

|  |
| --- |
| **Title: 6. Queries based on Join, Assertions and Triggers** |



**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 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 Hill

4. Elmasri and Navathe,”Fundamentals of database Systems”, 4th Edition,PEARSON Education.

**Pre Lab/ Prior Concepts**:

Resources used: MySQL/Oracle

**Theory**

MySQL JOINS are used to retrieve data from multiple tables. A MySQL JOIN is performed whenever two or more tables are joined in a SQL statement.

There are different types of MySQL joins:

• MySQL INNER JOIN (or sometimes called simple join)

• MySQL LEFT OUTER JOIN (or sometimes called LEFT JOIN)

• MySQL RIGHT OUTER JOIN (or sometimes called RIGHT JOIN)

**INNER JOIN :**

It is the most common type of join. MySQL INNER JOINS return all rows from multiple tables where the join condition is met.

**Syntax**

The syntax for the INNER JOIN in MySQL is:

SELECT columns

FROM table1

INNER JOIN table2

ON table1.column = table2.column;

**MySQL Left Outer Join**:

The LEFT OUTER JOIN returns all rows from the left hand table specified in the ON condition and only those rows from the other table where the join condition is fulfilled.

**Syntax:**

SELECT columns

FROM table1

LEFT OUTER JOIN table2

ON table1.column = table2.column;

**MySQL Right Outer Join:**

The MySQL Right Outer Join returns all rows from the RIGHT-hand table specified in the ON condition and only those rows from the other table where he join condition is fulfilled.

**Syntax:**

SELECT columns

FROM table1

RIGHT [OUTER] JOIN table2

ON table1.column = table2.column;

**Natural Join:**

A NATURAL JOIN is a JOIN operation that creates an implicit join clause for you based on the common columns in the two tables being joined. Common columns are columns that have the same name in both tables. A NATURAL JOIN can be an INNER join, a LEFT OUTER join, or a RIGHT OUTER join.

**Syntax**:

SELECT columns

FROM table1

Natural Join table2;

**Assertion:**

1. An **assertion** is a predicate expressing a condition we wish the database to always satisfy.
2. Domain constraints, functional dependency and referential integrity are special forms of assertion.
3. Where a constraint cannot be expressed in these forms, we use an assertion, e.g.
   * Ensuring the sum of loan amounts for each branch is less than the sum of all account balances at the branch.
   * Ensuring every loan customer keeps a minimum of $1000 in an account.

**Triggers:**

A trigger or database trigger is a stored program executed automatically to respond to a specific event e.g., insert, update or delete occurred in a table.

**MySQL: AFTER INSERT Trigger**

An AFTER INSERT Trigger means that MySQL will fire this trigger after the INSERT operation is executed.

**Syntax:**

The syntax to create an AFTER INSERT Trigger in MySQL is:

CREATE TRIGGER trigger\_name

AFTER INSERT

ON table\_name FOR EACH ROW

BEGIN

-- variable declarations

-- trigger code

END;

**Parameters or Arguments**

**Trigger\_name**

The name of the trigger to create.

**AFTER INSERT**

It indicates that the trigger will fire after the INSERT operation is executed.

**Table\_name**

The name of the table that the trigger is created on.

**Restrictions**

• You can not create an AFTER trigger on a view.

• You can not update the NEW values.

• You can not update the OLD values.

