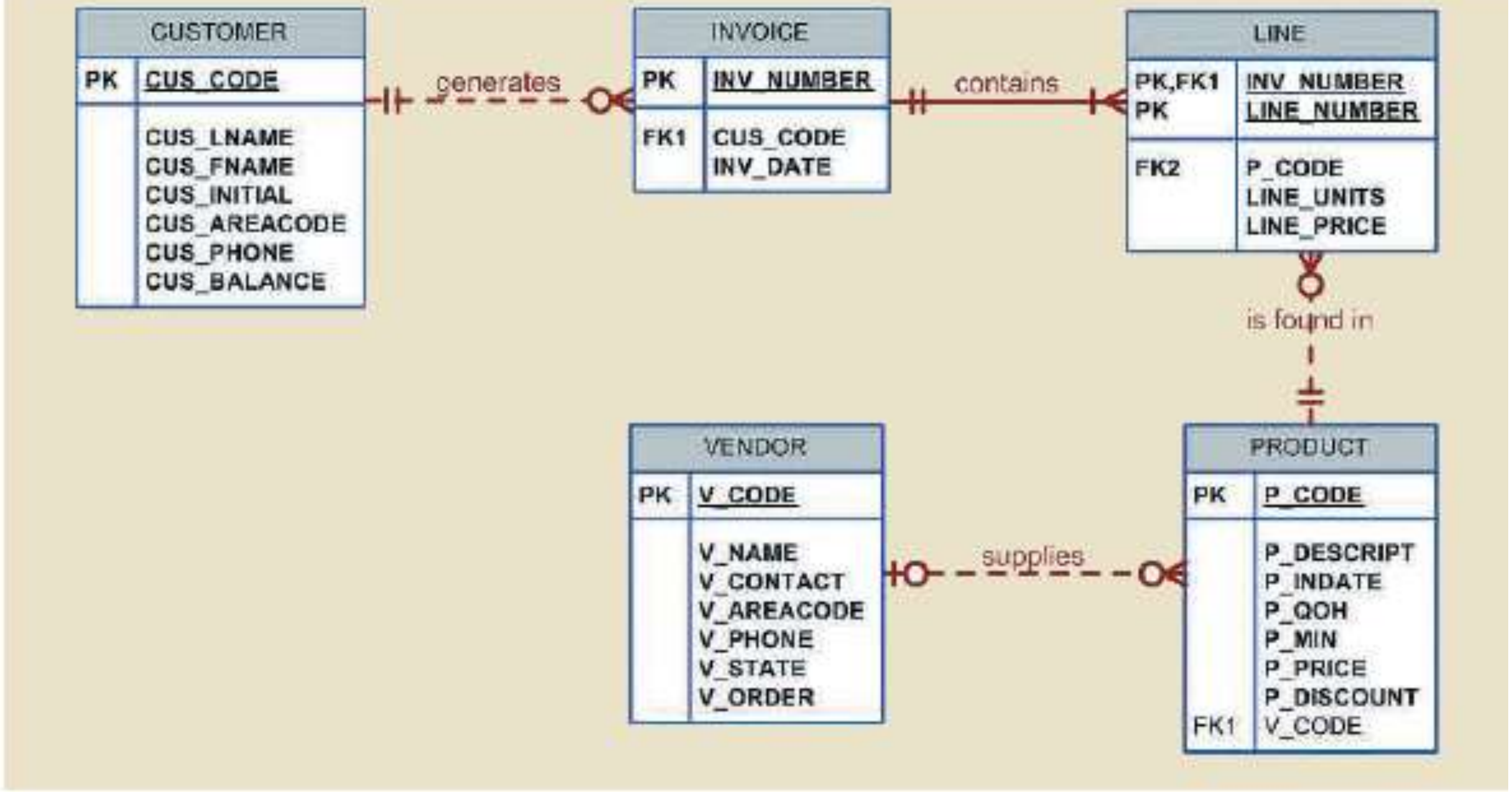
Advanced SQL

Objectives

- Use Advanced SQL JOIN Syntax in PostgreSQL
- Understand and Use Subqueries and Correlated Subqueries
- Manipulate Data Using PostgreSQL SQL Functions
- Apply Relational Set Operators in PostgreSQL
- Create and Use Views and Updatable Views
- Create and Use Triggers and Stored Procedures
- Create Embedded SQL

JOIN

FIGURE 7.1 THE DATABASE MODEL



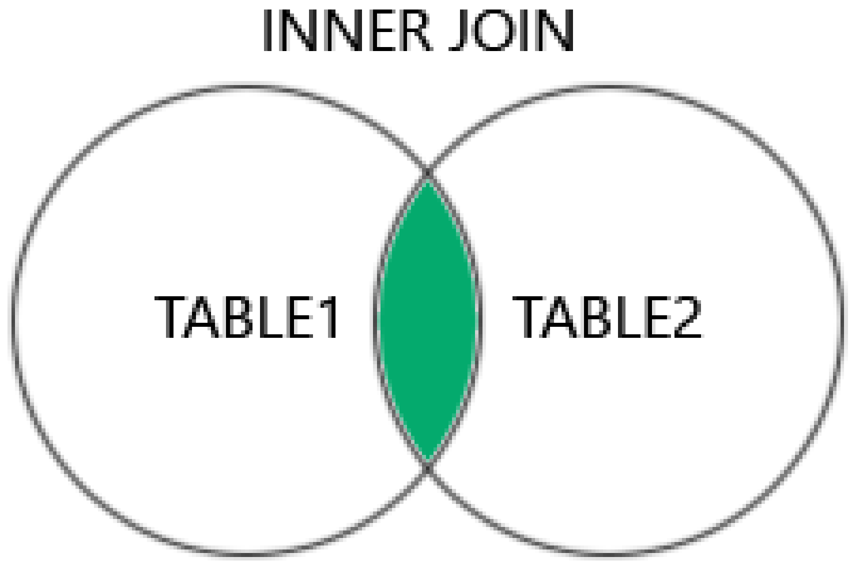
JOIN

- A JOIN in SQL lets you combine rows from two or more tables based on a related column between them.
- Basic Types of JOINs in PostgreSQL
    - INNER JOIN
    - LEFT JOIN
    - RIGHT JOIN
    - FULL JOIN
    - CROSS JOIN

INNER JOIN

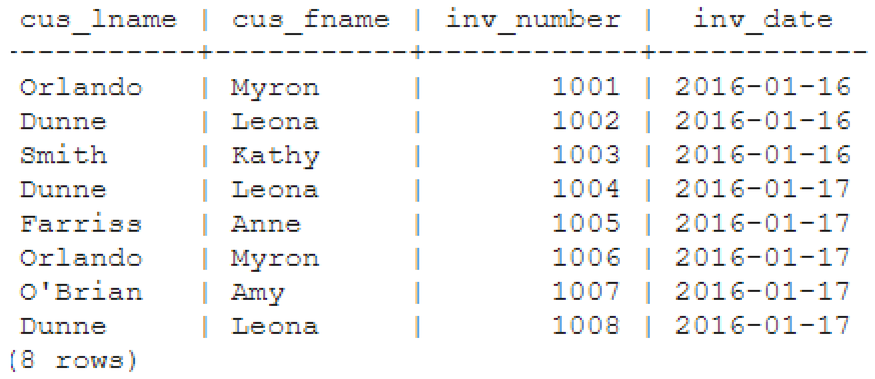
An **INNER JOIN** retrieves rows from two or more tables based on a related column between them. Rows are included in the result only when there is a match in both tables.

```
SELECT COLUMN_NAMES
FROM TABLE1 AS T1
INNER JOIN TABLE2 AS T2
ON T1.COLUMN_NAME =
T2.COLUMN_NAME;
```



INNER JOIN

```
SELECT C.CUS_LNAME, C.CUS_FNAME,
I.INV_NUMBER, I.INV_DATE
FROM CUSTOMER AS C
INNER JOIN INVOICE AS I
ON C.CUS_CODE = I.CUS_CODE;
```



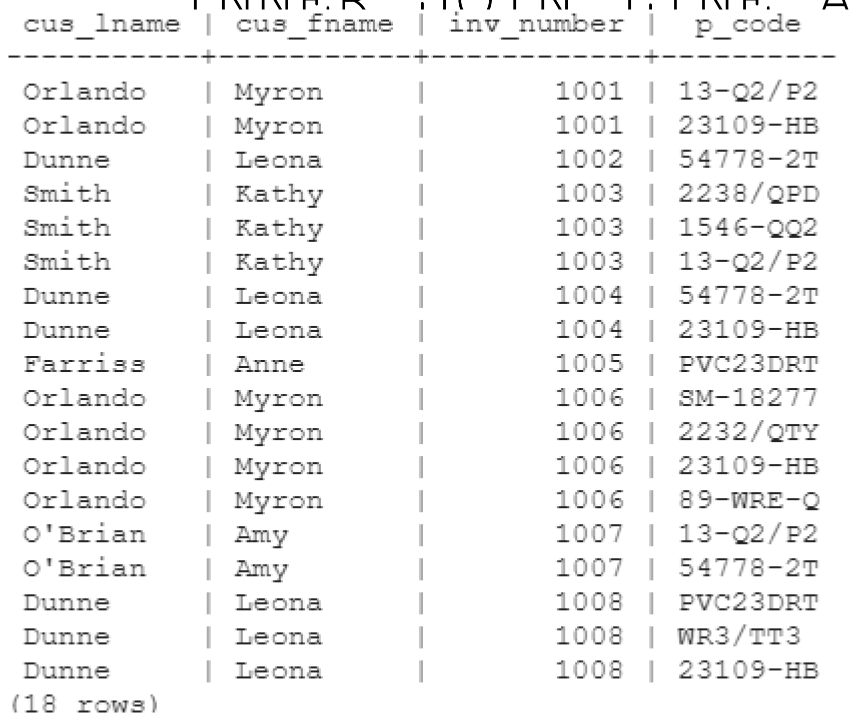
```
 cus_lname | cus_fname | inv_number |  inv_date
-----------+-----------+------------+------------
 Orlando   | Myron     |       1001 | 2016-01-16
 Dunne     | Leona     |       1002 | 2016-01-16
 Smith     | Kathy     |       1003 | 2016-01-16
 Dunne     | Leona     |       1004 | 2016-01-17
 Farriss   | Anne      |       1005 | 2016-01-17
 Orlando   | Myron     |       1006 | 2016-01-17
 O'Brian   | Amy       |       1007 | 2016-01-17
 Dunne     | Leona     |       1008 | 2016-01-17
(8 rows)
```

```
**w/o JOIN keyword
SELECT C.CUS_LNAME, C.CUS_FNAME,
I.INV_NUMBER, I.INV_DATE
FROM CUSTOMER AS C, INVOICE AS I
WHERE C.CUS_CODE = I.CUS_CODE
```

The query retrieves the customer's last name, first name, invoice number,
and invoice date where there is a matching CUS_CODE in both the Customer and Invoice tables.

INNER JOIN

```
SELECT C.CUS_LNAME, C.CUS_FNAME, I.INV_NUMBER,
L.P_CODE
FROM CUSTOMER AS C
INNER JOIN INVOICE AS I ON C.CUS_CODE = I.CUS_CODE
INNER JOIN LINE AS L ON I.INV_NUMBER = L.INV_NUMBER;
```



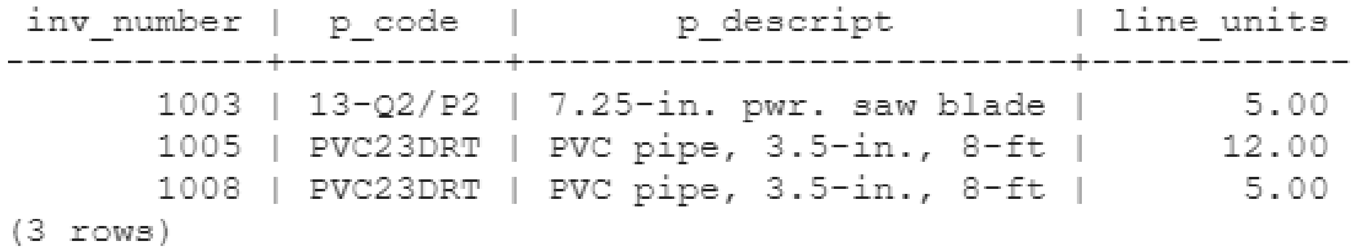
```
 cus_lname | cus_fname | inv_number |  p_code
-----------+-----------+------------+----------
 Orlando   | Myron     |       1001 | 13-Q2/P2
 Orlando   | Myron     |       1001 | 23109-HB
 Dunne     | Leona     |       1002 | 54778-2T
 Smith     | Kathy     |       1003 | 2238/QPD
 Smith     | Kathy     |       1003 | 1546-QQ2
 Smith     | Kathy     |       1003 | 13-Q2/P2
 Dunne     | Leona     |       1004 | 54778-2T
 Dunne     | Leona     |       1004 | 23109-HB
 Farriss   | Anne      |       1005 | PVC23DRT
 Orlando   | Myron     |       1006 | SM-18277
 Orlando   | Myron     |       1006 | 2232/QTY
 Orlando   | Myron     |       1006 | 23109-HB
 Orlando   | Myron     |       1006 | 89-WRE-Q
 O'Brian   | Amy       |       1007 | 13-Q2/P2
 O'Brian   | Amy       |       1007 | 54778-2T
 Dunne     | Leona     |       1008 | PVC23DRT
 Dunne     | Leona     |       1008 | WR3/TT3
 Dunne     | Leona     |       1008 | 23109-HB
(18 rows)
```

**w/o ALIAS

```
SELECT CUSTOMER.CUS_LNAME, CUSTOMER.CUS_FNAME,
INVOICE.INV_NUMBER, LINE.P_CODE
FROM CUSTOMER
INNER JOIN INVOICE ON CUSTOMER.CUS_CODE =
INVOICE.CUS_CODE
INNER JOIN LINE ON INVOICE.INV_NUMBER = LINE.INV_NUMBER;
```

INNER JOIN with Filtering

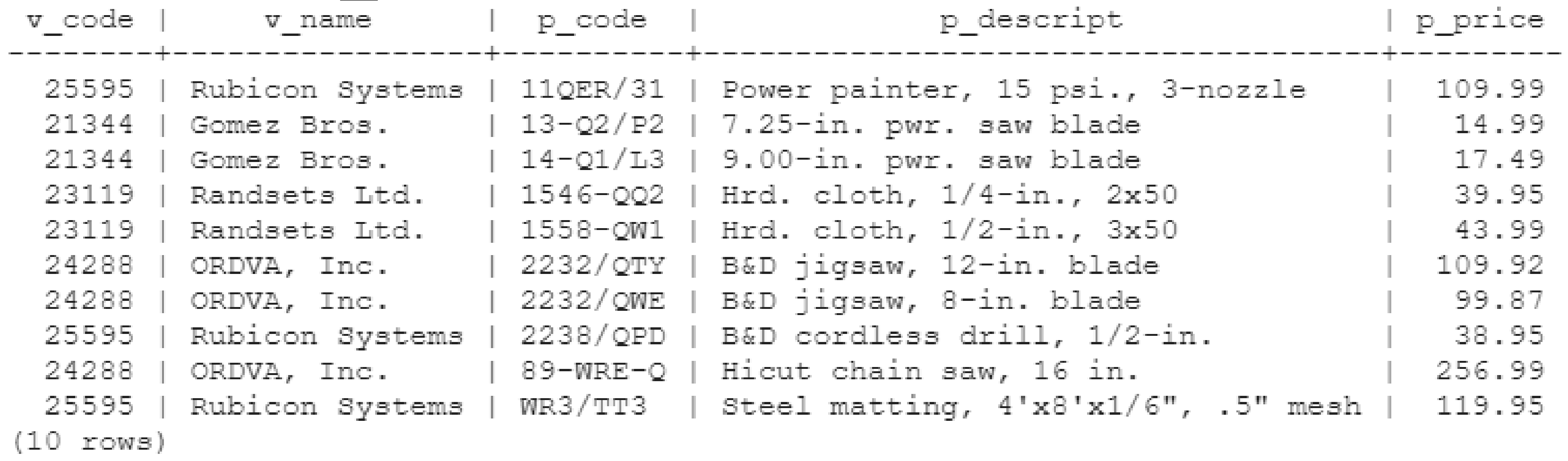
```
SELECT L.INV_NUMBER, L.P_CODE, P.P_DESCRIPT,
L.LINE_UNITS
FROM LINE AS L
INNER JOIN PRODUCT AS P ON L.P_CODE = P.P_CODE
WHERE L.LINE_UNITS > 3.0;
```



```
 inv_number |  p_code  |       p_descript        | line_units
------------+----------+-------------------------+------------
       1003 | 13-Q2/P2 | 7.25-in. pwr. saw blade |       5.00
       1005 | PVC23DRT | PVC pipe, 3.5-in., 8-ft |      12.00
       1008 | PVC23DRT | PVC pipe, 3.5-in., 8-ft |       5.00
(3 rows)
```

