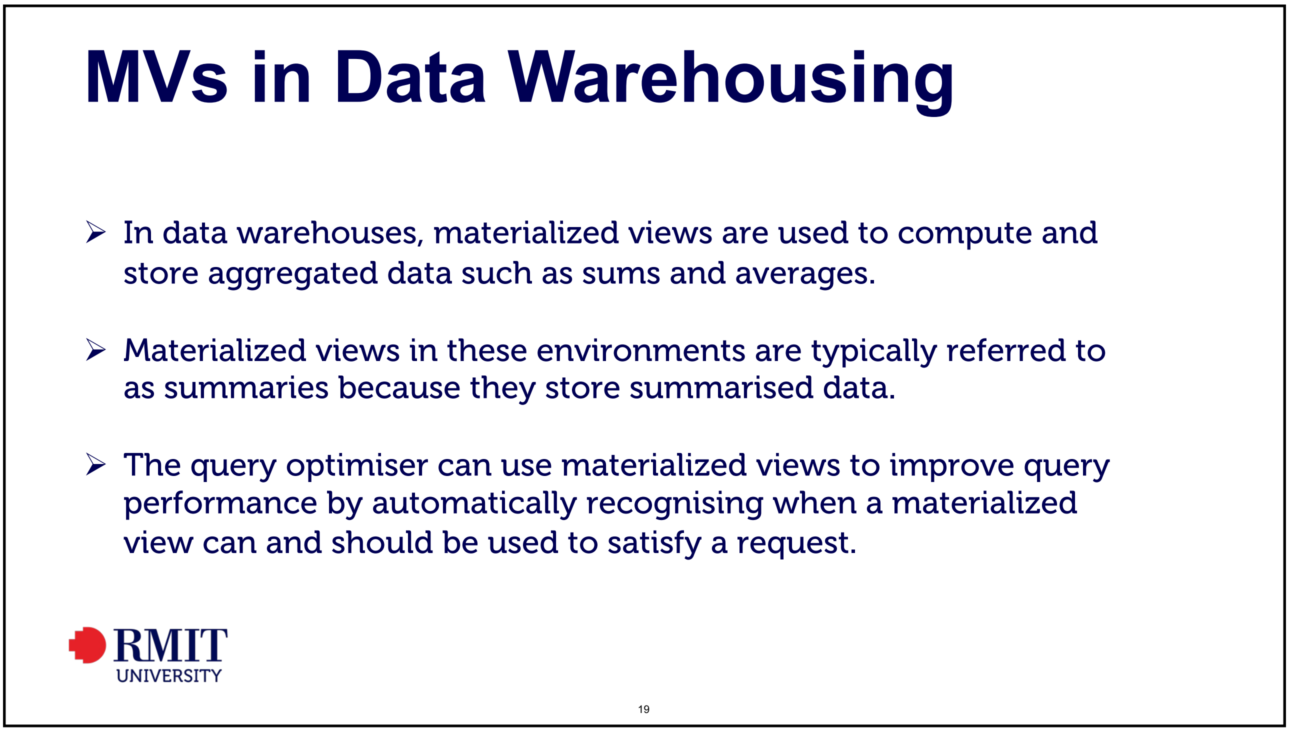
```
  QUERY PLAN
  text
1 Seq Scan on ontime_mv  (cost=0.00..83.74 rows=5374 width=14) (actual time=0.018.....
2 Planning Time: 0.026 ms
3 Execution Time: 1.035 ms
```

8000 seconds vs. 1 second!

Materiali[s|z]ed Views -- Example

- So enticing -- how `ontime_mv` was built?
- `ontime_mv` was built using the same SQL query.

```
CREATE MATERIALIZED VIEW ontime_mv
AS
    SELECT tailnum, COUNT(*)
        FROM ontime
        GROUP BY talinum;
```



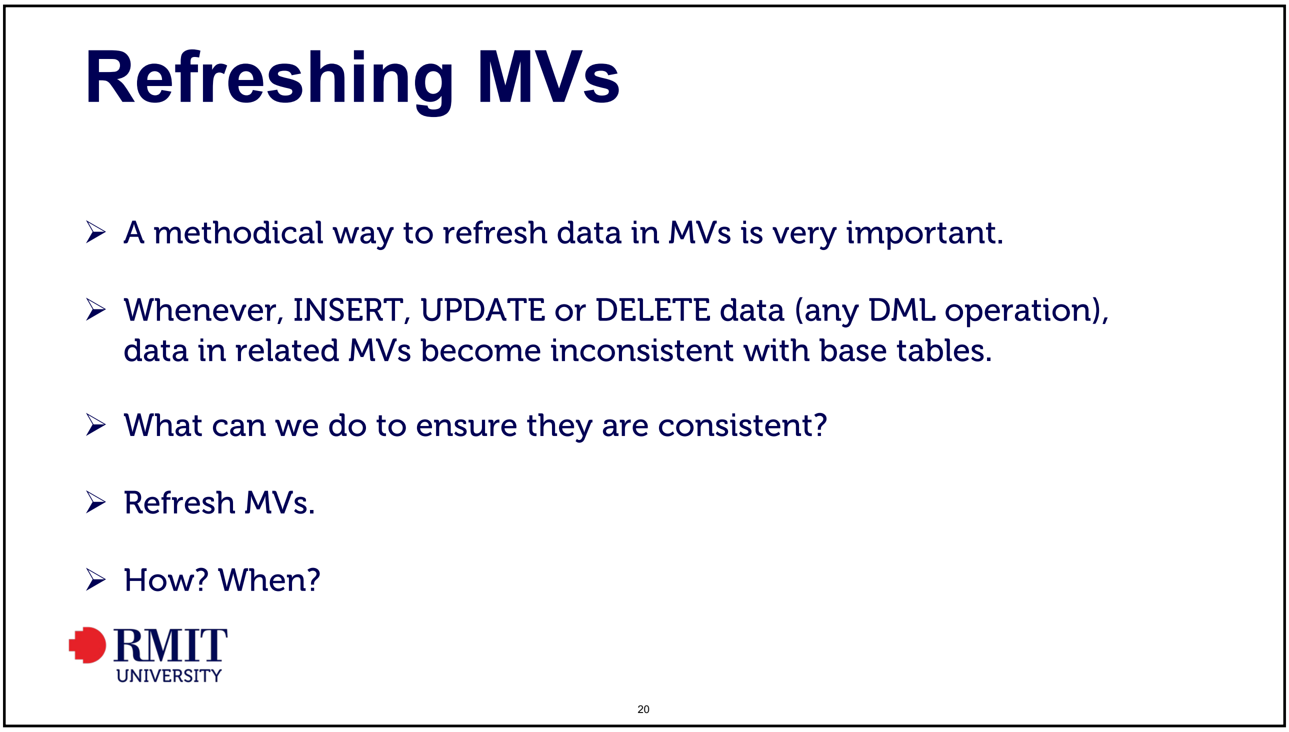
MVs in Data Warehousing

- In data warehouses, materialized views are used to compute and store aggregated data such as sums and averages.
- Materialized views in these environments are typically referred to as summaries because they store summarised data.
- The query optimiser can use materialized views to improve query performance by automatically recognising when a materialized view can and should be used to satisfy a request.

RMIT UNIVERSITY

19

19



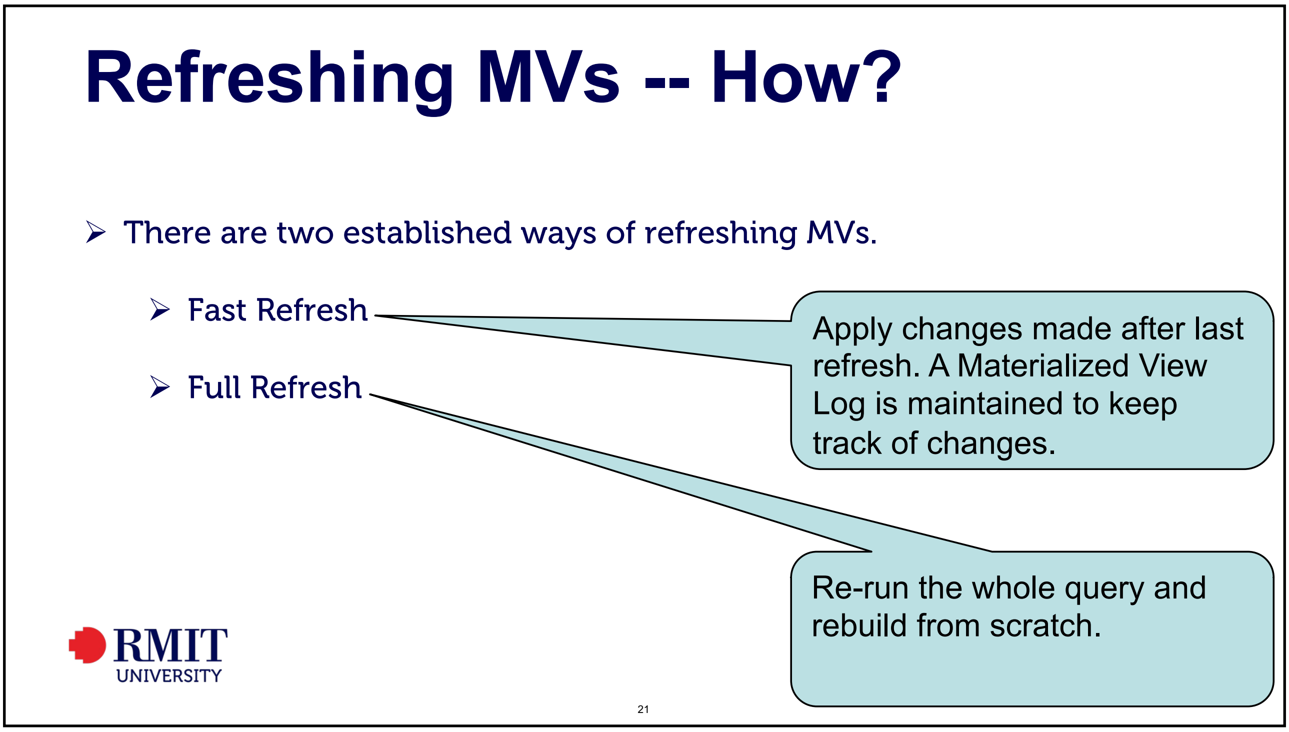
Refreshing MVs

- A methodical way to refresh data in MVs is very important.
- Whenever, INSERT, UPDATE or DELETE data (any DML operation), data in related MVs become inconsistent with base tables.
- What can we do to ensure they are consistent?
- Refresh MVs.
- How? When?