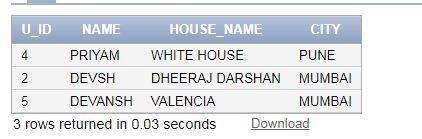
**Implementation Screenshots:**

**INNER JOIN**

Select USER\_1.U\_ID,USER\_1.NAME,PROPERTY.HOUSE\_NAME,PROPERTY.CITY

FROM USER\_1

INNER JOIN PROPERTY ON PROPERTY.U\_ID=USER\_1.U\_ID

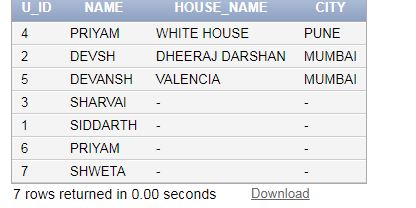


**LEFT JOIN**

Select USER\_1.U\_ID,USER\_1.NAME,PROPERTY.HOUSE\_NAME,PROPERTY.CITY

FROM USER\_1

LEFT JOIN PROPERTY ON PROPERTY.U\_ID=USER\_1.U\_ID

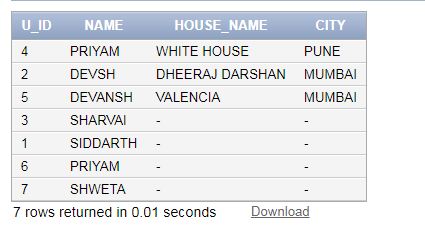


**RIGHT JOIN**

Select USER\_1.U\_ID,USER\_1.NAME,PROPERTY.HOUSE\_NAME,PROPERTY.CITY

FROM PROPERTY

RIGHT JOIN USER\_1 ON PROPERTY.U\_ID=USER\_1.U\_ID

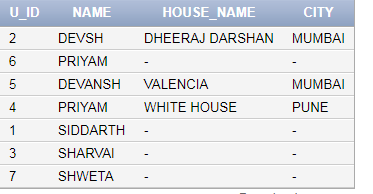


**FULL OUTER JOIN**

Select USER\_1.U\_ID,USER\_1.NAME,PROPERTY.HOUSE\_NAME,PROPERTY.CITY

FROM PROPERTY

FULL OUTER JOIN USER\_1 ON PROPERTY.U\_ID=USER\_1.U\_ID



**SELF JOIN**

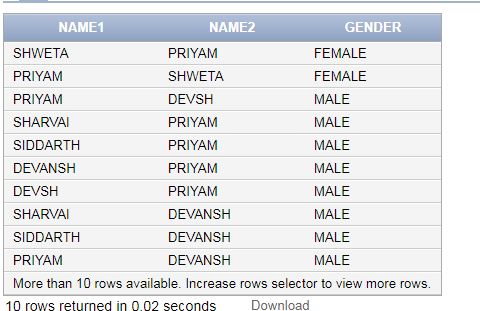
SELECT A.NAME AS NAME1, B.NAME AS NAME2, A.GENDER

FROM USER\_1 A, USER\_1 B

WHERE A.U\_ID <> B.U\_ID

AND A.GENDER = B.GENDER

ORDER BY A.GENDER;



**Views**

To create a view of property above average

CREATE VIEW [Property Above Average Price] AS

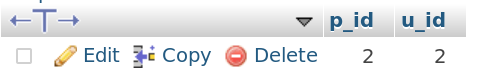
SELECT p\_id,u\_id

FROM Property

WHERE Price > (SELECT AVG(u\_id) FROM Property);

To display view

SELECT \* FROM [Property Above Average Price];



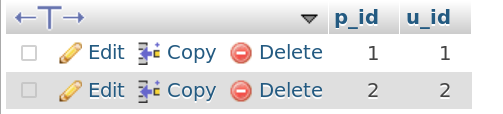
To create a view of flats

CREATE VIEW [Flats] AS

SELECT p\_id,u\_id

FROM Property

WHERE P\_Type = ‘Flat’;

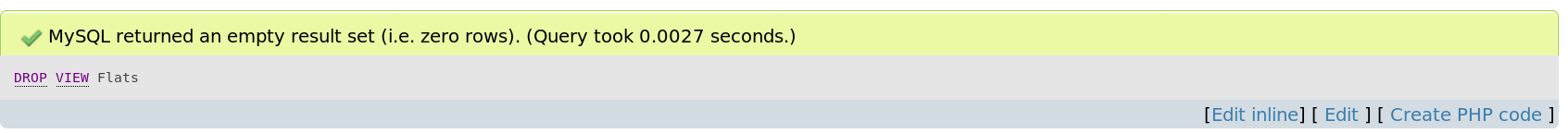


To display view

SELECT \* FROM [Flats];

To drop view

DROP VIEW Flats;



**Triggers**

**->To create a trigger that shows updated price and old price of a property with its difference.**

**CREATE OR REPLACE TRIGGER display\_price\_changes**

**BEFORE DELETE OR INSERT OR UPDATE ON property**

**FOR EACH ROW**

**WHEN (NEW.ID > 0)**

**DECLARE**

**price\_diff number;**

**BEGIN**

**price\_diff := :NEW.price - :OLD.price;**

**dbms\_output.put\_line('Old Price: ' || :OLD.price);**

**dbms\_output.put\_line('New Price: ' || :NEW.price);**

**dbms\_output.put\_line('Price difference: ' || price\_diff);**

**END;**

**->Insert Query**

**INSERT INTO property (P\_ID,NAME,P\_Type,Address,Price)**

**VALUES (1, 'White House', ‘FLAT’, 'HP', 750000 );**

**->Update Query**

**UPDATE property SET Price = Price + 500 WHERE id =1;**

# **Conclusion:**

Queries based on Join, Assertions and Triggers.

**Date: \_\_\_\_\_\_\_\_\_\_\_\_\_ Signature of faculty in-charge**

**Post Lab Questions:**

1. Explain the different types of Triggers which can be specified on the table with examples

Triggers are stored programs, which are automatically executed or fired when some events occur. Triggers are written to be executed in response to any of the following events −

A database manipulation (DML) statement (DELETE, INSERT, or UPDATE)

A database definition (DDL) statement (CREATE, ALTER, or DROP).

Triggers can be defined on the table, view, schema, or database with which the event is associated.

**Different Types of Triggers**

There are two types of triggers based on the which level it is triggered.

**1) Row level trigger** - An event is triggered for each row upated, inserted or deleted.

**2) Statement level trigger** - An event is triggered for each sql statement executed.

The following hierarchy is followed when a trigger is fired.