INNER JOIN with Filtering

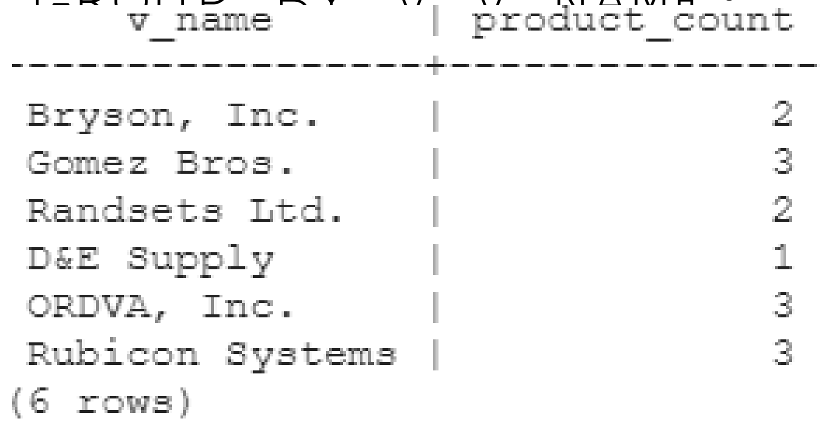
```
SELECT V.V_CODE, V_NAME, P.P_CODE, P.P_DESCRIPT,
P_PRICE
FROM VENDOR AS V
INNER JOIN PRODUCT AS P ON P.V_CODE = V.V_CODE
WHERE P.P_PRICE > 10;
```



```
 v_code |     v_name      |  p_code  |             p_descript              | p_price
--------+-----------------+----------+-------------------------------------+---------
  25595 | Rubicon Systems | 11QER/31 | Power painter, 15 psi., 3-nozzle    |  109.99
  21344 | Gomez Bros.     | 13-Q2/P2 | 7.25-in. pwr. saw blade             |   14.99
  21344 | Gomez Bros.     | 14-Q1/L3 | 9.00-in. pwr. saw blade             |   17.49
  23119 | Randsets Ltd.   | 1546-QQ2 | Hrd. cloth, 1/4-in., 2x50           |   39.95
  23119 | Randsets Ltd.   | 1558-QW1 | Hrd. cloth, 1/2-in., 3x50           |   43.99
  24288 | ORDVA, Inc.     | 2232/QTY | B&D jigsaw, 12-in. blade            |  109.92
  24288 | ORDVA, Inc.     | 2232/QWE | B&D jigsaw, 8-in. blade             |   99.87
  25595 | Rubicon Systems | 2238/QPD | B&D cordless drill, 1/2-in.         |   38.95
  24288 | ORDVA, Inc.     | 89-WRE-Q | Hicut chain saw, 16 in.             |  256.99
  25595 | Rubicon Systems | WR3/TT3  | Steel matting, 4'x8'x1/6", .5" mesh |  119.95
(10 rows)
```

INNER JOIN with Filtering

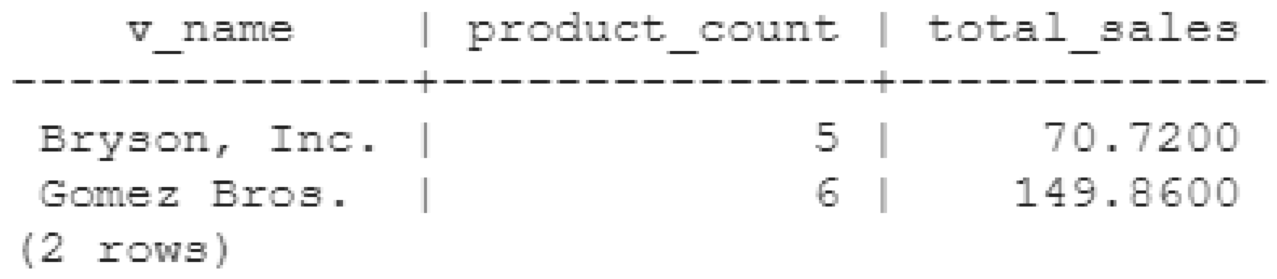
```
SELECT V.V_NAME,COUNT(P.P_CODE) AS
PRODUCT_COUNT
FROM VENDOR AS V
INNER JOIN PRODUCT AS P ON V.V_CODE = P.V_CODE
GROUP BY V.V_NAME;
```



```
     v_name      | product_count
-----------------+---------------
 Bryson, Inc.    |             2
 Gomez Bros.     |             3
 Randsets Ltd.   |             2
 D&E Supply      |             1
 ORDVA, Inc.     |             3
 Rubicon Systems |             3
(6 rows)
```

INNER JOIN with Filtering

```
SELECT V.V_NAME, COUNT(P.P_CODE) AS PRODUCT_COUNT, SUM(L.LINE_UNITS *
L.LINE_PRICE) AS TOTAL_SALES
FROM VENDOR AS V
INNER JOIN PRODUCT AS P ON V.V_CODE = P.V_CODE
INNER JOIN LINE AS L ON P.P_CODE = L.P_CODE
GROUP BY V.V_NAME
HAVING COUNT(P.P_CODE) > 2;
```

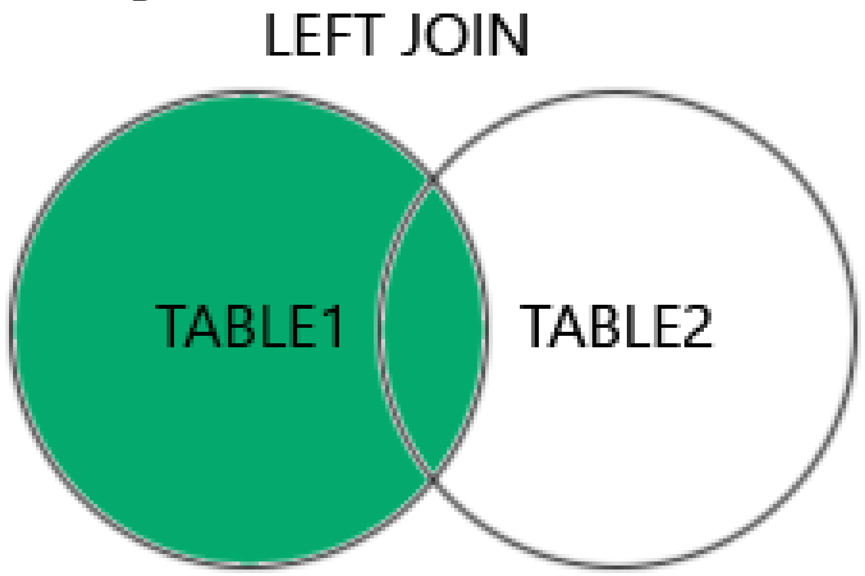


```
    v_name    | product_count | total_sales
--------------+---------------+-------------
 Bryson, Inc. |             5 |     70.7200
 Gomez Bros.  |             6 |    149.8600
(2 rows)
```

LEFT JOIN

A LEFT JOIN (or LEFT OUTER JOIN) returns all rows from the left table and the matching rows from the right table. If no match exists, the result is NULL on the side of the right table.
Keep everything on the left

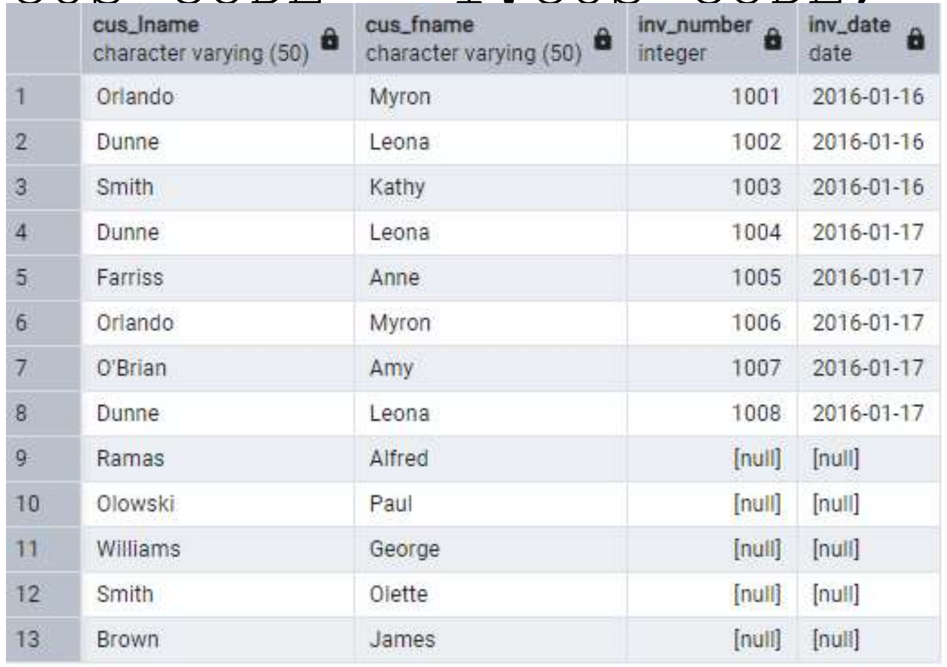
```
SELECT  COLUMN1, COLUMN2, ...
FROM  LEFT_TABLE AS L
LEFT JOIN   RIGHT_TABLE AS R ON L.COLUMN =
R.COLUMN;
```



LEFT JOIN

```
SELECT C.CUS_LNAME, C.CUS_FNAME, I.INV_NUMBER,
I.INV_DATE
FROM CUSTOMER AS C
LEFT JOIN INVOICE AS I ON C.CUS_CODE = I.CUS_CODE;
```

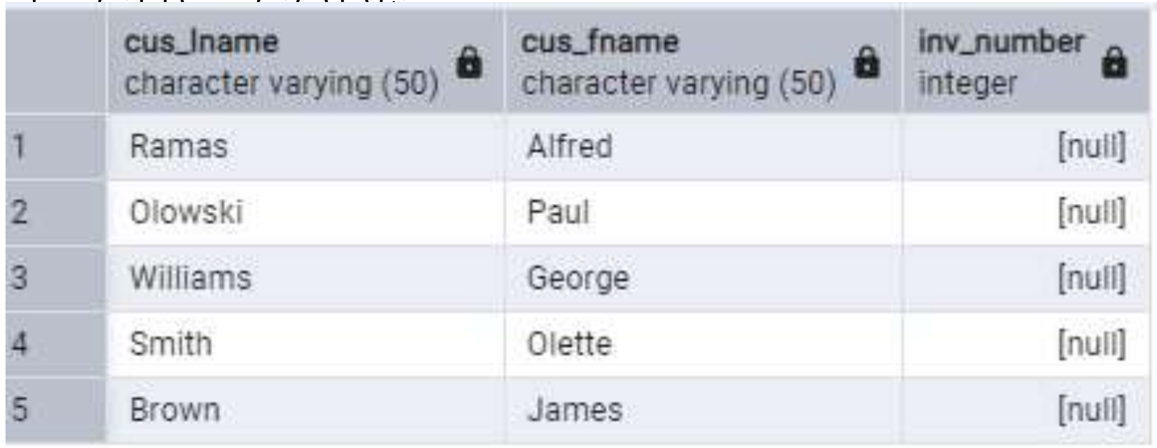
```
 cus_lname | cus_fname | inv_number |  inv_date
-----------+-----------+------------+------------
 Orlando   | Myron     |       1001 | 2016-01-16
 Dunne     | Leona     |       1002 | 2016-01-16
 Smith     | Kathy     |       1003 | 2016-01-16
 Dunne     | Leona     |       1004 | 2016-01-17
 Farriss   | Anne      |       1005 | 2016-01-17
 Orlando   | Myron     |       1006 | 2016-01-17
 O'Brian   | Amy       |       1007 | 2016-01-17
 Dunne     | Leona     |       1008 | 2016-01-17
 Ramas     | Alfred    |            |
 Olowski   | Paul      |            |
 Williams  | George    |            |
 Smith     | Olette    |            |
 Brown     | James     |            |
(13 rows)
```



	cus_lname character varying (50)	cus_fname character varying (50)	inv_number integer	inv_date date
1	Orlando	Myron	1001	2016-01-16
2	Dunne	Leona	1002	2016-01-16
3	Smith	Kathy	1003	2016-01-16
4	Dunne	Leona	1004	2016-01-17
5	Farriss	Anne	1005	2016-01-17
6	Orlando	Myron	1006	2016-01-17
7	O'Brian	Amy	1007	2016-01-17
8	Dunne	Leona	1008	2016-01-17
9	Ramas	Alfred	[null]	[null]
10	Olowski	Paul	[null]	[null]
11	Williams	George	[null]	[null]
12	Smith	Olette	[null]	[null]
13	Brown	James	[null]	[null]

LEFT JOIN with Filtering

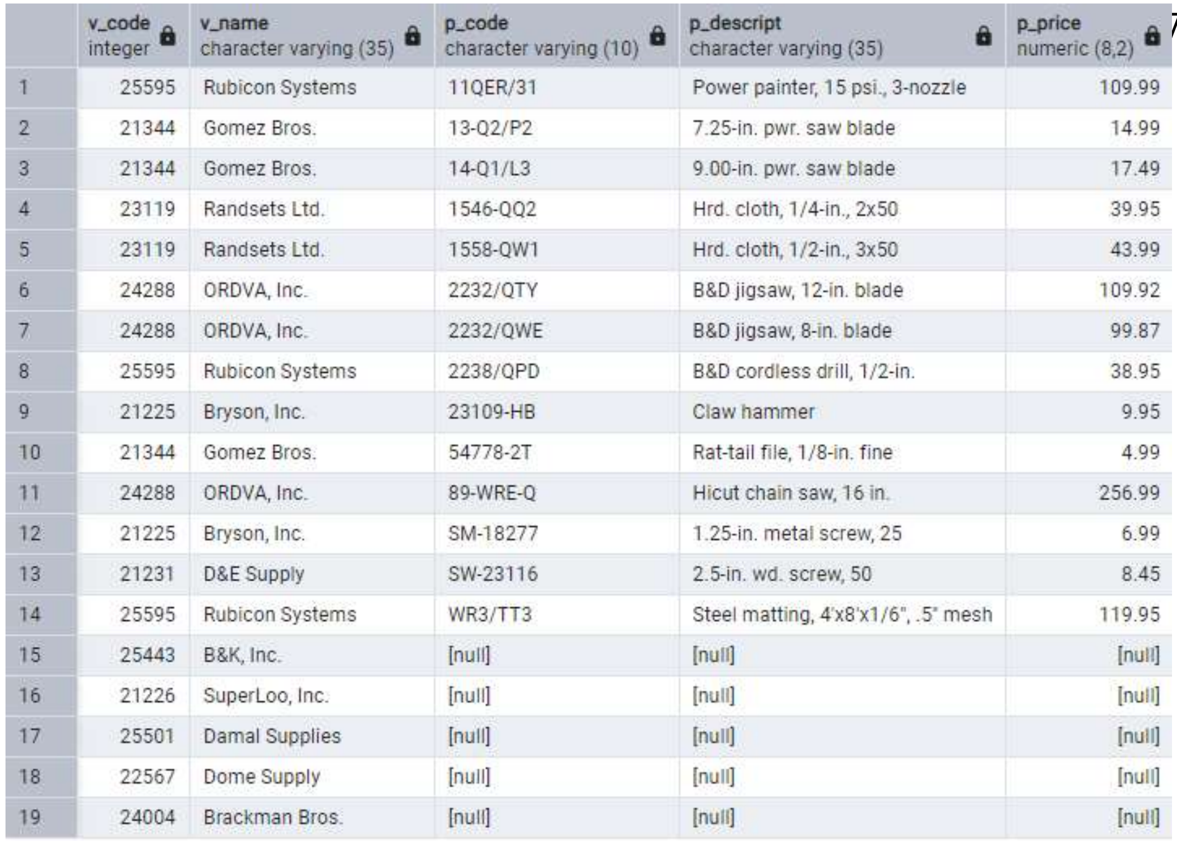
```
SELECT C.CUS_LNAME, C.CUS_FNAME,
I.INV_NUMBER
FROM CUSTOMER AS C
LEFT JOIN INVOICE AS I ON C.CUS_CODE =
I.CUS_CODE
```



	cus_lname character varying (50)	cus_fname character varying (50)	inv_number integer
1	Ramas	Alfred	[null]
2	Olowski	Paul	[null]
3	Williams	George	[null]
4	Smith	Olette	[null]
5	Brown	James	[null]

