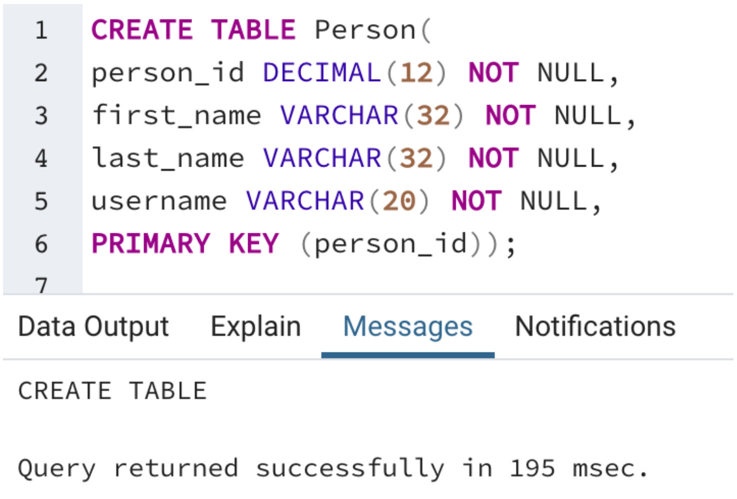
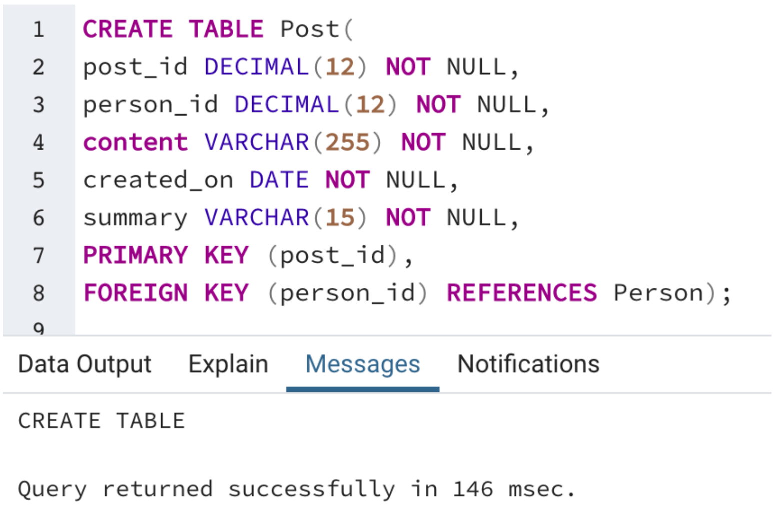
Section One – Stored Procedures
Step 1 – Create Table Structure



```sql
CREATE TABLE Person(
person_id DECIMAL(12) NOT NULL,
first_name VARCHAR(32) NOT NULL,
last_name VARCHAR(32) NOT NULL,
username VARCHAR(20) NOT NULL,
PRIMARY KEY (person_id));
```

```
CREATE TABLE

Query returned successfully in 195 msec.
```

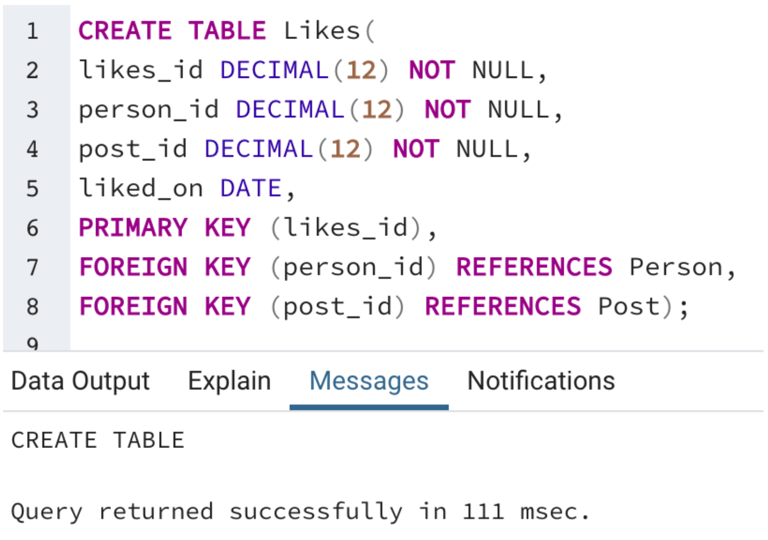
create person table



```sql
CREATE TABLE Post(
post_id DECIMAL(12) NOT NULL,
person_id DECIMAL(12) NOT NULL,
content VARCHAR(255) NOT NULL,
created_on DATE NOT NULL,
summary VARCHAR(15) NOT NULL,
PRIMARY KEY (post_id),
FOREIGN KEY (person_id) REFERENCES Person);
```

```
CREATE TABLE

Query returned successfully in 146 msec.
```

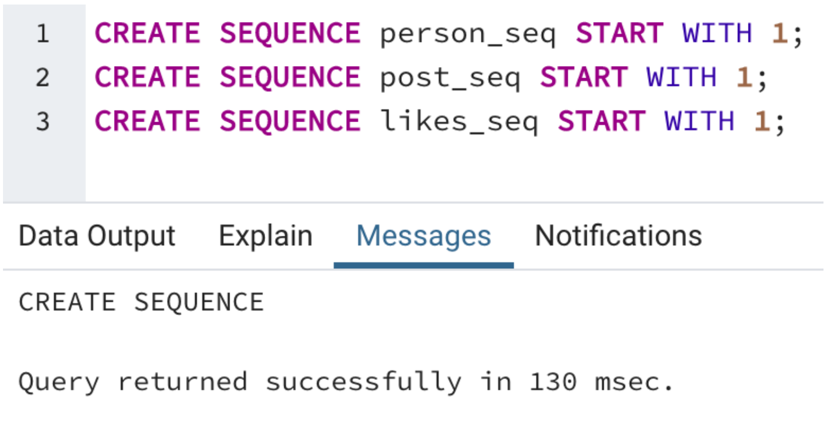
create post table



```sql
CREATE TABLE Likes(
likes_id DECIMAL(12) NOT NULL,
person_id DECIMAL(12) NOT NULL,
post_id DECIMAL(12) NOT NULL,
liked_on DATE,
PRIMARY KEY (likes_id),
FOREIGN KEY (person_id) REFERENCES Person,
FOREIGN KEY (post_id) REFERENCES Post);
```

```
CREATE TABLE

Query returned successfully in 111 msec.
```

create likes table



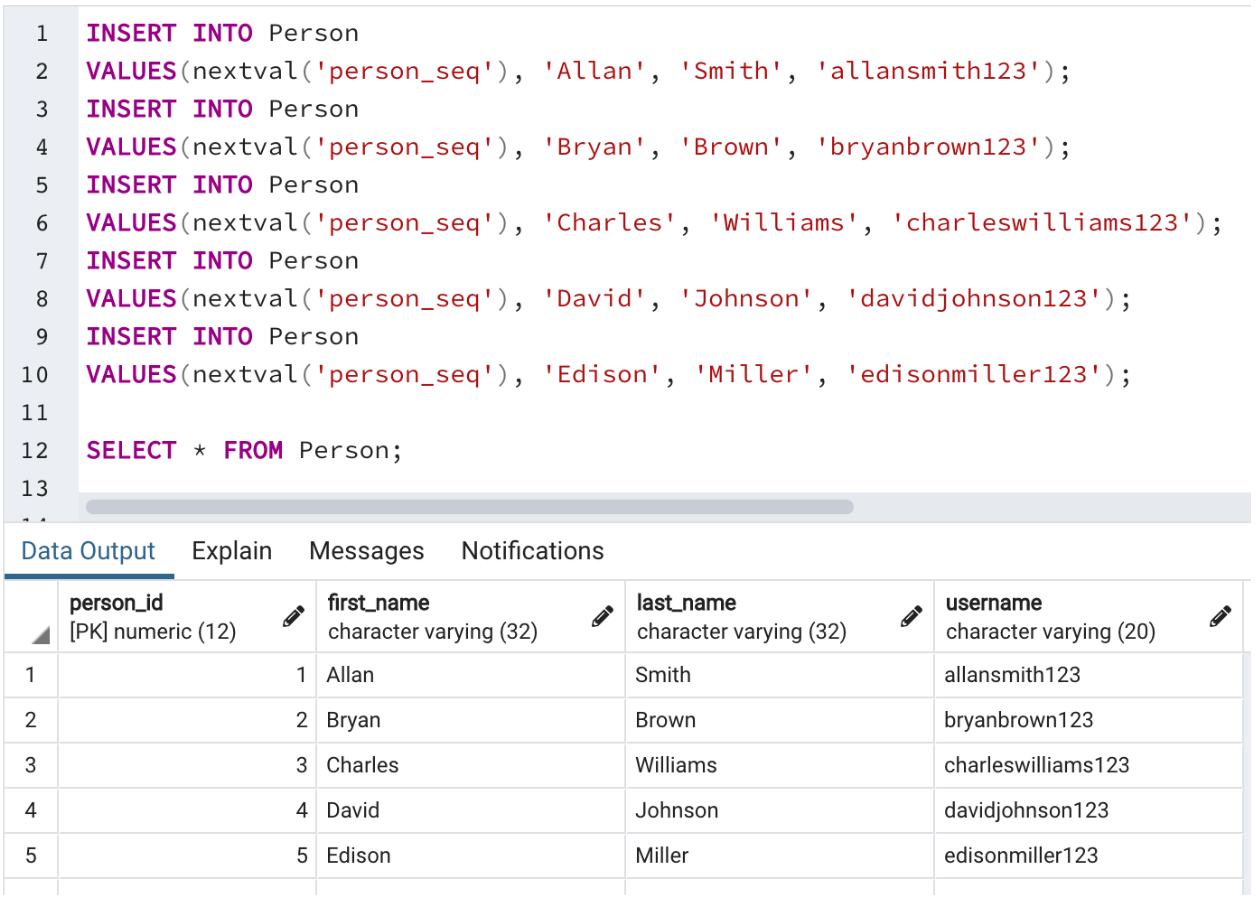
```
CREATE SEQUENCE person_seq START WITH 1;
CREATE SEQUENCE post_seq START WITH 1;
CREATE SEQUENCE likes_seq START WITH 1;
```

Data Output Explain Messages Notifications

```
CREATE SEQUENCE

Query returned successfully in 130 msec.
```

create sequences for each table

Step 2 – Populate Tables
at least 5 people, at least 8 posts, and at least 4 likes