RMIT UNIVERSITY

20

20



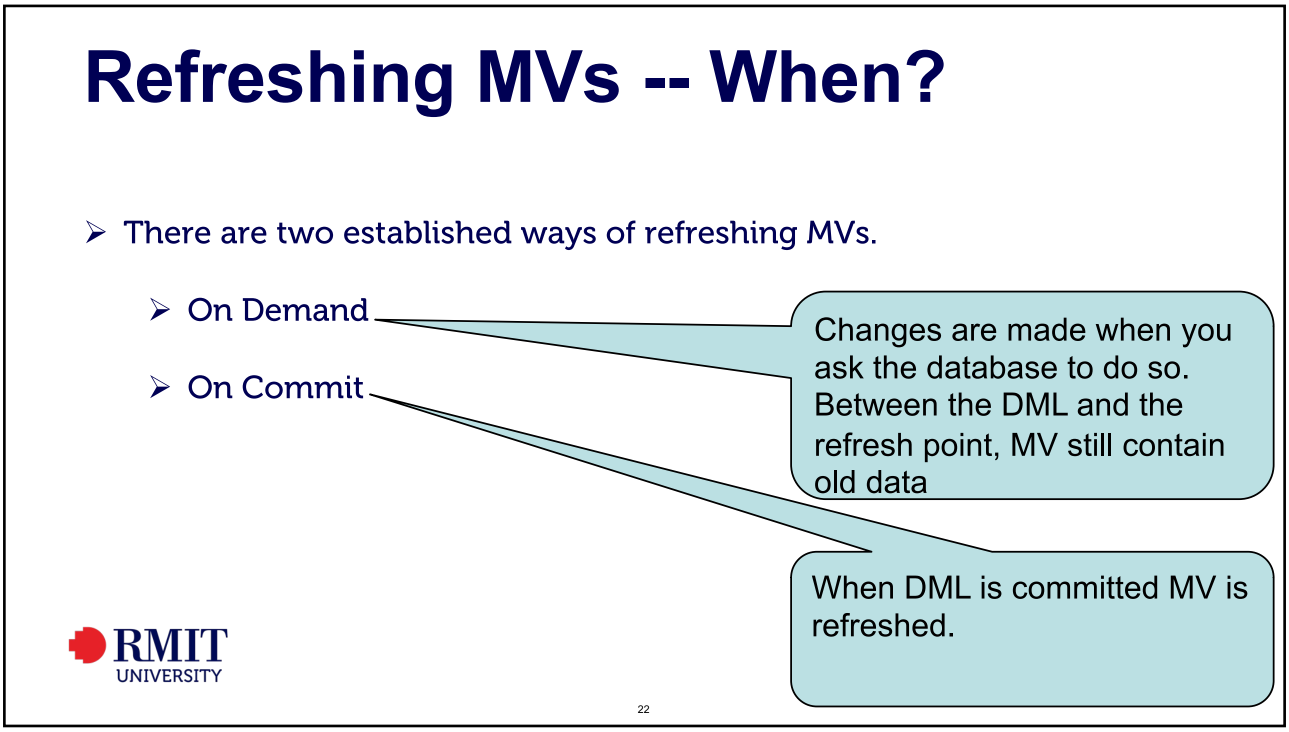
Refreshing MVs -- How?

- There are two established ways of refreshing MVs.
  - Fast Refresh
  - Full Refresh

Apply changes made after last refresh. A Materialized View Log is maintained to keep track of changes.

Re-run the whole query and rebuild from scratch.

21



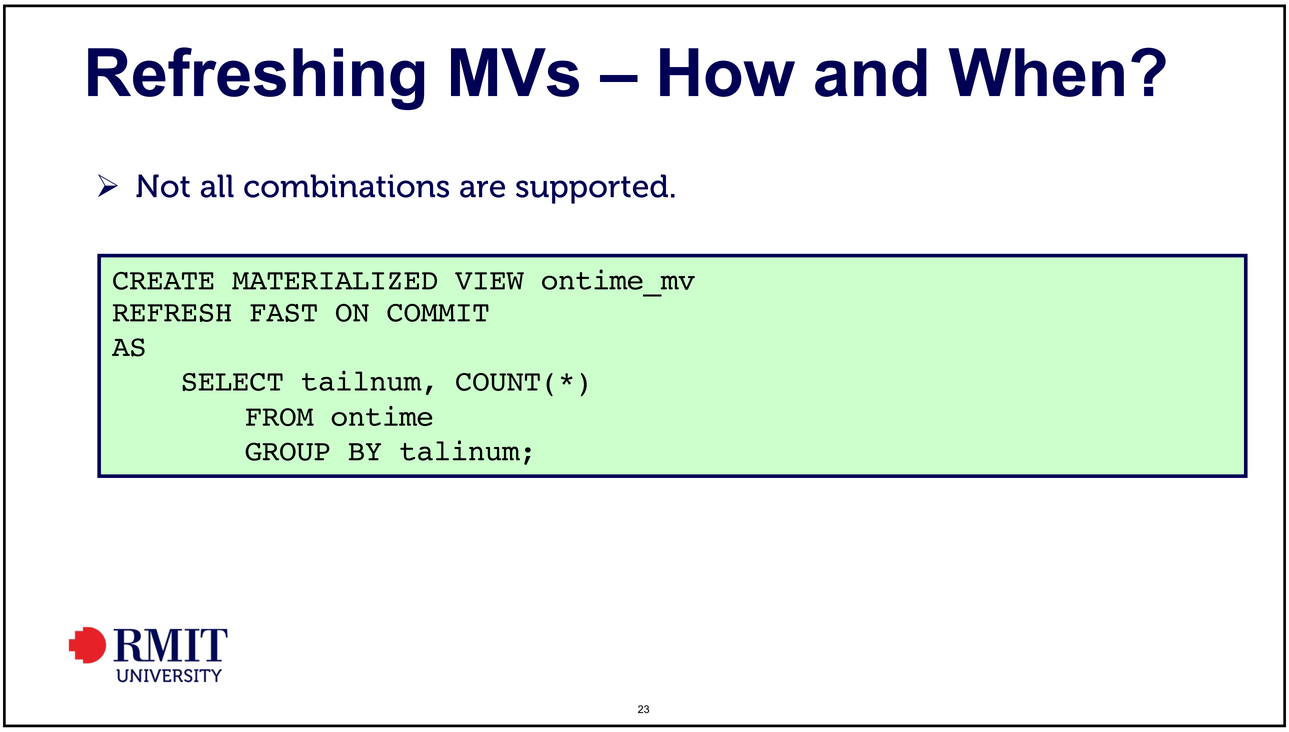
Refreshing MVs -- When?

- There are two established ways of refreshing MVs.
  - On Demand
  - On Commit

Changes are made when you ask the database to do so. Between the DML and the refresh point, MV still contain old data

When DML is committed MV is refreshed.

22



Refreshing MVs – How and When?

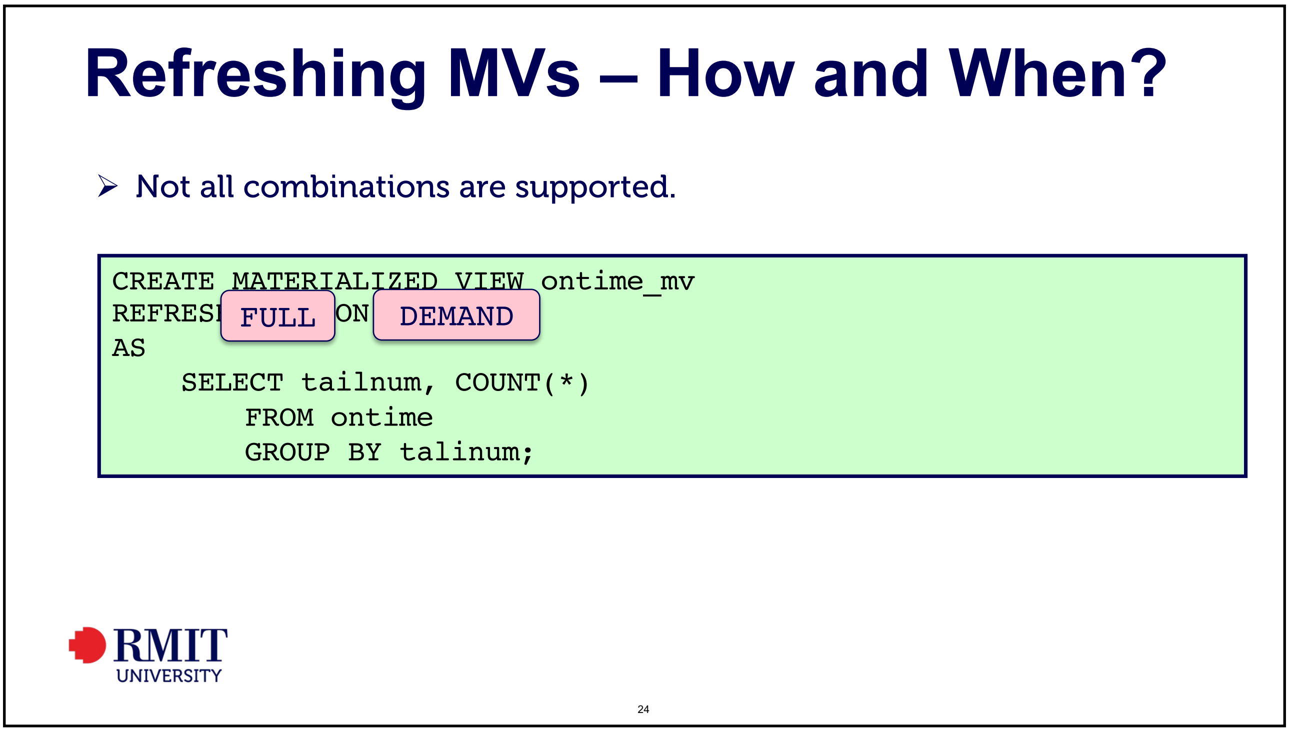
- ➢ Not all combinations are supported.

```
CREATE MATERIALIZED VIEW ontime_mv
REFRESH FAST ON COMMIT
AS
    SELECT tailnum, COUNT(*)
        FROM ontime
        GROUP BY talinum;
```

RMIT UNIVERSITY

23

23



Refreshing MVs – How and When?