LEFT JOIN

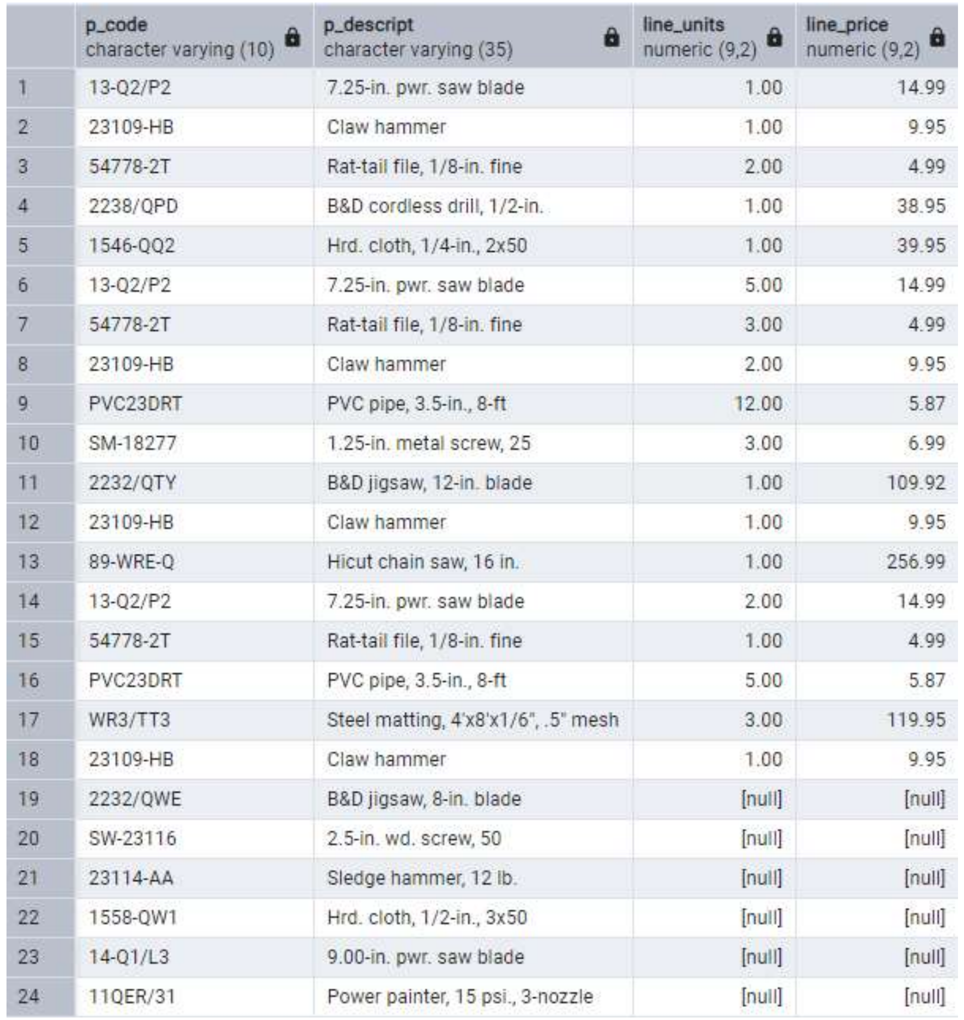
```
SELECT V.V_CODE, V.V_NAME, P.P_CODE, P_DESCRIPT,
P.P_PRICE
FROM VENDOR AS V
                                   _CODE = P.V_CODE;
```



	v_code integer	v_name character varying (35)	p_code character varying (10)	p_descript character varying (35)	p_price numeric (8,2)
1	25595	Rubicon Systems	11QER/31	Power painter, 15 psi., 3-nozzle	109.99
2	21344	Gomez Bros.	13-Q2/P2	7.25-in. pwr. saw blade	14.99
3	21344	Gomez Bros.	14-Q1/L3	9.00-in. pwr. saw blade	17.49
4	23119	Randsets Ltd.	1546-QQ2	Hrd. cloth, 1/4-in., 2x50	39.95
5	23119	Randsets Ltd.	1558-QW1	Hrd. cloth, 1/2-in., 3x50	43.99
6	24288	ORDVA, Inc.	2232/QTY	B&D jigsaw, 12-in. blade	109.92
7	24288	ORDVA, Inc.	2232/QWE	B&D jigsaw, 8-in. blade	99.87
8	25595	Rubicon Systems	2238/QPD	B&D cordless drill, 1/2-in.	38.95
9	21225	Bryson, Inc.	23109-HB	Claw hammer	9.95
10	21344	Gomez Bros.	54778-2T	Rat-tail file, 1/8-in. fine	4.99
11	24288	ORDVA, Inc.	89-WRE-Q	Hicut chain saw, 16 in.	256.99
12	21225	Bryson, Inc.	SM-18277	1.25-in. metal screw, 25	6.99
13	21231	D&E Supply	SW-23116	2.5-in. wd. screw, 50	8.45
14	25595	Rubicon Systems	WR3/TT3	Steel matting, 4'x8'x1/6", .5" mesh	119.95
15	25443	B&K, Inc.	[null]	[null]	[null]
16	21226	SuperLoo, Inc.	[null]	[null]	[null]
17	25501	Damal Supplies	[null]	[null]	[null]
18	22567	Dome Supply	[null]	[null]	[null]
19	24004	Brackman Bros.	[null]	[null]	[null]

LEFT JOIN

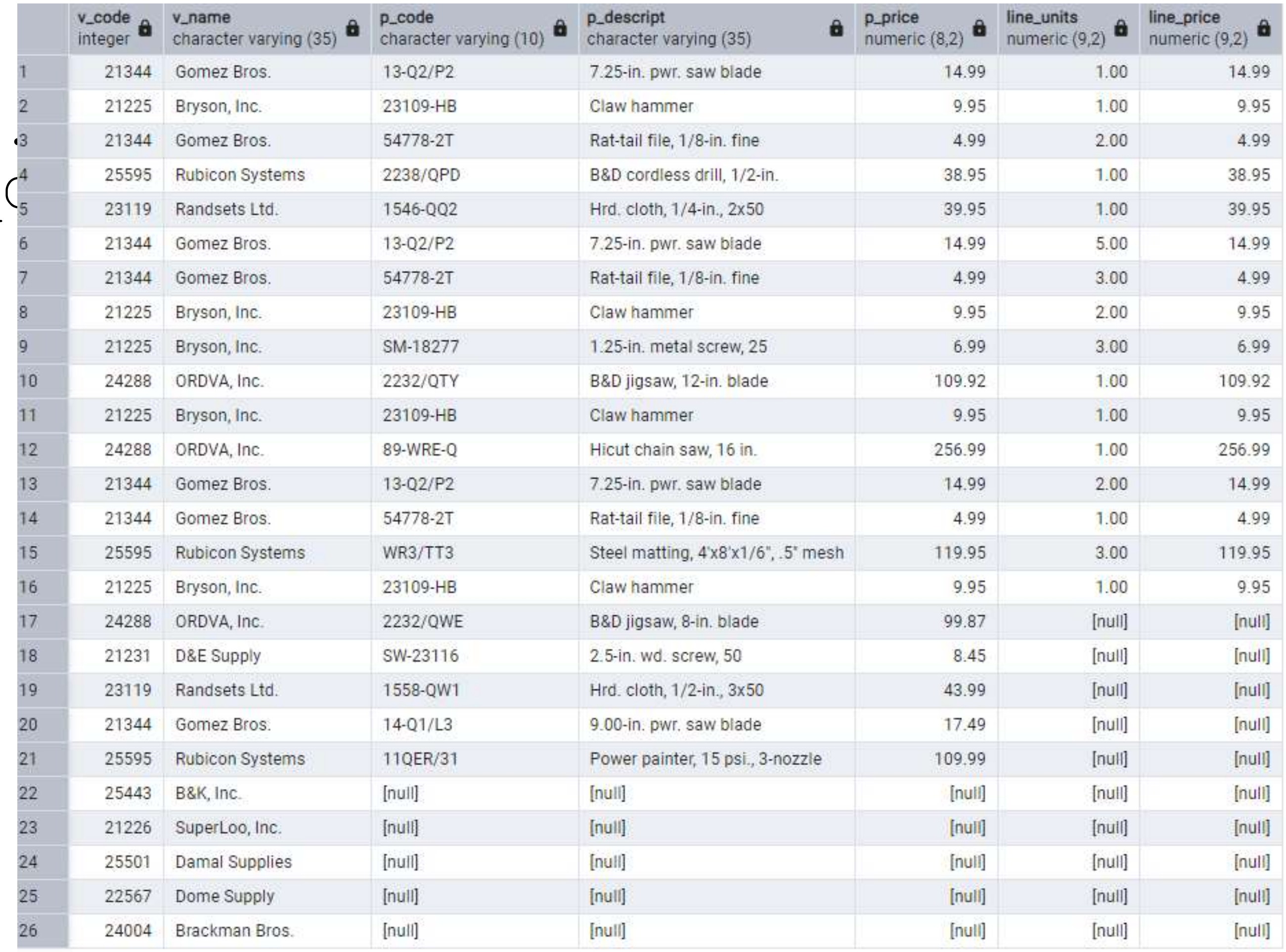
```
SELECT P.P_CODE, P.P_DESCRIPT, L.LINE_UNITS,
L.LINE_PRICE
FROM PRODUCT AS P
LEFT JOIN LINE AS L ON P.P_CODE = L
```



	p_code character varying (10)	p_descript character varying (35)	line_units numeric (9,2)	line_price numeric (9,2)
1	13-Q2/P2	7.25-in. pwr. saw blade	1.00	14.99
2	23109-HB	Claw hammer	1.00	9.95
3	54778-2T	Rat-tail file, 1/8-in. fine	2.00	4.99
4	2238/QPD	B&D cordless drill, 1/2-in.	1.00	38.95
5	1546-QQ2	Hrd. cloth, 1/4-in., 2x50	1.00	39.95
6	13-Q2/P2	7.25-in. pwr. saw blade	5.00	14.99
7	54778-2T	Rat-tail file, 1/8-in. fine	3.00	4.99
8	23109-HB	Claw hammer	2.00	9.95
9	PVC23DRT	PVC pipe, 3.5-in., 8-ft	12.00	5.87
10	SM-18277	1.25-in. metal screw, 25	3.00	6.99
11	2232/QTY	B&D jigsaw, 12-in. blade	1.00	109.92
12	23109-HB	Claw hammer	1.00	9.95
13	89-WRE-Q	Hicut chain saw, 16 in.	1.00	256.99
14	13-Q2/P2	7.25-in. pwr. saw blade	2.00	14.99
15	54778-2T	Rat-tail file, 1/8-in. fine	1.00	4.99
16	PVC23DRT	PVC pipe, 3.5-in., 8-ft	5.00	5.87
17	WR3/TT3	Steel matting, 4'x8'x1/6", .5" mesh	3.00	119.95
18	23109-HB	Claw hammer	1.00	9.95
19	2232/QWE	B&D jigsaw, 8-in. blade	[null]	[null]
20	SW-23116	2.5-in. wd. screw, 50	[null]	[null]
21	23114-AA	Sledge hammer, 12 lb.	[null]	[null]
22	1558-QW1	Hrd. cloth, 1/2-in., 3x50	[null]	[null]
23	14-Q1/L3	9.00-in. pwr. saw blade	[null]	[null]
24	11QER/31	Power painter, 15 psi., 3-nozzle	[null]	[null]

LEFT JOIN

```
SELECT V.V_CODE, V.V_NAME, L.P_CODE, P_DESCRIPT, P.P_PRICE,
L.LINE_UNITS, L.LINE_PRICE
FROM VENDOR AS V
LEFT JOIN PRODUCT AS P ON V.V_CODE = P.
LEFT JOIN LINE AS L ON P.P_CODE = L.P_
```

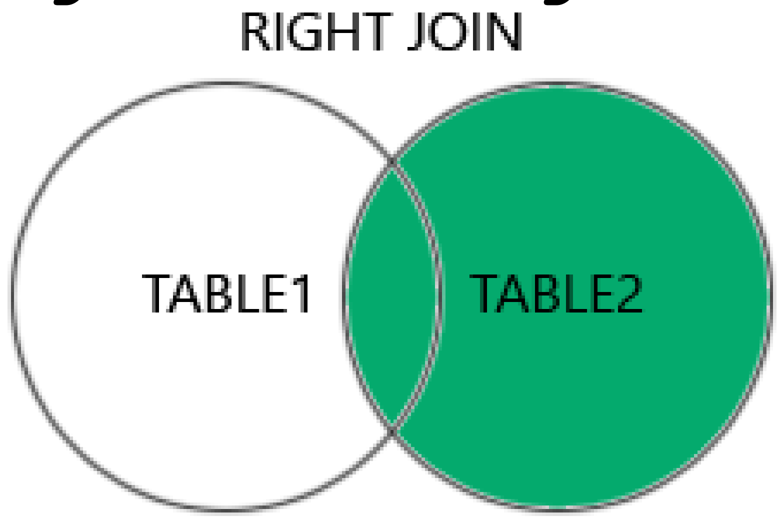


	v_code integer	v_name character varying (35)	p_code character varying (10)	p_descript character varying (35)	p_price numeric (8,2)	line_units numeric (9,2)	line_price numeric (9,2)
1	21344	Gomez Bros.	13-Q2/P2	7.25-in. pwr. saw blade	14.99	1.00	14.99
2	21225	Bryson, Inc.	23109-HB	Claw hammer	9.95	1.00	9.95
3	21344	Gomez Bros.	54778-2T	Rat-tail file, 1/8-in. fine	4.99	2.00	4.99
4	25595	Rubicon Systems	2238/QPD	B&D cordless drill, 1/2-in.	38.95	1.00	38.95
5	23119	Randsets Ltd.	1546-QQ2	Hrd. cloth, 1/4-in., 2x50	39.95	1.00	39.95
6	21344	Gomez Bros.	13-Q2/P2	7.25-in. pwr. saw blade	14.99	5.00	14.99
7	21344	Gomez Bros.	54778-2T	Rat-tail file, 1/8-in. fine	4.99	3.00	4.99
8	21225	Bryson, Inc.	23109-HB	Claw hammer	9.95	2.00	9.95
9	21225	Bryson, Inc.	SM-18277	1.25-in. metal screw, 25	6.99	3.00	6.99
10	24288	ORDVA, Inc.	2232/QTY	B&D jigsaw, 12-in. blade	109.92	1.00	109.92
11	21225	Bryson, Inc.	23109-HB	Claw hammer	9.95	1.00	9.95
12	24288	ORDVA, Inc.	89-WRE-Q	Hicut chain saw, 16 in.	256.99	1.00	256.99
13	21344	Gomez Bros.	13-Q2/P2	7.25-in. pwr. saw blade	14.99	2.00	14.99
14	21344	Gomez Bros.	54778-2T	Rat-tail file, 1/8-in. fine	4.99	1.00	4.99
15	25595	Rubicon Systems	WR3/TT3	Steel matting, 4'x8'x1/6", .5" mesh	119.95	3.00	119.95
16	21225	Bryson, Inc.	23109-HB	Claw hammer	9.95	1.00	9.95
17	24288	ORDVA, Inc.	2232/QWE	B&D jigsaw, 8-in. blade	99.87	[null]	[null]
18	21231	D&E Supply	SW-23116	2.5-in. wd. screw, 50	8.45	[null]	[null]
19	23119	Randsets Ltd.	1558-QW1	Hrd. cloth, 1/2-in., 3x50	43.99	[null]	[null]
20	21344	Gomez Bros.	14-Q1/L3	9.00-in. pwr. saw blade	17.49	[null]	[null]
21	25595	Rubicon Systems	11QER/31	Power painter, 15 psi., 3-nozzle	109.99	[null]	[null]
22	25443	B&K, Inc.	[null]	[null]	[null]	[null]	[null]
23	21226	SuperLoo, Inc.	[null]	[null]	[null]	[null]	[null]
24	25501	Damal Supplies	[null]	[null]	[null]	[null]	[null]
25	22567	Dome Supply	[null]	[null]	[null]	[null]	[null]
26	24004	Brackman Bros.	[null]	[null]	[null]	[null]	[null]