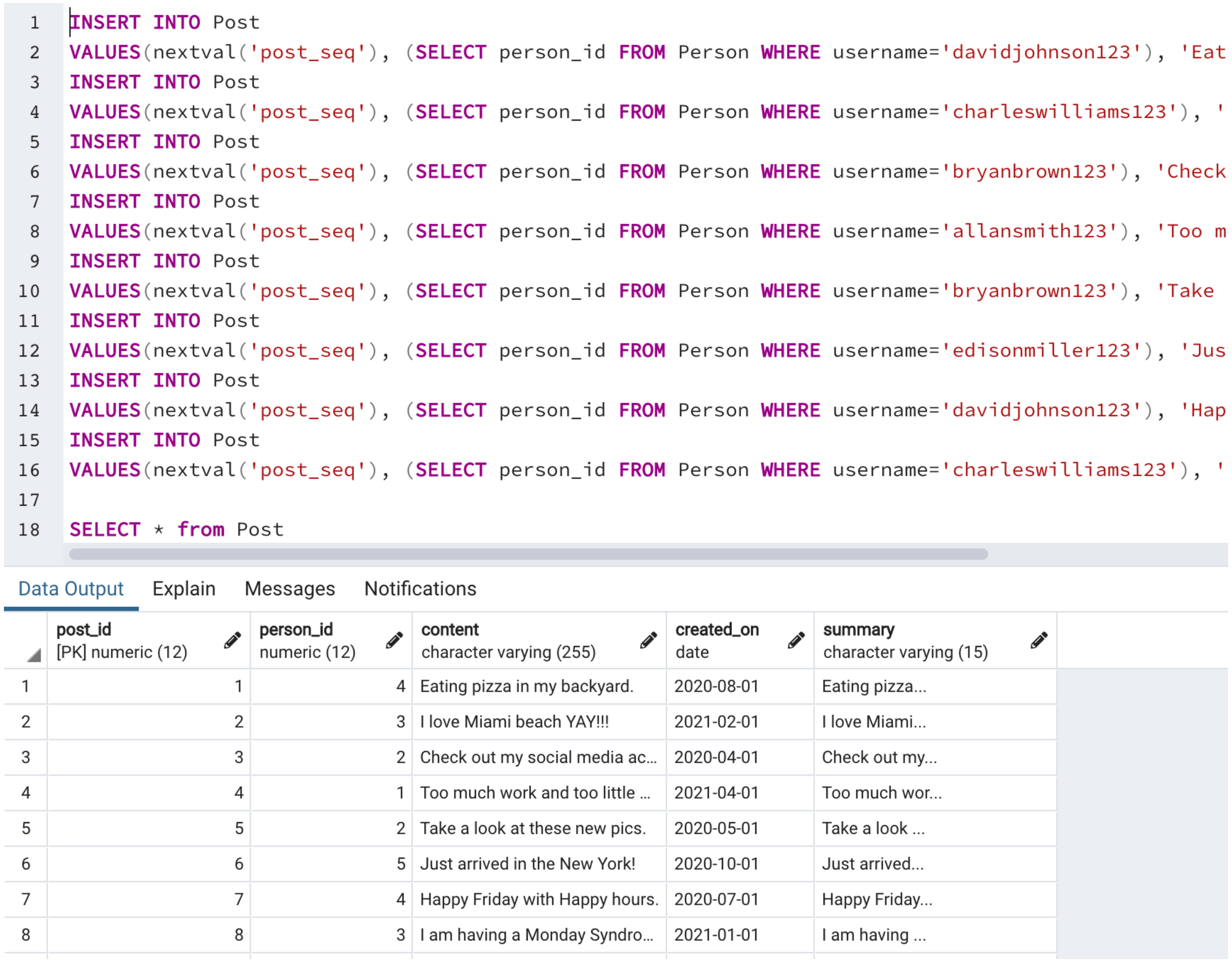


```
INSERT INTO Person
VALUES(nextval('person_seq'), 'Allan', 'Smith', 'allansmith123');
INSERT INTO Person
VALUES(nextval('person_seq'), 'Bryan', 'Brown', 'bryanbrown123');
INSERT INTO Person
VALUES(nextval('person_seq'), 'Charles', 'Williams', 'charleswilliams123');
INSERT INTO Person
VALUES(nextval('person_seq'), 'David', 'Johnson', 'davidjohnson123');
INSERT INTO Person
VALUES(nextval('person_seq'), 'Edison', 'Miller', 'edisonmiller123');

SELECT * FROM Person;

```

Data Output Explain Messages Notifications

	person_id [PK] numeric (12)	first_name character varying (32)	last_name character varying (32)	username character varying (20)
1	1	Allan	Smith	allansmith123
2	2	Bryan	Brown	bryanbrown123
3	3	Charles	Williams	charleswilliams123
4	4	David	Johnson	davidjohnson123
5	5	Edison	Miller	edisonmiller123

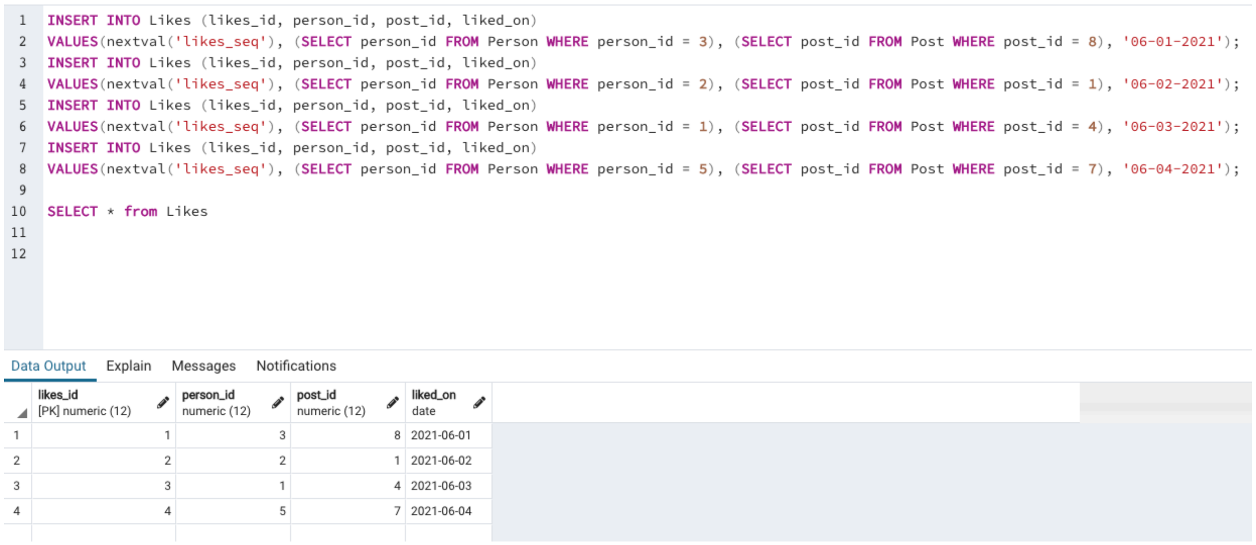
-- Person Table



```sql
INSERT INTO Post
VALUES(nextval('post_seq'), (SELECT person_id FROM Person WHERE username='davidjohnson123'), 'Eat
INSERT INTO Post
VALUES(nextval('post_seq'), (SELECT person_id FROM Person WHERE username='charleswilliams123'), '
INSERT INTO Post
VALUES(nextval('post_seq'), (SELECT person_id FROM Person WHERE username='bryanbrown123'), 'Check
INSERT INTO Post
VALUES(nextval('post_seq'), (SELECT person_id FROM Person WHERE username='allansmith123'), 'Too m
INSERT INTO Post
VALUES(nextval('post_seq'), (SELECT person_id FROM Person WHERE username='bryanbrown123'), 'Take
INSERT INTO Post
VALUES(nextval('post_seq'), (SELECT person_id FROM Person WHERE username='edisonmiller123'), 'Jus
INSERT INTO Post
VALUES(nextval('post_seq'), (SELECT person_id FROM Person WHERE username='davidjohnson123'), 'Hap
INSERT INTO Post
VALUES(nextval('post_seq'), (SELECT person_id FROM Person WHERE username='charleswilliams123'), '

SELECT * from Post
```

	post_id [PK] numeric (12)	person_id numeric (12)	content character varying (255)	created_on date	summary character varying (15)
1	1	4	Eating pizza in my backyard.	2020-08-01	Eating pizza...
2	2	3	I love Miami beach YAY!!!	2021-02-01	I love Miami...
3	3	2	Check out my social media ac...	2020-04-01	Check out my...
4	4	1	Too much work and too little ...	2021-04-01	Too much wor...
5	5	2	Take a look at these new pics.	2020-05-01	Take a look ...
6	6	5	Just arrived in the New York!	2020-10-01	Just arrived...
7	7	4	Happy Friday with Happy hours.	2020-07-01	Happy Friday...
8	8	3	I am having a Monday Syndro...	2021-01-01	I am having ...

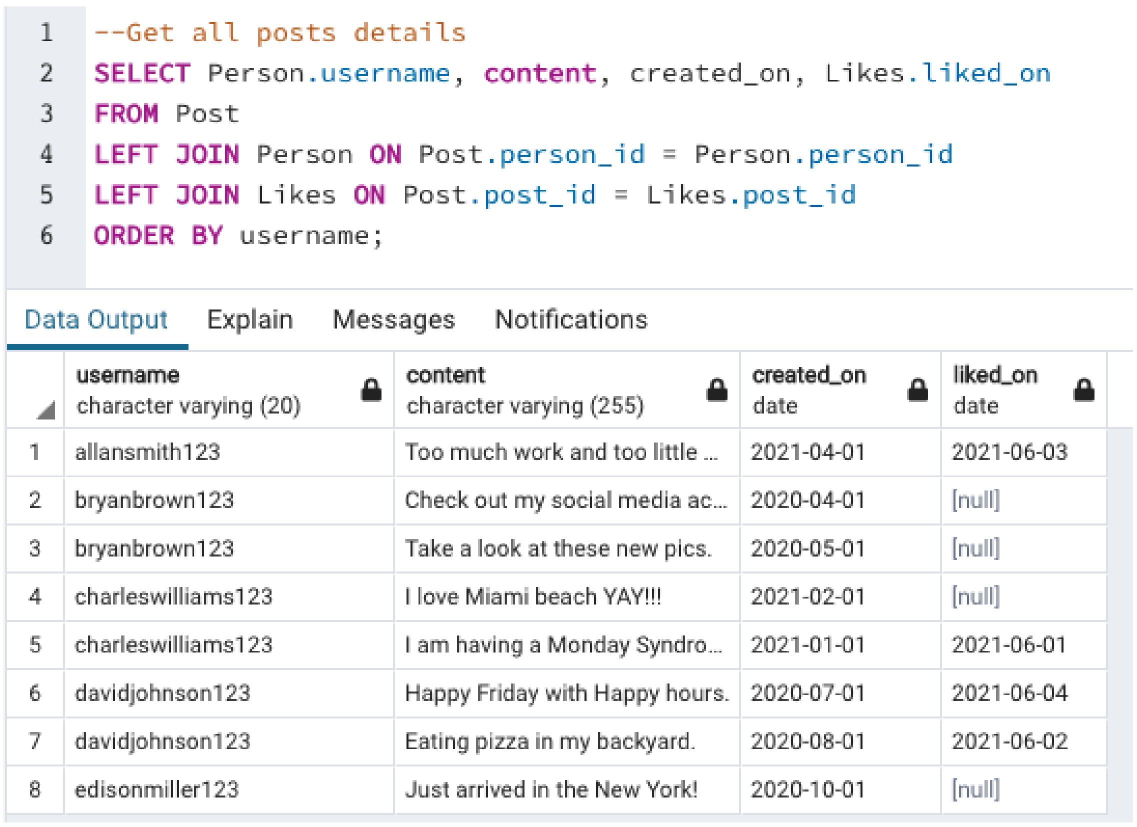
-- Post Table



```sql
INSERT INTO Likes (likes_id, person_id, post_id, liked_on)
VALUES(nextval('likes_seq'), (SELECT person_id FROM Person WHERE person_id = 3), (SELECT post_id FROM Post WHERE post_id = 8), '06-01-2021');
INSERT INTO Likes (likes_id, person_id, post_id, liked_on)
VALUES(nextval('likes_seq'), (SELECT person_id FROM Person WHERE person_id = 2), (SELECT post_id FROM Post WHERE post_id = 1), '06-02-2021');
INSERT INTO Likes (likes_id, person_id, post_id, liked_on)
VALUES(nextval('likes_seq'), (SELECT person_id FROM Person WHERE person_id = 1), (SELECT post_id FROM Post WHERE post_id = 4), '06-03-2021');
INSERT INTO Likes (likes_id, person_id, post_id, liked_on)
VALUES(nextval('likes_seq'), (SELECT person_id FROM Person WHERE person_id = 5), (SELECT post_id FROM Post WHERE post_id = 7), '06-04-2021');

SELECT * from Likes
```

	likes_id [PK] numeric (12)	person_id numeric (12)	post_id numeric (12)	liked_on date
1	1	3	8	2021-06-01
2	2	2	1	2021-06-02
3	3	1	4	2021-06-03
4	4	5	7	2021-06-04

