

Batch: B1 Roll No.: 1711072

Experiment / assignment / tutorial No. 5

Grade: AA / AB / BB / BC / CC / CD /DD

Title: Queries based on Joins, and Views

Objective: To be able to use SQL JOIN clause to extract data from 2 (or more) tables, we need a relationship between certain columns in these tables.

Expected Outcome of Experiment:

CO 2: Convert entity-relationship diagrams into relational tables, populate a relational database and formulate SQL queries on the data Use SQL for creation and query the database

CO 3: Define and apply integrity constraints and improve database design using normalization techniques.

Books/ Journals/ Websites referred:

- 1. Dr. P.S. Deshpande, SQL and PL/SQL for Oracle 10g.Black book, Dreamtech Press
- 2. www.db-book.com
- 3. Korth, Slberchatz, Sudarshan: "Database Systems Concept", 5th Edition, McGraw
- 4. Elmasri and Navathe, "Fundamentals of database Systems", 4th Edition, PEARSON Education.

Resources used: Postgresql

Theory

Join is a combination of a Cartesian product followed by a selection process. A Join operation pairs two tuples from different relations, if and only if a given join condition is satisfied. Or JOINS are used to retrieve data from multiple tables. A JOIN is performed whenever two or more tables are joined in a SQL statement.



There are different types of Joins:

- The CROSS JOIN
- The INNER JOIN
- The LEFT OUTER JOIN
- The RIGHT OUTER JOIN
- The FULL OUTER JOIN

A **CROSS JOIN** matches every row of the first table with every row of the second table. If the input tables have x and y columns, respectively, the resulting table will have x+y columns. Because CROSS JOINs have the potential to generate extremely large tables, care must be taken to use them only when appropriate.

Ex. SELECT EMP_ID, NAME, DEPT FROM COMPANY CROSS JOIN DEPARTMENT:

A **INNER JOIN** creates a new result table by combining column values of two tables (table1 and table2) based upon the join-predicate. The query compares each row of table1 with each row of table2 to find all pairs of rows, which satisfy the join-predicate. When the join-predicate is satisfied, column values for each matched pair of rows of table1 and table2 are combined into a result row.

Ex. SELECT EMP_ID, NAME, DEPT FROM COMPANY INNER JOIN DEPARTMENT ON COMPANY.ID = DEPARTMENT.EMP_ID;

The **OUTER JOIN** is an extension of the INNER JOIN. SQL standard defines three types of OUTER JOINs: LEFT, RIGHT, and FULL and PostgreSQL supports all of these.

In case of **LEFT OUTER JOIN**, an inner join is performed first. Then, for each row in table T1 that does not satisfy the join condition with any row in table T2, a joined row is added with null values in columns of T2. Thus, the joined table always has at least one row for each row in T1.

Ex. SELECT EMP_ID, NAME, DEPT FROM COMPANY LEFT OUTER JOIN DEPARTMENT ON COMPANY.ID = DEPARTMENT.EMP_ID;

The RIGHT OUTER JOIN

First, an inner join is performed. Then, for each row in table T2 that does not satisfy the join condition with any row in table T1, a joined row is added with null values in



columns of T1. This is the converse of a left join; the result table will always have a row for each row in T2.

Ex. SELECT EMP_ID, NAME, DEPT FROM COMPANY RIGHT OUTER JOIN DEPARTMENT ON COMPANY.ID = DEPARTMENT.EMP ID;

The FULL OUTER JOIN

First, an inner join is performed. Then, for each row in table T1 that does not satisfy the join condition with any row in table T2, a joined row is added with null values in columns of T2. In addition, for each row of T2 that does not satisfy the join condition with any row in T1, a joined row with null values in the columns of T1 is added.

SELECT EMP_ID, NAME, DEPT FROM COMPANY FULL OUTER JOIN DEPARTMENT ON COMPANY.ID = DEPARTMENT.EMP ID;

Views are pseudo-tables. That is, they are not real tables; nevertheless appear as ordinary tables to SELECT. A view can represent a subset of a real table, selecting certain columns or certain rows from an ordinary table. A view can even represent joined tables. Because views are assigned separate permissions, you can use them to restrict table access so that the users see only specific rows or columns of a table.

A view can contain all rows of a table or selected rows from one or more tables. A view can be created from one or many tables, which depends on the written PostgreSQL query to create a view.

Views, which are kind of virtual tables, allow users to do the following –

- Structure data in a way that users or classes of users find natural or intuitive.
- Restrict access to the data such that a user can only see limited data instead of complete table.
- Summarize data from various tables, which can be used to generate reports.

Since views are not ordinary tables, you may not be able to execute a DELETE, INSERT, or UPDATE statement on a view. However, you can create a RULE to correct this problem of using DELETE, INSERT or UPDATE on a view.

Syntax

CREATE [TEMP | TEMPORARY] VIEW view name AS

SELECT column1, column2.....