RIGHT JOIN

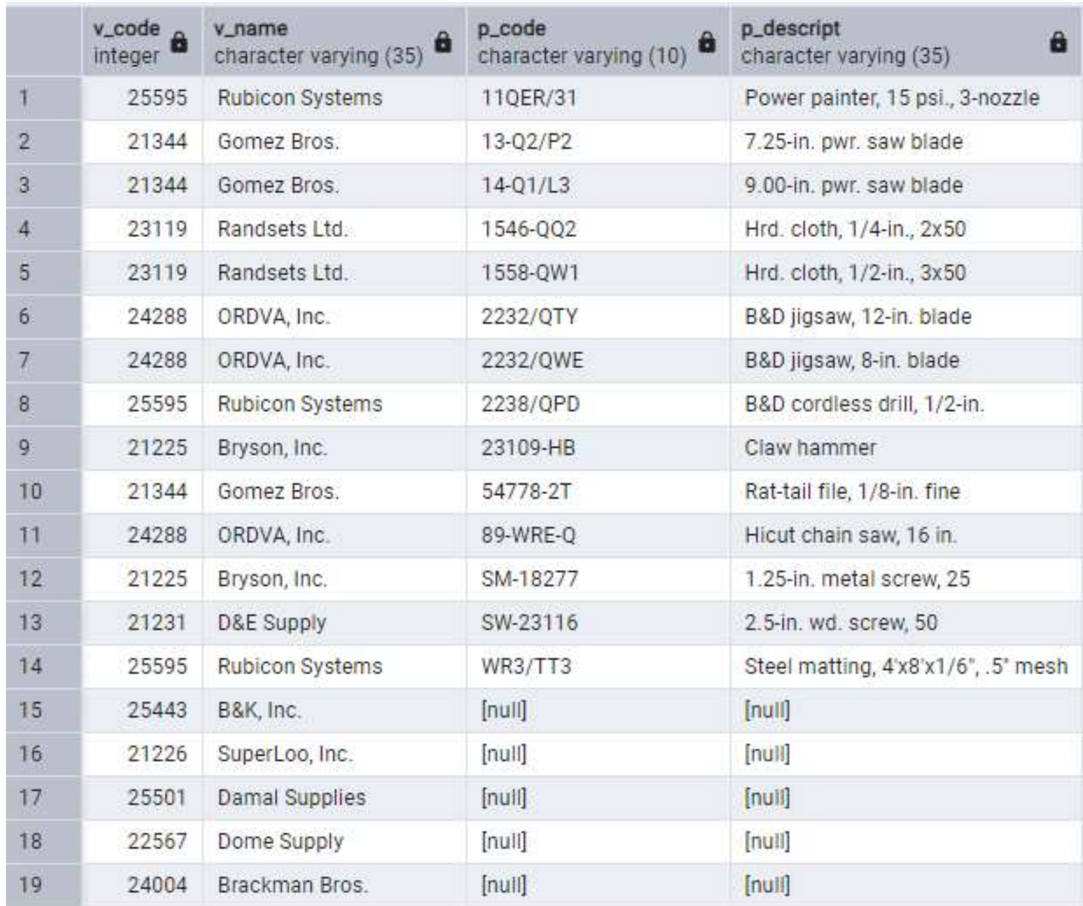
returns all rows from the right table, and the matching rows from the left table. If there's no match from the left table, the result is NULL for the left side. ***Keep everything on the right.***

```
SELECT  COLUMN1, COLUMN2, ...
FROM  RIGHT_TABLE AS R
RIGHT JOIN  LEFT_TABLE AS L ON R.COLUMN =
L.COLUMN;
```



RIGHT JOIN

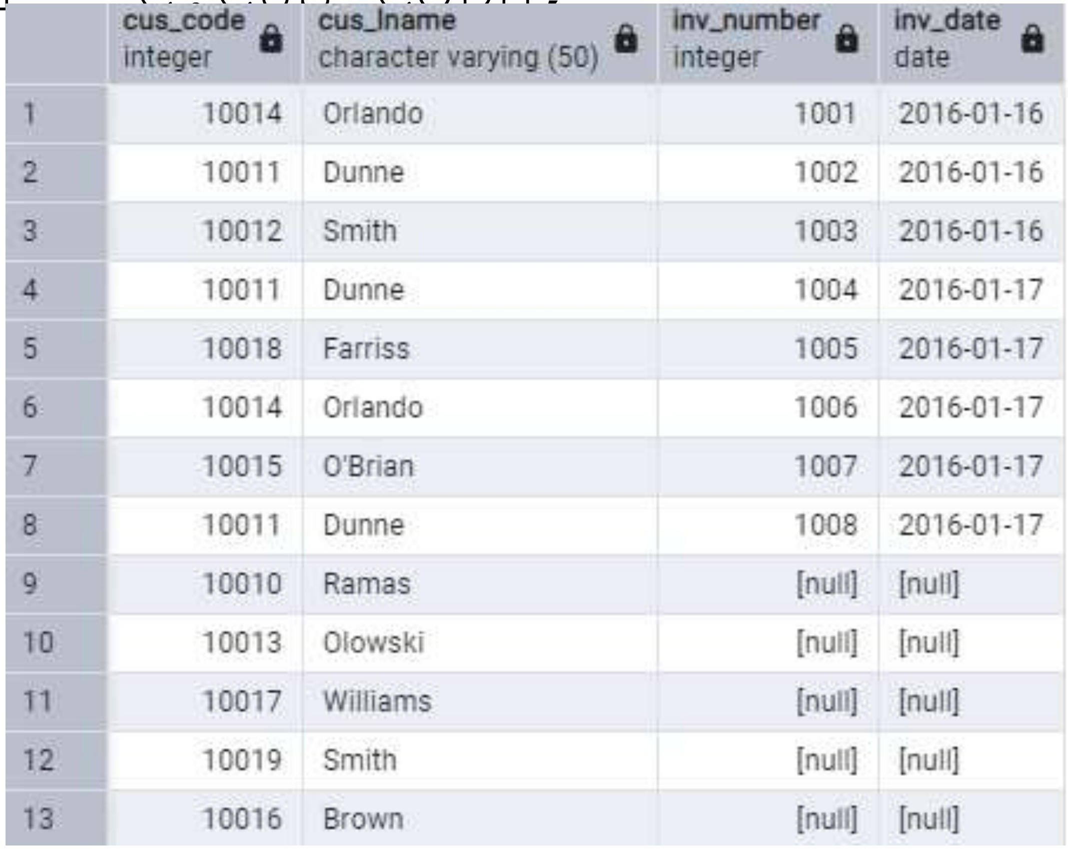
```
SELECT V.V_CODE, V.V_NAME, P.P_CODE, P.P_DESCRIPT
FROM  PRODUCT P
RIGHT JOIN VENDOR V ON P.V_CODE = V.V_CODE;
```



	v_code integer	v_name character varying (35)	p_code character varying (10)	p_descript character varying (35)
1	25595	Rubicon Systems	11QER/31	Power painter, 15 psi., 3-nozzle
2	21344	Gomez Bros.	13-Q2/P2	7.25-in. pwr. saw blade
3	21344	Gomez Bros.	14-Q1/L3	9.00-in. pwr. saw blade
4	23119	Randsets Ltd.	1546-QQ2	Hrd. cloth, 1/4-in., 2x50
5	23119	Randsets Ltd.	1558-QW1	Hrd. cloth, 1/2-in., 3x50
6	24288	ORDVA, Inc.	2232/QTY	B&D jigsaw, 12-in. blade
7	24288	ORDVA, Inc.	2232/QWE	B&D jigsaw, 8-in. blade
8	25595	Rubicon Systems	2238/QPD	B&D cordless drill, 1/2-in.
9	21225	Bryson, Inc.	23109-HB	Claw hammer
10	21344	Gomez Bros.	54778-2T	Rat-tail file, 1/8-in. fine
11	24288	ORDVA, Inc.	89-WRE-Q	Hicut chain saw, 16 in.
12	21225	Bryson, Inc.	SM-18277	1.25-in. metal screw, 25
13	21231	D&E Supply	SW-23116	2.5-in. wd. screw, 50
14	25595	Rubicon Systems	WR3/TT3	Steel matting, 4'x8'x1/6", .5" mesh
15	25443	B&K, Inc.	[null]	[null]
16	21226	SuperLoo, Inc.	[null]	[null]
17	25501	Damal Supplies	[null]	[null]
18	22567	Dome Supply	[null]	[null]
19	24004	Brackman Bros.	[null]	[null]

RIGHT JOIN

```
SELECT C.CUS_CODE, C.CUS_LNAME, I.INV_NUMBER,
I.INV_DATE
FROM INVOICE I
RIGHT JOIN CUSTOMER C ON I.CUS_CODE = C.CUS_CODE;
```

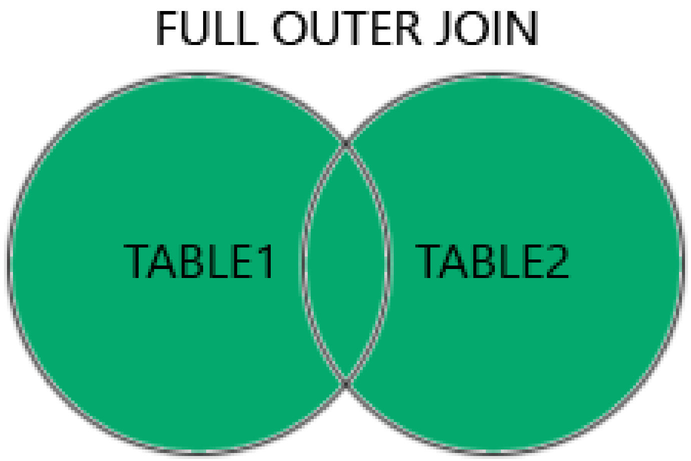


	cus_code integer	cus_lname character varying (50)	inv_number integer	inv_date date
1	10014	Orlando	1001	2016-01-16
2	10011	Dunne	1002	2016-01-16
3	10012	Smith	1003	2016-01-16
4	10011	Dunne	1004	2016-01-17
5	10018	Farriss	1005	2016-01-17
6	10014	Orlando	1006	2016-01-17
7	10015	O'Brian	1007	2016-01-17
8	10011	Dunne	1008	2016-01-17
9	10010	Ramas	[null]	[null]
10	10013	Olowski	[null]	[null]
11	10017	Williams	[null]	[null]
12	10019	Smith	[null]	[null]
13	10016	Brown	[null]	[null]

FULL OUTER JOIN

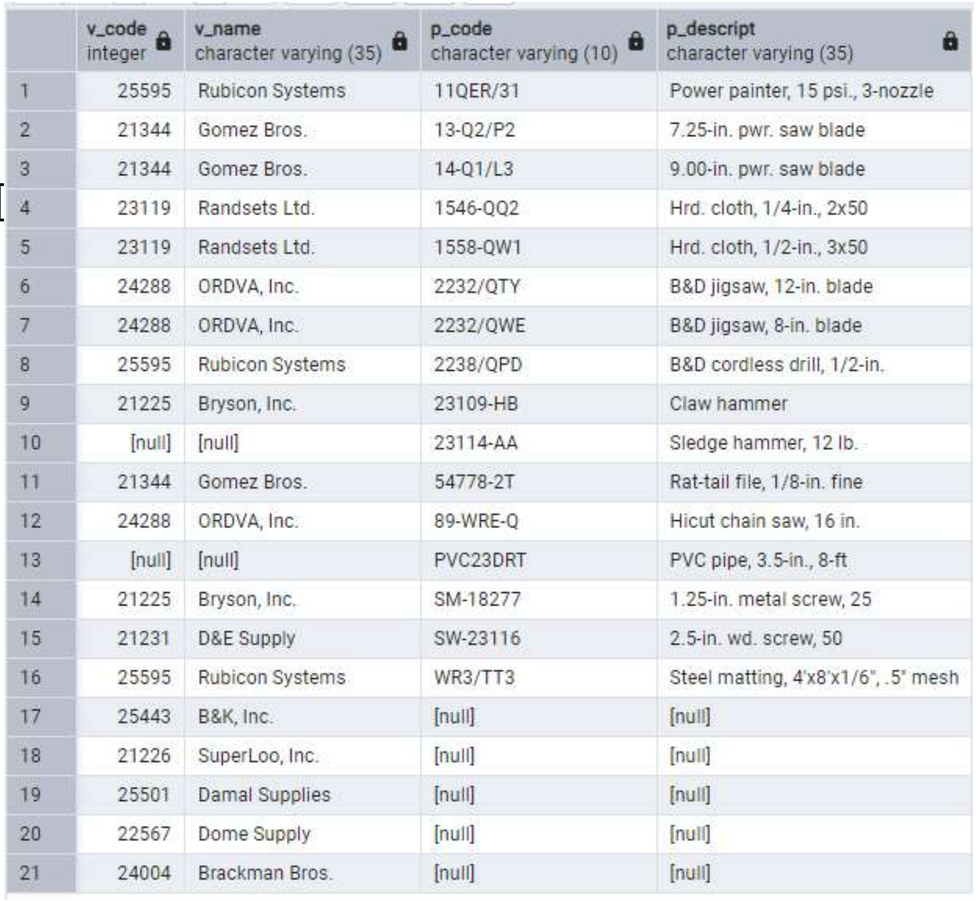
A FULL OUTER JOIN (sometimes just called FULL JOIN) is a type of SQL join that combines the results of both a LEFT JOIN and a RIGHT JOIN. It returns all rows from both participating tables.

```
SELECT column1, column2, ...
FROM table1
FULL OUTER JOIN table2 ON table1.join_column =
table2.join_column;
```