-- Like Table

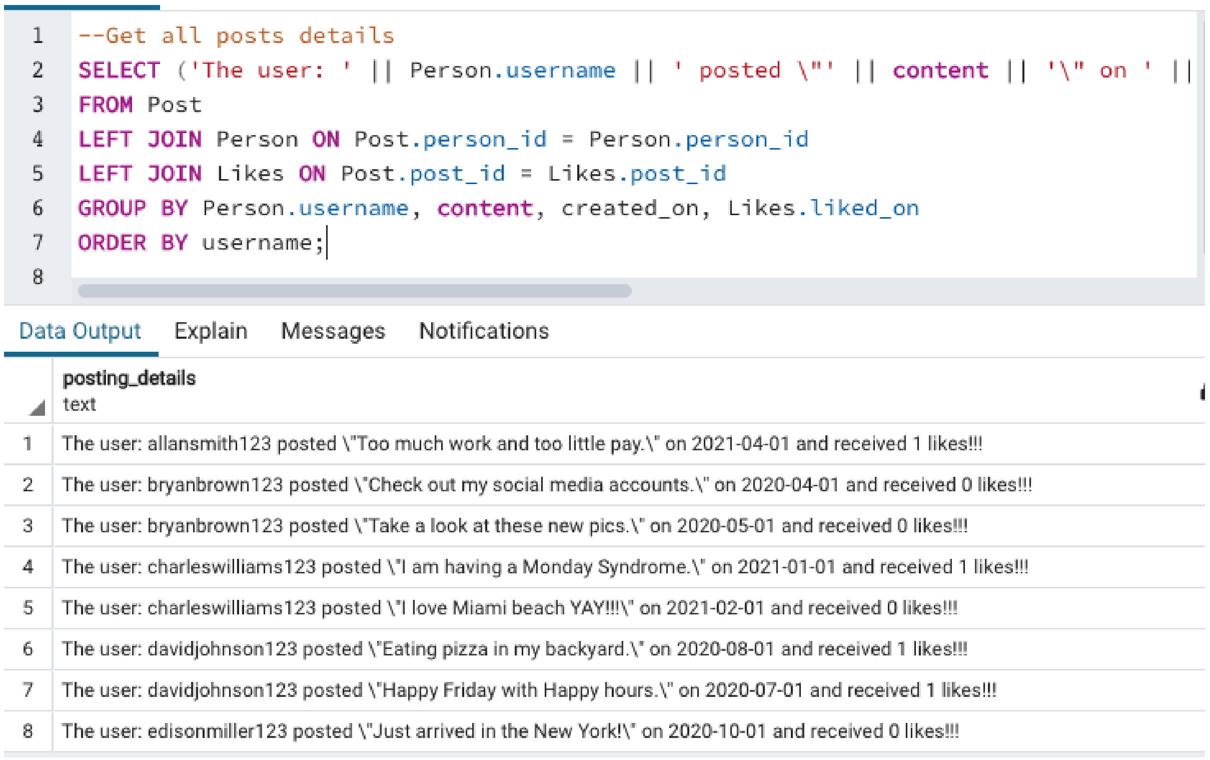


```sql
--Get all posts details
SELECT Person.username, content, created_on, Likes.liked_on
FROM Post
LEFT JOIN Person ON Post.person_id = Person.person_id
LEFT JOIN Likes ON Post.post_id = Likes.post_id
ORDER BY username;
```

Data Output | Explain | Messages | Notifications

	username character varying (20)	content character varying (255)	created_on date	liked_on date
1	allansmith123	Too much work and too little ...	2021-04-01	2021-06-03
2	bryanbrown123	Check out my social media ac...	2020-04-01	[null]
3	bryanbrown123	Take a look at these new pics.	2020-05-01	[null]
4	charleswilliams123	I love Miami beach YAY!!!	2021-02-01	[null]
5	charleswilliams123	I am having a Monday Syndro...	2021-01-01	2021-06-01
6	davidjohnson123	Happy Friday with Happy hours.	2020-07-01	2021-06-04
7	davidjohnson123	Eating pizza in my backyard.	2020-08-01	2021-06-02
8	edisonmiller123	Just arrived in the New York!	2020-10-01	[null]

--Posts Details



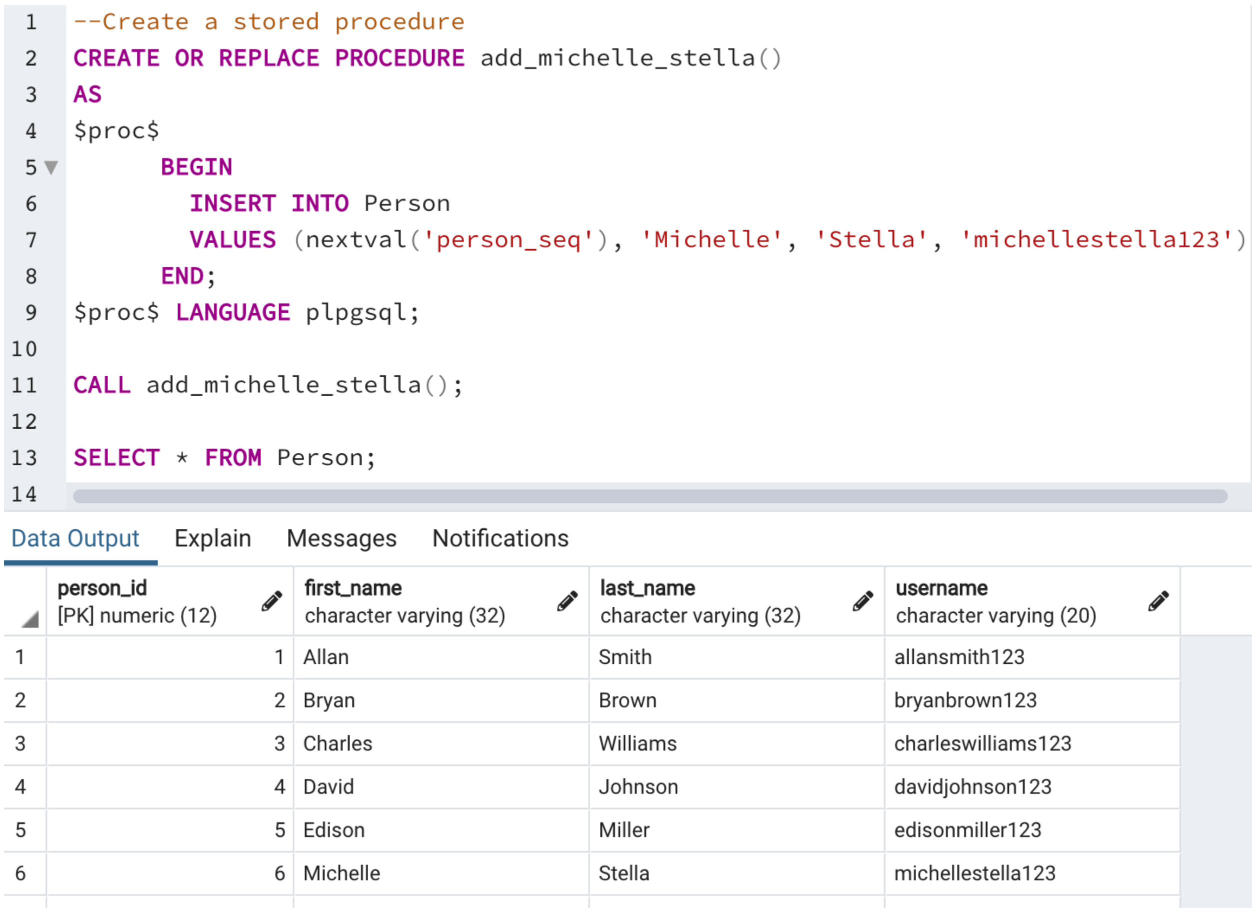
```sql
--Get all posts details
SELECT ('The user: ' || Person.username || ' posted \"' || content || '\" on ' ||
FROM Post
LEFT JOIN Person ON Post.person_id = Person.person_id
LEFT JOIN Likes ON Post.post_id = Likes.post_id
GROUP BY Person.username, content, created_on, Likes.liked_on
ORDER BY username;

```

Data Output | Explain | Messages | Notifications

	posting_details text
1	The user: allansmith123 posted \"Too much work and too little pay.\" on 2021-04-01 and received 1 likes!!!
2	The user: bryanbrown123 posted \"Check out my social media accounts.\" on 2020-04-01 and received 0 likes!!!
3	The user: bryanbrown123 posted \"Take a look at these new pics.\" on 2020-05-01 and received 0 likes!!!
4	The user: charleswilliams123 posted \"I am having a Monday Syndrome.\" on 2021-01-01 and received 1 likes!!!
5	The user: charleswilliams123 posted \"I love Miami beach YAY!!!\" on 2021-02-01 and received 0 likes!!!
6	The user: davidjohnson123 posted \"Eating pizza in my backyard.\" on 2020-08-01 and received 1 likes!!!
7	The user: davidjohnson123 posted \"Happy Friday with Happy hours.\" on 2020-07-01 and received 1 likes!!!
8	The user: edisonmiller123 posted \"Just arrived in the New York!\" on 2020-10-01 and received 0 likes!!!

