Outer Joins, View, Temp Table, DM with Queries

CSC365
Spring 2019

Outer Joins

In addition to [INNER] JOIN (theta join in relational algebra) SQL supports another type of join (OUTER) that has three variations:

LEFT [OUTER] JOIN

RIGHT [OUTER] JOIN

FULL [OUTER] JOIN -- Not Supported in MySQL

LEFT OUTER JOIN

```
SELECT *
FROM A
  LEFT OUTER JOIN B ON (A.id = B.id)
```

Produces a result containing *all* records from A, paired with matching records from B. If a record from A has no match in B, the record from A is listed along with empty (null-padded) columns from B.

Example

Print all movies directed by Christopher Nolan and with Tom Hardy’s name if he starred in it.

```
SELECT *

FROM Movies m

LEFT JOIN StarsIn s

ON m.mid=s.mid AND sname='Tom Hardy'

WHERE director='Christopher Nolan';
```

Examples

With OUTER JOIN putting condition in ON or WHERE matters!

```
SELECT *

FROM Movies

LEFT JOIN StarsIn USING (mid)

WHERE director='Christopher Nolan' AND sname='Tom Hardy';
```

Is equivalent to:

```
SELECT * FROM Movies m JOIN StarsIn s ON m.mid=s.mid AND sname='Tom
Hardy' WHERE director='Christopher Nolan';
```

Examples with more than one LEFT JOIN

```
SELECT *

FROM Movies m

LEFT JOIN StarsIn s ON m.mid=s.mid AND sname='Tom Hardy'

LEFT JOIN Stars ON name=sname

WHERE director='Christopher Nolan';
```

RIGHT OUTER JOIN

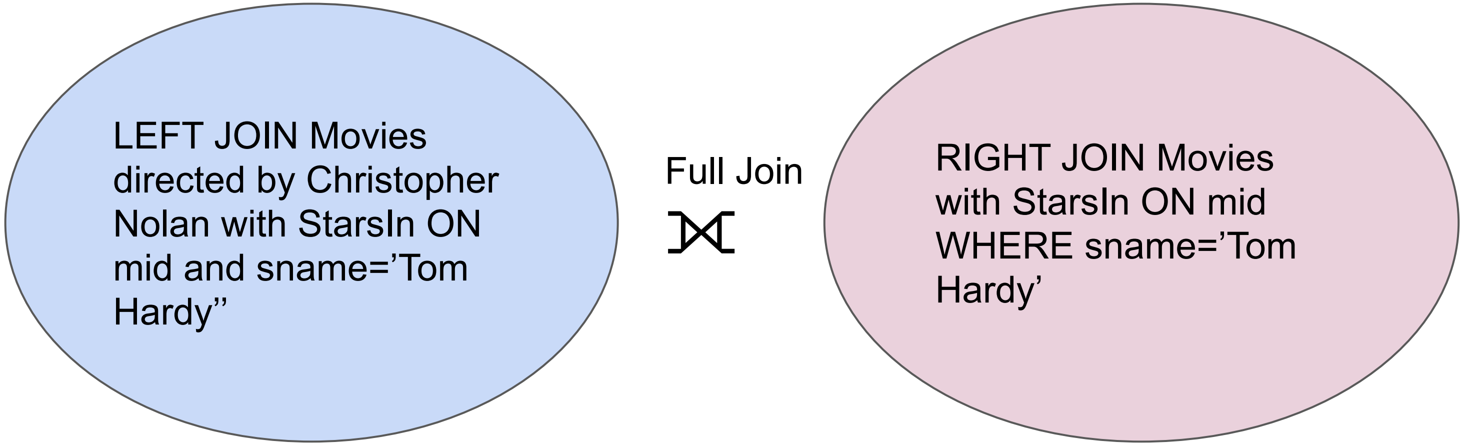
```
SELECT A.id, A.a_val, B.b_val
FROM A RIGHT OUTER JOIN B ON (A.id = B.id)
```

...is equivalent to…

```
SELECT A.id, A.a_val, B.b_val
FROM B LEFT OUTER JOIN A ON (A.id = B.id)
```

Here we switched A & B,
changed `RIGHT` to `LEFT`

RIGHT OUTER JOIN - Example



LEFT JOIN Movies directed by Christopher Nolan with StarsIn ON mid and sname='Tom Hardy''

