# Experiment No. 3

**Environment:** Microsoft Windows

**Tools/ Language:** Oracle/SQL

## Objective: Write the SQL queries using Set Operations and Joins. Theory & Concepts:

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

There are different types of SQL joins:

SQL INNER JOIN (or sometimes called simple join) SQL CROSS JOIN

SQL NATURAL JOIN

SQL LEFT OUTER JOIN (or sometimes called LEFT JOIN) SQL RIGHT OUTER JOIN (or sometimes called RIGHT JOIN) SQL FULL OUTER JOIN (or sometimes called FULL JOIN)

## SQL INNER JOIN (SIMPLE JOIN)

SQL INNER JOINS return all rows from multiple tables where the join condition is met.

Syntax

The syntax for the SQL INNER JOIN is:

SELECT columns FROM table1 INNER JOIN table2

ON table1.column = table2.column;

If the tables COUNTRIES and CITIES have two common columns named POPULATION and COUNTRY\_ISO\_CODE, JOIN applies equality condition on ISO codes with cities having less POPULATION attributes:

**SELECT \* FROM**

***COUNTRIES***

**INNER JOIN *CITIES***

**On *COUNTRIES*. COUNTRY\_ISO\_CODE=*CITIES*. COUNTRY\_ISO\_CODE And *COUNTRIES*.POPULATION > *CITIES*.POPULATION;**

## SQL LEFT OUTER JOIN

Another type of join is called a LEFT OUTER JOIN. This type of join returns all rows from the LEFT-hand table specified in the ON condition and only those rows from the other table where the joined fields are equal (join condition is met).

Syntax

The syntax for the SQL LEFT OUTER JOIN is:

SELECT columns FROM table1

LEFT [OUTER] JOIN table2

ON table1.column = table2.column;

In some databases, the LEFT OUTER JOIN keywords are replaced with LEFT JOIN.

**SELECT \* FROM**

***COUNTRIES***

**LEFT JOIN *CITIES***

**On *COUNTRIES*. COUNTRY\_ISO\_CODE=*CITIES*. COUNTRY\_ISO\_CODE And *COUNTRIES*.POPULATION > *CITIES*.POPULATION;**

## SQL RIGHT OUTER JOIN

Another type of join is called a SQL RIGHT OUTER JOIN. This type of join returns all rows from the RIGHT-hand table specified in the ON condition and only those rows from the other table where the joined fields are equal (join condition is met).

Syntax

The syntax for the SQL RIGHT OUTER JOIN is:

SELECT columns FROM table1

RIGHT [OUTER] JOIN table2

ON table1.column = table2.column;

In some databases, the RIGHT OUTER JOIN keywords are replaced with RIGHT JOIN.

**SELECT \* FROM**

***COUNTRIES***

**RIGHT JOIN *CITIES***

**On *COUNTRIES*. COUNTRY\_ISO\_CODE=*CITIES*. COUNTRY\_ISO\_CODE And *COUNTRIES*.POPULATION > *CITIES*.POPULATION;**

## SQL FULL OUTER JOIN

Another type of join is called a SQL FULL OUTER JOIN. This type of join returns all rows from the LEFT-hand table and RIGHT-hand table with nulls in place where the join condition is not met.

Syntax

The syntax for the SQL FULL OUTER JOIN is:

SELECT columns FROM table1

FULL [OUTER] JOIN table2

ON table1.column = table2.column;

In some databases, the FULL OUTER JOIN keywords are replaced with FULL JOIN.

**SELECT \* FROM**

***COUNTRIES***

**FULL JOIN *CITIES***

**On *COUNTRIES*. COUNTRY\_ISO\_CODE=*CITIES*. COUNTRY\_ISO\_CODE And *COUNTRIES*.POPULATION > *CITIES*.POPULATION;**

## SQL NATURAL JOIN

A NATURAL JOIN is a [JOIN operation](http://docs.oracle.com/javadb/10.8.3.0/ref/rrefsqlj29840.html#rrefsqlj29840) 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.

If the SELECT statement in which the NATURAL JOIN operation appears has an asterisk (\*) in the select list, the asterisk will be expanded to the following list of columns (in this order):

* All the common columns
* Every column in the first (left) table that is not a common column
* Every column in the second (right) table that is not a common column

An asterisk qualified by a table name (for example, COUNTRIES.\*) will be expanded to every column of that table that is not a common column.