--Concatenation

Step 3 – Create Hardcoded Procedure

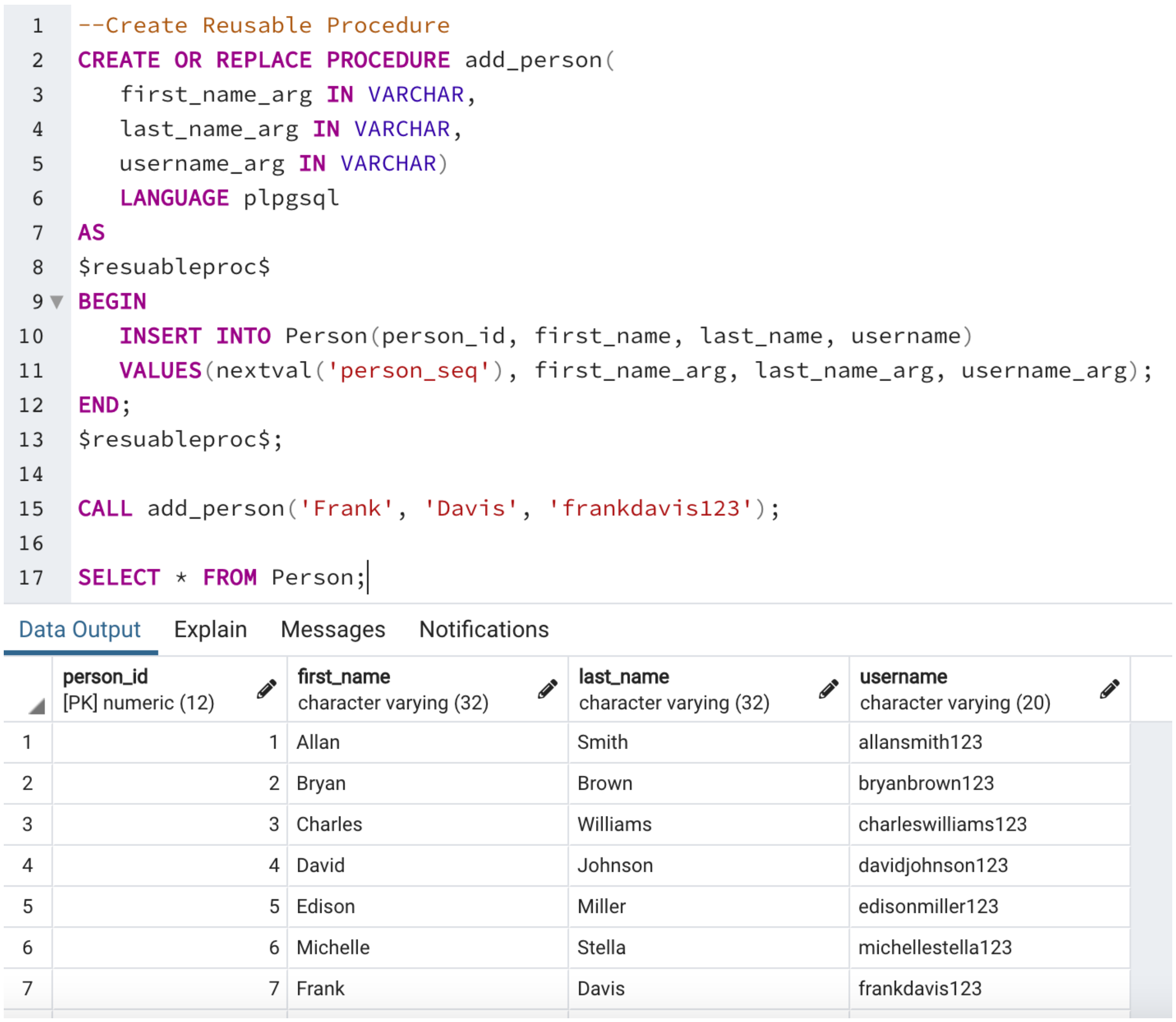


```
--Create a stored procedure
CREATE OR REPLACE PROCEDURE add_michelle_stella()
AS
$proc$
      BEGIN
        INSERT INTO Person
        VALUES (nextval('person_seq'), 'Michelle', 'Stella', 'michellestella123')
      END;
$proc$ LANGUAGE plpgsql;

CALL add_michelle_stella();

SELECT * FROM Person;

```

Data Output   Explain   Messages   Notifications

	person_id [PK] numeric (12)	first_name character varying (32)	last_name character varying (32)	username character varying (20)
1	1	Allan	Smith	allansmith123
2	2	Bryan	Brown	bryanbrown123
3	3	Charles	Williams	charleswilliams123
4	4	David	Johnson	davidjohnson123
5	5	Edison	Miller	edisonmiller123
6	6	Michelle	Stella	michellestella123

Step 4 – Create Reusable Procedure

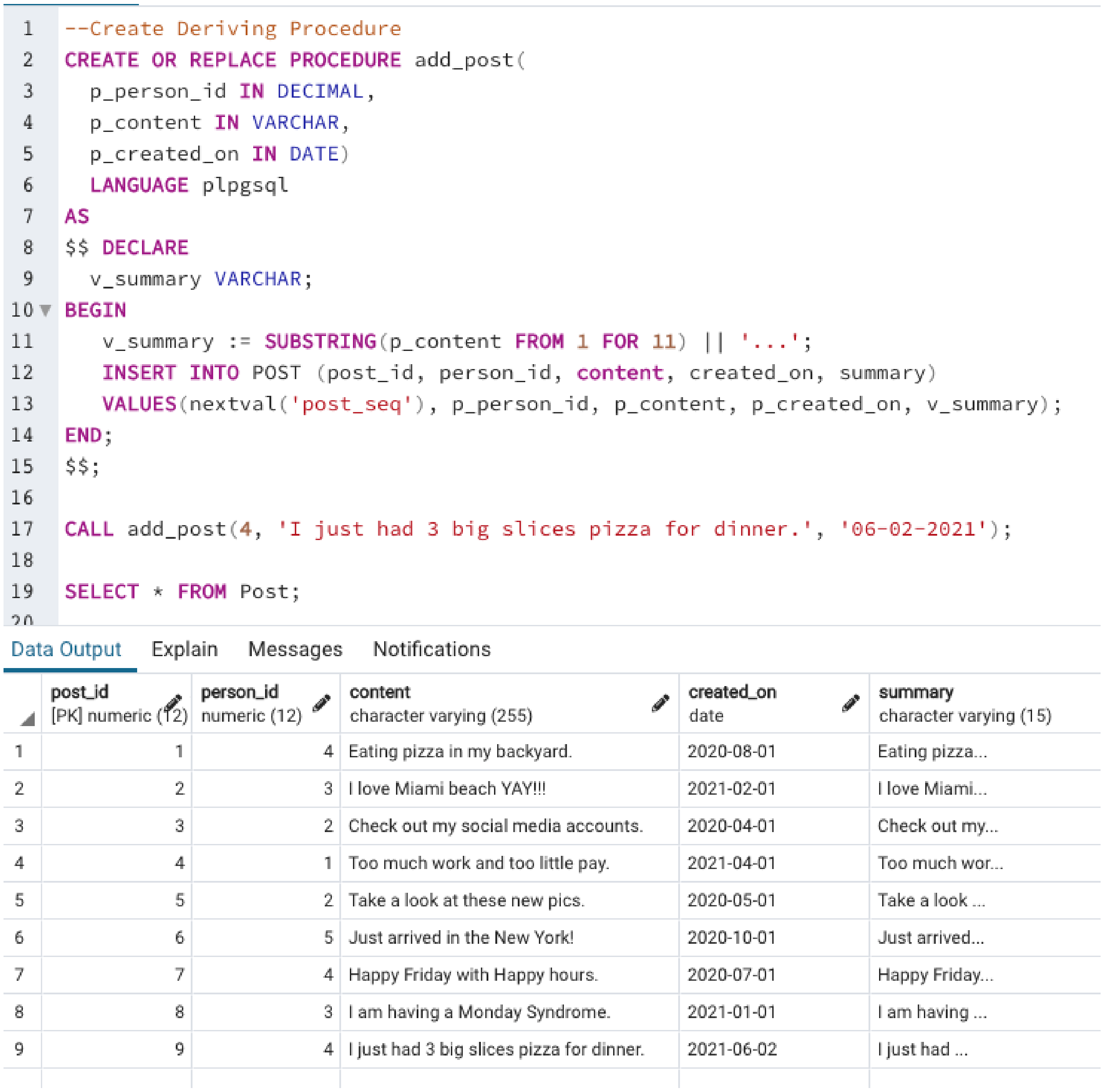


```
--Create Reusable Procedure
CREATE OR REPLACE PROCEDURE add_person(
   first_name_arg IN VARCHAR,
   last_name_arg IN VARCHAR,
   username_arg IN VARCHAR)
   LANGUAGE plpgsql
AS
$resuableproc$
BEGIN
   INSERT INTO Person(person_id, first_name, last_name, username)
   VALUES(nextval('person_seq'), first_name_arg, last_name_arg, username_arg);
END;
$resuableproc$;

CALL add_person('Frank', 'Davis', 'frankdavis123');

SELECT * FROM Person;
```

Data Output Explain Messages Notifications

	person_id [PK] numeric (12)	first_name character varying (32)	last_name character varying (32)	username character varying (20)
1	1	Allan	Smith	allansmith123
2	2	Bryan	Brown	bryanbrown123
3	3	Charles	Williams	charleswilliams123
4	4	David	Johnson	davidjohnson123
5	5	Edison	Miller	edisonmiller123
6	6	Michelle	Stella	michellestella123
7	7	Frank	Davis	frankdavis123

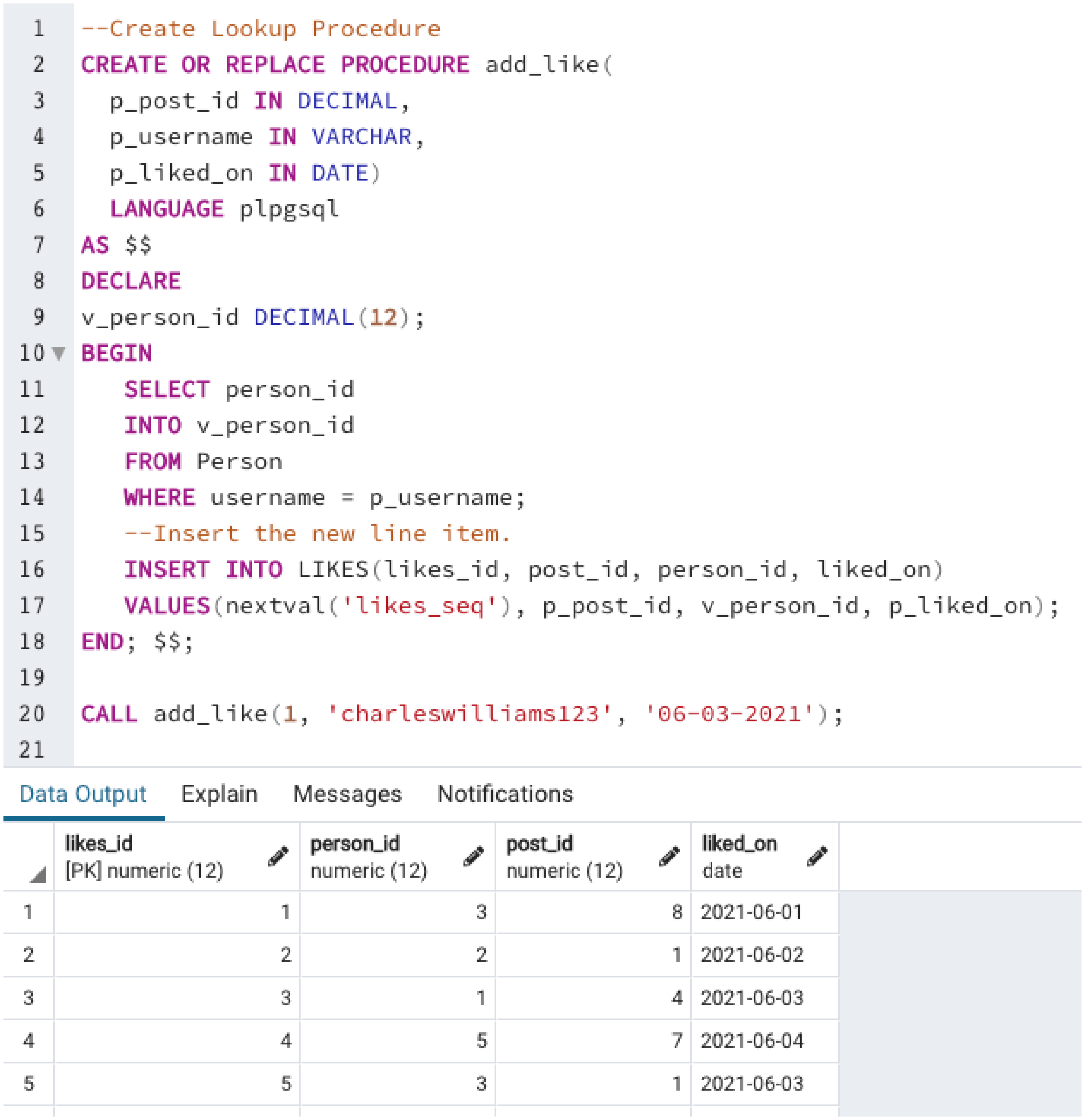
Step 5 – Create Deriving Procedure



```
--Create Deriving Procedure
CREATE OR REPLACE PROCEDURE add_post(
  p_person_id IN DECIMAL,
  p_content IN VARCHAR,
  p_created_on IN DATE)
  LANGUAGE plpgsql
AS
$$ DECLARE
  v_summary VARCHAR;
BEGIN
   v_summary := SUBSTRING(p_content FROM 1 FOR 11) || '...';
   INSERT INTO POST (post_id, person_id, content, created_on, summary)
   VALUES(nextval('post_seq'), p_person_id, p_content, p_created_on, v_summary);
END;
$$;

CALL add_post(4, 'I just had 3 big slices pizza for dinner.', '06-02-2021');

SELECT * FROM Post;
```