Full Join

⟗

RIGHT JOIN Movies with StarsIn ON mid WHERE sname='Tom Hardy'

```
+------+----------------------+------+-----------+----------+-------+----------+-----------------------+------+-----------+------+
| mid  | title                | year | gross     | duration | color | language | director              | imdb | sname     | mid  |
+------+----------------------+------+-----------+----------+-------+----------+-----------------------+------+-----------+------+
|    4 | The Dark Knight Rises | 2012 | 448130642 |      164 |     0 | English  | Christopher Nolan     |  8.5 | Tom Hardy |    4 |
|   67 | The Dark Knight      | 2008 | 533316061 |      152 |     0 | English  | Christopher Nolan     |    9 | NULL      | NULL |
|   97 | Interstellar         | 2014 | 187991439 |      169 |     0 | English  | Christopher Nolan     |  8.6 | NULL      | NULL |
|   98 | Inception            | 2010 | 292568851 |      148 |     0 | English  | Christopher Nolan     |  8.8 | Tom Hardy |   98 |
|  121 | Batman Begins        | 2005 | 205343774 |      128 |     0 | English  | Christopher Nolan     |  8.3 | NULL      | NULL |
|  129 | Mad Max: Fury Road   | 2015 | 153629485 |      120 |     0 | English  | George Miller         |  8.1 | Tom Hardy |  129 |
|  180 | The Revenant         | 2015 | 183635922 |      156 |     0 | English  | Alejandro G. Iñárritu |  8.1 | Tom Hardy |  180 |
+------+----------------------+------+-----------+----------+-------+----------+-----------------------+------+-----------+------+
```

Simulating FULL JOIN in MySQL

Use UNION

```
-- FULL JOIN between Movies directed by Christopher Nolan and Movies with Tom Hardy.
SELECT *
FROM Movies m1
   LEFT JOIN StarsIn s1
      ON m1.mid=s1.mid AND s1.sname = 'Tom Hardy'
      WHERE director = 'Christopher Nolan'
UNION
SELECT *
FROM Movies m2
      RIGHT JOIN StarsIn s2
         ON m2.mid=s2.mid
      WHERE sname = 'Tom Hardy';
```

Creating View

- Used to transform queries used frequently into virtual tables.
- The query result will not be saved. So, the query still needs to be executed every time.
- Materialized View
    - Maintain values all the time
    - NOT Supported by MySQL but the same effect can be achieved by creating a table and periodically recreating it.

Examples

```
CREATE VIEW test.v AS SELECT * FROM t;
```

Examples

```
CREATE OR REPLACE View TeamWins AS

SELECT t.id, t.name, count(g.date) as wins

FROM Game g

RIGHT JOIN Team t

ON t.id = g.home_team_id and g.score_home > score_away or t.id =
g.away_team_id and g.score_away > g.score_home group by t.id;
```

Creating Temporary Table

If you need a table only for a short time, and then you want it to disappear automatically.

Create a TEMPORARY table, and let MySQL take care of removing it.

MySQL creates a transient table that disappears when your connection to the server closes, if you haven’t already removed it yourself.

```
CREATE TEMPORARY TABLE new_tbl SELECT * FROM orig_tbl LIMIT 0;
```

Limitations with Temporary Table

- The SHOW TABLES statement does not list TEMPORARY tables.
- You cannot refer to a TEMPORARY table more than once in the same query. For example, the following does not work:
  - `SELECT * FROM temp_table JOIN temp_table AS t2;`
  - The statement produces this error:
  - `ERROR 1137: Can't reopen table: 'temp_table'`

Examples

```
CREATE TEMPORARY TABLE TempWins

select t.id, t.name, count(g.date) as wins from Game g RIGHT JOIN Team t ON
t.id = g.home_team_id and g.score_home > score_away or t.id = g.away_team_id
and g.score_away > g.score_home group by t.id;



-- This will fail!

SELECT id, name FROM Team WHERE id not in (SELECT w1.id FROM
TempWins w1, TempWins w2 WHERE w1.wins < w2.wins);
```

Common Table Expressions (CTE)

- Introduced into standard SQL in order to simplify various classes of SQL Queries for which a derived table was just unsuitable.
- CTE is a temporary named result set that you can reference within a SELECT, INSERT, UPDATE, or DELETE statement.
- Syntax

    -
      ```
      WITH cte AS (SELECT 1 AS col_a, 2 AS col_b)

      SELECT * FROM cte AS t1 JOIN cte AS t2;
      ```

Not Supported in MySQL before v8!