FULL OUTER JOIN

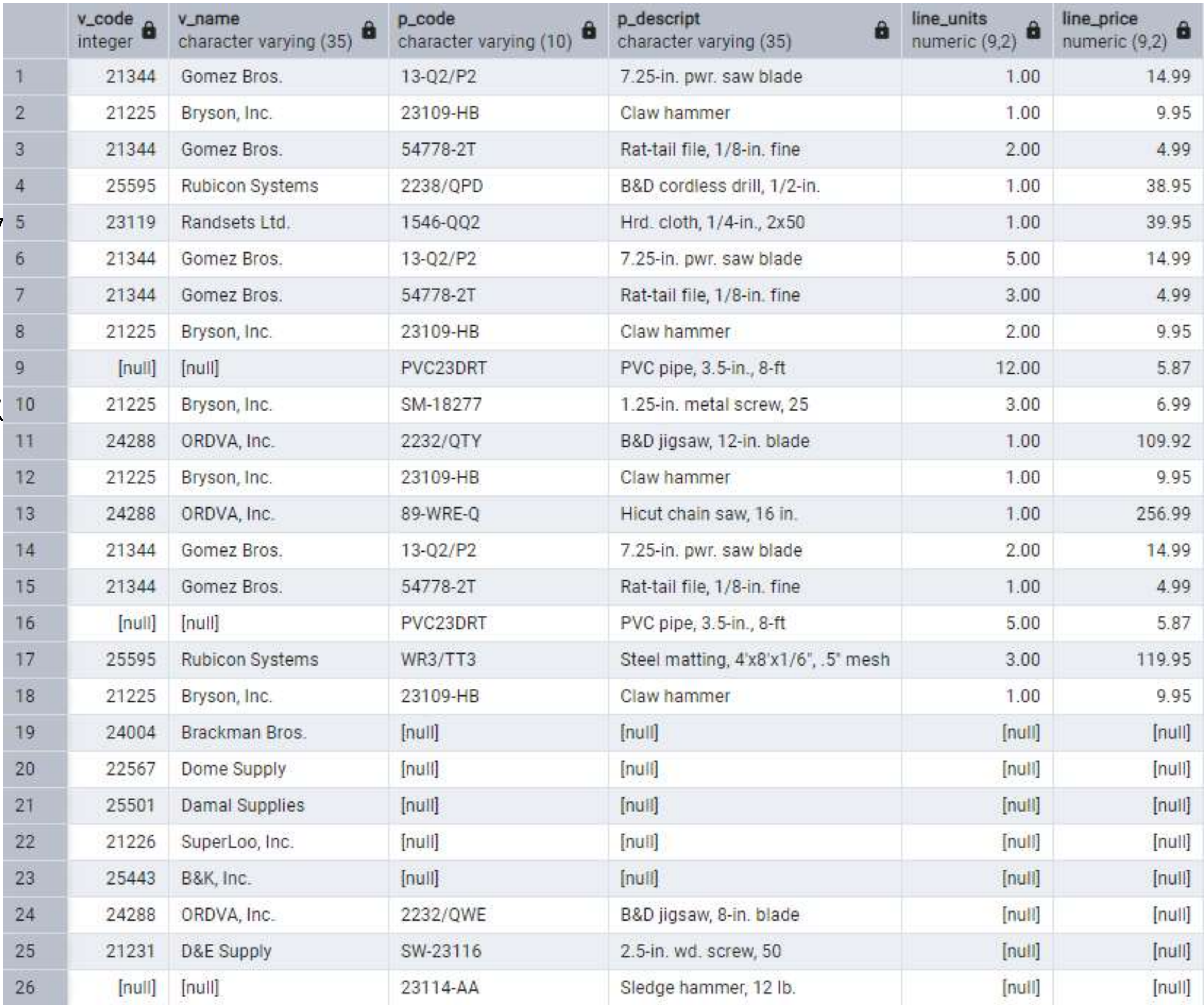
```
SELECT  V.V_CODE, V.V_NAME, P.P_CODE,
P.P_DESCRIPT
FROM  PRODUCT P
FULL OUTER JOIN VENDOR V ON P.V_CO
V.V_CODE;
```



	v_code integer	v_name character varying (35)	p_code character varying (10)	p_descript character varying (35)
1	25595	Rubicon Systems	11QER/31	Power painter, 15 psi., 3-nozzle
2	21344	Gomez Bros.	13-Q2/P2	7.25-in. pwr. saw blade
3	21344	Gomez Bros.	14-Q1/L3	9.00-in. pwr. saw blade
4	23119	Randsets Ltd.	1546-QQ2	Hrd. cloth, 1/4-in., 2x50
5	23119	Randsets Ltd.	1558-QW1	Hrd. cloth, 1/2-in., 3x50
6	24288	ORDVA, Inc.	2232/QTY	B&D jigsaw, 12-in. blade
7	24288	ORDVA, Inc.	2232/QWE	B&D jigsaw, 8-in. blade
8	25595	Rubicon Systems	2238/QPD	B&D cordless drill, 1/2-in.
9	21225	Bryson, Inc.	23109-HB	Claw hammer
10	[null]	[null]	23114-AA	Sledge hammer, 12 lb.
11	21344	Gomez Bros.	54778-2T	Rat-tail file, 1/8-in. fine
12	24288	ORDVA, Inc.	89-WRE-Q	Hicut chain saw, 16 in.
13	[null]	[null]	PVC23DRT	PVC pipe, 3.5-in., 8-ft
14	21225	Bryson, Inc.	SM-18277	1.25-in. metal screw, 25
15	21231	D&E Supply	SW-23116	2.5-in. wd. screw, 50
16	25595	Rubicon Systems	WR3/TT3	Steel matting, 4'x8'x1/6", .5" mesh
17	25443	B&K, Inc.	[null]	[null]
18	21226	SuperLoo, Inc.	[null]	[null]
19	25501	Damal Supplies	[null]	[null]
20	22567	Dome Supply	[null]	[null]
21	24004	Brackman Bros.	[null]	[null]

FULL OUTER

```
SELECT V.V_CODE, V
L.LINE_UNITS, L.LI
FROM VENDOR V
FULL OUTER JOIN PR
FULL OUTER JOIN LI
```



	v_code integer	v_name character varying (35)	p_code character varying (10)	p_descript character varying (35)	line_units numeric (9,2)	line_price numeric (9,2)
1	21344	Gomez Bros.	13-Q2/P2	7.25-in. pwr. saw blade	1.00	14.99
2	21225	Bryson, Inc.	23109-HB	Claw hammer	1.00	9.95
3	21344	Gomez Bros.	54778-2T	Rat-tail file, 1/8-in. fine	2.00	4.99
4	25595	Rubicon Systems	2238/QPD	B&D cordless drill, 1/2-in.	1.00	38.95
5	23119	Randsets Ltd.	1546-QQ2	Hrd. cloth, 1/4-in., 2x50	1.00	39.95
6	21344	Gomez Bros.	13-Q2/P2	7.25-in. pwr. saw blade	5.00	14.99
7	21344	Gomez Bros.	54778-2T	Rat-tail file, 1/8-in. fine	3.00	4.99
8	21225	Bryson, Inc.	23109-HB	Claw hammer	2.00	9.95
9	[null]	[null]	PVC23DRT	PVC pipe, 3.5-in., 8-ft	12.00	5.87
10	21225	Bryson, Inc.	SM-18277	1.25-in. metal screw, 25	3.00	6.99
11	24288	ORDVA, Inc.	2232/QTY	B&D jigsaw, 12-in. blade	1.00	109.92
12	21225	Bryson, Inc.	23109-HB	Claw hammer	1.00	9.95
13	24288	ORDVA, Inc.	89-WRE-Q	Hicut chain saw, 16 in.	1.00	256.99
14	21344	Gomez Bros.	13-Q2/P2	7.25-in. pwr. saw blade	2.00	14.99
15	21344	Gomez Bros.	54778-2T	Rat-tail file, 1/8-in. fine	1.00	4.99
16	[null]	[null]	PVC23DRT	PVC pipe, 3.5-in., 8-ft	5.00	5.87
17	25595	Rubicon Systems	WR3/TT3	Steel matting, 4'x8'x1/6", .5" mesh	3.00	119.95
18	21225	Bryson, Inc.	23109-HB	Claw hammer	1.00	9.95
19	24004	Brackman Bros.	[null]	[null]	[null]	[null]
20	22567	Dome Supply	[null]	[null]	[null]	[null]
21	25501	Damal Supplies	[null]	[null]	[null]	[null]
22	21226	SuperLoo, Inc.	[null]	[null]	[null]	[null]
23	25443	B&K, Inc.	[null]	[null]	[null]	[null]
24	24288	ORDVA, Inc.	2232/QWE	B&D jigsaw, 8-in. blade	[null]	[null]
25	21231	D&E Supply	SW-23116	2.5-in. wd. screw, 50	[null]	[null]
26	[null]	[null]	23114-AA	Sledge hammer, 12 lb.	[null]	[null]

CROSS JOIN

- A CROSS JOIN is a type of join that produces the Cartesian product of the rows from the joined tables. This means that every row from the first table is combined with every row from the second table.
- No ON clause: Unlike INNER JOIN, LEFT JOIN, RIGHT JOIN, or FULL OUTER JOIN, a CROSS JOIN does not have an ON clause to specify a join condition. It simply combines all possible pairs of rows
- Result Size: The number of rows in the result of a CROSS JOIN is the product of the number of rows in each of the joined tables. For example, if table A has 3 rows and table B has 4 rows, their CROSS JOIN will result in 3 * 4 = 12 rows

```
SELECT column1_table1, column2_table1,
column1_table2, column2_table2, ...
FROM table1
CROSS JOIN table2;
```

RELATIONAL SET OPERATORS

RELATIONAL SET OPERATORS

- The relational set operators in SQL (including PostgreSQL) allow you to combine and manipulate results from multiple queries, working on whole result sets rather than just individual rows.
- UNION, UNION ALL, INTERSECT, EXCEPT (or MINUS)

RELATIONAL SET OPERATORS

- The relational set operators in SQL (including PostgreSQL) allow you to combine and manipulate results from multiple queries, working on whole result sets rather than just individual rows.
- UNION, UNION ALL, INTERSECT, EXCEPT (or MINUS)

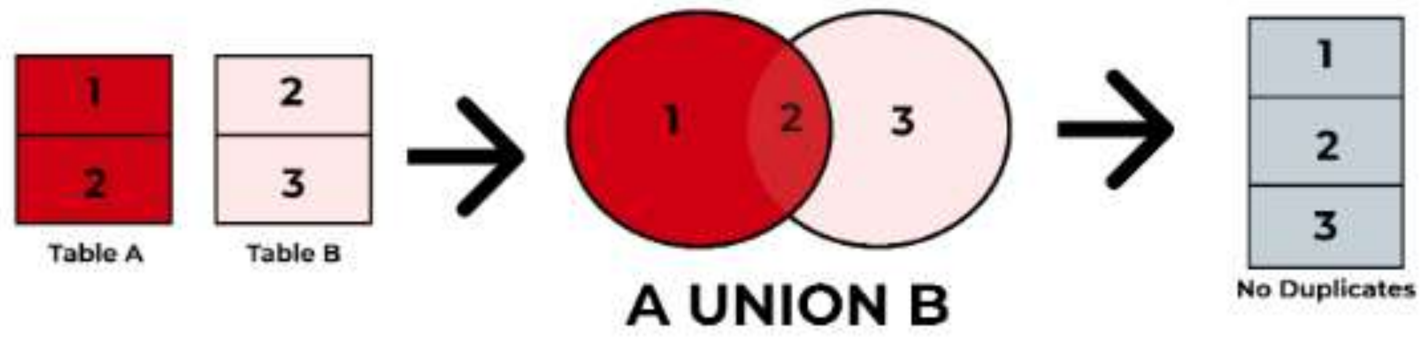
UNION

- The UNION operator in SQL is used to combine the result sets of two or more SELECT queries. It combines the rows of the result sets, removing duplicates by default, and returns a single result set.
- Removes Duplicates: The UNION operator eliminates duplicate rows in the result set. If you want to include duplicates, you can use UNION ALL (which we will cover later).
- Column Consistency: All SELECT statements involved in the UNION must have the same number of columns, and the corresponding columns must

UNION

- The UNION operator in SQL is used to combine the result sets of two or more SELECT queries. It combines the rows of the result sets, removing duplicates by default, and returns a single result set.
- Removes Duplicates: The UNION operator eliminates duplicate rows in the result set. If you want to include duplicates, you can use UNION ALL (which we will cover later).
- Column Consistency: All SELECT statements involved in the UNION must have the same number of columns, and the correspond data types.

```
SELECT column1, column2, ..
FROM table1
UNION
SELECT column1, column2, ..
FROM table2;
```



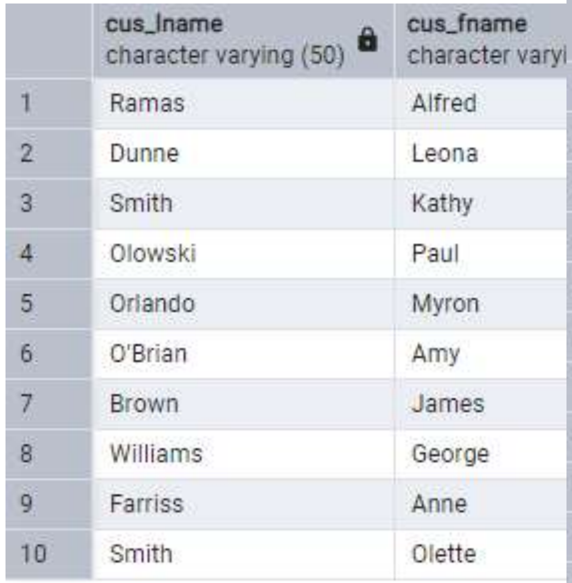
UNION (important rules)

- **Number of columns** must be the same in all queries.
- Data types must be compatible (e.g., VARCHAR can match TEXT; INTEGER can match NUMERIC with conversion).
- ORDER BY must appear after the last SELECT.

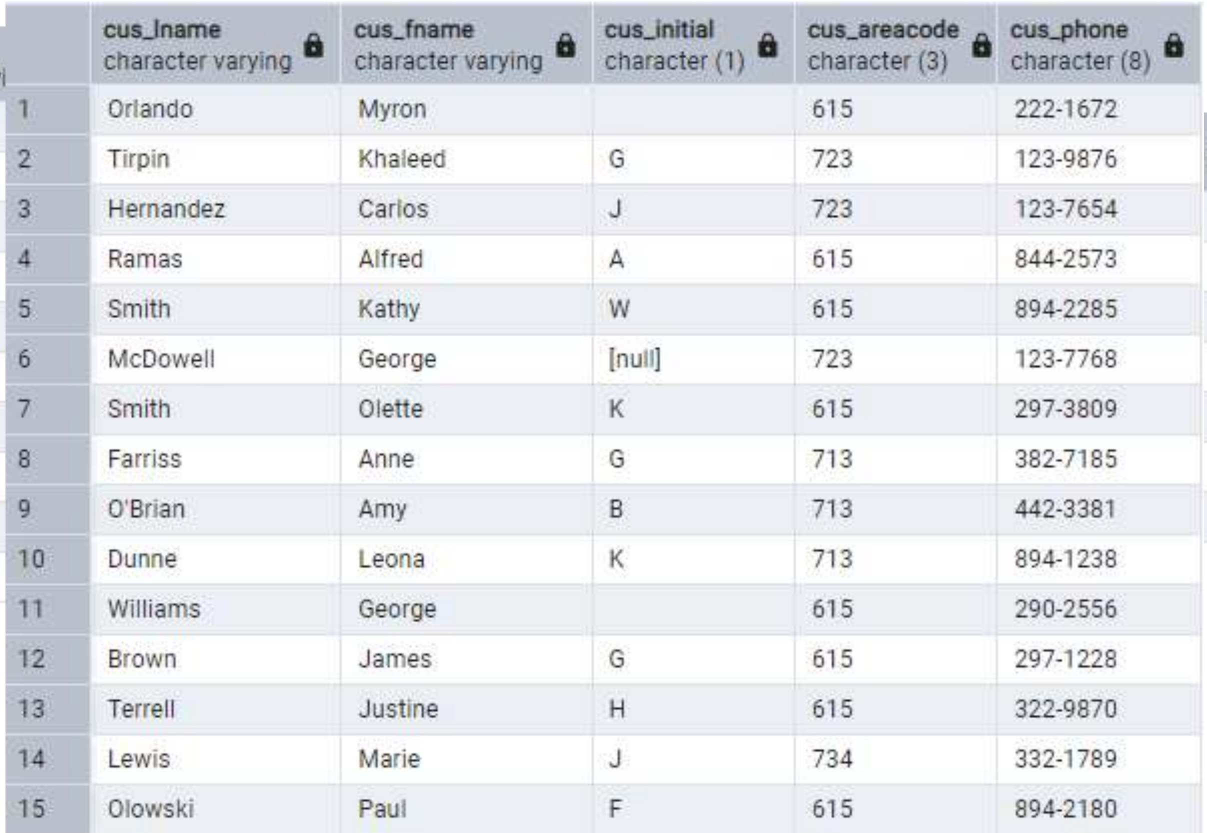
UNION

```
SELECT CUS_LNAME, CUS_FNAME, CUS_INITIAL,
CUS_AREACODE,
CUS_PHONE
FROM CUSTOMER
UNION
SELECT CUS_LNAME, CUS_FNAME, CUS_INITIAL,
CUS_AREACODE,
CUS_PHONE
FROM CUSTOMER_2;
```