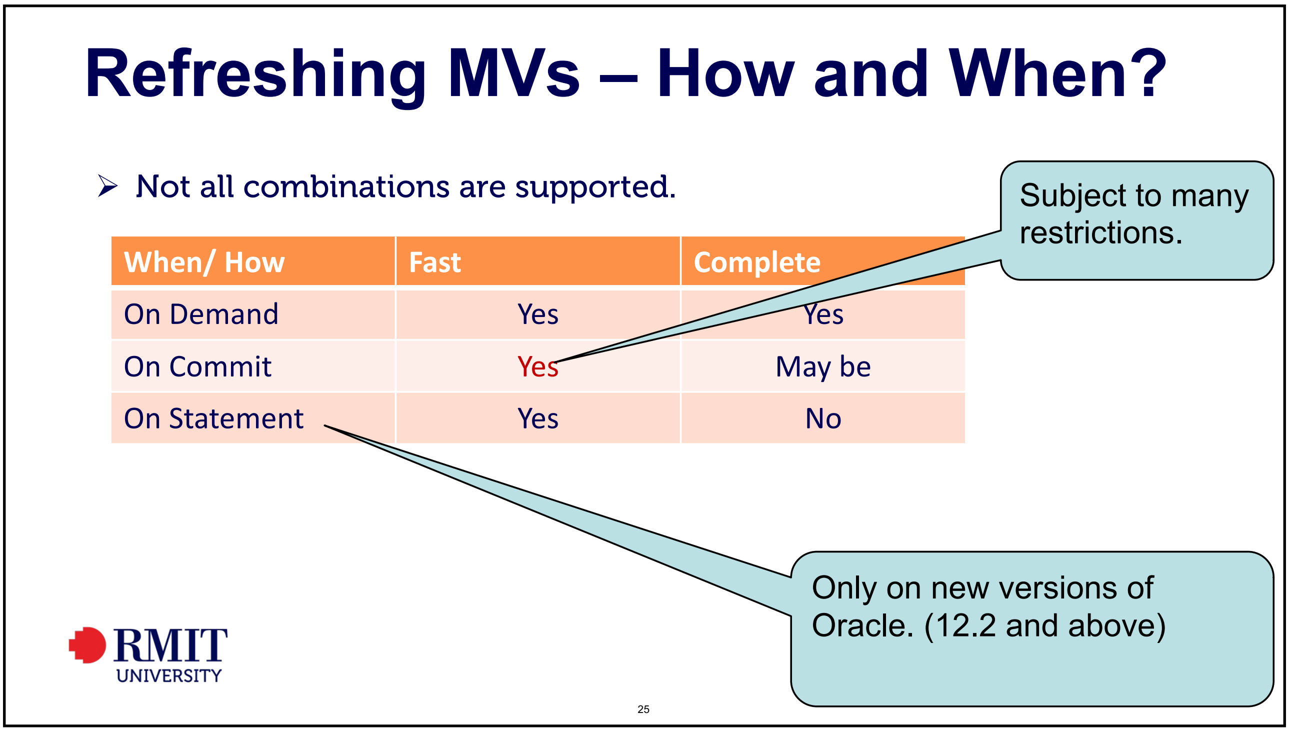
- ➢ Not all combinations are supported.

```
CREATE MATERIALIZED VIEW ontime_mv
REFRESH FULL ON DEMAND
AS
    SELECT tailnum, COUNT(*)
        FROM ontime
        GROUP BY talinum;
```

RMIT UNIVERSITY

24

24



Refreshing MVs – How and When?

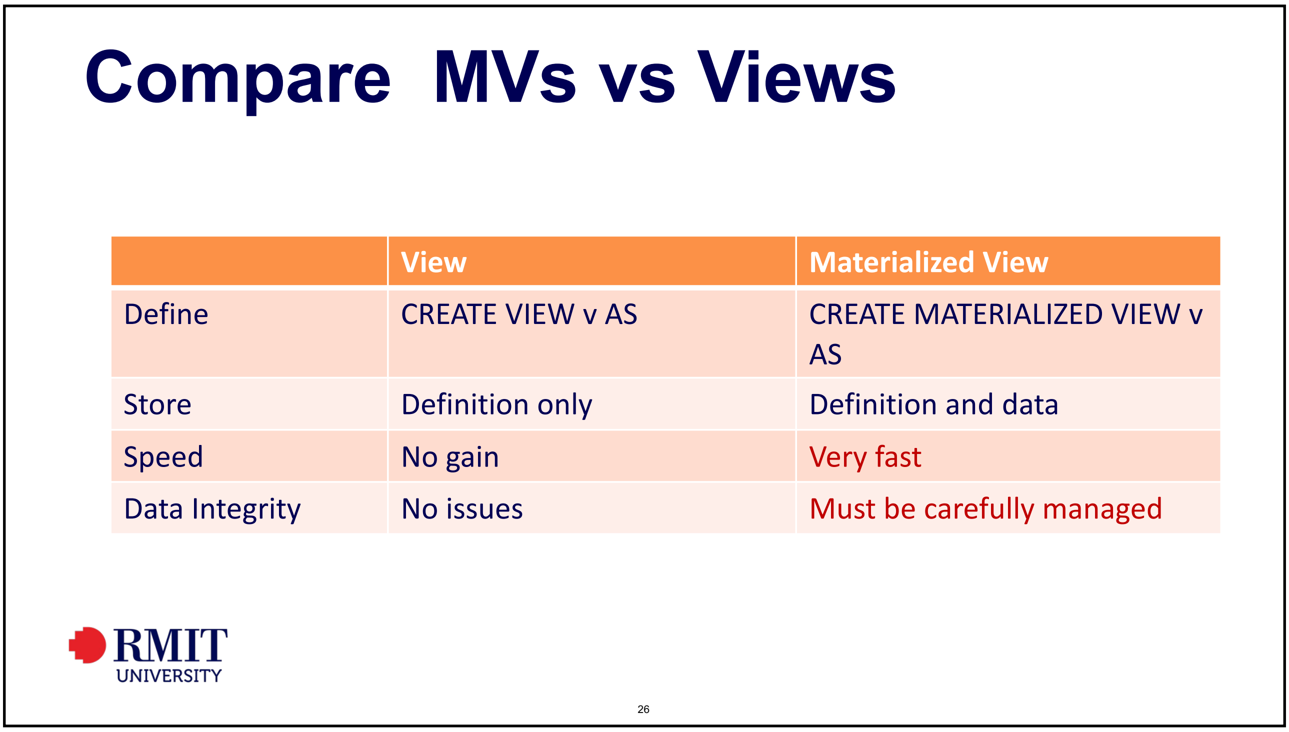
- Not all combinations are supported.

When/ How	Fast	Complete
On Demand	Yes	Yes
On Commit	Yes	May be
On Statement	Yes	No

Subject to many restrictions.

Only on new versions of Oracle. (12.2 and above)

RMIT UNIVERSITY



Compare MVs vs Views

	View	Materialized View
Define	CREATE VIEW v AS	CREATE MATERIALIZED VIEW v AS
Store	Definition only	Definition and data
Speed	No gain	Very fast
Data Integrity	No issues	Must be carefully managed

RMIT UNIVERSITY

Triggers

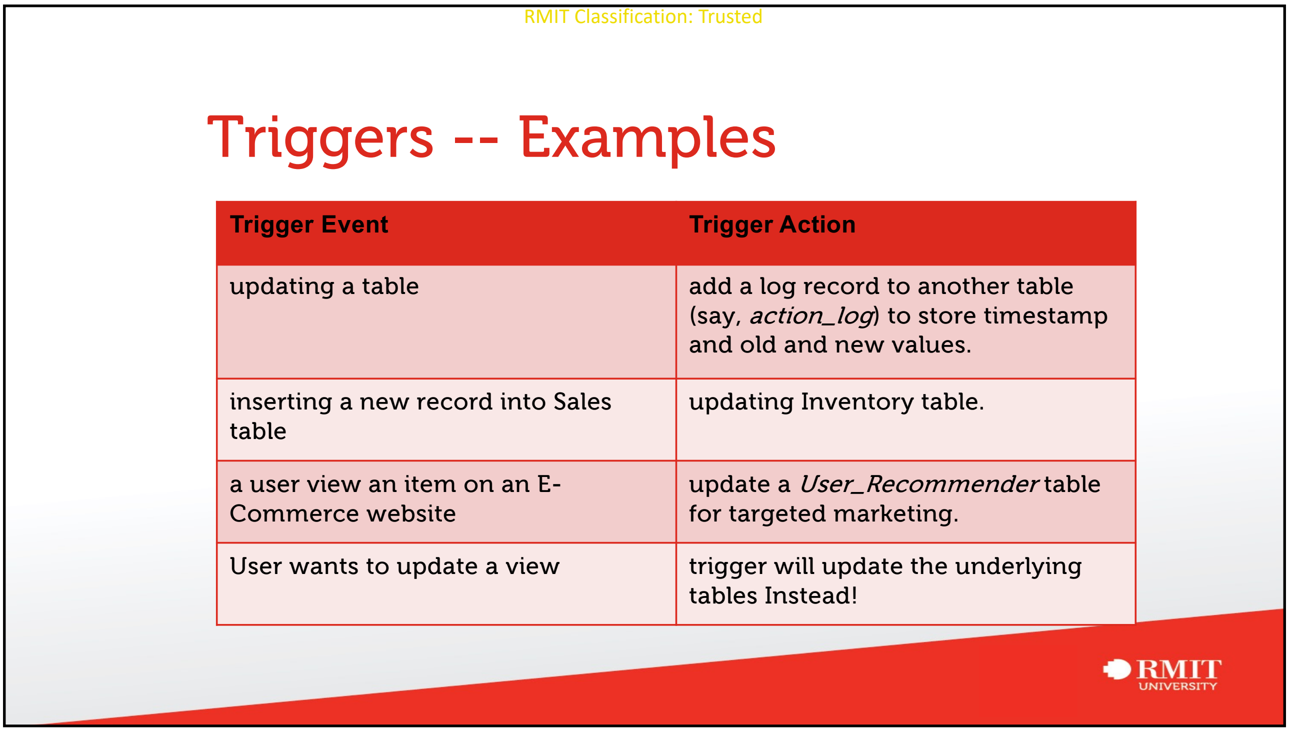
Sorry, not about “Tigger”😀!