Syntax ***Select \* FROM table1***

***NATURAL JOIN table2;***

## Examples

If the tables COUNTRIES and CITIES have two common columns named COUNTRY and COUNTRY\_ISO\_CODE, NATURAL JOIN applies equality condition on both attributes:

**SELECT \* FROM COUNTRIES NATURAL JOIN CITIES;**

**CROSS JOIN operation**

A CROSS JOIN is a [JOIN operation](https://docs.oracle.com/javadb/10.10.1.2/ref/rrefsqlj29840.html#rrefsqlj29840) that produces the Cartesian product of two tables. Unlike other JOIN operators, it does not let you specify a join clause. You may, however, specify a WHERE clause in the SELECT statement.

# Examples

The following SELECT statements are equivalent:

**SELECT \* FROM *CITIES* CROSS JOIN**

***FLIGHTS***

**SELECT \* FROM *CITIES*, *FLIGHTS***

**Practical Assignment - 3**



**Department:** Computer Engineering & Applications

**Course:** B.Tech. (CSE)

**Subject:** Database Management System Lab (BCSC0802)

**Year:** 2nd **Semester:** 3rd

Imagine you are managing a comprehensive database system for an academic institution that tracks essential information about colleges, students, and their application records. Create Student, Apply and College tables using script.

Run the following Script:

BEGIN

FOR cur\_rec IN (SELECT object\_name, object\_type FROM user\_objects

WHERE object\_type IN ('TABLE',

'VIEW', 'PACKAGE', 'PROCEDURE', 'FUNCTION', 'SEQUENCE'

))

LOOP BEGIN

IF cur\_rec.object\_type = 'TABLE' THEN

EXECUTE IMMEDIATE 'DROP '

|| cur\_rec.object\_type

|| ' "'

|| cur\_rec.object\_name

|| '" CASCADE CONSTRAINTS';

ELSE

EXECUTE IMMEDIATE 'DROP '

|| cur\_rec.object\_type

|| ' "'

|| cur\_rec.object\_name

|| '"';

END IF; EXCEPTION

WHEN OTHERS THEN

DBMS\_OUTPUT.put\_line ( 'FAILED: DROP '

|| cur\_rec.object\_type

|| ' "'

|| cur\_rec.object\_name

|| '"'

);

END;

END LOOP;

END;

/

commit;

drop table College; drop table Student; drop table Apply;

create table College(collegeName varchar2(10) primary key, state varchar2(10), enrollment int);

create table Student(sID int primary key, sName varchar2(10), GPA real, sizeHS int);

create table Apply(sID int, cName varchar2(10), major varchar2(20), decision char(1), primary key(sID, major, cName), constraint sID\_fk Foreign key(sID) references Student, constraint cName\_fk Foreign key(cName) references College);

delete from Student; delete from College; delete from Apply;

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| insert | into | Student | values | (123, | 'Amy', 3.9, 1000); |
| insert | into | Student | values | (234, | 'Bob', 3.6, 1500); |
| insert | into | Student | values | (345, | 'Craig', 3.5, 500); |
| insert | into | Student | values | (456, | 'Doris', 3.9, 1000); |
| insert | into | Student | values | (567, | 'Edward', 2.9, 2000); |
| insert | into | Student | values | (678, | 'Fay', 3.8, 200); |
| insert | into | Student | values | (789, | 'Gary', 3.4, 800); |
| insert | into | Student | values | (987, | 'Helen', 3.7, 800); |
| insert | into | Student | values | (876, | 'Irene', 3.9, 400); |
| insert | into | Student | values | (765, | 'Jay', 2.9, 1500); |
| insert | into | Student | values | (654, | 'Amy', 3.9, 1000); |
| insert | into | Student | values | (543, | 'Craig', 3.4, 2000); |

insert into College values ('Stanford', 'CA', 15000); insert into College values ('Berkeley', 'CA', 36000); insert into College values ('MIT', 'MA', 10000); insert into College values ('Cornell', 'NY', 21000); insert into College values ('Harvard', 'MA', 50040); insert into Apply values (123, 'Stanford', 'CS', 'Y'); insert into Apply values (123, 'Stanford', 'EE', 'N'); insert into Apply values (123, 'Berkeley', 'CS', 'Y'); insert into Apply values (123, 'Cornell', 'EE', 'Y');

insert into Apply values (234, 'Berkeley', 'biology', 'N'); insert into Apply values (345, 'MIT', 'bioengineering', 'Y'); insert into Apply values (345, 'Cornell', 'bioengineering', 'N'); insert into Apply values (345, 'Cornell', 'CS', 'Y');

