SQL JOINS:

http://www.sqlservertutorial.net/sql-server-basics/

SQL INNER JOIN syntax

Scenario Explanation Link: http://www.sqlservertutorial.net/sql-server-basics/sql-server-inner-join/

The following illustrates INNER JOIN syntax for joining two tables:

```
SELECT column_name(s)
FROM table1
INNER JOIN table2
ON table1.column_name = table2.column_name;
```

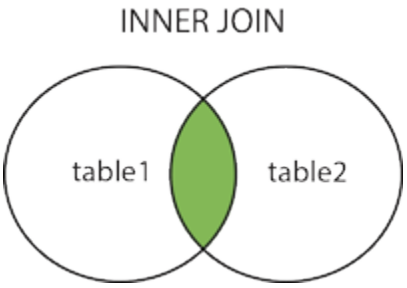
Let's examine the syntax above in greater detail:

The table_1 and table_2 are called joined-tables.

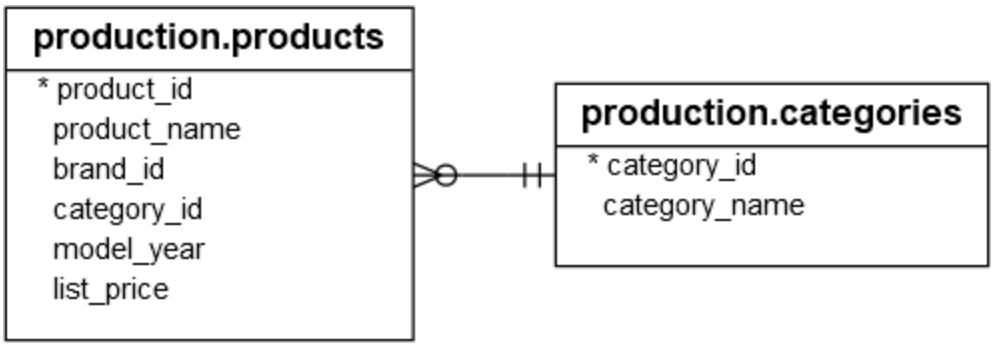
For each row in the table_1, the query find the corresponding row in the table_2 that meet the join condition.

If the corresponding row found, the query returns a row that contains data from both tables.

Otherwise, it examines next row in the table_1, and this process continues until all the rows in the table_1 are examined.



The inner join is one of the most commonly used joins in SQL Server. The inner join clause allows you to query data from two or more related tables.



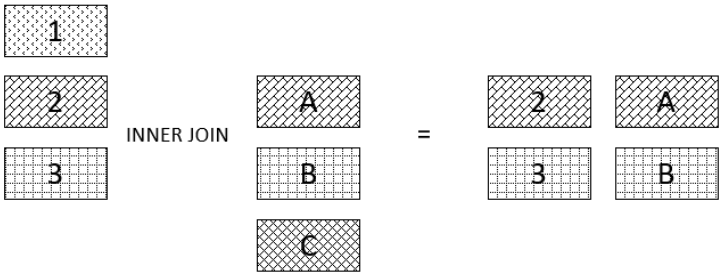

```
SELECT
   product_name,
   list_price,
   category_id
FROM
   production.products
ORDER BY
   product_name DESC;
```

product_name	list_price	category_id
Trek XM700+ Lowstep - 2018	3499.99	5
Trek XM700+ - 2018	3499.99	5
Trek X-Caliber Frameset - 2018	1499.99	6
Trek X-Caliber 8 - 2018	999.99	6
Trek X-Caliber 8 - 2017	999.99	6
Trek X-Caliber 7 - 2018	919.99	6
Trek Verve+ Lowstep - 2018	2299.99	5
Trek Verve+ - 2018	2299.99	5
Trek Ticket S Frame - 2018	1469.99	6
Trek Superfly 24 - 2017/2018	489.99	1
Trek Superfly 20 - 2018	399.99	1