27

Triggers

- Triggers are operations that are automatically performed when a specified database event occurs.
- A trigger event can be a changing data action -- INSERT, UPDATE, or DELETE.
- A trigger action can be another action (SELECT, INSERT, UPDATE, DELETE) action.

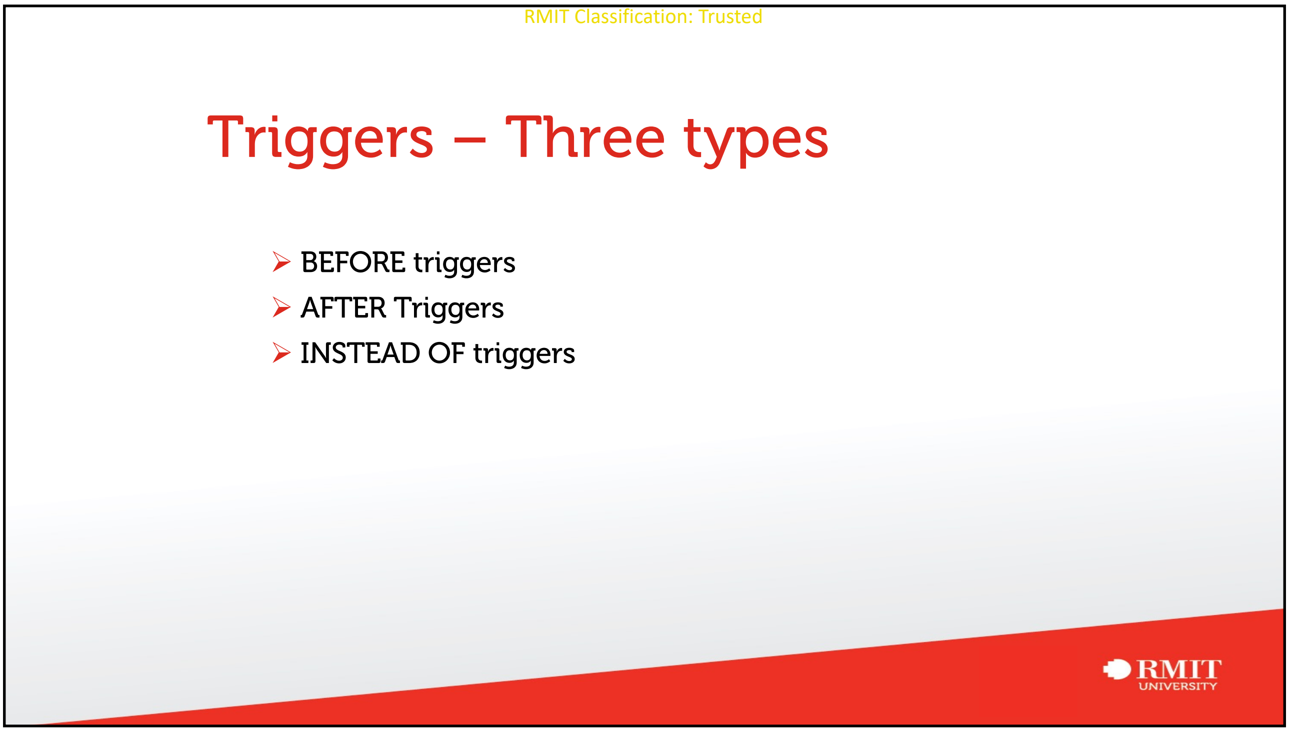
28



Triggers -- Examples

Trigger Event	Trigger Action
updating a table	add a log record to another table (say, *action_log*) to store timestamp and old and new values.
inserting a new record into Sales table	updating Inventory table.
a user view an item on an E-Commerce website	update a *User_Recommender* table for targeted marketing.
User wants to update a view	trigger will update the underlying tables Instead!

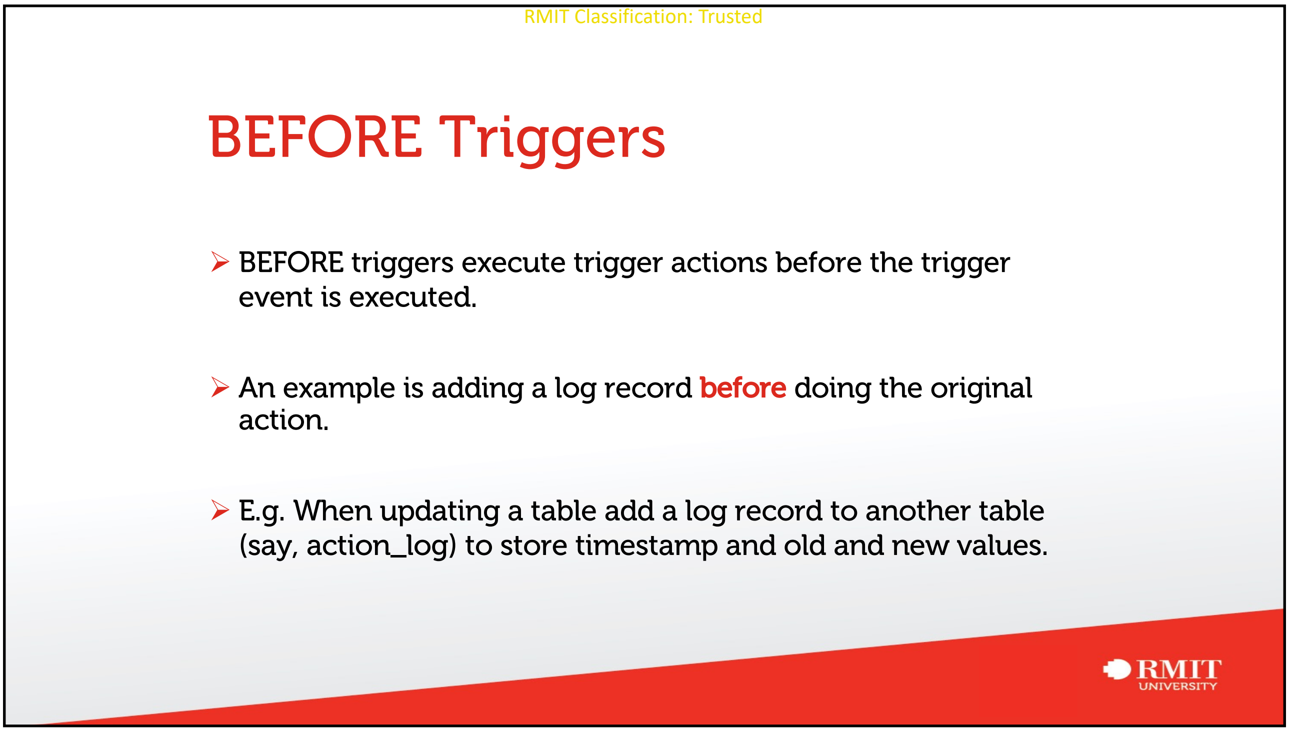
29



Triggers – Three types

- BEFORE triggers
- AFTER Triggers
- INSTEAD OF triggers

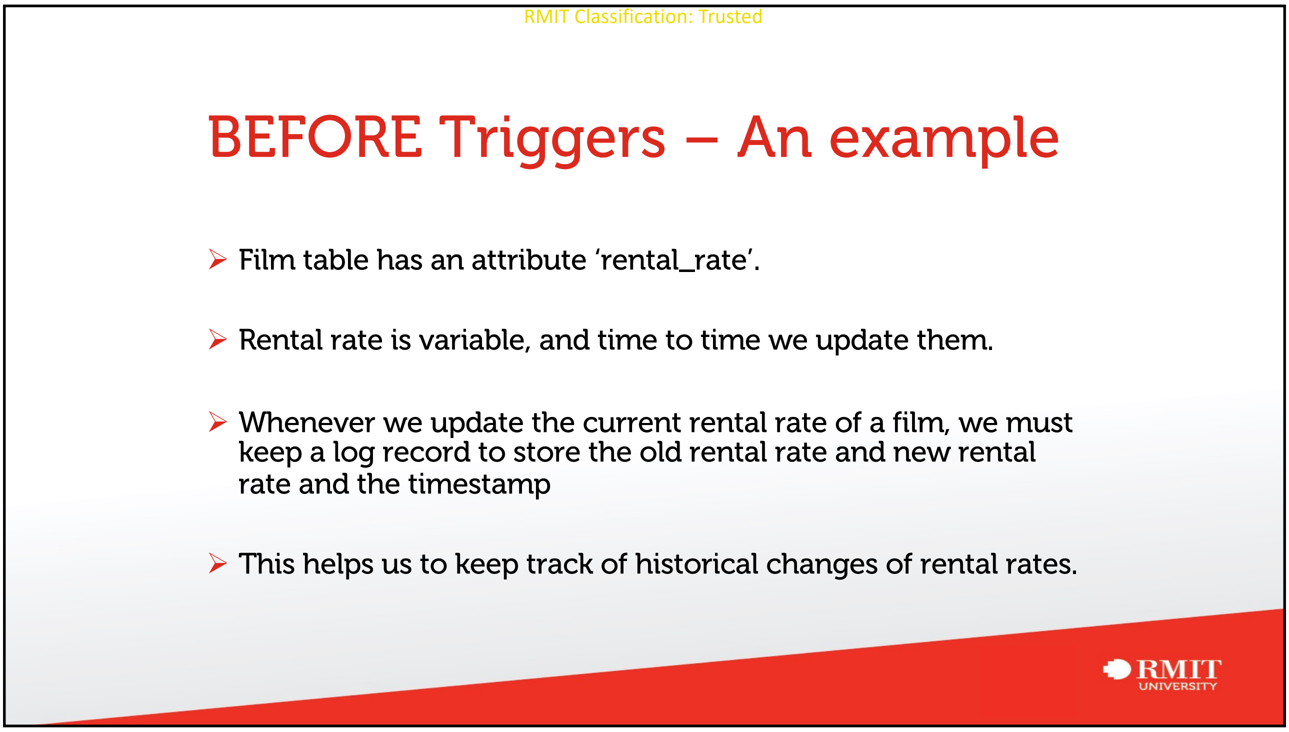
30



BEFORE Triggers

- ➢ BEFORE triggers execute trigger actions before the trigger event is executed.
- ➢ An example is adding a log record **before** doing the original action.
- ➢ E.g. When updating a table add a log record to another table (say, action_log) to store timestamp and old and new values.