	post_id [PK] numeric (12)	person_id numeric (12)	content character varying (255)	created_on date	summary character varying (15)
1	1	4	Eating pizza in my backyard.	2020-08-01	Eating pizza...
2	2	3	I love Miami beach YAY!!!	2021-02-01	I love Miami...
3	3	2	Check out my social media accounts.	2020-04-01	Check out my...
4	4	1	Too much work and too little pay.	2021-04-01	Too much wor...
5	5	2	Take a look at these new pics.	2020-05-01	Take a look ...
6	6	5	Just arrived in the New York!	2020-10-01	Just arrived...
7	7	4	Happy Friday with Happy hours.	2020-07-01	Happy Friday...
8	8	3	I am having a Monday Syndrome.	2021-01-01	I am having ...
9	9	4	I just had 3 big slices pizza for dinner.	2021-06-02	I just had ...

Step 6 – Create Lookup Procedure

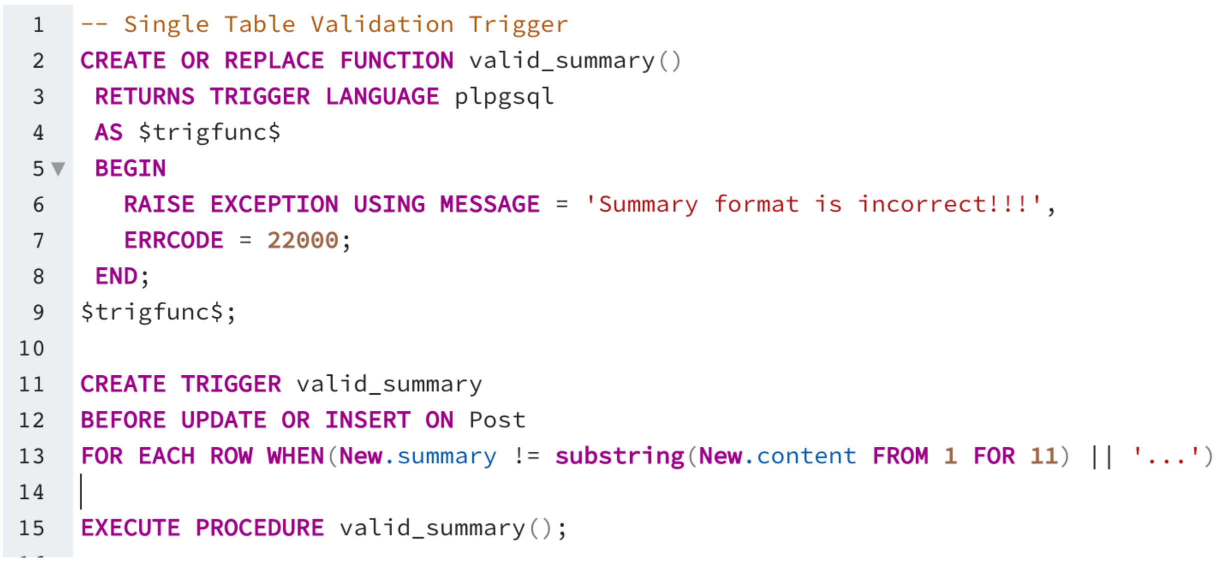


```
--Create Lookup Procedure
CREATE OR REPLACE PROCEDURE add_like(
  p_post_id IN DECIMAL,
  p_username IN VARCHAR,
  p_liked_on IN DATE)
  LANGUAGE plpgsql
AS $$
DECLARE
v_person_id DECIMAL(12);
BEGIN
   SELECT person_id
   INTO v_person_id
   FROM Person
   WHERE username = p_username;
   --Insert the new line item.
   INSERT INTO LIKES(likes_id, post_id, person_id, liked_on)
   VALUES(nextval('likes_seq'), p_post_id, v_person_id, p_liked_on);
END; $$;

CALL add_like(1, 'charleswilliams123', '06-03-2021');
```

Data Output | Explain | Messages | Notifications

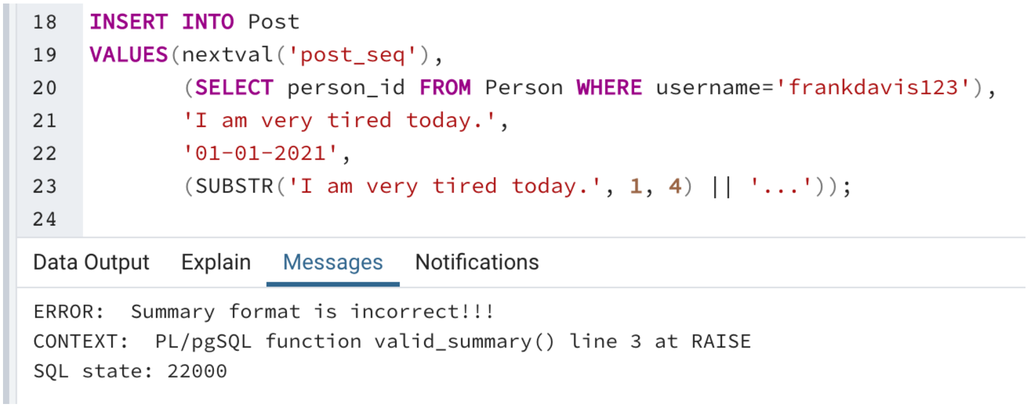
	likes_id [PK] numeric (12)	person_id numeric (12)	post_id numeric (12)	liked_on date
1	1	3	8	2021-06-01
2	2	2	1	2021-06-02
3	3	1	4	2021-06-03
4	4	5	7	2021-06-04
5	5	3	1	2021-06-03

Section Two – Triggers
Step 7 – Single Table Validation Trigger



```
-- Single Table Validation Trigger
CREATE OR REPLACE FUNCTION valid_summary()
 RETURNS TRIGGER LANGUAGE plpgsql
 AS $trigfunc$
 BEGIN
   RAISE EXCEPTION USING MESSAGE = 'Summary format is incorrect!!!',
   ERRCODE = 22000;
 END;
$trigfunc$;

CREATE TRIGGER valid_summary
BEFORE UPDATE OR INSERT ON Post
FOR EACH ROW WHEN(New.summary != substring(New.content FROM 1 FOR 11) || '...')

EXECUTE PROCEDURE valid_summary();
```

--create trigger

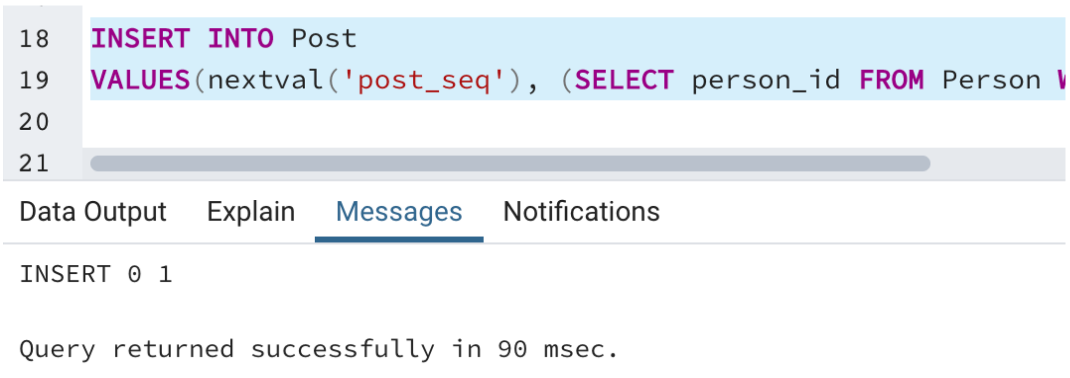


```
INSERT INTO Post
VALUES(nextval('post_seq'),
        (SELECT person_id FROM Person WHERE username='frankdavis123'),
        'I am very tired today.',
        '01-01-2021',
        (SUBSTR('I am very tired today.', 1, 4) || '...'));
```

Data Output Explain Messages Notifications

```
ERROR:  Summary format is incorrect!!!
CONTEXT:  PL/pgSQL function valid_summary() line 3 at RAISE
SQL state: 22000
```

--incorrect one

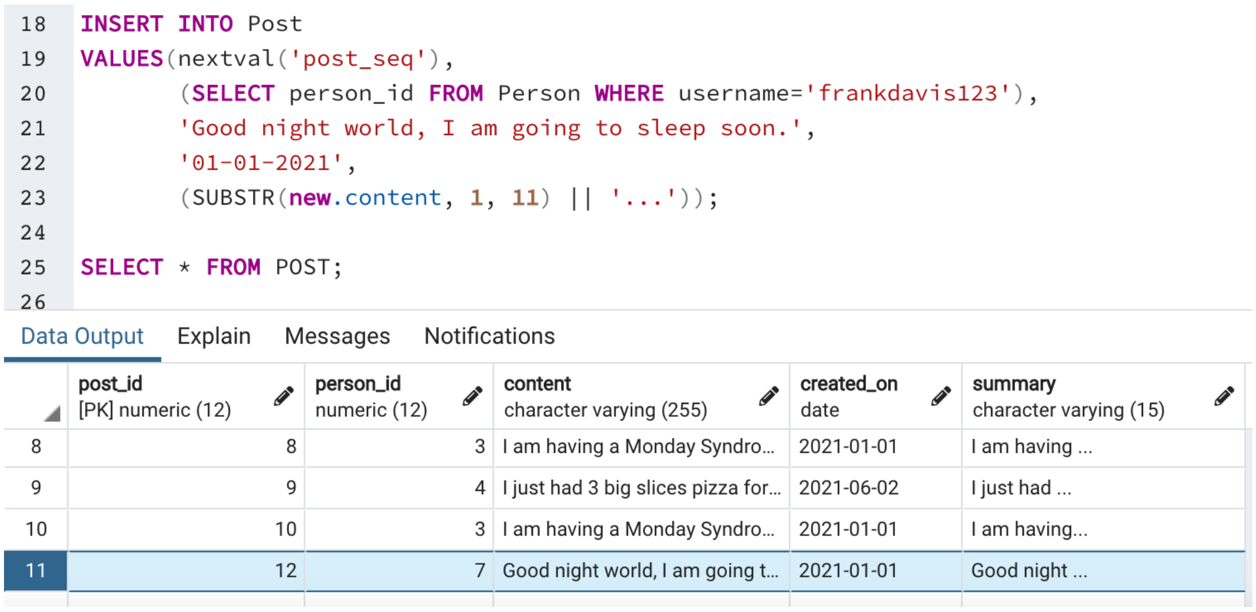


```
INSERT INTO Post
VALUES(nextval('post_seq'), (SELECT person_id FROM Person W
```

Data Output Explain Messages Notifications

```
INSERT 0 1

Query returned successfully in 90 msec.
```

--correct one: detail in next screenshot picture.