**1)** BEFORE statement trigger fires first.

**2)** Next BEFORE row level trigger fires, once for each row affected.

**3)** Then AFTER row level trigger fires once for each affected row. This events will alternates between BEFORE and AFTER row level triggers.

**4)** Finally the AFTER statement level trigger fires.

**For Example:** Let's create a table 'product\_check' which we can use to store messages when triggers are fired.

1. CREATE TABLE product
2. (Message varchar2(50),
3. Current\_Date number(32)
4. );

Let's create a BEFORE and AFTER statement and row level triggers for the product table.

**1) BEFORE UPDATE, Statement Level:** This trigger will insert a record into the table 'product\_check' before a sql update statement is executed, at the statement level.

1. CREATE **or** REPLACE TRIGGER Before\_Update\_Stat\_product
2. BEFORE
3. UPDATE ON product
4. Begin
5. INSERT INTO product\_check
6. Values('Before update, statement level',sysdate);
7. **END**;
8. /

**2) BEFORE UPDATE, Row Level:** This trigger will insert a record into the table 'product\_check' before each row is updated.

1. CREATE **or** REPLACE TRIGGER Before\_Upddate\_Row\_product
2. BEFORE
3. UPDATE ON product
4. FOR EACH ROW
5. **BEGIN**
6. INSERT INTO product\_check
7. Values('Before update row level',sysdate);
8. **END**;
9. /

**3) AFTER UPDATE, Statement Level:** This trigger will insert a record into the table 'product\_check' after a sql update statement is executed, at the statement level.

1. CREATE **or** REPLACE TRIGGER After\_Update\_Stat\_product
2. AFTER
3. UPDATE ON product
4. **BEGIN**
5. INSERT INTO product\_check
6. Values('After update, statement level', sysdate);
7. End;
8. /

**4) AFTER UPDATE, Row Level:** This trigger will insert a record into the table 'product\_check' after each row is updated.

1. CREATE **or** REPLACE TRIGGER After\_Update\_Row\_product
2. AFTER
3. insert On product
4. FOR EACH ROW
5. **BEGIN**
6. INSERT INTO product\_check
7. Values('After update, Row level',sysdate);
8. **END**;
9. /

Now lets execute a update statement on table product.

1. UPDATE PRODUCT SET unit\_price = 800
2. WHERE product\_id **in** (100,101);

Lets check the data in 'product\_check' table to see the order in which the trigger is fired.

1. SELECT \* FROM product\_check;

**Output:**

Message Current\_Date

------------------------------------------------------------

Before update, statement level 26-Nov-2008

Before update, row level 26-Nov-2008

After update, Row level 26-Nov-2008

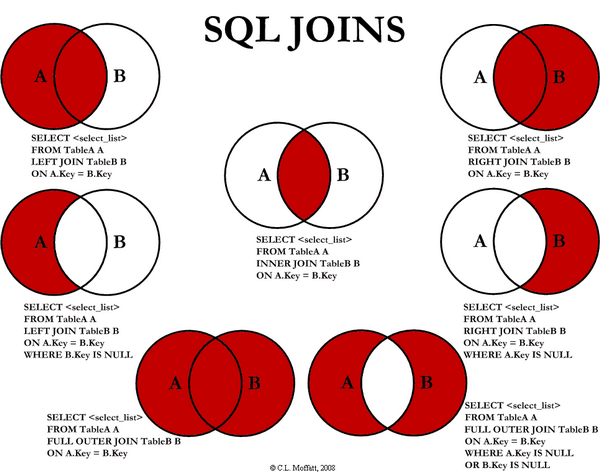
Before update, row level 26-Nov-2008

After update, Row level 26-Nov-2008

After update, statement level 26-Nov-2008

The above result shows 'before update' and 'after update' row level events have occured twice, since two records were updated. But 'before update' and 'after update' statement level events are fired only once per sql statement.

1. Explain the difference between Inner Join and Left Join with example.



TableA

id firstName lastName

.......................................

1 arun prasanth

2 ann antony

3 sruthy abc

6 new abc

TableB

id2 age Place

................

1 24 kerala

2 24 usa

3 25 ekm

5 24 chennai

**INNER JOIN** gets all records that are common between both tables based on the foreign key

**Example:**

**SELECT TableA.firstName,TableA.lastName,TableB.age,TableB.Place**

**FROM TableA**

**INNER JOIN TableB**

**ON TableA.id = TableB.id2;**

**firstName lastName age Place**

**..............................................**

**arun prasanth 24 kerala**

**ann antony 24 usa**

**sruthy abc 25 ekm**

**LEFT JOIN** gets all records from the LEFT linked table but if you have selected some columns from the RIGHT table, if there is no related records, these columns will contain NULL

**Example:**

**SELECT TableA.firstName,TableA.lastName,TableB.age,TableB.Place**

**FROM TableA**

**LEFT JOIN TableB**

**ON TableA.id = TableB.id2;**

**firstName lastName age Place**

**...............................................................................**

**arun prasanth 24 kerala**

**ann antony 24 usa**

**sruthy abc 25 ekm**

**new abc NULL NULL**