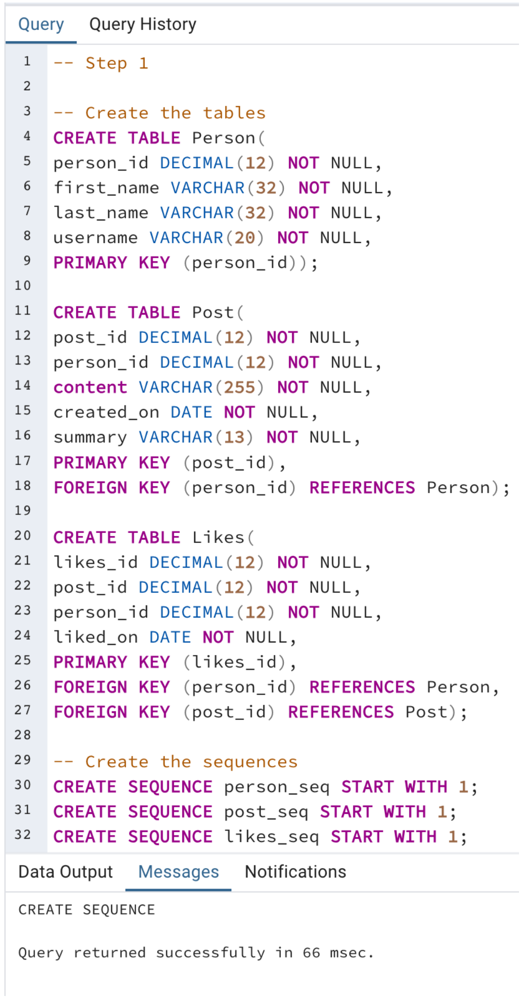
# MET CS 669 Database Design and Implementation for Business
# Lab 4: Procedural SQL and Concurrency

## *Section One – Stored Procedures*

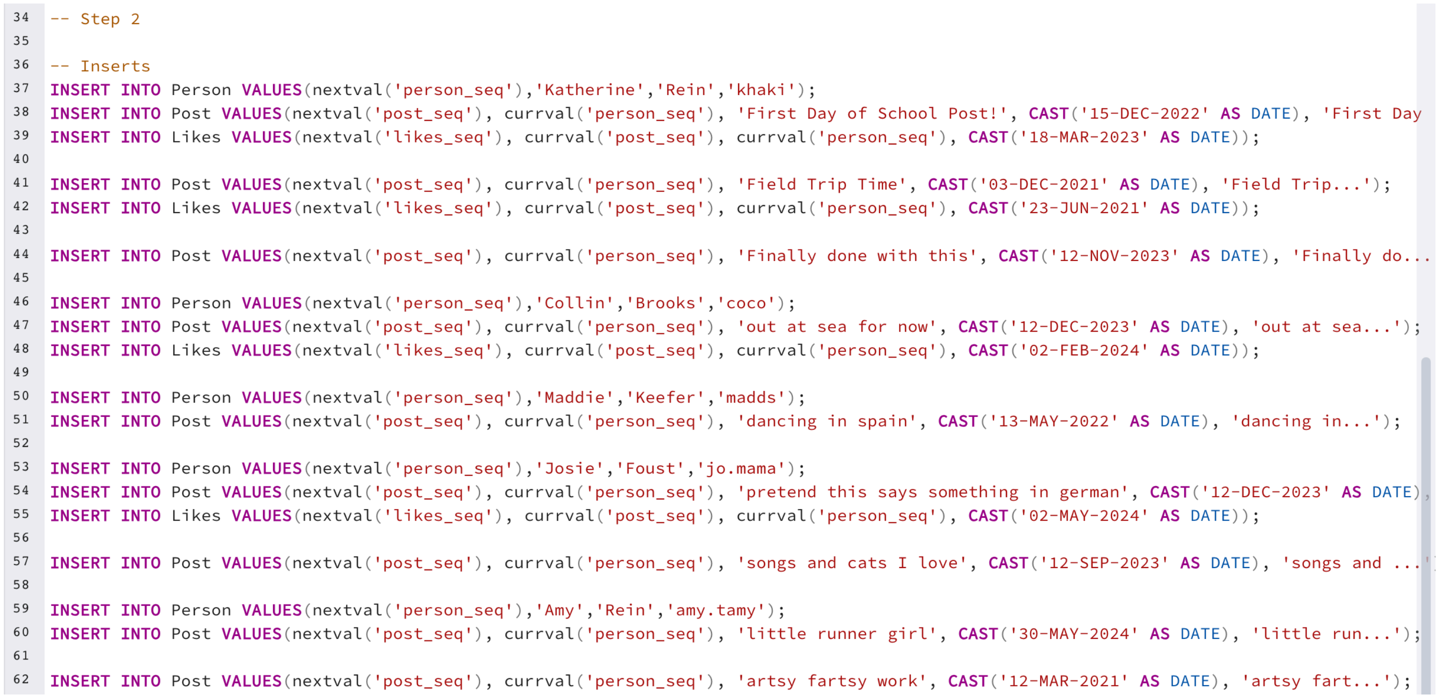
1. *Create Table Structure* – Create the tables in the social networking schema, including all of their columns, datatypes, and constraints. Create sequences for each table; these will be used to generate the primary and foreign key values in Step #2.



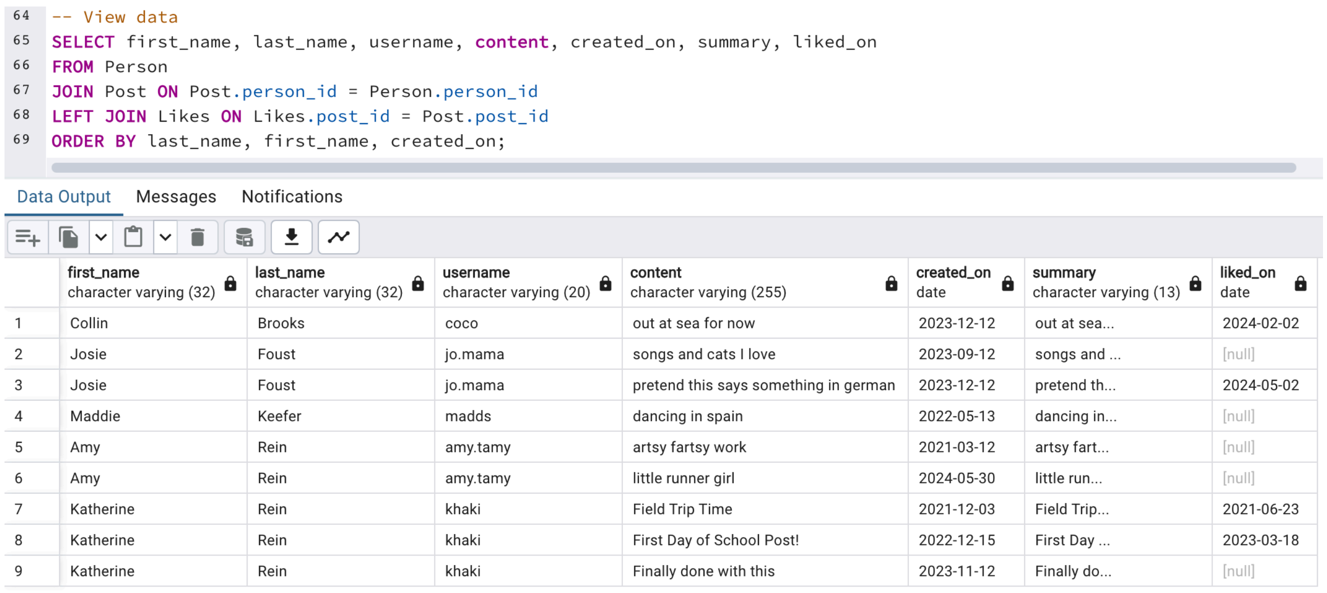
```
-- Step 1

-- Create the tables
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 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(13) NOT NULL,
PRIMARY KEY (post_id),
FOREIGN KEY (person_id) REFERENCES Person);

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

-- Create the sequences
CREATE SEQUENCE person_seq START WITH 1;
CREATE SEQUENCE post_seq START WITH 1;
CREATE SEQUENCE likes_seq START WITH 1;
```

```
CREATE SEQUENCE

Query returned successfully in 66 msec.