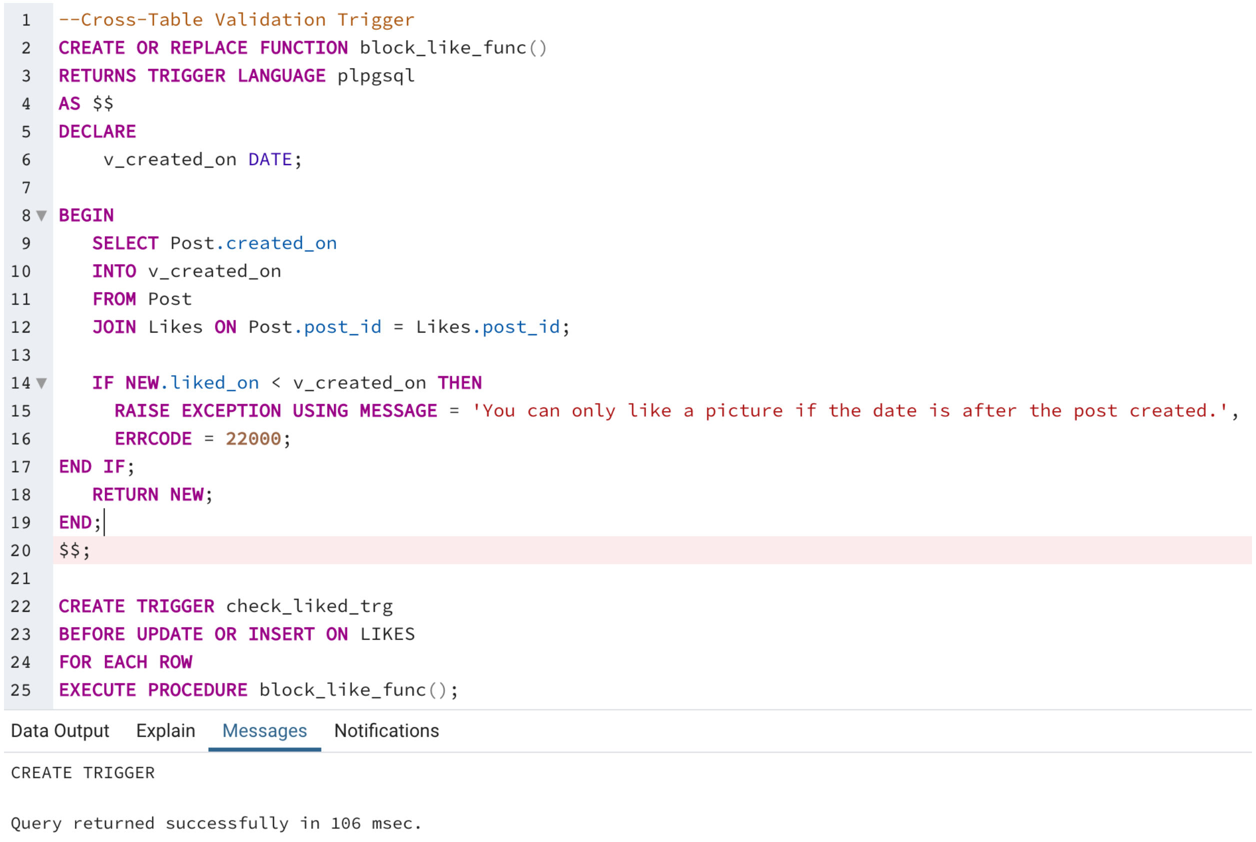
```sql
INSERT INTO Post
VALUES(nextval('post_seq'),
        (SELECT person_id FROM Person WHERE username='frankdavis123'),
        'Good night world, I am going to sleep soon.',
        '01-01-2021',
        (SUBSTR(new.content, 1, 11) || '...'));

SELECT * FROM POST;
```

Data Output | Explain | Messages | Notifications

	post_id [PK] numeric (12)	person_id numeric (12)	content character varying (255)	created_on date	summary character varying (15)
8	8	3	I am having a Monday Syndro...	2021-01-01	I am having ...
9	9	4	I just had 3 big slices pizza for...	2021-06-02	I just had ...
10	10	3	I am having a Monday Syndro...	2021-01-01	I am having...
11	12	7	Good night world, I am going t...	2021-01-01	Good night ...

--posted successfully

Step 8 – Cross-Table Validation Trigger

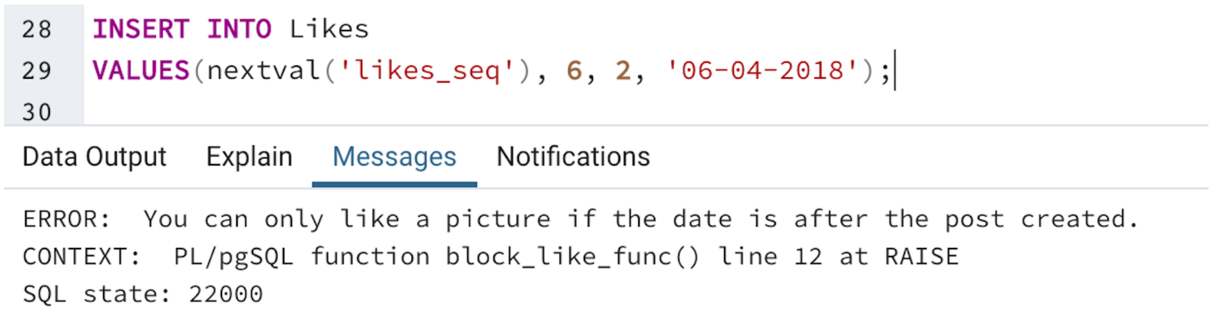


```sql
--Cross-Table Validation Trigger
CREATE OR REPLACE FUNCTION block_like_func()
RETURNS TRIGGER LANGUAGE plpgsql
AS $$
DECLARE
    v_created_on DATE;

BEGIN
   SELECT Post.created_on
   INTO v_created_on
   FROM Post
   JOIN Likes ON Post.post_id = Likes.post_id;

   IF NEW.liked_on < v_created_on THEN
     RAISE EXCEPTION USING MESSAGE = 'You can only like a picture if the date is after the post created.',
     ERRCODE = 22000;
END IF;
   RETURN NEW;
END;
$$;

CREATE TRIGGER check_liked_trg
BEFORE UPDATE OR INSERT ON LIKES
FOR EACH ROW
EXECUTE PROCEDURE block_like_func();
```

Data Output | Explain | Messages | Notifications

```
CREATE TRIGGER

Query returned successfully in 106 msec.
```

--create trigger

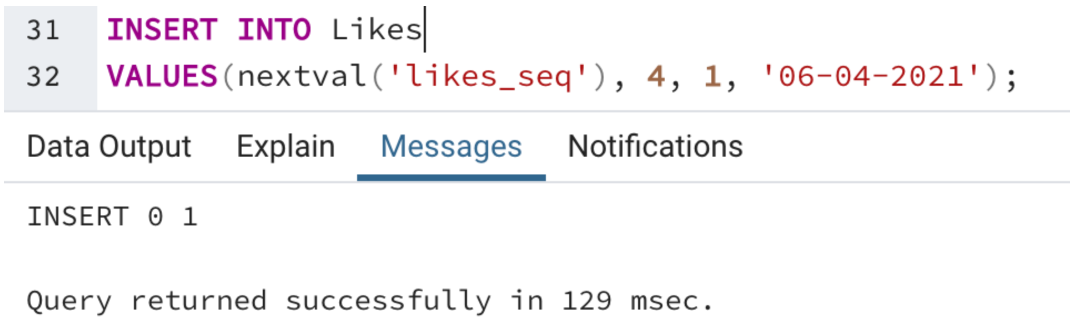


```
INSERT INTO Likes
VALUES(nextval('likes_seq'), 6, 2, '06-04-2018');
```

Data Output Explain Messages Notifications

```
ERROR:  You can only like a picture if the date is after the post created.
CONTEXT:  PL/pgSQL function block_like_func() line 12 at RAISE
SQL state: 22000
```

--incorrect one

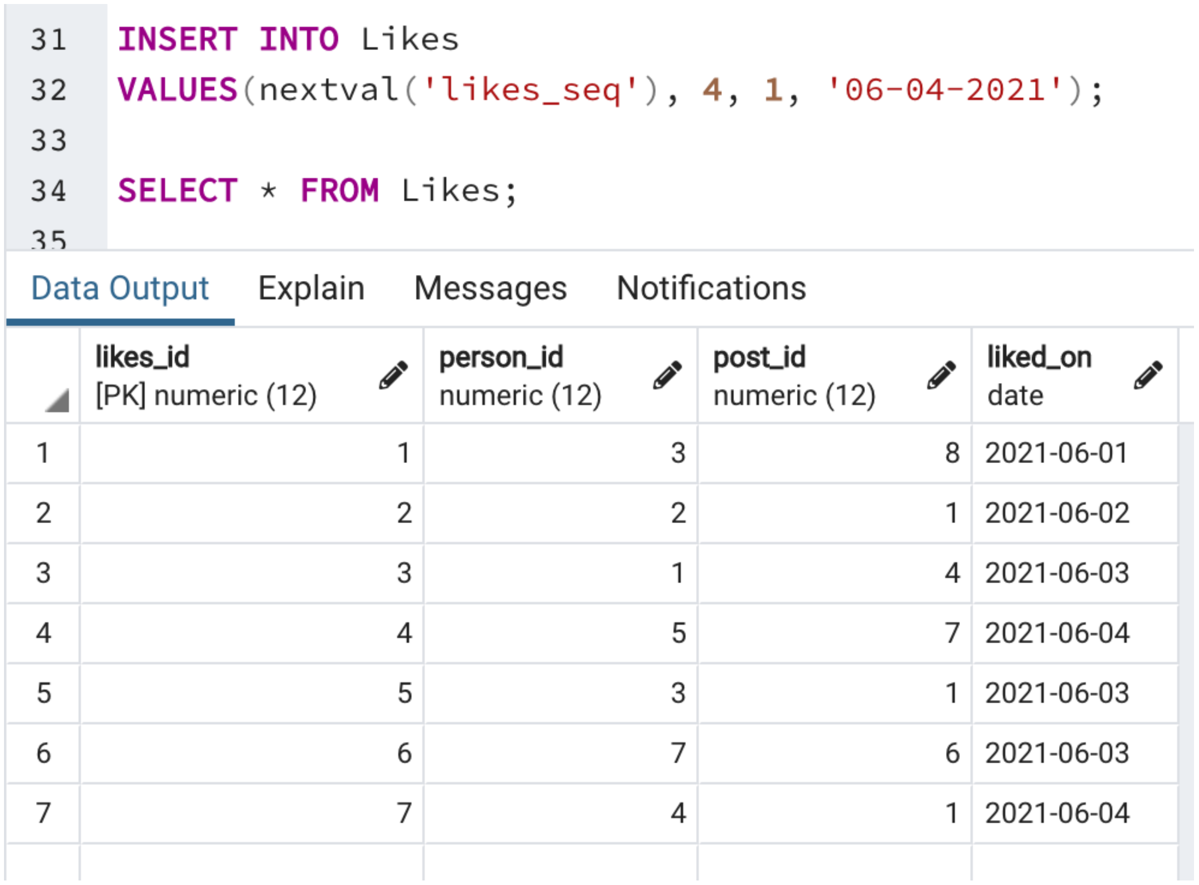


```
INSERT INTO Likes
VALUES(nextval('likes_seq'), 4, 1, '06-04-2021');
```