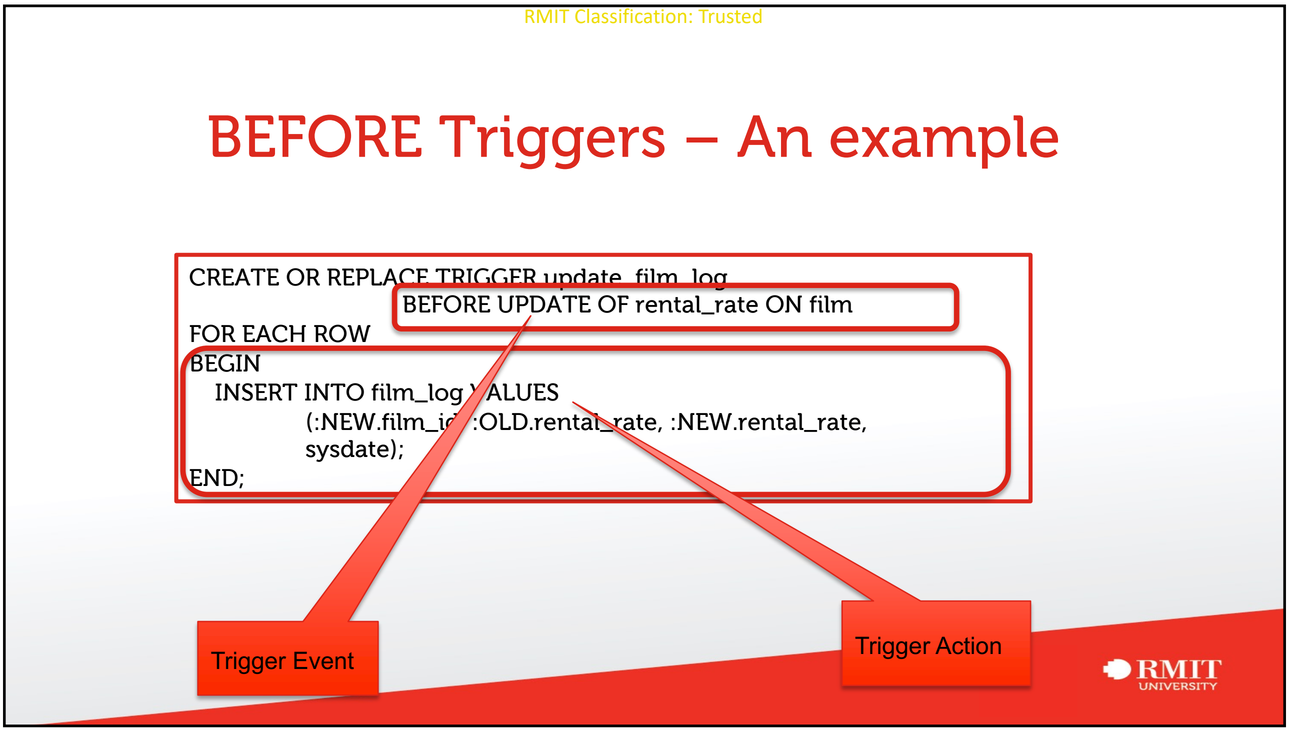
31



BEFORE Triggers – An example

- ➢ Film table has an attribute ‘rental_rate’.
- ➢ Rental rate is variable, and time to time we update them.
- ➢ Whenever we update the current rental rate of a film, we must keep a log record to store the old rental rate and new rental rate and the timestamp
- ➢ This helps us to keep track of historical changes of rental rates.

32



RMIT Classification: Trusted

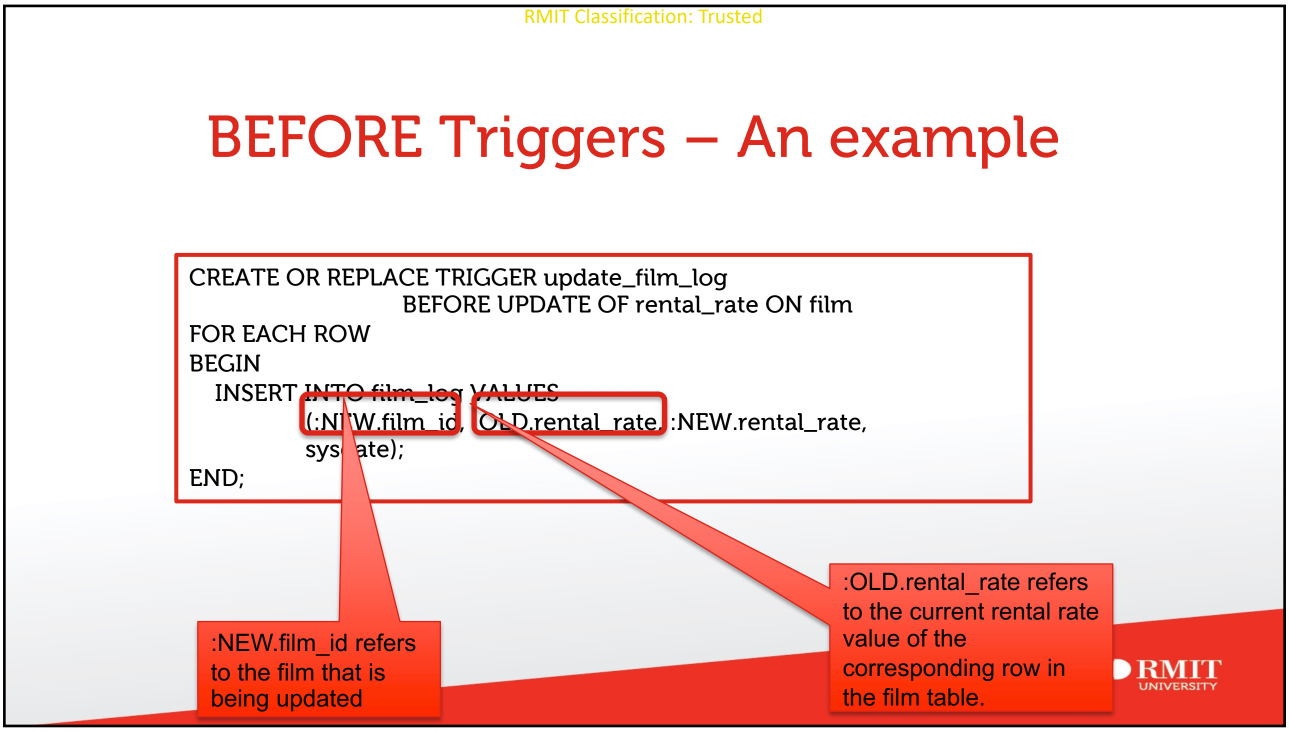
BEFORE Triggers – An example

```
CREATE OR REPLACE TRIGGER update_film_log
                BEFORE UPDATE OF rental_rate ON film
FOR EACH ROW
BEGIN
   INSERT INTO film_log VALUES
             (:NEW.film_id, :OLD.rental_rate, :NEW.rental_rate,
             sysdate);
END;
```

Trigger Event

Trigger Action

33



RMIT Classification: Trusted

BEFORE Triggers – An example

```
CREATE OR REPLACE TRIGGER update_film_log
                BEFORE UPDATE OF rental_rate ON film
FOR EACH ROW
BEGIN
   INSERT INTO film_log VALUES
             (:NEW.film_id, :OLD.rental_rate, :NEW.rental_rate,
             sysdate);
END;
```

:NEW.film_id refers to the film that is being updated

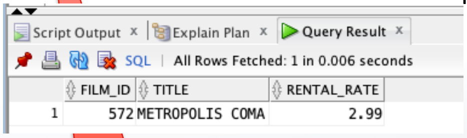
:OLD.rental_rate refers to the current rental rate value of the corresponding row in the film table.

34

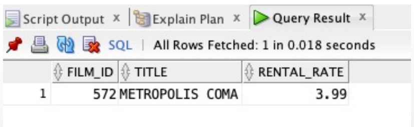
BEFORE Triggers – An example

```
UPDATE film SET
  rental_rate = 3.99
  WHERE LOWER(title) = 'metropolis coma'
```

film

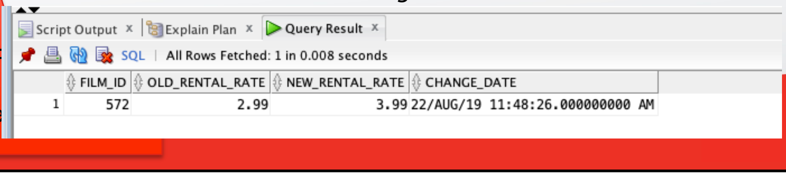


```
     FILM_ID  TITLE             RENTAL_RATE
1        572  METROPOLIS COMA          2.99
```



```
     FILM_ID  TITLE             RENTAL_RATE
1        572  METROPOLIS COMA          3.99
```

film_Log



```
   FILM_ID  OLD_RENTAL_RATE  NEW_RENTAL_RATE  CHANGE_DATE
1      572             2.99             3.99  22/AUG/19 11:48:26.000000000 AM
```

This UP
fire the
"update
trigger

AFTER Triggers

- AFTER triggers execute trigger actions after the trigger event is executed.
- An example is display before and after values **after** doing and update action.
- Operation of the AFTER trigger is very similar to BEFORE trigger, only the timing is different.