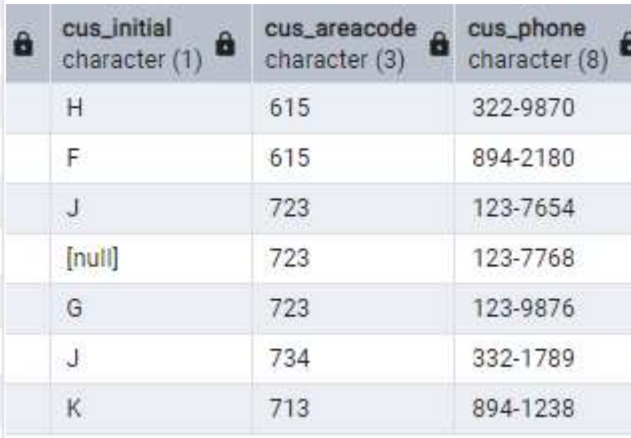
```
SELECT CUS_LNAME,                                              _FNAME, CUS_INITIAL,
CUS_AREACODE,
CUS_PHONE
FROM CUSTOMER
```



	cus_lname character varying (50)	cus_fname character varyi
1	Ramas	Alfred
2	Dunne	Leona
3	Smith	Kathy
4	Olowski	Paul
5	Orlando	Myron
6	O'Brian	Amy
7	Brown	James
8	Williams	George
9	Farriss	Anne
10	Smith	Olette



	cus_lname character varying	cus_fname character varying	cus_initial character (1)	cus_areacode character (3)	cus_phone character (8)
1	Orlando	Myron		615	222-1672
2	Tirpin	Khaleed	G	723	123-9876
3	Hernandez	Carlos	J	723	123-7654
4	Ramas	Alfred	A	615	844-2573
5	Smith	Kathy	W	615	894-2285
6	McDowell	George	[null]	723	123-7768
7	Smith	Olette	K	615	297-3809
8	Farriss	Anne	G	713	382-7185
9	O'Brian	Amy	B	713	442-3381
10	Dunne	Leona	K	713	894-1238
11	Williams	George		615	290-2556
12	Brown	James	G	615	297-1228
13	Terrell	Justine	H	615	322-9870
14	Lewis	Marie	J	734	332-1789
15	Olowski	Paul	F	615	894-2180

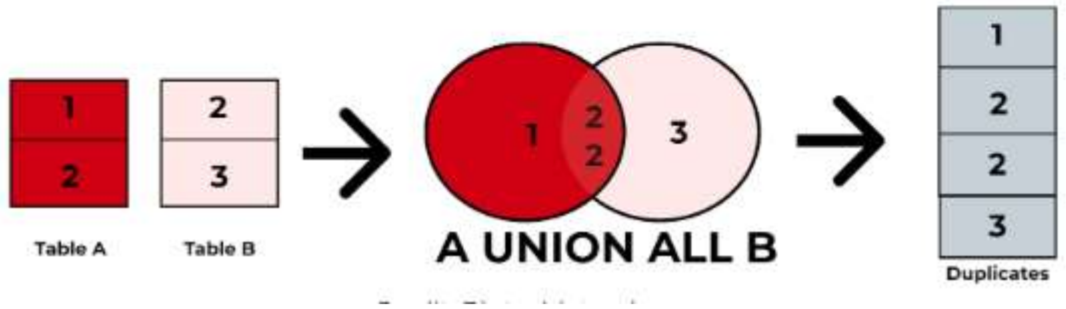


cus_initial character (1)	cus_areacode character (3)	cus_phone character (8)
H	615	322-9870
F	615	894-2180
J	723	123-7654
[null]	723	123-7768
G	723	123-9876
J	734	332-1789
K	713	894-1238

UNION ALL

- UNION ALL is a SQL set operator that combines the result sets of two or more SELECT queries into a single result set.
- It includes all rows from all queries
- It does NOT remove duplicates — every row from every SELECT is returned.
- It is faster and uses less memory than UNION because PostgreSQL doesn't sort and remove duplicates.

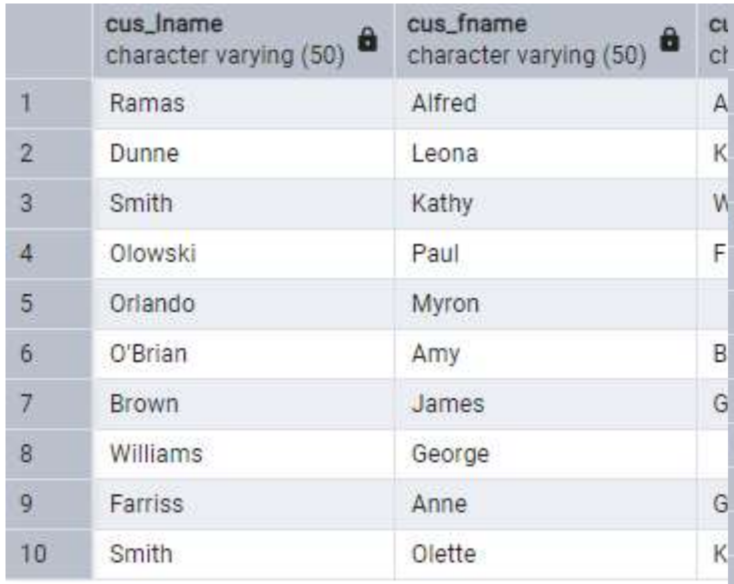
```
SELECT column1, column2, ...
FROM table1
WHERE condition1
UNION ALL
SELECT column1, column2, ...
FROM table2
WHERE condition2
```



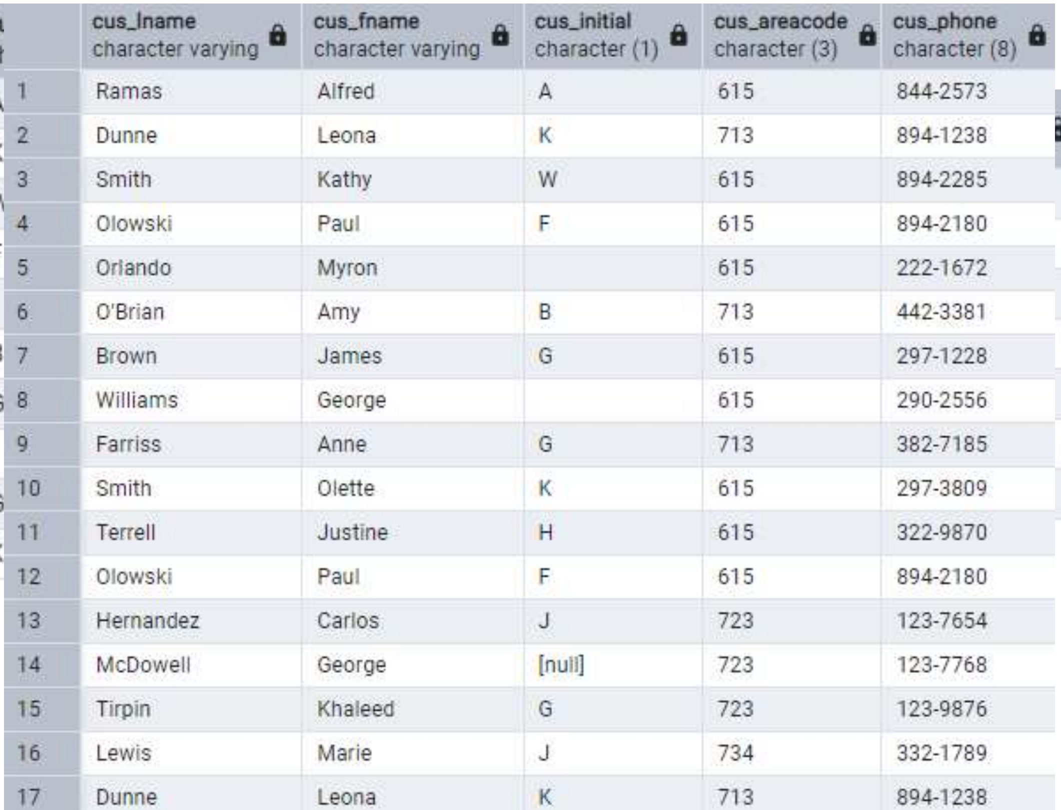
UNION ALL

```
SELECT CUS_LNAME, CUS_F                                        , CUS_INITIAL,
CUS_AREACODE,
CUS_PHONE
FROM CUSTOMER
```

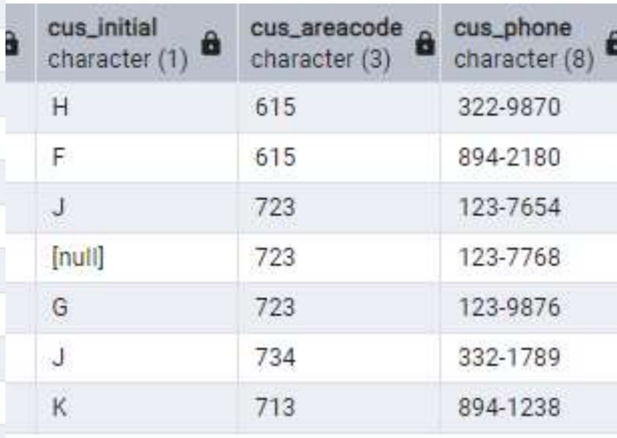
```
SELECT CUS_LNAME, CUS_FNAME, CUS_INITIAL,
CUS_AREACODE,
CUS_PHONE
FROM CUSTOMER
UNION ALL
SELECT CUS_LNAME, CUS_FNAME, CUS_INITIAL,
CUS_AREACODE,
CUS_PHONE
FROM CUSTOMER_2;
```



	cus_lname character varying (50)	cus_fname character varying (50)	cu ch
1	Ramas	Alfred	A
2	Dunne	Leona	K
3	Smith	Kathy	W
4	Olowski	Paul	F
5	Orlando	Myron	
6	O'Brian	Amy	B
7	Brown	James	G
8	Williams	George	
9	Farriss	Anne	G
10	Smith	Olette	K



	cus_lname character varying	cus_fname character varying	cus_initial character (1)	cus_areacode character (3)	cus_phone character (8)
1	Ramas	Alfred	A	615	844-2573
2	Dunne	Leona	K	713	894-1238
3	Smith	Kathy	W	615	894-2285
4	Olowski	Paul	F	615	894-2180
5	Orlando	Myron		615	222-1672
6	O'Brian	Amy	B	713	442-3381
7	Brown	James	G	615	297-1228
8	Williams	George		615	290-2556
9	Farriss	Anne	G	713	382-7185
10	Smith	Olette	K	615	297-3809
11	Terrell	Justine	H	615	322-9870
12	Olowski	Paul	F	615	894-2180
13	Hernandez	Carlos	J	723	123-7654
14	McDowell	George	[null]	723	123-7768
15	Tirpin	Khaleed	G	723	123-9876
16	Lewis	Marie	J	734	332-1789
17	Dunne	Leona	K	713	894-1238

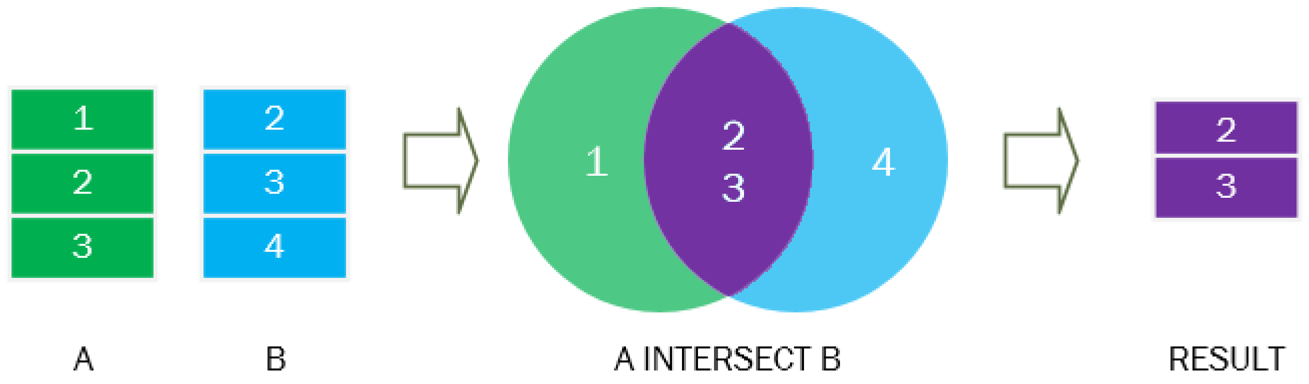


cus_initial character (1)	cus_areacode character (3)	cus_phone character (8)
H	615	322-9870
F	615	894-2180
J	723	123-7654
[null]	723	123-7768
G	723	123-9876
J	734	332-1789
K	713	894-1238

INTERSECT

- INTERSECT is a SQL set operator that returns only the rows that are common to two or more SELECT queries.
- It shows only the rows that exist in both (or all) query results.
- It removes duplicates automatically (just like UNION does).
- Only rows that appear exactly (same values) in both queries are returne

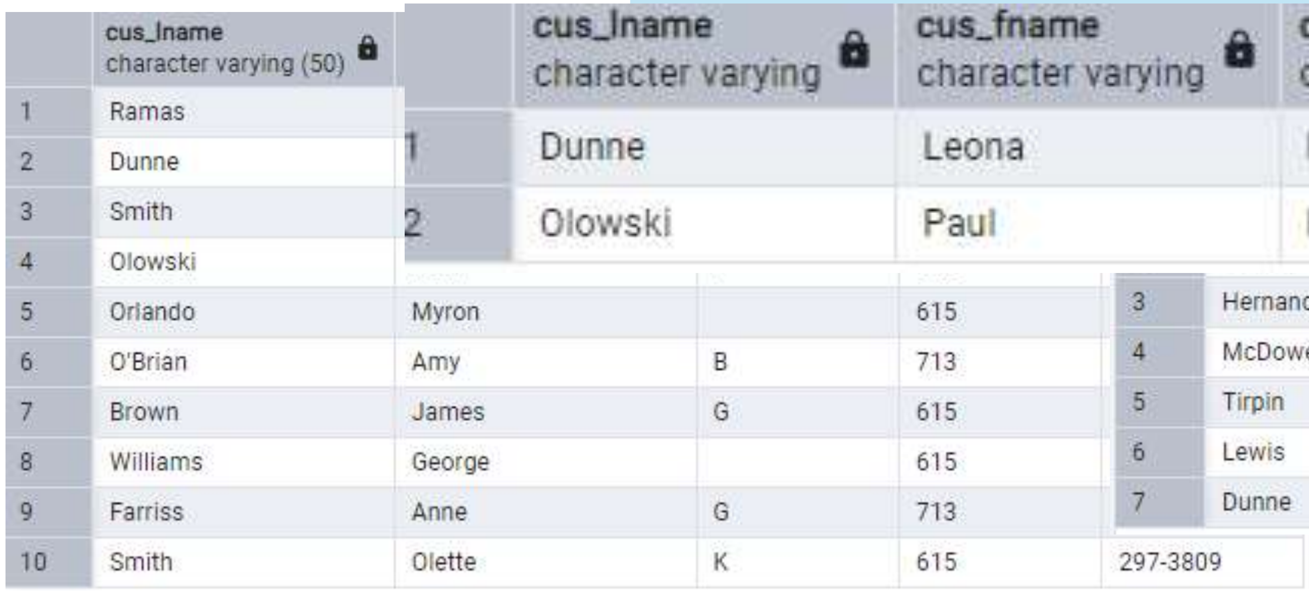
```
SELECT column1, column2, ...
FROM tableA
WHERE conditionA

INTERSECT

SELECT column1, column2, ...
FROM tableB
WHERE conditionB

ORDER BY column;
```



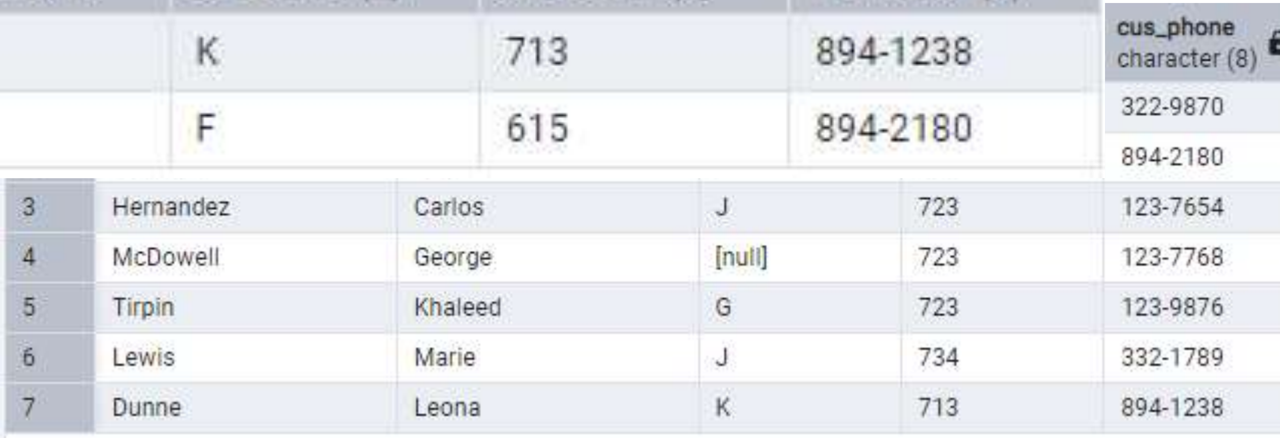
INTERSECT

```
SELECT CUS_LNAME, CUS_                                    E, CUS_INITIAL,
CUS_AREACODE,
CUS_PHONE
FROM CUSTOMER
```