Data Output Explain Messages Notifications

```
INSERT 0 1

Query returned successfully in 129 msec.
```

--correct one



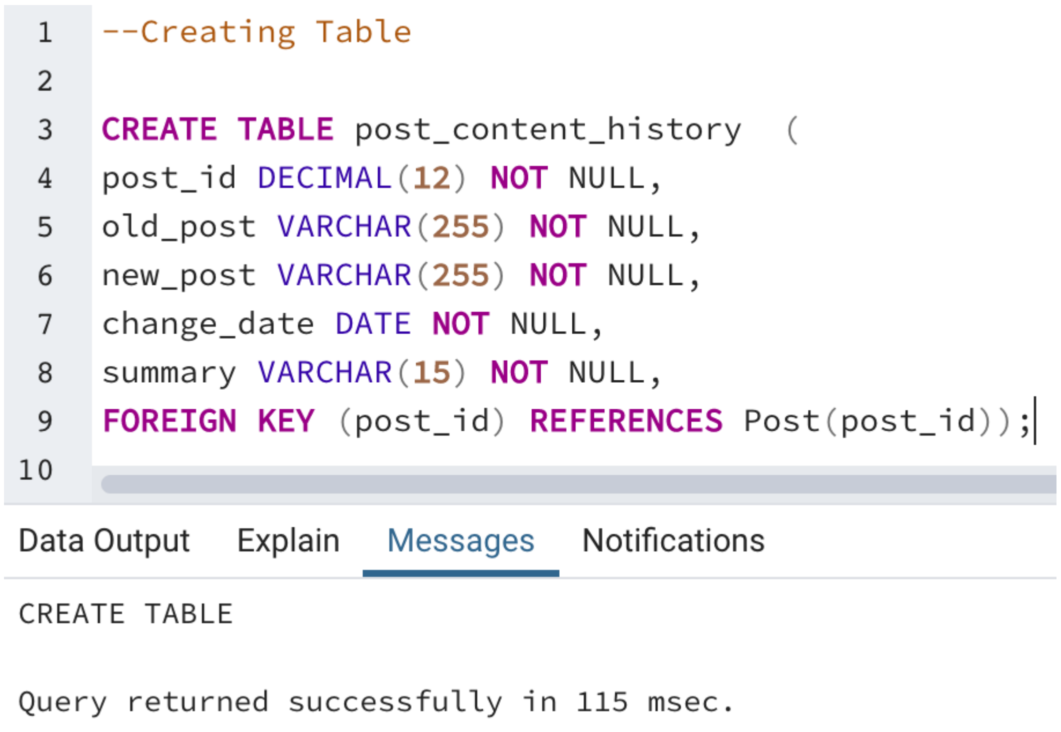
```
INSERT INTO Likes
VALUES(nextval('likes_seq'), 4, 1, '06-04-2021');

SELECT * FROM Likes;
```

Data Output Explain Messages Notifications

	likes_id [PK] numeric (12)	person_id numeric (12)	post_id numeric (12)	liked_on date
1	1	3	8	2021-06-01
2	2	2	1	2021-06-02
3	3	1	4	2021-06-03
4	4	5	7	2021-06-04
5	5	3	1	2021-06-03
6	6	7	6	2021-06-03
7	7	4	1	2021-06-04

--posted successfully

Step 9 – History Trigger

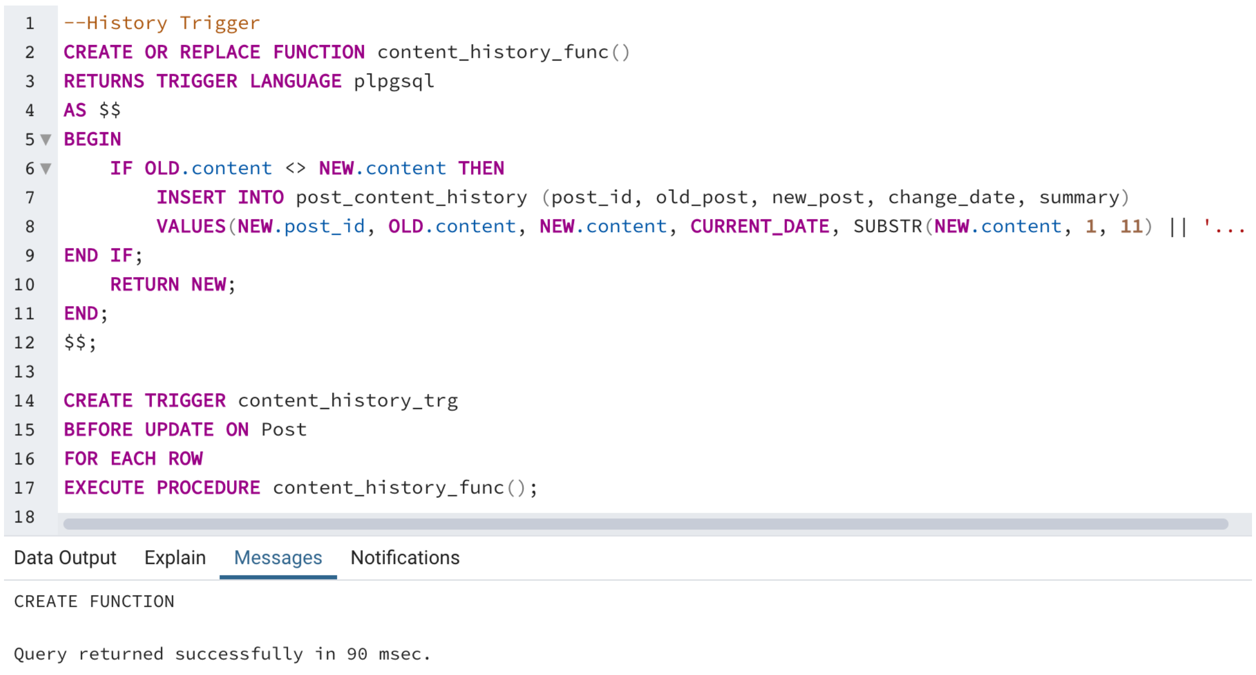


```
--Creating Table

CREATE TABLE post_content_history  (
post_id DECIMAL(12) NOT NULL,
old_post VARCHAR(255) NOT NULL,
new_post VARCHAR(255) NOT NULL,
change_date DATE NOT NULL,
summary VARCHAR(15) NOT NULL,
FOREIGN KEY (post_id) REFERENCES Post(post_id));
```

Data Output Explain Messages Notifications

```
CREATE TABLE

Query returned successfully in 115 msec.
```

--create table

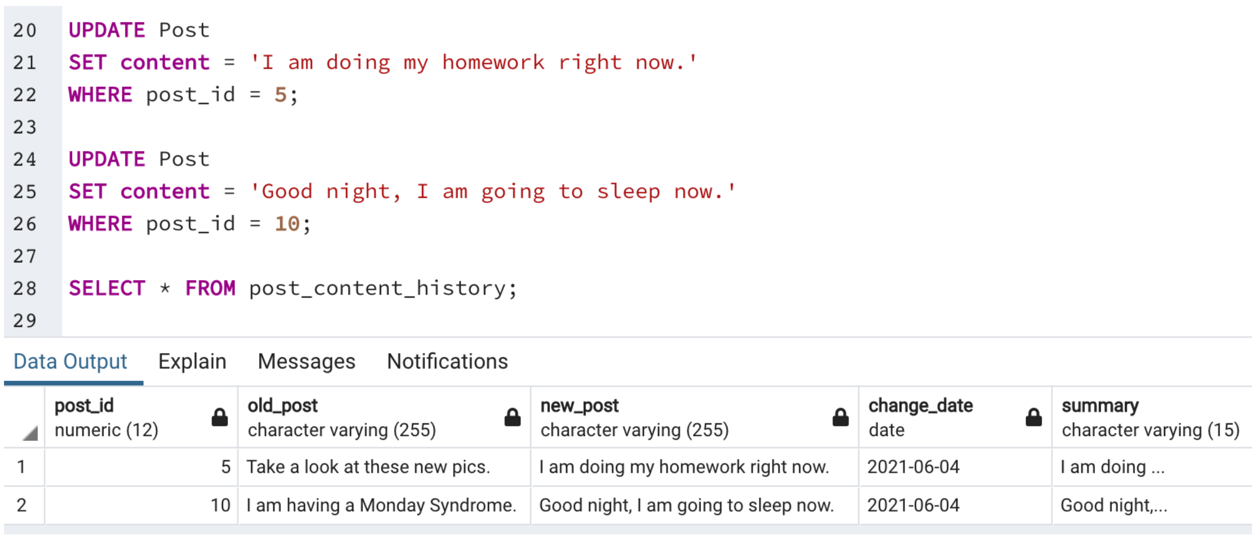


```
--History Trigger
CREATE OR REPLACE FUNCTION content_history_func()
RETURNS TRIGGER LANGUAGE plpgsql
AS $$
BEGIN
    IF OLD.content <> NEW.content THEN
        INSERT INTO post_content_history (post_id, old_post, new_post, change_date, summary)
        VALUES(NEW.post_id, OLD.content, NEW.content, CURRENT_DATE, SUBSTR(NEW.content, 1, 11) || '...
END IF;
    RETURN NEW;
END;
$$;

CREATE TRIGGER content_history_trg
BEFORE UPDATE ON Post
FOR EACH ROW
EXECUTE PROCEDURE content_history_func();
```

Data Output Explain Messages Notifications

```
CREATE FUNCTION

Query returned successfully in 90 msec.
```

--create function and trigger

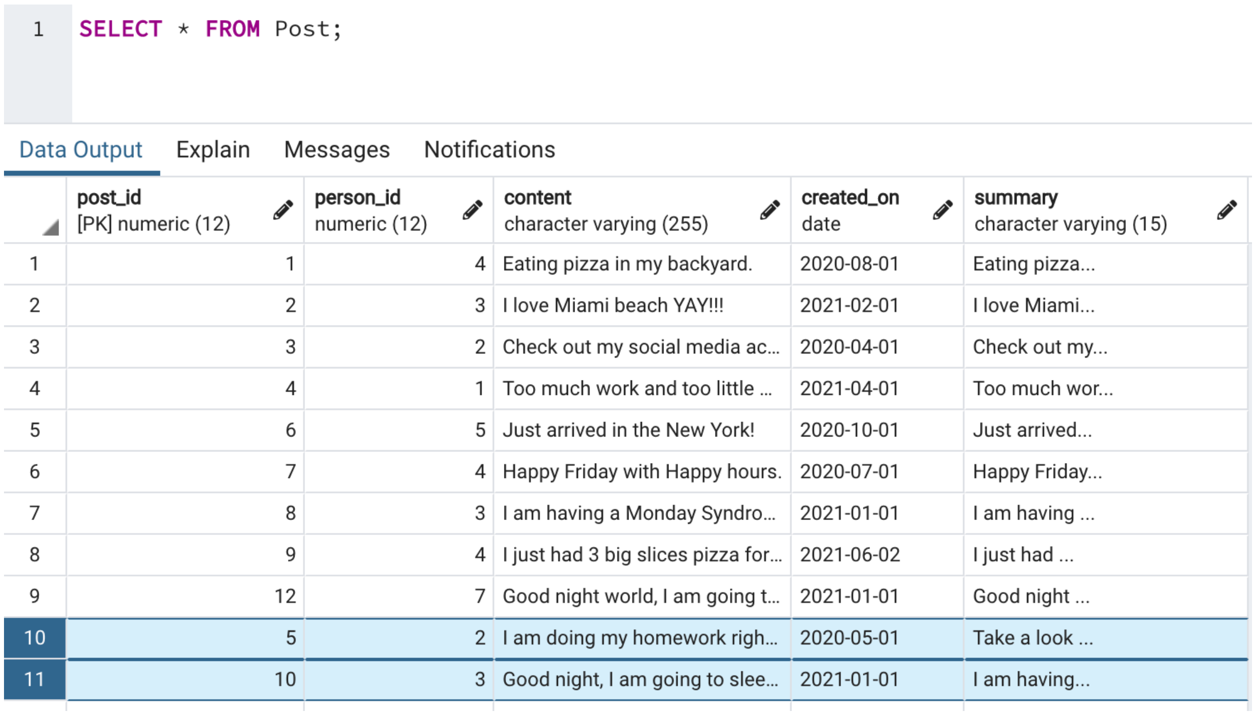


```sql
UPDATE Post
SET content = 'I am doing my homework right now.'
WHERE post_id = 5;

UPDATE Post
SET content = 'Good night, I am going to sleep now.'
WHERE post_id = 10;

SELECT * FROM post_content_history;

```

Data Output | Explain | Messages | Notifications

	post_id numeric (12)	old_post character varying (255)	new_post character varying (255)	change_date date	summary character varying (15)
1	5	Take a look at these new pics.	I am doing my homework right now.	2021-06-04	I am doing ...
2	10	I am having a Monday Syndrome.	Good night, I am going to sleep now.	2021-06-04	Good night,...