INSTEAD OF Triggers

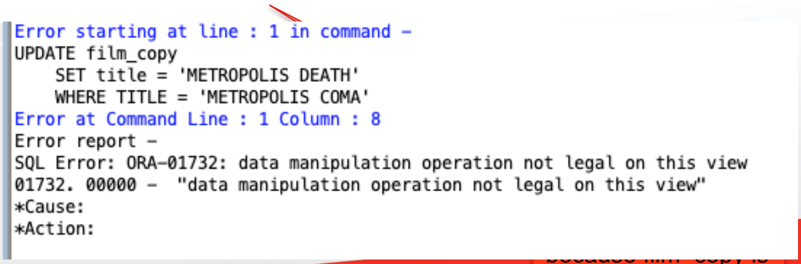
- INSTEAD OF triggers are mainly used for updating VIEWs.
- When a user asks for updating a view, normally it is not permitted. However, you can use a trigger to update the underlying tables instead.

37

INSTEAD OF Triggers – An example

- A user wants to update the name of a continent.

```
UPDATE film_copy
   SET title = 'METROPOLIS DEATH'
   WHERE TITLE = 'METROPOLIS COMA';
```



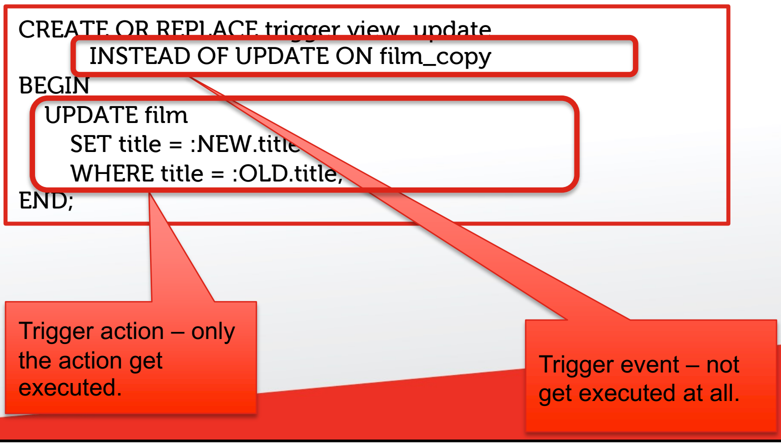
```
Error starting at line : 1 in command -
UPDATE film_copy
    SET title = 'METROPOLIS DEATH'
    WHERE TITLE = 'METROPOLIS COMA'
Error at Command Line : 1 Column : 8
Error report -
SQL Error: ORA-01732: data manipulation operation not legal on this view
01732. 00000 -  "data manipulation operation not legal on this view"
*Cause:
*Action:
```

a view.

38

INSTEAD OF Triggers – An example

- Define an INSTEAD OF trigger to do the job.



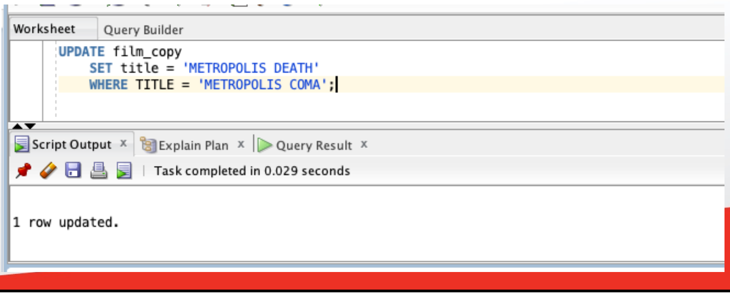
```
CREATE OR REPLACE trigger view_update
        INSTEAD OF UPDATE ON film_copy
BEGIN
   UPDATE film
      SET title = :NEW.title
      WHERE title = :OLD.title;
END;
```

Trigger action – only the action get executed.

Trigger event – not get executed at all.

INSTEAD OF Triggers – An example

```
CREATE OR REPLACE trigger view_update
        INSTEAD OF UPDATE ON film_copy
BEGIN
   UPDATE film
      SET title = :NEW.title
      WHERE title = :OLD.title;
END
```

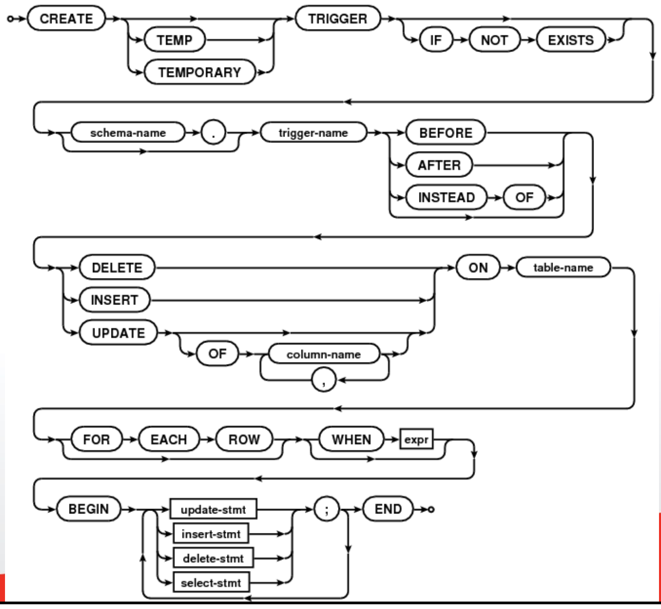


```
UPDATE film_copy
    SET title = 'METROPOLIS DEATH'
    WHERE TITLE = 'METROPOLIS COMA';
```

Task completed in 0.029 seconds

```
1 row updated.
```

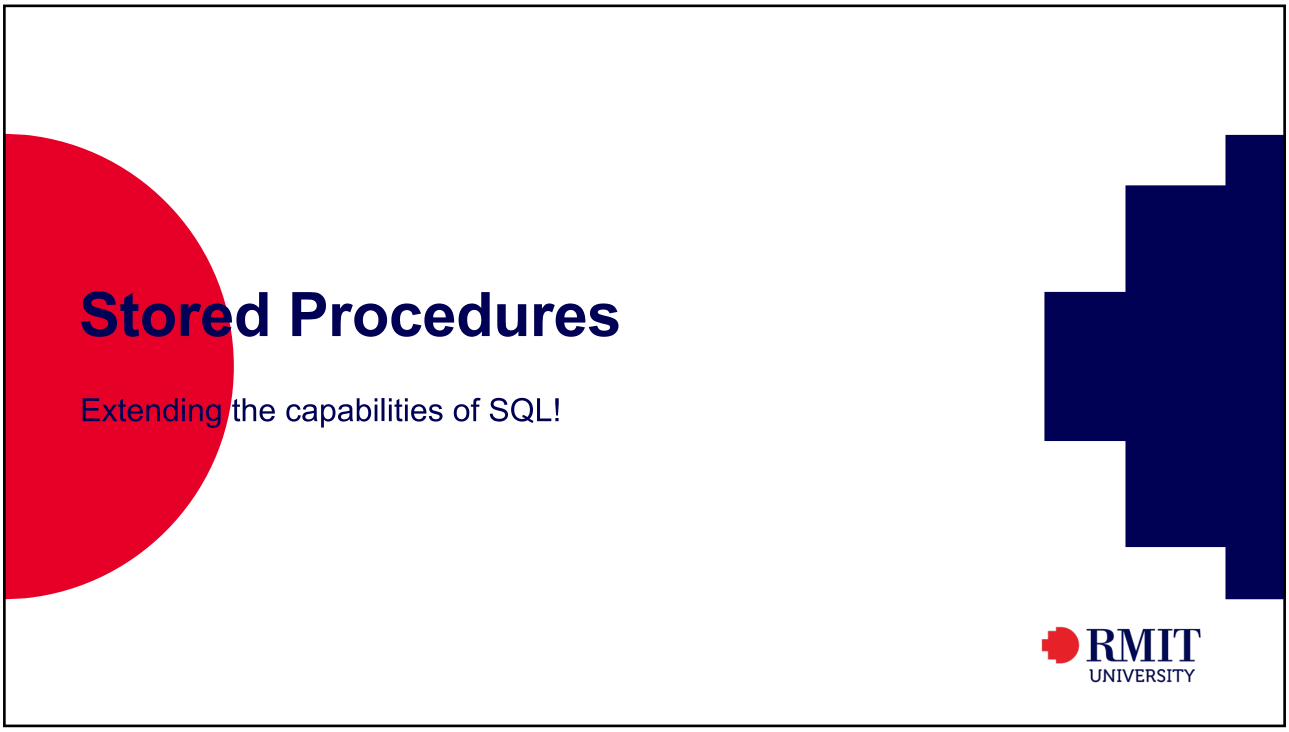
CREATE TRIGGER syntax



About use of OLD and NEW

- When executing a trigger, the database engine allows you to refer old and new rows the trigger event is dealing with.
- They can be used within trigger action.
- Depend on the trigger event, we may use either of them or both.