The query returned only a list of category identification numbers, not the category names. To include the **category names** in the result set, you use the `INNER JOIN` clause as follows:

```
SELECT product_name, category_name, list_price FROM production.products p
INNER JOIN production.categories c ON c.category_id = p.category_id
ORDER BY product_name DESC;
```

product_name	category_name	list_price
Trek XM700+ Lowstep - 2018	Electric Bikes	3499.99
Trek XM700+ - 2018	Electric Bikes	3499.99
Trek X-Caliber Frameset - 2018	Mountain Bikes	1499.99
Trek X-Caliber 8 - 2018	Mountain Bikes	999.99
Trek X-Caliber 8 - 2017	Mountain Bikes	999.99
Trek X-Caliber 7 - 2018	Mountain Bikes	919.99
Trek Verve+ Lowstep - 2018	Electric Bikes	2299.99
Trek Verve+ - 2018	Electric Bikes	2299.99
Trek Ticket S Frame - 2018	Mountain Bikes	1469.99
Trek Superfly 24 - 2017/2018	Children Bicycles	489.99
Trek Superfly 20 - 2018	Children Bicycles	399.99
Trek Super Commuter+ 8S - 2018	Electric Bikes	4999.99
Trek Super Commuter+ 7 - 2018	Electric Bikes	3599.99
Trek Stache Carbon Frameset - 2018	Mountain Bikes	919.99



==========

SQL OUTER JOIN – left outer join

Scenario Explanation Link: http://www.sqlservertutorial.net/sql-server-basics/sql-server-left-join/

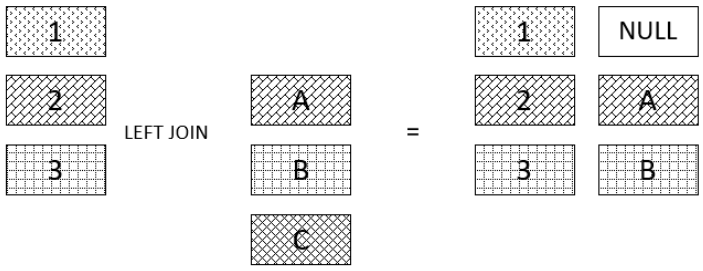
```
SELECT column_name(s)
FROM table1
LEFT JOIN table2
ON table1.column_name = table2.column_name;
```

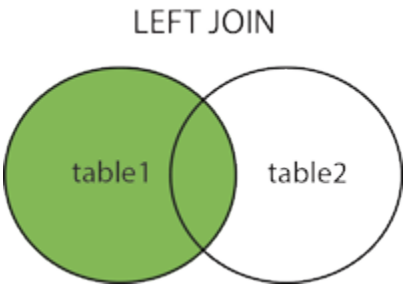
My understanding-Things that we want to achieve using right join can also be achieved using left join if we place the table name in left hand side of “LEFT JOIN” keyword.

SQL left outer join is also known as SQL left join.

Suppose, we want to join two tables: A and B. SQL left outer join returns all rows in the left table (A)

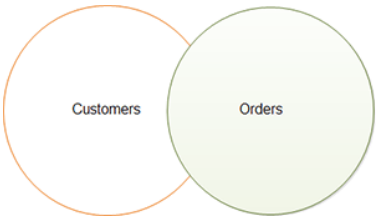
and all the matching rows(matching the join condition) found in the right table (B). It means the result of the SQL left join always contains the rows in the left table.






```
SELECT Customers.CustomerName, Orders.OrderID
FROM Customers
LEFT JOIN Orders ON Customers.CustomerID = Orders.CustomerID
ORDER BY Customers.CustomerName;
```

Right outer join: opposite of left outer join.



==========

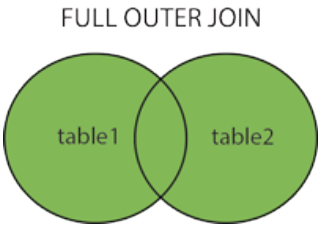
SQL full outer join returns:

```
SELECT column_name(s)
FROM table1
FULL OUTER JOIN table2
ON table1.column_name = table2.column_name
WHERE condition;
```

all rows in the left table table_A.

all rows in the right table table_B.

and all matching rows in both tables.