--SELECT * FROM post_content_history;
--updated successfully



```sql
SELECT * FROM Post;
```

Data Output | Explain | Messages | Notifications

	post_id [PK] numeric (12)	person_id numeric (12)	content character varying (255)	created_on date	summary character varying (15)
1	1	4	Eating pizza in my backyard.	2020-08-01	Eating pizza...
2	2	3	I love Miami beach YAY!!!	2021-02-01	I love Miami...
3	3	2	Check out my social media ac...	2020-04-01	Check out my...
4	4	1	Too much work and too little ...	2021-04-01	Too much wor...
5	6	5	Just arrived in the New York!	2020-10-01	Just arrived...
6	7	4	Happy Friday with Happy hours.	2020-07-01	Happy Friday...
7	8	3	I am having a Monday Syndro...	2021-01-01	I am having ...
8	9	4	I just had 3 big slices pizza for...	2021-06-02	I just had ...
9	12	7	Good night world, I am going t...	2021-01-01	Good night ...
10	5	2	I am doing my homework righ...	2020-05-01	Take a look ...
11	10	3	Good night, I am going to slee...	2021-01-01	I am having...

--SELECT * FROM Post;
--updated successfully (last two)

Section Three – Concurrency
Step 10 – Issues with No Concurrency Control

I am choosing to show the “Lost Updates Issue”

For virtual purposes, I am using yellow color for transaction1 and green color for transaction 2.

Lost Update Schedule				
Time	**Step**	**Explanation**	**Data**	
1	Transaction 1: Read the value from row 4.	Data table is 1, 2, 3, 4, 5; so the transaction 1 read the value as "4"	T1 = 4	
2	Transaction 2: Read the value from row 2.	Data table is 1, 2, 3, 4, 5; so the transaction 2 read the value as "2"	T2 = 2	
3	Transaction 1: Multiply that value times 3.	Multiply the value "4" times 3 from the row 4 by transaction 1	T1*3 = 12	
4	Transaction 2: Write that value to row 4.	Write the value "2" gotten from transaction 2 to the row 4	1, 2, 3, 2, 5	
5	Transaction 1: Write the result to row 3.	Write the result '12' gotten from transaction 1 to the row 3	1, 2, 12, 4, 5	
6	Transaction 1: Write the literal value “8” to row 2.	Write the number '8' to the row 2 according to transaction 1	1, 8, 12, 4, 5	
7	Transaction 2: Write the literal value “15” to row 3.	Write the number '15' to the row 3 according to transaction 2	1, 2, 15, 2, 5	
8	Transaction 1: Write the literal value “20” to row 5.	Write the number '20' to the row 5 according to transaction 1	1, 8, 12, 4, 20	
9	Transaction 1: Commit.	Transaction 1 commited its current value, but it has no effects because next step, 10, will overwrite this update, and the value associated with transaction 1 will be discarded.	1, 8, 12, 4, 20	thrown
10	Transaction 2: Commit.	This transaction 2 wins because it is the last commit, whichever transaction updates last wins, the previous update (transaction 1 here) has no effect and will be lost when the same transaction is executing on the same thing at the same time.	1, 2, 15, 2, 5	upated

The first update does not matter but the last transaction wins, that is what happens when the same transaction is executing on the same thing at the same time, the results from the last transaction will overwrite the previous transactions if the transactions run simultaneously without concurrency control. In short, whichever transaction updates last wins, the previous update has no effect and will be lost.

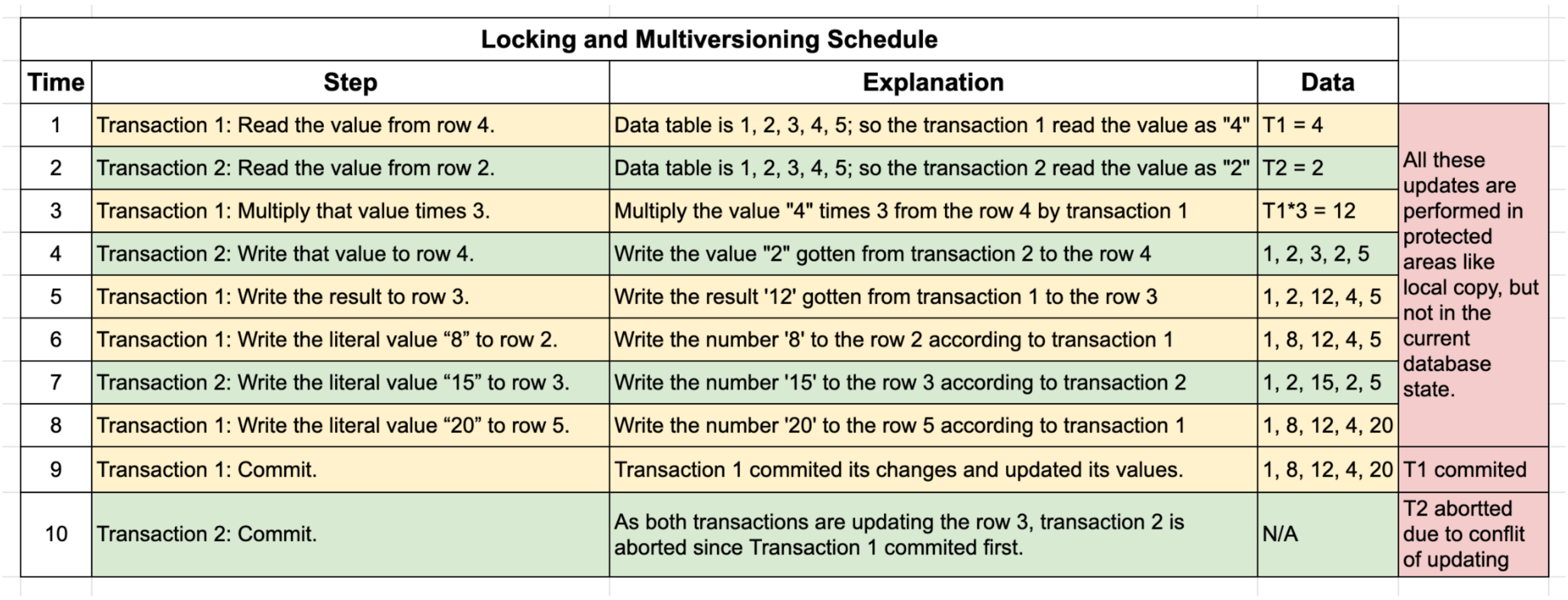
Data Table
1
2
15
2
5

--The current updated data table after the transactions complete.

Step 11 – Issues with Locking and Multiversioning

a. The use of locking and multiversioning

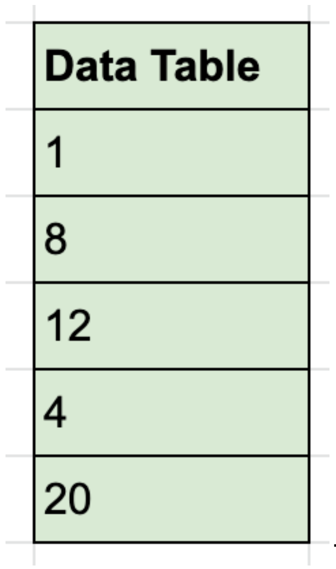
With multiversioning, it is like timestamping that remembers both old and new values and gives you the history of the values. When a transaction goes to read the value, it is always reading the value as it was when it just started, like a snapshot at that moment. Share locks are no longer required, and continually assigns a new version to the database each time changes are applied.



Locking and Multiversioning Schedule

Time	Step	Explanation	Data	
1	Transaction 1: Read the value from row 4.	Data table is 1, 2, 3, 4, 5; so the transaction 1 read the value as "4"	T1 = 4	All these updates are performed in protected areas like local copy, but not in the current database state.
2	Transaction 2: Read the value from row 2.	Data table is 1, 2, 3, 4, 5; so the transaction 2 read the value as "2"	T2 = 2	
3	Transaction 1: Multiply that value times 3.	Multiply the value "4" times 3 from the row 4 by transaction 1	T1*3 = 12	
4	Transaction 2: Write that value to row 4.	Write the value "2" gotten from transaction 2 to the row 4	1, 2, 3, 2, 5	
5	Transaction 1: Write the result to row 3.	Write the result '12' gotten from transaction 1 to the row 3	1, 2, 12, 4, 5	
6	Transaction 1: Write the literal value “8” to row 2.	Write the number '8' to the row 2 according to transaction 1	1, 8, 12, 4, 5	
7	Transaction 2: Write the literal value “15” to row 3.	Write the number '15' to the row 3 according to transaction 2	1, 2, 15, 2, 5	
8	Transaction 1: Write the literal value “20” to row 5.	Write the number '20' to the row 5 according to transaction 1	1, 8, 12, 4, 20	
9	Transaction 1: Commit.	Transaction 1 commited its changes and updated its values.	1, 8, 12, 4, 20	T1 commited
10	Transaction 2: Commit.	As both transactions are updating the row 3, transaction 2 is aborted since Transaction 1 commited first.	N/A	T2 aborted due to conflit of updating

Transaction 1 committed the change first, so if transaction 2 wants to update row 3 which has already been updated by transaction 1, it has to restart the transaction again.



Data Table
1
8
12
4
20

The current updated data table after the transactions complete.

b. Could a schedule of these transactions result in a deadlock?

Since I am using Postgres, it always combines multiversioning with locking. When multiversioning is used, shared locks are no longer necessary. With multiversioning, only updating or deleting the same rows in a different order can cause deadlocks between concurrent transactions. Reads do not enter the picture.

Because the transactions above are using both multiversioning and locking; therefore, these transactions do not result in a deadlock. With multiversioning, only updating or deleting the same rows in a different order can cause deadlocks between concurrent transactions. Based on the transaction above, a deadlock is not occurring since there is no updating and deleting the same row at the same time during the concurrent transaction. While both transactions 1 and 2 try to update the same row, row 3, but transaction 1 updates it first, and when transaction 2 tries to update it, transaction 1 is already in row 2 updating row 2. Moreover, reading alone does not cause any deadlock. Therefore, the schedule of these transactions do not result in a deadlock.