FROM table name

WHERE [condition];

Ex

CREATE VIEW COMPANY_VIEW AS SELECT ID, NAME, AGE FROM COMPANY;

Dropping Views

Syntax: DROP VIEW view_name;

Implementation:

select * from movie join review on movie_id = mov_id

select * from movie inner join review on movie_id = mov_id

select * from movie full outer join review on movie_id = mov_id

select * from movie left outer join review on movie_id = mov_id

select * from movie right outer join review on movie_id = mov_id

select * from movie natural join review

select * from movie cross join cast_movie

select * from movie t, movie m where

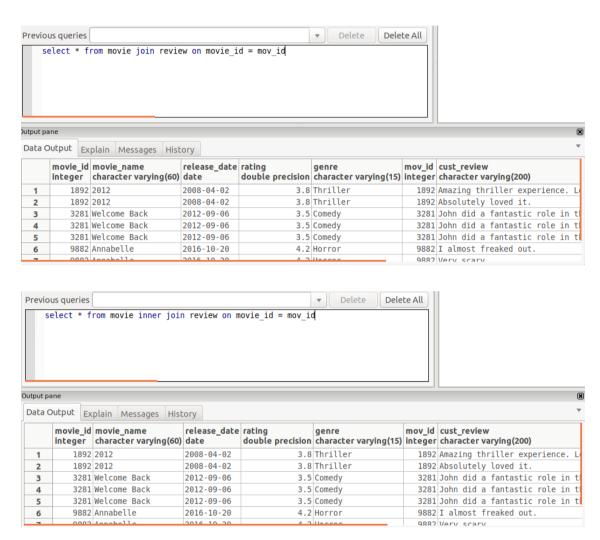
m.movie_name=t.movie_name; self join

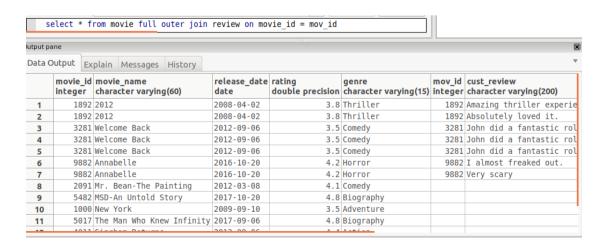
create or replace view reviewer as select * from movie join review

on movie_id = mov_id

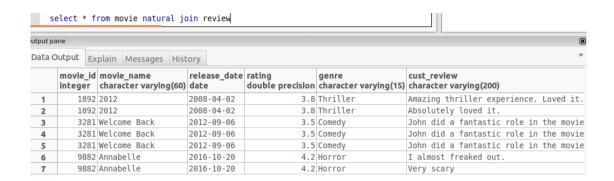


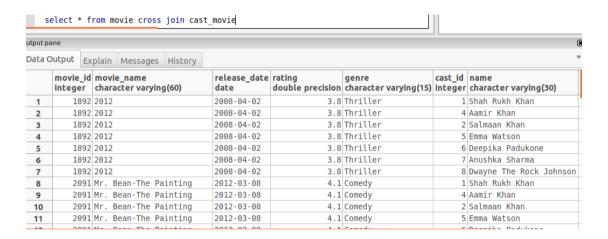
Screenshots:

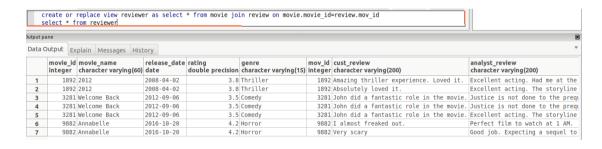


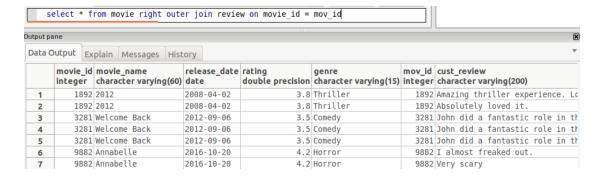




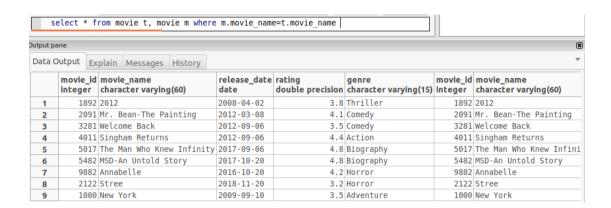


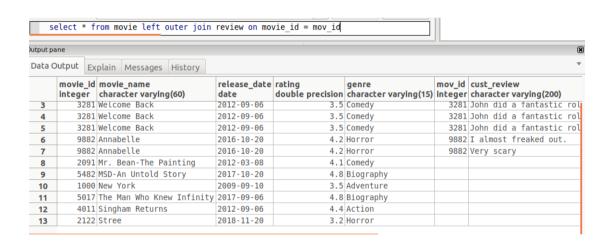












Conclusion: Various kinds of join operations were performed and views were created.