```

2. *Populate Tables* – Populate the tables with data, ensuring that there are at least 5 people, at least 8 posts, and at least 4 likes. Make sure to use sequences to generate the primary and foreign key values. Most of the fields are self-explanatory. As far as the "content" field in Post, make them whatever you like, such as "Take a look at these new pics" or "Just arrived in the Bahamas", and set the summary as the first 10 characters of the content, followed by "…".



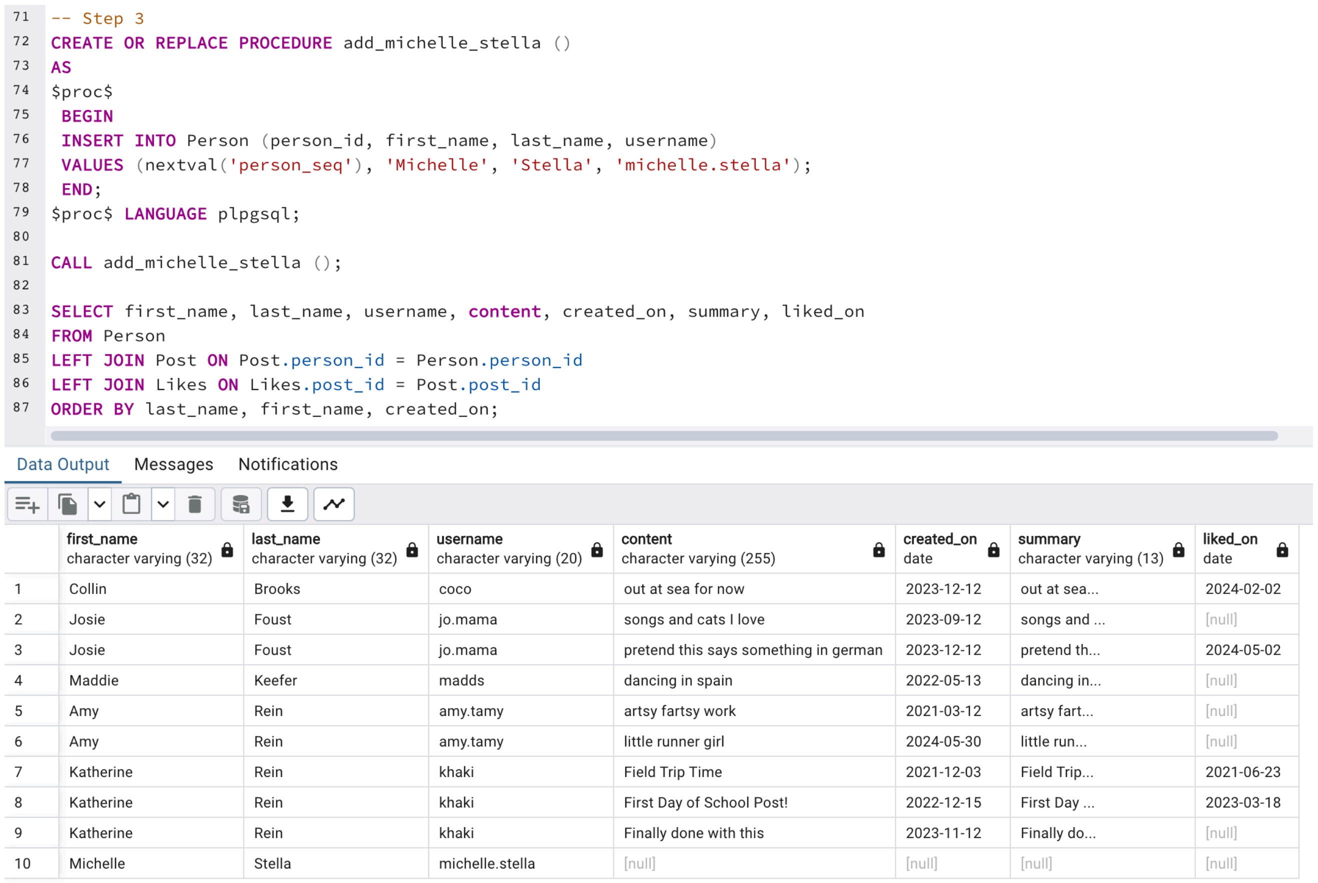
```sql
-- Step 2

-- Inserts
INSERT INTO Person VALUES(nextval('person_seq'),'Katherine','Rein','khaki');
INSERT INTO Post VALUES(nextval('post_seq'), currval('person_seq'), 'First Day of School Post!', CAST('15-DEC-2022' AS DATE), 'First Day
INSERT INTO Likes VALUES(nextval('likes_seq'), currval('post_seq'), currval('person_seq'), CAST('18-MAR-2023' AS DATE));

INSERT INTO Post VALUES(nextval('post_seq'), currval('person_seq'), 'Field Trip Time', CAST('03-DEC-2021' AS DATE), 'Field Trip...');
INSERT INTO Likes VALUES(nextval('likes_seq'), currval('post_seq'), currval('person_seq'), CAST('23-JUN-2021' AS DATE));

INSERT INTO Post VALUES(nextval('post_seq'), currval('person_seq'), 'Finally done with this', CAST('12-NOV-2023' AS DATE), 'Finally do...

INSERT INTO Person VALUES(nextval('person_seq'),'Collin','Brooks','coco');
INSERT INTO Post VALUES(nextval('post_seq'), currval('person_seq'), 'out at sea for now', CAST('12-DEC-2023' AS DATE), 'out at sea...');
INSERT INTO Likes VALUES(nextval('likes_seq'), currval('post_seq'), currval('person_seq'), CAST('02-FEB-2024' AS DATE));

INSERT INTO Person VALUES(nextval('person_seq'),'Maddie','Keefer','madds');
