

TE-COMPUTER		Roll number : 9914	
Experiment no. : 6		Date of Implementation : 12/ 3/ 2024	
Aim : To implement Join and complex SQL commands			
Tool Used : PostgreSQL			
Related Course outcome : At the end of the course, Students will be able to Use SQL : Standard language of relational database			
Rubrics for assessment of Experiment:			
Indicator	Poor	Average	Good
Timeliness <ul style="list-style-type: none"> Maintains assignment deadline (3) 	Assignment not done (0)	One or More than One week late (1-2)	Maintains deadline (3)
Completeness and neatness <ul style="list-style-type: none"> Complete all parts of assignment(3) 	N/A	< 80% complete (1-2)	100% complete (3)
Originality <ul style="list-style-type: none"> Extent of plagiarism(2) 	Copied it from someone else(0)	At least few questions have been done without copying(1)	Assignment has been solved completely without copying (2)
Knowledge <ul style="list-style-type: none"> In depth knowledge of the assignment(2) 	Unable to answer 2 questions(0)	Unable to answer 1 question (1)	Able to answer 2 questions (2)
Assessment Marks :			
Timeliness			
Completeness and neatness			
Originality			
Knowledge			
Total			
Total : (Out of 10)			

Teacher's Sign :**EXPE
RIME
NT 5**

Complex SQL commands

Aim

To implement complex SQL queries

Tools

PostgreSQL

Theor
y**Joining Tables**

The FROM clause allows more than 1 table in its list, however simply listing more than one table will very rarely produce the expected results. The rows from one table must be correlated with the rows of the others. This correlation is known as *joining*. In the subsequent text, the following 3 example tables are used:

p Table (parts)

pno	descr	color
P1	Widget	Blue
P2	Widget	Red
P3	Dongle	Green

s Table (suppliers)

sno	name	city
S1	Pierre	Paris
S2	John	London
S3	Mario	Rome

sp Table (suppliers & parts)

sno	pno	qty
S1	P1	NULL
S2	P1	200
S3	P1	1000
S3	P2	200

An example can best illustrate the rationale behind joins. The following query:

SELECT * FROM sp, p

Produces:

sno	pno	qty	pno	descr	color
S1	P1	NULL	P1	Widget	Blue
S1	P1	NULL	P2	Widget	Red
S1	P1	NULL	P3	Dongle	Green
S2	P1	200	P1	Widget	Blue
S2	P1	200	P2	Widget	Red
S2	P1	200	P3	Dongle	Green
S3	P1	1000	P1	Widget	Blue
S3	P1	1000	P2	Widget	Red
S3	P1	1000	P3	Dongle	Green
S3	P2	200	P1	Widget	Blue
S3	P2	200	P2	Widget	Red
S3	P2	200	P3	Dongle	Green

Each row in *sp* is arbitrarily combined with each row in *p*, giving 12 result rows (4 rows in *sp* X 3 rows in *p*.) This is known as a *cartesian product*.

A more usable query would correlate the rows from *sp* with rows from *p*, for instance matching on the common column -- *pno*:

```
SELECT *  
FROM sp, p  
WHERE sp.pno = p.pno
```

This produces:

sno	pno	qty	pno	descr	color
S1	P1	NULL	P1	Widget	Blue
S2	P1	200	P1	Widget	Blue
S3	P1	1000	P1	Widget	Blue
S3	P2	200	P2	Widget	Red

More information refer this

<https://www.tutorialspoint.com/sql/sql-using-joins.htm>

Procedure

1. Create following table:
Table name : sales_order_details









Column Name	Data type	Size
order_no	varchar	6
Product_no	varchar	6
Qty_ordered	numeric	8
Qty_disp	numeric	8
Product_rate	numeric	10,2

Create table- customer(cid, cname, address, pno)

Create table- cust_order(cid foreign key, order_no foreign key)

```
1 CREATE TABLE sales_order_details (  
2     order_no VARCHAR(6) REFERENCES salesorder(order_no),  
3     Product_no VARCHAR(6),  
4     Qty_ordered NUMERIC(8),  
5     Qty_disp NUMERIC(8),  
6     Product_rate NUMERIC(10,2)  
7 );  
8 SELECT * FROM sales_order_details;  
9
```

Scratch Pad × Messages Data output Notifications

							
order_no	product_no	qty_ordered	qty_disp	product_rate			
character varying (6)	character varying (6)	numeric (8)	numeric (8)	numeric (10,2)			

```
9 CREATE TABLE customer (  
10     cid SERIAL UNIQUE PRIMARY KEY,  
11     cname VARCHAR(255),  
12     address VARCHAR(255),  
13     pno VARCHAR(15)  
14 );  
15 SELECT * FROM customer;
```

Scratch Pad × Messages Data output Notifications

							
cid	cname	address	pno				
[PK] integer	character varying (255)	character varying (255)	character varying (15)				

```

16 CREATE TABLE cust_order (
17     cid INT REFERENCES customer(cid) UNIQUE,
18     order_no VARCHAR(6) REFERENCES sales_order_details(order_no)
19 );
20 SELECT * FROM cust_order;