Trigger Event	Available references
INSERT	:NEW
UPDATE	:NEW, :OLD
DELETE	:OLD

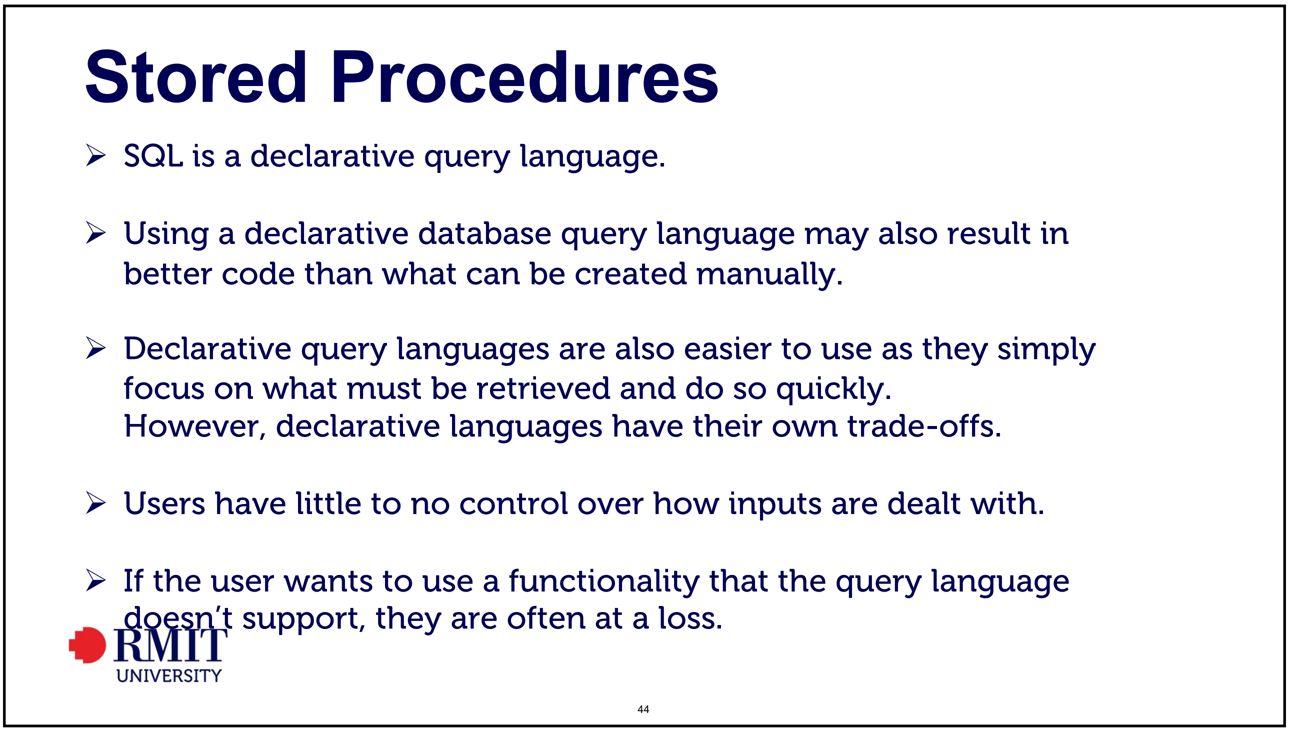


Stored Procedures

Extending the capabilities of SQL!

RMIT UNIVERSITY

43



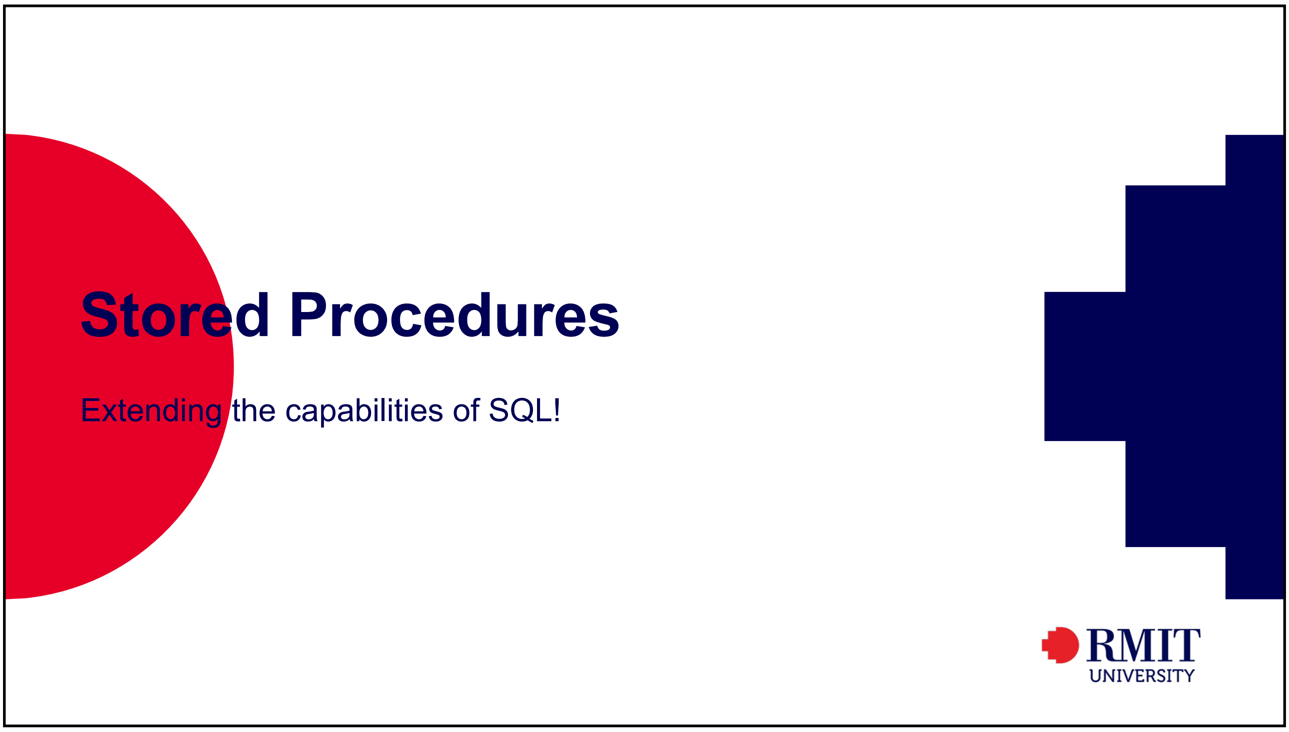
Stored Procedures

- ➢ SQL is a declarative query language.
- ➢ Using a declarative database query language may also result in better code than what can be created manually.
- ➢ Declarative query languages are also easier to use as they simply focus on what must be retrieved and do so quickly. However, declarative languages have their own trade-offs.
- ➢ Users have little to no control over how inputs are dealt with.
- ➢ If the user wants to use a functionality that the query language doesn't support, they are often at a loss.

RMIT UNIVERSITY

44

44

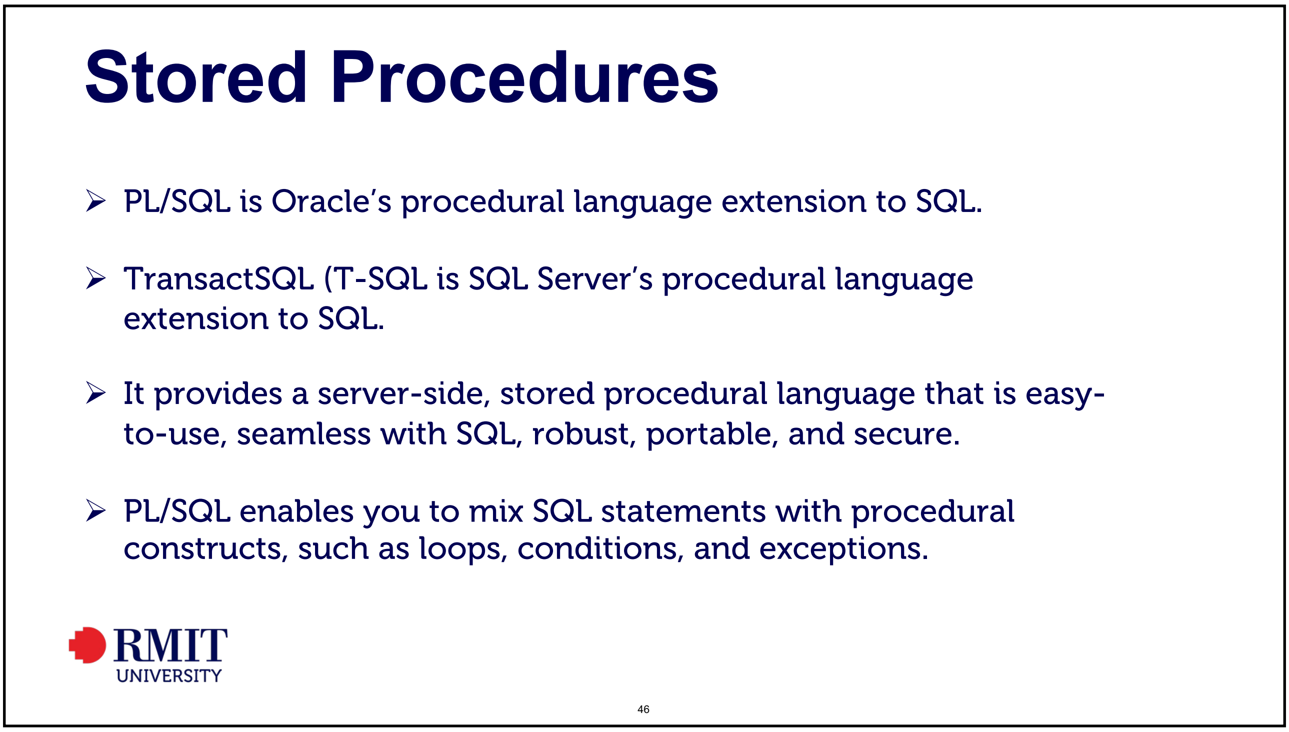


Stored Procedures

Extending the capabilities of SQL!

RMIT UNIVERSITY

45



Stored Procedures

- PL/SQL is Oracle's procedural language extension to SQL.
- TransactSQL (T-SQL is SQL Server's procedural language extension to SQL.
- It provides a server-side, stored procedural language that is easy-to-use, seamless with SQL, robust, portable, and secure.
- PL/SQL enables you to mix SQL statements with procedural constructs, such as loops, conditions, and exceptions.