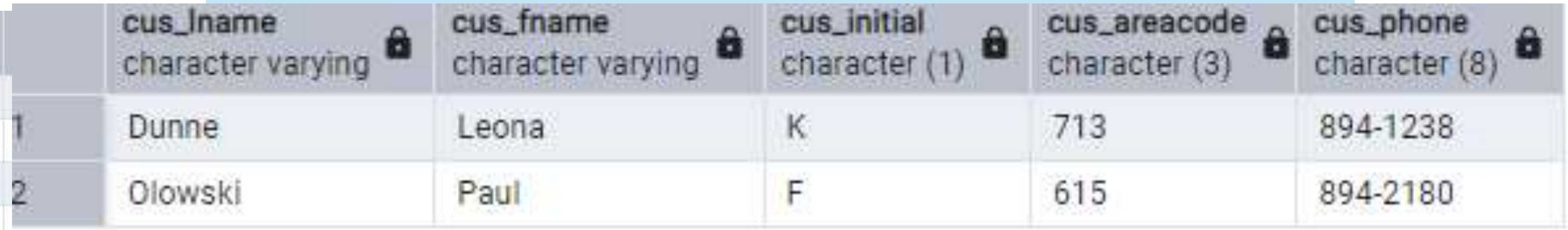
```
SELECT CUS_LNAME, CUS_FNAME, CUS_INITIAL,
CUS_AREACODE,
CUS_PHONE
FROM CUSTOMER
INTERSECT
SELECT CUS_LNAME, CUS_FNAME, CUS_INITIAL,
CUS_AREACODE,
CUS_PHONE
FROM CUSTOMER_2;
```



	cus_lname character varying (50)				
1	Ramas				
2	Dunne				
3	Smith				
4	Olowski				
5	Orlando	Myron		615	
6	O'Brian	Amy	B	713	
7	Brown	James	G	615	
8	Williams	George		615	
9	Farriss	Anne	G	713	
10	Smith	Olette	K	615	297-3809



					cus_phone character (8)
					322-9870
					894-2180
3	Hernandez	Carlos	J	723	123-7654
4	McDowell	George	[null]	723	123-7768
5	Tirpin	Khaleed	G	723	123-9876
6	Lewis	Marie	J	734	332-1789
7	Dunne	Leona	K	713	894-1238

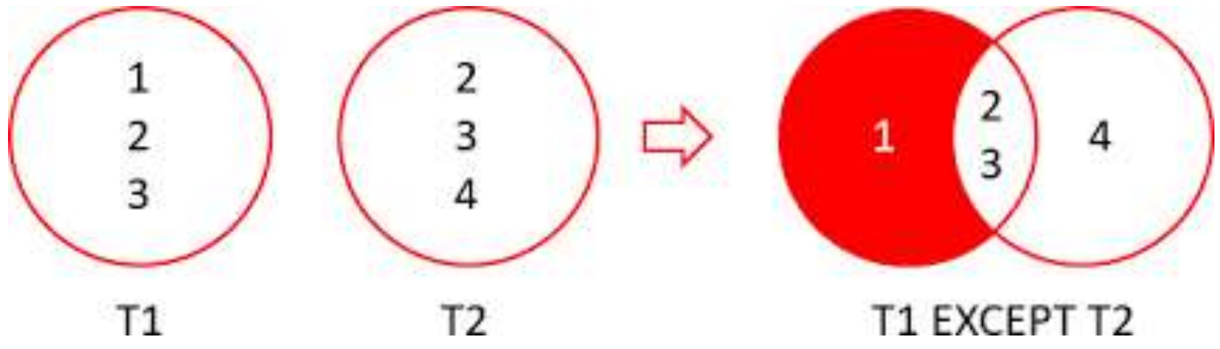


	cus_lname character varying	cus_fname character varying	cus_initial character (1)	cus_areacode character (3)	cus_phone character (8)
1	Dunne	Leona	K	713	894-1238
2	Olowski	Paul	F	615	894-2180

EXCEPT

- combines rows from two queries and returns only the rows that appear in the first set but not in the second.
- It is basically "Query 1 - Query 2" (subtract the second from the first)
- Duplicates are automatically removed in the result (just like UNION and INTERSECT).
- Only unique rows that exist in the first set but not in the second are shown.

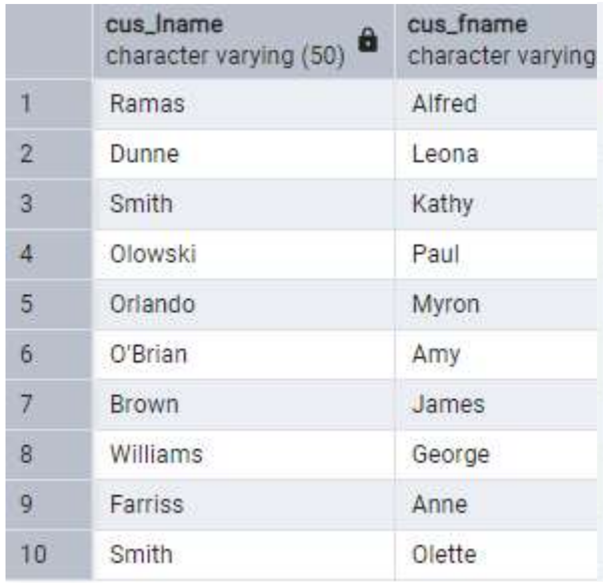
```
SELECT column1, column2, ...
FROM table1
WHERE condition1
EXCEPT
SELECT column1, column2, ...
FROM table2
WHERE condition2;
```



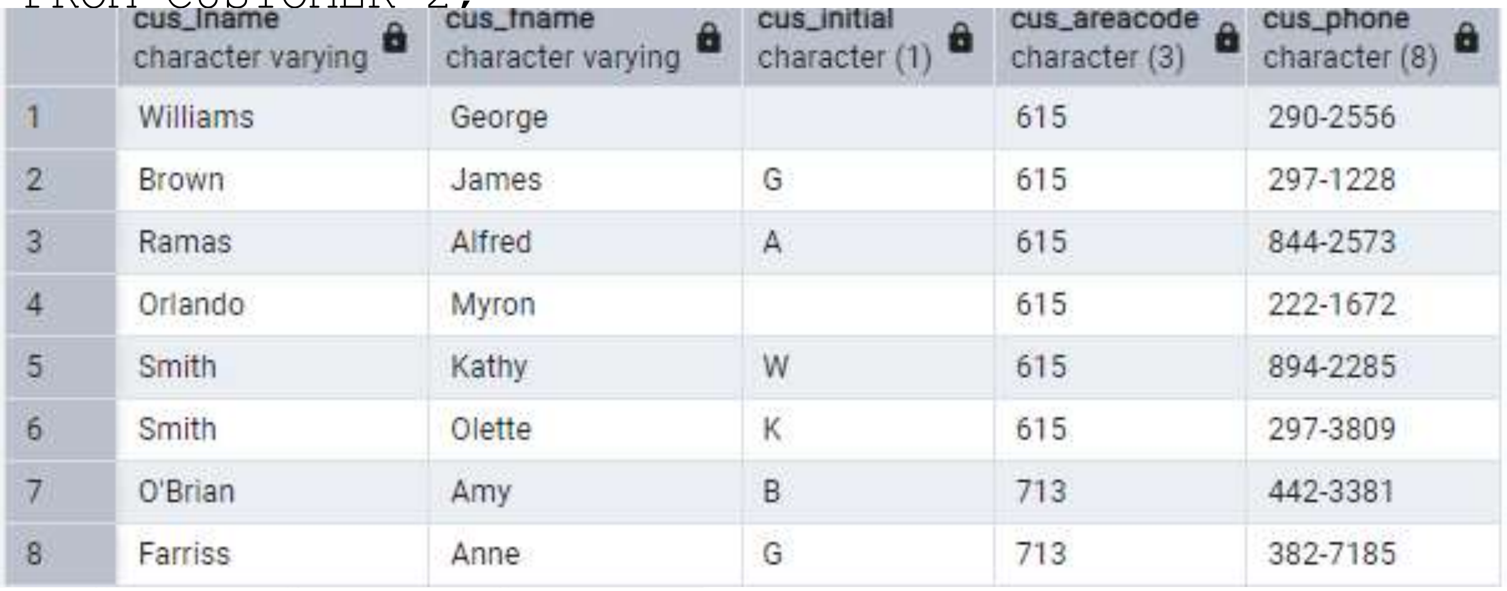
EXCEPT

```
SELECT CUS_LNAME, CUS                                                    ME, CUS_INITIAL,
CUS_AREACODE,
CUS_PHONE
FROM CUSTOMER
```

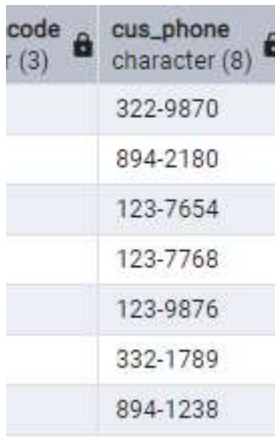
```
SELECT CUS_LNAME, CUS_FNAME, CUS_INITIAL,
CUS_AREACODE,
CUS_PHONE
FROM CUSTOMER
EXCEPT
CUS_LNAME, CUS_FNAME, CUS_INITIAL,
CUS_AREACODE,
CUS_PHONE
FROM CUSTOMER 2;
```



	cus_lname character varying (50)	cus_fname character varying
1	Ramas	Alfred
2	Dunne	Leona
3	Smith	Kathy
4	Olowski	Paul
5	Orlando	Myron
6	O'Brian	Amy
7	Brown	James
8	Williams	George
9	Farriss	Anne
10	Smith	Olette



	cus_lname character varying	cus_fname character varying	cus_initial character (1)	cus_areacode character (3)	cus_phone character (8)
1	Williams	George		615	290-2556
2	Brown	James	G	615	297-1228
3	Ramas	Alfred	A	615	844-2573
4	Orlando	Myron		615	222-1672
5	Smith	Kathy	W	615	894-2285
6	Smith	Olette	K	615	297-3809
7	O'Brian	Amy	B	713	442-3381
8	Farriss	Anne	G	713	382-7185



code (3)	cus_phone character (8)
	322-9870
	894-2180
	123-7654
	123-7768
	123-9876
	332-1789
	894-1238

FUNCTION

FUNCTION

- A **function** in PostgreSQL is a stored program written in SQL or procedural languages (like PL/pgSQL) that **takes input parameters**, **processes data**, and **returns a result**.
- Use functions when you need to return a value or perform computations that are used in queries.
- Functions are good for calculations, data formatting, or manipulating values.

Key Characteristics

Feature	**Description**
Reusable	Can be called multiple times with different parameters
Returns a value	Always returns a result (scalar, record, or table)
Used in SQL	Can be used in SELECT, WHERE, JOIN, etc.
No transaction control	Cannot use COMMIT or ROLLBACK inside
Supports logic	Can include IF, LOOP, CASE, etc., especially with PL/pgSQL

Basic Syntax

```
CREATE FUNCTION function_name(param1 TYPE, param2
TYPE)
RETURNS return_type
LANGUAGE plpgsql
AS $$
BEGIN
    -- logic here
    RETURN some_value;
END;
$$;
```

Function that adds two numbers

```
CREATE FUNCTION add_numbers(a INTEGER, b INTEGER)
RETURNS INTEGER
LANGUAGE plpgsql
AS $$
BEGIN
    RETURN a + b;
END;
$$;
```

```
 SELECT add_numbers(5, 10);
```

Limitations

- Cannot commit or roll back transactions.
- More limited than procedures when it comes to side effects (e.g., bulk updates).
- Should not be used when you don't need a return value — use a procedure instead.

Types of Returns

- Scalar: e.g., INTEGER, TEXT
- Composite/Record: like a row with multiple columns
- Table: returns a set of rows, like a mini query

CALCULATION OF GROSS PAY

```
CREATE OR REPLACE FUNCTION get_grosspay(hours_worked NUMERIC,
hourly_rate NUMERIC)
RETURNS NUMERIC
LANGUAGE plpgsql
AS $$
DECLARE
       gross_pay NUMERIC;
BEGIN
       gross_pay := hours_worked * hourly_rate;

       RETURN gross_pay;
END;
$$;
```

```
SELECT get_grosspay(50,100);
```

Return Full Name of a Customer

```
CREATE OR REPLACE FUNCTION get_customer_full_name(p_cus_code INT)
RETURNS TEXT AS $$
DECLARE
    full_name TEXT;
BEGIN
    SELECT CUS_FNAME || ' ' || CUS_LNAME
    INTO full_name
    FROM CUSTOMER
    WHERE CUS_CODE = p_cus_code;

    RETURN full_name;
END;
$$ LANGUAGE plpgsql;
```

```
SELECT
get_customer_full_name(10010);
```

Calculate Total Invoice Amount

```
CREATE OR REPLACE FUNCTION get_invoice_total(p_inv_number INT)
RETURNS NUMERIC(10,2) AS $$
DECLARE
    total NUMERIC(10,2);
BEGIN
    SELECT SUM(LINE_UNITS * LINE_PRICE)
    INTO total
    FROM LINE
    WHERE INV_NUMBER = p_inv_number;

    RETURN COALESCE(total, 0.00);
END;
$$ LANGUAGE plpgsql;
```

```
SELECT get_invoice_total(1003);
```

List All Products by a Vendor

```
CREATE OR REPLACE FUNCTION
get_products_by_vendor(p_v_code INT)
RETURNS TABLE (p_code VARCHAR, p_descript VARCHAR)
AS $$
BEGIN
    RETURN QUERY
    SELECT p.P_CODE, p.P_DESCRIPT
    FROM PRODUCT AS p
    WHERE V_CODE = p_v_code;
END;
$$ LANGUAGE plpgsql;
```

```
SELECT * FROM
get_products_by_vendor(25595);
```

Check If Customer Has Any Invoices

```
CREATE OR REPLACE FUNCTION customer_has_invoice(p_cus_code INT)
RETURNS BOOLEAN AS $$
DECLARE
    has_invoice BOOLEAN;
BEGIN
    SELECT EXISTS (
        SELECT 1
        FROM INVOICE
        WHERE CUS_CODE = p_cus_code
    ) INTO has_invoice;

    RETURN has_invoice;
END;
$$ LANGUAGE plpgsql;
```

```
SELECT
customer_has_invoice(10010);
```

Stored Procedure

Stored Procedure

- A **stored procedure** is a set of SQL statements that are stored and executed within a database.
- It is precompiled and saved, allowing you to execute the same set of operations multiple times without needing to retype or recompile the logic each time.
- They are primarily used for tasks like INSERT, UPDATE, DELETE, and complex queries, as well as for implementing business logic directly inside the database.

Key Characteristics

Feature	**Description**
Reusable	Can be executed multiple times, but only as a procedure call, not as part of a SQL query.
No return value	Typically does not return a value (though it can return values via OUT parameters or RETURN QUERY).
Used in SQL	Cannot be used directly within SELECT, WHERE, JOIN, etc., but can execute SQL commands within the procedure body.
Transaction control	Can manage transactions by using COMMIT, ROLLBACK, or SAVEPOINT within the procedure.
Supports logic	Can include control structures like IF, LOOP, CASE, etc., especially with PL/pgSQL.
Side effects	Typically modifies data (e.g., UPDATE, INSERT, DELETE) and may have side effects on the database state.