Database Modifications

CREATE TABLE FROM Query Result

```
CREATE [TEMPORARY] TABLE [IF NOT EXISTS] tbl_name
    [(create_definition,...)]
    [table_options]
    [partition_options]
    [IGNORE | REPLACE]
    [AS] query_expression
query_expression:
    SELECT ...   (Some valid select or union statement)
```

CREATE

```
CREATE TABLE mTeamWins

select t.id, t.name, count(g.date) as wins from Game g
RIGHT JOIN Team t ON t.id = g.home_team_id and
g.score_home > score_away or t.id = g.away_team_id and
g.score_away > g.score_home group by t.id;
```

INSERT From Query Results

```
INSERT [LOW_PRIORITY | HIGH_PRIORITY] [IGNORE]
    [INTO] tbl_name
    [(col_name [, col_name] ...)]
    SELECT ...
    [ON DUPLICATE KEY UPDATE assignment_list]
```

Insertion

```
INSERT INTO mTeamWins
select t.id, t.name, count(g.date) as wins from Game g
RIGHT JOIN Team t ON t.id = g.home_team_id and
g.score_home > score_away or t.id = g.away_team_id and
g.score_away > g.score_home group by t.id;
```

UPDATE

```
UPDATE [LOW_PRIORITY] [IGNORE] table_reference
    SET assignment_list
    [WHERE where_condition]
    [ORDER BY ...]
    [LIMIT row_count]

value:
    {expr | DEFAULT}

assignment:
    col_name = value

assignment_list:
    assignment [, assignment] ...
```

UPDATE Example

```
UPDATE t SET id = id + 1;
```

```
UPDATE t SET id = id + 1 ORDER BY id DESC;
```

For example, if the table contains 1 and 2 in the id column and 1 is updated to 2 before 2 is updated to 3, an error occurs. To avoid this problem, add an ORDER BY clause to cause the rows with larger id values to be updated before those with smaller values

Update Involving Multiple Tables

```
UPDATE [LOW_PRIORITY] [IGNORE] table_references
    SET assignment_list
    [WHERE where_condition]
```

Update With INNER JOIN

```
UPDATE items,month SET items.price=month.price
WHERE items.id=month.id;
```

Updates With Subqueries

```
UPDATE mTeamWins

SET wins = (SELECT count(g.date) from Game g JOIN Team
t ON t.id = 1 and g.score_home > score_away or t.id = 1 and
g.score_away > g.score_home)

WHERE id = 1;
```

Examples

```
UPDATE mTeamWins SET wins = 9

WHERE id in

(SELECT id FROM Team WHERE id not in (SELECT w1.id
FROM TeamWins w1, TeamWins w2 WHERE w1.wins <
w2.wins));
```

Examples

```
UPDATE mTeamWins

SET wins =

(SELECT wins FROM TeamWins WHERE id = (SELECT id
FROM Team WHERE id not in (SELECT w1.id FROM
TeamWins w1, TeamWins w2 WHERE w1.wins < w2.wins)))

WHERE id in (SELECT id FROM Team WHERE id not in
(SELECT w1.id FROM TeamWins w1, TeamWins w2 WHERE
w1.wins < w2.wins));
```

DELETE

```
DELETE [LOW_PRIORITY] [QUICK] [IGNORE] FROM tbl_name
    [WHERE where_condition]
    [ORDER BY ...]
    [LIMIT row_count]
```

Deletion

```
DELETE FROM mTeamWins

WHERE id =

 (SELECT id FROM Team WHERE id not in (SELECT w1.id
FROM TeamWins w1, TeamWins w2 WHERE w1.wins <
w2.wins));
```

Summary

- SELECT statements can be used almost anywhere in SQL: in DQL, DDL, and DML.
- Queries can be stored as Views.
- Query results can be stored as Materialized Views*, Temporary Table, or CTE**.
  - * not supported by MySQL
  - ** not supported by MySQL in versions earlier than v8