```

Scratch Pad × Messages Data output Notifications

cid	integer	order_no	character varying (6)		

2. Insert 5-6 records in table in each tables.

```

-- Insert into sales_order_details
/*INSERT INTO sales_order_details (ord
VALUES
    ('S0001', 'P001', 10, 8, 25.50),
    ('S0002', 'P002', 15, 12, 30.75),
    ('S0003', 'P003', 20, 18, 15.20),
    ('S0004', 'P004', 12, 10, 40.00),
    ('S0005', 'P005', 8, 6, 18.75),
    ('S0006', 'P006', 25, 20, 22.50);

```

Data Output Messages Notifications

	order_no [PK] character varying (6)	product_no character varying (6)	qty_ordered numeric (8)	qty_disp numeric (8)	product_rate numeric (10,2)
1	S0001	P001	10	8	25.50
2	S0002	P002	15	12	30.75
3	S0003	P003	20	18	15.20
4	S0004	P004	12	10	40.00
5	S0005	P005	8	6	18.75
6	S0006	P006	25	20	22.50

```

-- Insert into customer
/*INSERT INTO customer (cid, cname, address, pno)
VALUES
    (1, 'John Doe', '123 Main St, Cityville', '555-1234'),
    (2, 'Jane Smith', '456 Oak St, Townsville', '555-5678'),
    (3, 'Bob Johnson', '789 Maple St, Villagetown', '555-9101'),
    (4, 'Alice Williams', '101 Pine St, Hamletville', '555-1122'),
    (5, 'Charlie Brown', '202 Cedar St, Boroughville', '555-3344'),
    (6, 'Eva Davis', '303 Elm St, Villageville', '555-5566');*/

-- Insert into cust_order
/*INSERT INTO cust_order (cid, order_no)

```

cid [PK] integer	cname character varying (255)	address character varying (255)	pno character varying (15)
1	John Doe	123 Main St, Cityville	555-1234
2	Jane Smith	456 Oak St, Townsville	555-5678
3	Bob Johnson	789 Maple St, Villagetown	555-9101
4	Alice Williams	101 Pine St, Hamletville	555-1122
5	Charlie Brown	202 Cedar St, Boroughville	555-3344
6	Eva Davis	303 Elm St, Villageville	555-5566

```
-- Insert into cust_order
/*INSERT INTO cust_order (cid, order_no)
VALUES
  (1, 'S0001'),
  (2, 'S0002'),
  (3, 'S0003'),
  (4, 'S0004'),
  (5, 'S0005'),
  (6, 'S0006');*/

--SELECT * FROM sales_order_details;
--SELECT * FROM customer;
SELECT * FROM cust_order;
```

ta Output Messages Notifications

order_no [PK] character varying (6)	cid integer
S0001	1
S0002	2
S0003	3
S0004	4
S0005	5
S0006	6

3. Print the description and total qty sold for each product

```

SELECT
    p.product_no,
    p.description,
    COALESCE(SUM(sod.qty_disp), 0) AS total_qty_sold
FROM
    products p
LEFT JOIN
    sales_order_details sod ON p.product_no = sod.product_no
GROUP BY
    p.product_no, p.description;

--SELECT * FROM sales_order_details;
--SELECT * FROM customer;
--SELECT * FROM cust_order;
--SELECT * FROM products;

```

Output Messages Notifications

product_no [PK] character varying (6)	description character varying (255)	total_qty_sold numeric
P005	Product 5 Description	6
P001	Product 1 Description	8
P006	Product 6 Description	20
P002	Product 2 Description	12
P004	Product 4 Description	10
P003	Product 3 Description	18

4. Find the value of each product sold

```

-- Query to find the value of each product sold
SELECT
    p.product_no,
    p.description,
    COALESCE(SUM(sod.qty_disp * sod.product_rate), 0) AS total_value_sold
FROM
    products p
LEFT JOIN
    sales_order_details sod ON p.product_no = sod.product_no
GROUP BY
    p.product_no, p.description;

--SELECT * FROM sales_order_details;
--SELECT * FROM customer;
--SELECT * FROM cust_order;
--SELECT * FROM products;

```

Output Messages Notifications

product_no [PK] character varying (6)	description character varying (255)	total_value_sold numeric
P005	Product 5 Description	112.50
P001	Product 1 Description	204.00
P006	Product 6 Description	450.00
P002	Product 2 Description	369.00
P004	Product 4 Description	400.00
P003	Product 3 Description	273.60