insert into Apply values (345, 'Cornell', 'EE', 'N'); insert into Apply values (678, 'Stanford', 'history', 'Y'); insert into Apply values (987, 'Stanford', 'CS', 'Y'); insert into Apply values (987, 'Berkeley', 'CS', 'Y'); insert into Apply values (876, 'Stanford', 'CS', 'N'); insert into Apply values (876, 'MIT', 'biology', 'Y');

insert into Apply values (876, 'MIT', 'marine biology', 'N'); insert into Apply values (765, 'Stanford', 'history', 'Y'); insert into Apply values (765, 'Cornell', 'history', 'N'); insert into Apply values (765, 'Cornell', 'psychology', 'Y'); insert into Apply values (543, 'MIT', 'CS', 'N');

commit;

## Student

|  |  |  |  |
| --- | --- | --- | --- |
| **Apply** | | | |
| **sID** | **cName** | **major** | **decision** |
| 123 | Stanford | CS | Y |
| 123 | Stanford | EE | N |
| 123 | Berkeley | CS | Y |
| 123 | Cornell | EE | Y |
| 234 | Berkeley | biology | N |
| 345 | MIT | bioengineering | Y |
| 345 | Cornell | bioengineering | N |
| 345 | Cornell | CS | Y |
| 345 | Cornell | EE | N |
| 678 | Stanford | history | Y |
| 987 | Stanford | CS | Y |
| 987 | Berkeley | CS | Y |
| 876 | Stanford | CS | N |
| 876 | MIT | biology | Y |
| 876 | MIT | marine biology | N |
| 765 | Stanford | history | Y |
| 765 | Cornell | history | N |
| 765 | Cornell | psychology | Y |
| 543 | MIT | CS | N |

|  |  |  |  |
| --- | --- | --- | --- |
| **sID** | **sName** | **GPA** | **sizeHS** |
| 123 | Amy | 3.9 | 1000 |
| 234 | Bob | 3.6 | 1500 |
| 345 | Craig | 3.5 | 500 |
| 456 | Doris | 3.9 | 1000 |
| 567 | Edward | 2.9 | 2000 |
| 678 | Fay | 3.8 | 200 |
| 789 | Gary | 3.4 | 800 |
| 987 | Helen | 3.7 | 800 |
| 876 | Irene | 3.9 | 400 |
| 765 | Jay | 2.9 | 1500 |
| 654 | Amy | 3.9 | 1000 |
| 543 | Craig | 3.4 | 2000 |

**College**

|  |  |  |
| --- | --- | --- |
| **collegeName** | **state** | **enrollment** |
| Stanford | CA | 15000 |
| Berkeley | CA | 36000 |
| MIT | MA | 10000 |
| Cornell | NY | 21000 |
| Harvard | MA | 50040 |

# Write SQL Queries for the following:

**Q1.** Produce a combine table in which each student is combine with every other application.

**Q2.** Give Student ID, name, GPA and name of college and major each student applied to.

**Q3.** Find detail of applications who applied to California State.

**Q4.** IDs, name, GPA of students and name of college with GPA > 3.7 applying to Stanford

**Q5.** Find detail of Student who apply to CS major and their application are rejected **Q6.** Find detail of student and application who applied to colleges at New York **Q7.** Find detail of student who have not applied to any of college

**Q8.** Find college where no student have applied

**Q9.** Find sID who have only one application

**Q10.** Find name and GPA of applicants who apply to any college whose enrollment is not more than 25000.

**Q11.** Find pair of students (sID) having same GPA. (*each pair should occur just once in result*)

# Exercise

**For each of the following you need to write three queries**

**i.e. three version first using :CROSS Join Second using: Natural Join**

**And third using: Inner Join**

***You are also advised to observe output of all three***

**Q12.** Find student and major he / she applied to.

**Q13.** Find detail of student who came from high school have size less than 20000 and applied to CS at Stanford.

**Q14.** Provide complete detail of each student where they applied what major they applied to what was the decision and complete detail of college they applied.

**Q15.** Names and GPAs of students with HS>1000 who applied to CS and were rejected

**Q16.** Names and GPAs of students with HS>1000 who applied to CS at college with enr>20,000 and were rejected

## Pre Experiment Questions

* 1. When we need to combine two tables?
  2. Difference between Equi Join and Theta Join
  3. Difference between Natural join and Inner Join

## Post Experiment Questions

1. When can we use natural join?
2. When we are bound to use inner join?
3. Can we implement all joins using cross join?
4. Where and in what kind of queries require outer joins?