RMIT UNIVERSITY

46

```
SET SERVEROUTPUT ON
    DECLARE
        CURSOR movie_data IS
        SELECT mvtitle,dirname
            FROM movie, director
            WHERE movie.dirnumb = director.dirnumb;

        director_name director.dirname%type;
        movie_name movie.mvtitle%type;
    BEGIN
        OPEN movie_data;
        LOOP
            FETCH movie_data INTO movie_name, director_name;
            EXIT WHEN movie_data%NOTFOUND;

            DBMS_OUTPUT.put_line(movie_name || '   ' ||
director_name);
        END LOOP;
        CLOSE movie_data;
    END;
/
```

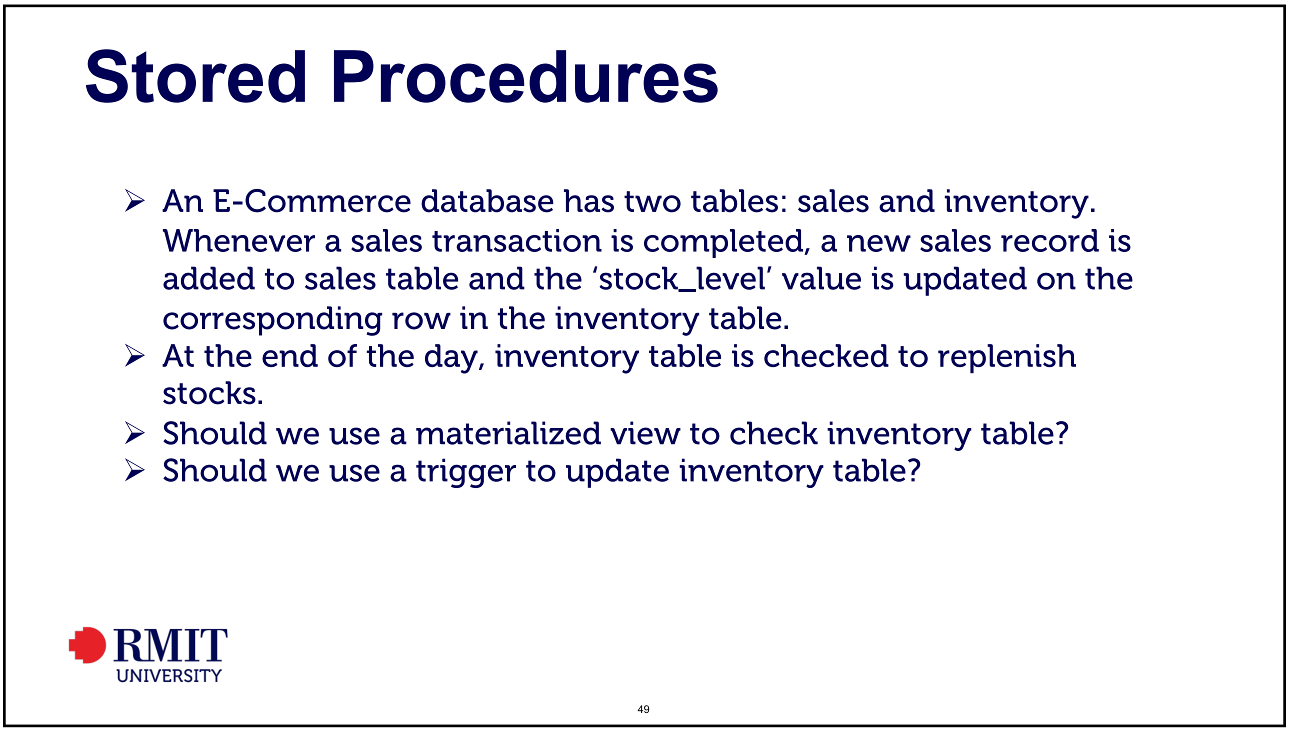
47

Stored Procedures

- ➢ We will have a hands-on PL/SQL discussion next week.

RMIT UNIVERSITY

48

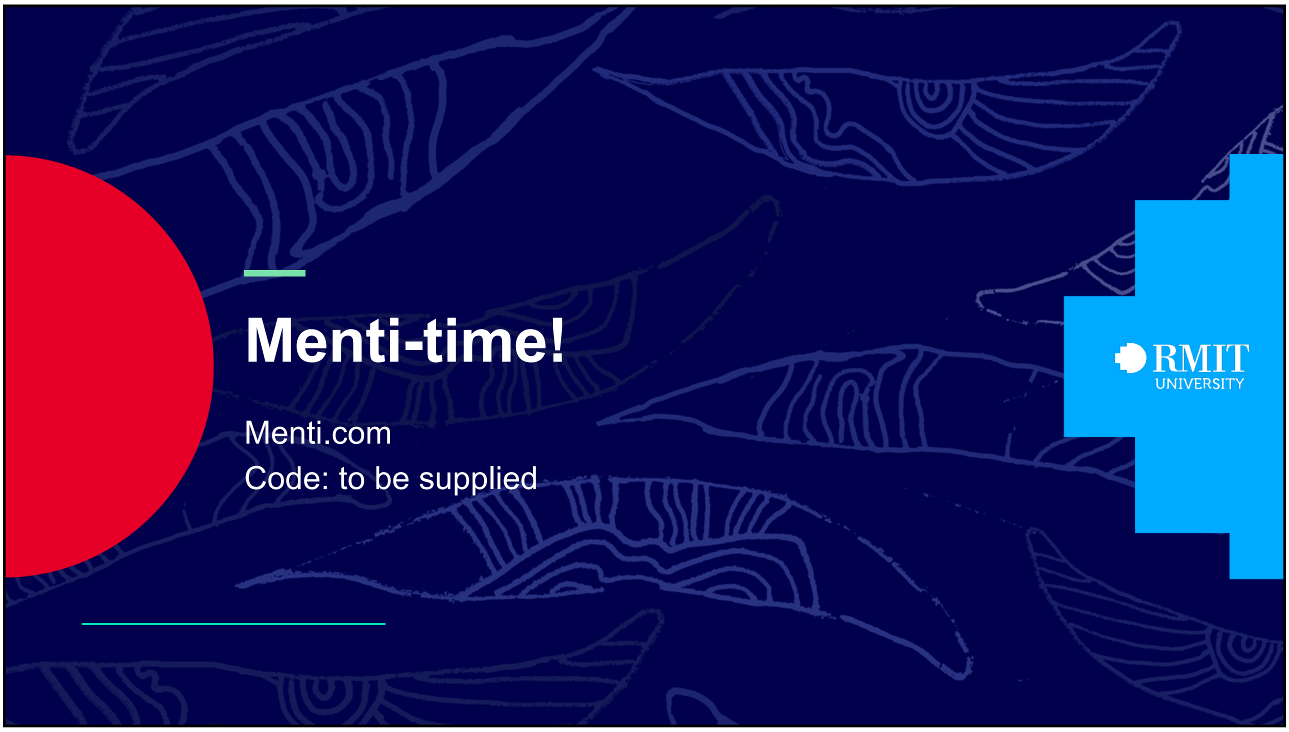


Stored Procedures

- ➢ An E-Commerce database has two tables: sales and inventory. Whenever a sales transaction is completed, a new sales record is added to sales table and the ‘stock_level’ value is updated on the corresponding row in the inventory table.
- ➢ At the end of the day, inventory table is checked to replenish stocks.
- ➢ Should we use a materialized view to check inventory table?
- ➢ Should we use a trigger to update inventory table?

RMIT UNIVERSITY

49

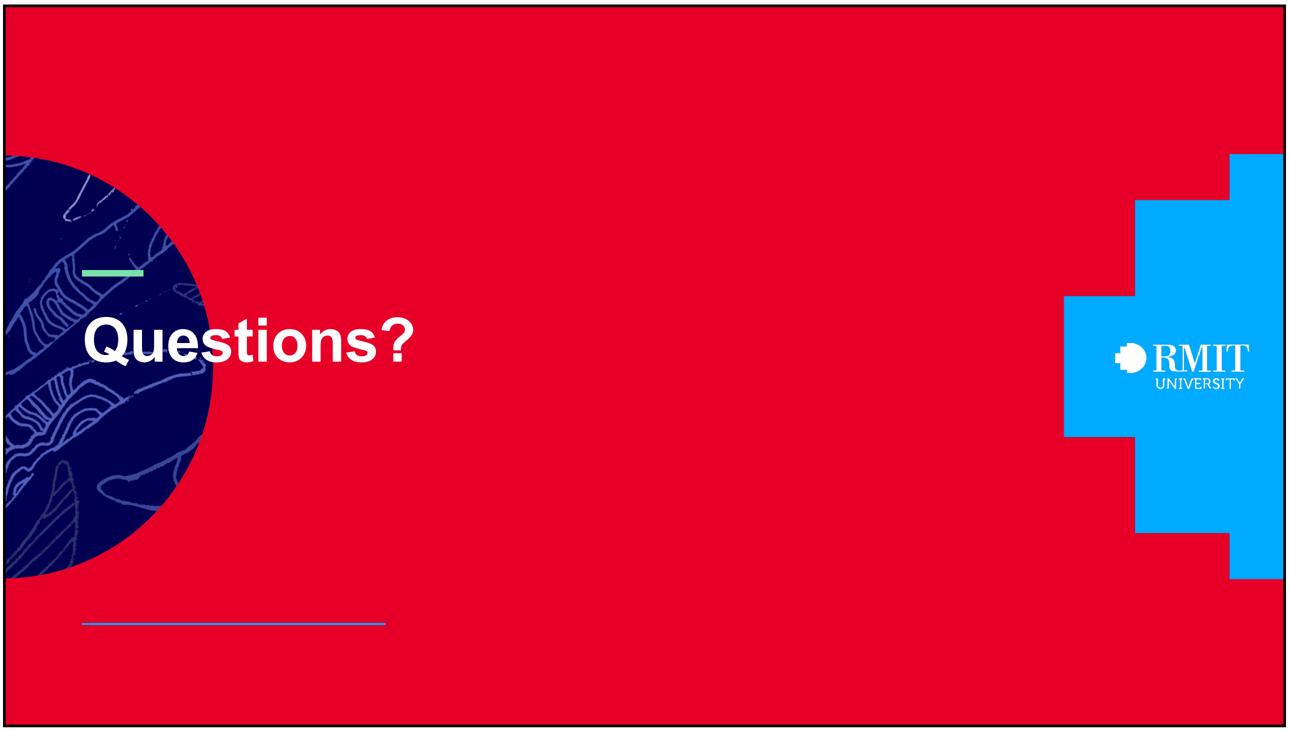


Menti-time!

Menti.com

Code: to be supplied

50



Questions?

51