JOIN Three Tables

```
SELECT Orders.OrderID, Customers.CustomerName, Shippers.ShipperName
FROM ((Orders INNER JOIN Customers ON Orders.CustomerID
Customers.CustomerID)INNER JOIN Shippers ON Orders.ShipperID =
Shippers.ShipperID);
```

The following statement uses two `INNER JOIN` clauses to query data from the three tables

```
SELECT
   product_name,
   category_name,
   brand_name,
   list_price
FROM
   production.products p
INNER JOIN production.categories c ON c.category_id = p.category_id
INNER JOIN production.brands b ON b.brand_id = p.brand_id
ORDER BY
   product_name DESC;
```

SQL - CARTESIAN or CROSS JOINS

The basic syntax of the **CARTESIAN JOIN** or the **CROSS JOIN** is as follows −

```
SELECT table1.column1, table2.column2...
FROM  table1, table2 [, table3 ]
```


```
SELECT table1.column1, table2.column2...
FROM  table1, table2 [, table3 ]
```


SQL - SELF JOINS

```
SELECT a.column_name, b.column_name...
FROM table1 a, table1 b
WHERE a.common_field = b.common_field;
```

Using WHERE with Inner Join

```
SELECT pz.CompanyId, pz.CompanyCity, pz.CompanyName,f.ItemName, f.UnitsSold
FROM PizzaCompany pz
INNER JOIN Foods f ON pz.CompanyId = f.CompanyId
WHERE f.UnitsSold > 6
ORDER BY pz.CompanyCity
```

Using Group By with Inner Join

```
SELECT pz.CompanyCity, pz.CompanyName, SUM(f.UnitsSold) AS TotalQuantitySold
FROM PizzaCompany pz
INNER JOIN Foods f ON pz.CompanyId = f.CompanyId
GROUP BY pz.CompanyCity, pz.CompanyName
ORDER BY pz.CompanyCity
```

A brief note on Equi and Non-Equi(Theta) Join

Equi Join

As the name suggests, equi join contains an equality operator '=' either in the Join clause or in the WHERE condition. SQL Inner, Left, Right are all equi joins when '=' operator is being used as a

comparison operator. Usually, when there is a mention of SQL Inner Join, it is considered as an Inner equi Join, in an unusual situation only, equality operator is not used.

Theta Join (Non-equi join)

Non-equi join is basically opposite of equi-join and is used when we join on a condition other than '=' operator. This type is rarely used in practice. Below is an example that makes use of theta join with an inequality operator (<) to evaluate profit by estimating cost and selling prices in two tables.

```
SELECT * FROM Table1 T1, Table2 T2 WHERE T1.ProductCost < T2.SalesPrice
```

---

salesman

salesman_id	name	city	commission
5001	James Hoog	New York	0.15
5002	Nail Knite	Paris	0.13
5005	Pit Alex	London	0.11
5006	Mc Lyon	Paris	0.14
5007	Paul Adam	Rome	0.13
5003	Lauson Hen	San Jose	0.12

cutsomer

customer_id	cust_name	city	grade	salesman_id
3002	Nick Rimando	New York	100	5001
3007	Brad Davis	New York	200	5001
3005	Graham Zusi	California	200	5002
3008	Julian Green	London	300	5002
3004	Fabian Johnson	Paris	300	5006

3009 | Geoff Cameron | Berlin | 100 | 5003

3003 | Jozy Altidor | Moscow | 200 | 5007

orders

ord_no	purch_amt	ord_date	customer_id	salesman_id
70001	150.5	2012-10-05	3005	5002
70009	270.65	2012-09-10	3001	5005
70002	65.26	2012-10-05	3002	5001
70004	110.5	2012-08-17	3009	5003
70007	948.5	2012-09-10	3005	5002
70005	2400.6	2012-07-27	3007	5001
70008	5760	2012-09-10	3002	5001

Write a SQL statement to prepare a list with salesman name, customer name and their cities for the salesmen and customer who belongs to the same city.

select s.name, c.cust_name, s.city from salesman as s, customer as c where s.city=c.city;

Comments:

-This is supposed to test knowledge of joins, yet there's no join in the solution. The question is not worded well.

It should ask that we produce a list of any combination of customers/salesman as long as they live in the same city.

-To join two or more tables, it is not necessary to use JOIN keyword always.

FROM customer, salesman >> this is cartesian join

FROM customer c JOIN salesman s >> which you used, is inner join. so you missed the results derived from salesman to customer

Write a SQL statement to make a list with order no, purchase amount, customer name and their cities for those orders which order amount between 500 and 2000.

select o.ord_no, o.purch_amt,c.cust_name,c.city from orders as o,customer as c where o.customer_id=c.customer_id and o.purch_amt between 500 and 2000;