5. Calculate the average quantity sold for each client that has a maximum order value of 15000

```
-- Query to calculate the average quantity sold for each cli
WITH MaxOrderClients AS (
    SELECT
        co.cid,
        MAX(sod.qty_disp * sod.product_rate) AS max_order_val
    FROM
        cust_order co
    JOIN
        sales_order_details sod ON co.order_no = sod.order_no
    GROUP BY
        co.cid
    HAVING
        MAX(sod.qty_disp * sod.product_rate) <= 15000
)

SELECT
    co.cid,
    c.cname,
    AVG(sod.qty_disp) AS avg_qty_sold
FROM
    cust_order co
JOIN
    sales_order_details sod ON co.order_no = sod.order_no
JOIN
    customer c ON co.cid = c.cid
JOIN
    MaxOrderClients moc ON co.cid = moc.cid
GROUP BY
    co.cid, c.cname;
```

ta Output Messages Notifications

cid	cname	avg_qty_sold
integer	character varying (255)	numeric
5	Charlie Brown	6.0000000000000000
3	Bob Johnson	18.0000000000000000
2	Jane Smith	12.0000000000000000
6	Eva Davis	20.0000000000000000
1	John Doe	8.0000000000000000
4	Alice Williams	10.0000000000000000

6. Find out the sum total of all the billed orders for the month of January

```
-- Assuming the correct column name is "order_date"
SELECT
    SUM(qty_disp * product_rate) AS total_billed_amount
FROM
    sales_order_details
WHERE
    EXTRACT(MONTH FROM order_date) = 1; -- January
```

ata Output Messages Notifications

order_no	product_no	qty_ordered	qty_disp
[PK] character varying (6)	character varying (6)	numeric (8)	numeric (8)
S0001	P001	10	8
S0002	P002	15	12
S0003	P003	20	18
S0004	P004	12	10
S0005	P005	8	6
S0006	P006	25	20

7. Find out the name of customers who have given the order of more than 10 qty.


```

SELECT
    c.cname
FROM
    customer c
JOIN
    cust_order co ON c.cid = co.cid
JOIN
    sales_order_details sod ON co.order_no = sod.order_no
WHERE
    sod.qty_ordered > 10;

--SELECT * FROM sales_order_details;
--SELECT * FROM customer;

```

Output Messages Notifications

cname

character varying (255)

Jane Smith

Bob Johnson

Alice Williams

Eva Davis

8. Find out the customer names with product no with maximum qty ordered.

```

WITH MaxQtyPerProduct AS (
    SELECT
        co.cid,
        sod.product_no,
        MAX(sod.qty_ordered) AS max_qty_ordered
    FROM
        cust_order co
    JOIN
        sales_order_details sod ON co.order_no = sod.order_no
    GROUP BY
        co.cid, sod.product_no
)
SELECT
    c.cname,
    m.product_no
FROM
    customer c
JOIN
    MaxQtyPerProduct m ON c.cid = m.cid
WHERE

```

```

WHERE
    (c.cid, m.max_qty_ordered) IN (
        SELECT
            cid,
            MAX(max_qty_ordered) AS max_qty_ordered
        FROM
            MaxQtyPerProduct
        GROUP BY
            cid
    )

```

cname	product_no
character varying (255)	character va
Eva Davis	P006
Charlie Brown	P005
Bob Johnson	P003
Alice Williams	P004
Jane Smith	P002
John Doe	P001

9. Find out most frequent orders

```

SELECT
    order_no,
    COUNT(*) AS order_frequency
FROM
    cust_order
GROUP BY
    order_no
ORDER BY
    order_frequency DESC
LIMIT 1;

```

[a Output](#) Messages Notifications

order_no	order_frequency
[PK] character varying (6)	bigint
S0005	1

**Post
Lab
Quest
ions:**

1. What is the difference between inner Join and outer Join.
⇒ The main difference between inner join and outer join in SQL is that an inner join returns only the rows with matching values in both tables, while an outer join returns all the rows from the database tables, including those that do not have a match in the other table. There are three types of outer joins: left outer join, right outer join, and full outer join. An inner join is a simple join that provides the result directly, while an outer join is a complex join that requires additional syntax to specify the type of join. Outer joins are generally faster than inner joins because they are less restrictive and do not require precise matches.
2. Give one example for equi_join and non equi_join.
⇒
 - In an Equi Join, the join operation is based on an equality condition using the equals sign (=). For instance, consider two tables: "state" and "city." The "state" table contains State_ID and State_Name columns, while the "city" table contains City_ID and City_Name columns. An Equi Join can be used to map cities with the states they belong to based on a common column.
 - In a Non Equi Join, the join condition involves comparison operators other than the equals sign, such as >, <, >=, <=. For example, consider two tables: "orders" and "customer." To retrieve order numbers and order amounts from the "orders" table and customer names and working areas from the "customer" table where the order amount matches any opening amount in the customer table, a Non Equi Join can be used.
3. complete online exercise and add screen shots
https://www.w3schools.com/sql/exercise.asp?filename=exercise_join1

Exercise:

Insert the missing parts in the JOIN clause to join the two tables Orders and Customers , using the CustomerID field in both the relationship between the two tables.

Correct!

Next >

Exercise:

Choose the correct JOIN clause to select all records from the two tables where there is a match in both tables.

Correct!

Next >



**BUILD YOUR CAREER. GET
FULL ACCESS. SAVE 770\$**

Start today



Exercise:

Choose the correct `JOIN` clause to select all the records from the `Customers` table plus all the matches from the `Orders` table.

Correct!

Next >