Basic Syntax

```
CREATE OR REPLACE PROCEDURE procedure_name(param1 TYPE,
param2 TYPE)
LANGUAGE plpgsql
AS $$
BEGIN
    -- logic here
    -- you can use SQL stat
DELETE
    -- transaction control
ROLLBACK)
END;
$$;
```

```
CREATE OR REPLACE PROCEDURE
update_product_price(p_code VARCHAR, new_price
NUMERIC)
LANGUAGE plpgsql
AS $$
BEGIN
    UPDATE product
    SET p_price = new_price
    WHERE p_code = p_code;
END;
$$;

CALL update_product_price('13-Q2/P2', 17.99);
```

Insert a new vendor

```
CREATE OR REPLACE PROCEDURE set_vendor(
    p_v_code INT,
    p_v_name VARCHAR,
    p_v_contact VARCHAR,
    p_v_areacode CHAR(3),
    p_v_phone CHAR(8),
    p_v_state CHAR(2),
    p_v_order CHAR(1)
)
LANGUAGE plpgsql
AS $$
BEGIN
    IF NOT EXISTS (SELECT 1 FROM vendor WHERE v_code = p_v_code) THEN
        INSERT INTO vendor
        VALUES (p_v_code, p_v_name, p_v_contact, p_v_areacode,
p_v_phone, p_v_state, p_v_order);
        RAISE NOTICE 'Vendor % inserted.', p_v_name;
    ELSE
        RAISE NOTICE 'Vendor % already exists.', p_v_name;
    END IF;
END;
$$;
```

```
CALL set_vendor(400, 'ABC Tools', 'Maria
Rivera', '999', '9999999', 'TX', 'Y');
```

Delete a vendor

```
CREATE OR REPLACE PROCEDURE remove_vendor(p_v_code INT)
LANGUAGE plpgsql
AS $$
BEGIN
    IF EXISTS (SELECT 1 FROM vendor WHERE v_code =
p_v_code) THEN
        DELETE FROM vendor WHERE v_code = p_v_code;
        RAISE NOTICE 'Vendor with code % removed.',
p_v_code;
    ELSE
        RAISE NOTICE 'Vendor with code % not found.',
p_v_code;
    END IF;
END;
$$;
```

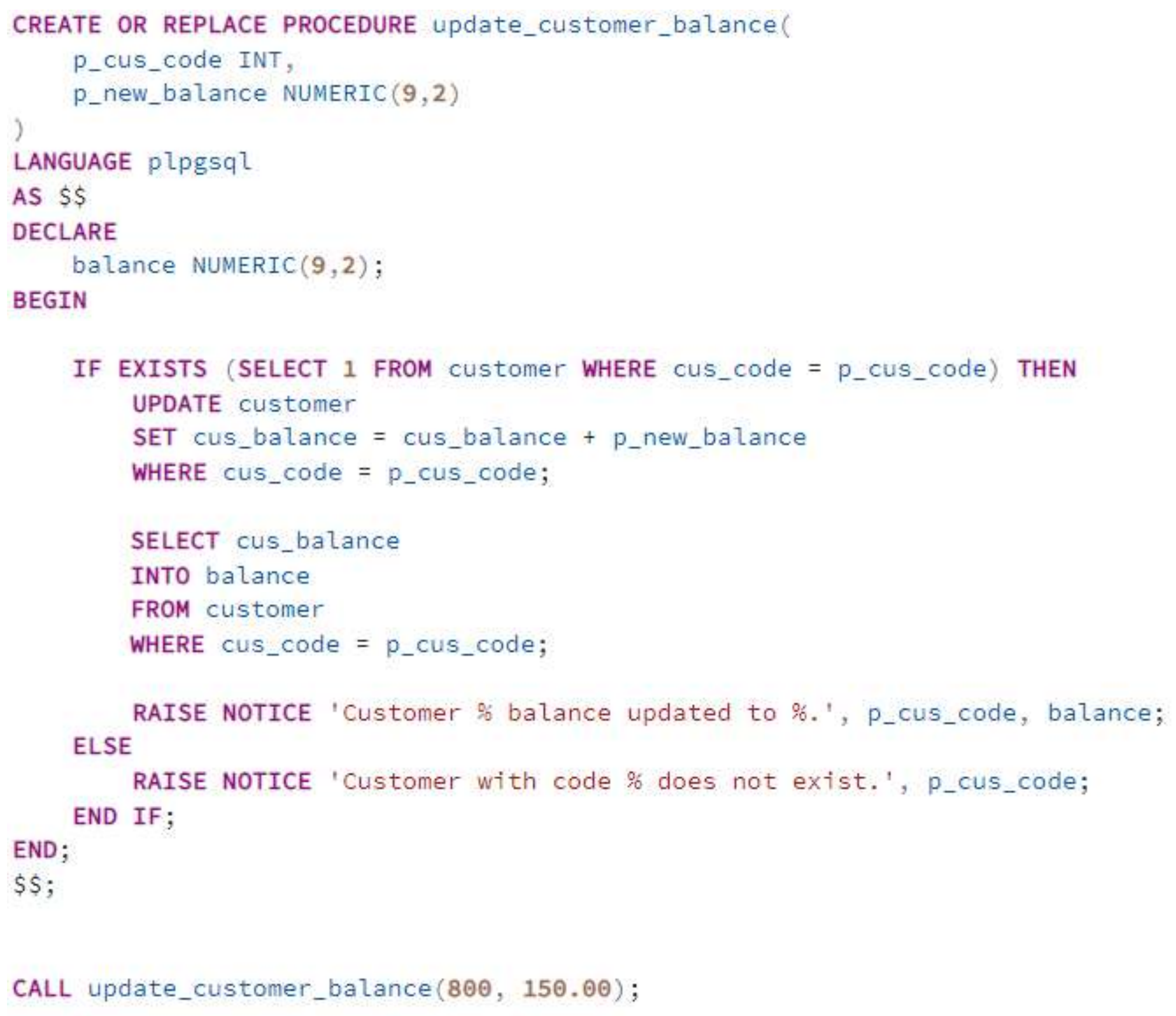
`CALL remove_vendor(400);`

Insert a new customer

```
CREATE OR REPLACE PROCEDURE set_customer(
    p_cus_code INT,
    p_lname VARCHAR,
    p_fname VARCHAR,
    p_initial CHAR(1),
    p_areacode CHAR(3),
    p_phone CHAR(8),
    p_balance NUMERIC(9,2)
)
LANGUAGE plpgsql
AS $$
BEGIN
    IF NOT EXISTS (SELECT 1 FROM customer WHERE cus_code = p_cus_code) THEN
        INSERT INTO customer
        VALUES (p_cus_code, p_lname, p_fname, p_initial, p_areacode, p_phone,
p_balance);
        RAISE NOTICE 'Customer % % inserted.', p_fname, p_lname;
    ELSE
        RAISE NOTICE 'Customer % % already exists.', p_fname, p_lname;
    END IF;
END;
$$;
```

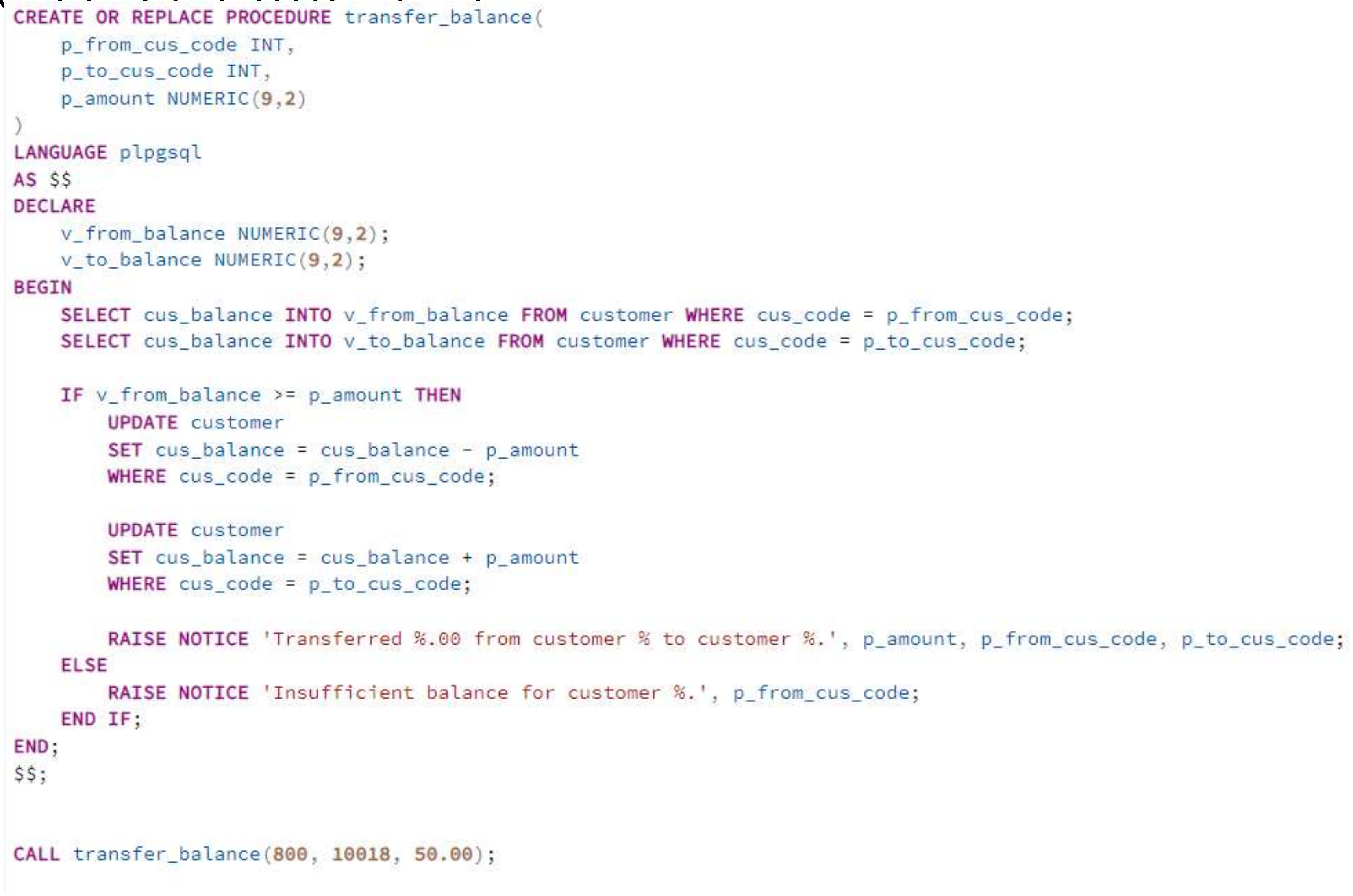
`CALL set_customer(800, 'Garcia', 'Luis', 'M', '615', '1234567', 120.00);`

Update Customer Balance



```
CREATE OR REPLACE PROCEDURE update_customer_balance(
    p_cus_code INT,
    p_new_balance NUMERIC(9,2)
)
LANGUAGE plpgsql
AS $$
DECLARE
    balance NUMERIC(9,2);
BEGIN

    IF EXISTS (SELECT 1 FROM customer WHERE cus_code = p_cus_code) THEN
        UPDATE customer
        SET cus_balance = cus_balance + p_new_balance
        WHERE cus_code = p_cus_code;

        SELECT cus_balance
        INTO balance
        FROM customer
        WHERE cus_code = p_cus_code;

        RAISE NOTICE 'Customer % balance updated to %.', p_cus_code, balance;
    ELSE
        RAISE NOTICE 'Customer with code % does not exist.', p_cus_code;
    END IF;
END;
$$;


CALL update_customer_balance(800, 150.00);
```

Transfer Balance Between Customers



```
CREATE OR REPLACE PROCEDURE transfer_balance(
    p_from_cus_code INT,
    p_to_cus_code INT,
    p_amount NUMERIC(9,2)
)
LANGUAGE plpgsql
AS $$
DECLARE
    v_from_balance NUMERIC(9,2);
    v_to_balance NUMERIC(9,2);
BEGIN
    SELECT cus_balance INTO v_from_balance FROM customer WHERE cus_code = p_from_cus_code;
    SELECT cus_balance INTO v_to_balance FROM customer WHERE cus_code = p_to_cus_code;

    IF v_from_balance >= p_amount THEN
        UPDATE customer
        SET cus_balance = cus_balance - p_amount
        WHERE cus_code = p_from_cus_code;

        UPDATE customer
        SET cus_balance = cus_balance + p_amount
        WHERE cus_code = p_to_cus_code;

        RAISE NOTICE 'Transferred %.00 from customer % to customer %.', p_amount, p_from_cus_code, p_to_cus_code;
    ELSE
        RAISE NOTICE 'Insufficient balance for customer %.', p_from_cus_code;
    END IF;
END;
$$;


CALL transfer_balance(800, 10018, 50.00);
```

Delete Customer if No Invoices Exist

```
CREATE OR REPLACE PROCEDURE delete_customer_if_no_invoices(p_cus_code
INT)
LANGUAGE plpgsql
AS $$
BEGIN
    IF NOT EXISTS (SELECT 1 FROM invoice WHERE cus_code = p_cus_code)
THEN
        DELETE FROM customer WHERE cus_code = p_cus_code;
        RAISE NOTICE 'Customer % deleted as they have no invoices.',
p_cus_code;
    ELSE
        RAISE NOTICE 'Customer % has invoices and cannot be deleted.',
p_cus_code;
    END IF;
END;
$$;
```

```
CALL delete_customer_if_no_invoices(800);
```

FOUND

```
CREATE OR REPLACE PROCEDURE upsert_customer_and_invoice(
    p_cus_code INT,
    p_cus_lname VARCHAR,
    p_cus_fname VARCHAR,
    p_cus_initial CHAR,
    p_cus_areacode CHAR(3),
    p_cus_phone CHAR(8),
    p_cus_balance NUMERIC(9,2),
    p_inv_number INT,
    p_inv_date DATE
)
LANGUAGE plpgsql
AS $$
BEGIN
    IF EXISTS (SELECT 1 FROM customer WHERE cus_code = p_cus_code) THEN
        UPDATE customer
        SET cus_lname = p_cus_lname,
            cus_fname = p_cus_fname,
            cus_initial = p_cus_initial,
            cus_areacode = p_cus_areacode,
            cus_phone = p_cus_phone,
            cus_balance = p_cus_balance
        WHERE cus_code = p_cus_code;
    ELSE
        INSERT INTO customer(cus_code, cus_lname, cus_fname, cus_initial, cus_areacode, cus_phone, cus_balance)
        VALUES (p_cus_code, p_cus_lname, p_cus_fname, p_cus_initial, p_cus_areacode, p_cus_phone, p_cus_balance);
    END IF;

    IF EXISTS (SELECT 1 FROM invoice WHERE inv_number = p_inv_number) THEN
        UPDATE invoice
        SET inv_date = p_inv_date,
            cus_code = p_cus_code
        WHERE inv_number = p_inv_number;
    ELSE
        INSERT INTO invoice(inv_number, inv_date, cus_code)
        VALUES (p_inv_number, p_inv_date, p_cus_code);
    END IF;
END;
$$;
```

```
CALL upsert_customer_and_invoice(
    10020, 'Johnson', 'Mark', 'T', '615', '555-1234', 100.00,
    1010, '2024-05-15'
);

CALL upsert_customer_and_invoice(
    10010, 'Ramas', 'Alfred', 'A', '615', '844-2573', 50.00,
    1001, '2016-01-16'
);
```

INSERT, UPDATE, and DELETE operations

```
CREATE OR REPLACE PROCEDURE manage_customer(p_cus_code INT, p_lname VARCHAR, p_fname VARCHAR,
    p_initial CHAR(1), p_areacode CHAR(3), p_phone CHAR(8), p_cus_balance NUMERIC(9,2))
LANGUAGE plpgsql
AS $$
BEGIN
    IF EXISTS (SELECT 1 FROM customer WHERE cus_code = p_cus_code) THEN
        UPDATE customer
        SET cus_balance = cus_balance + p_cus_balance
        WHERE cus_code = p_cus_code;

        RAISE NOTICE 'Customer % updated.', p_cus_code;
    ELSEsn
      INSERT INTO customer
      VALUES (p_cus_code, p_lname, p_fname, p_initial, p_areacode, p_phone, p_cus_balance);

      RAISE NOTICE 'Customer % inserted.', p_cus_code;
    END IF;

    DELETE FROM customer
    WHERE cus_balance <= 0;

    RAISE NOTICE 'Customers with zero or negative balance deleted.';
END;
$$;
```

```
CALL manage_customer(10021, 'Pacibe', 'Luis', 'M', '615', '1234567', 120.00);
```