INSERT INTO Post VALUES(nextval('post_seq'), currval('person_seq'), 'dancing in spain', CAST('13-MAY-2022' AS DATE), 'dancing in...');

INSERT INTO Person VALUES(nextval('person_seq'),'Josie','Foust','jo.mama');
INSERT INTO Post VALUES(nextval('post_seq'), currval('person_seq'), 'pretend this says something in german', CAST('12-DEC-2023' AS DATE),
INSERT INTO Likes VALUES(nextval('likes_seq'), currval('post_seq'), currval('person_seq'), CAST('02-MAY-2024' AS DATE));

INSERT INTO Post VALUES(nextval('post_seq'), currval('person_seq'), 'songs and cats I love', CAST('12-SEP-2023' AS DATE), 'songs and ...'

INSERT INTO Person VALUES(nextval('person_seq'),'Amy','Rein','amy.tamy');
INSERT INTO Post VALUES(nextval('post_seq'), currval('person_seq'), 'little runner girl', CAST('30-MAY-2024' AS DATE), 'little run...');

INSERT INTO Post VALUES(nextval('post_seq'), currval('person_seq'), 'artsy fartsy work', CAST('12-MAR-2021' AS DATE), 'artsy fart...');
```



```
-- View data
SELECT first_name, last_name, username, content, created_on, summary, liked_on
FROM Person
JOIN Post ON Post.person_id = Person.person_id
LEFT JOIN Likes ON Likes.post_id = Post.post_id
ORDER BY last_name, first_name, created_on;
```

| | first_name character varying (32) | last_name character varying (32) | username character varying (20) | content character varying (255) | created_on date | summary character varying (13) | liked_on date |
|---|---|---|---|---|---|---|---|
| 1 | Collin | Brooks | coco | out at sea for now | 2023-12-12 | out at sea... | 2024-02-02 |
| 2 | Josie | Foust | jo.mama | songs and cats I love | 2023-09-12 | songs and ... | [null] |
| 3 | Josie | Foust | jo.mama | pretend this says something in german | 2023-12-12 | pretend th... | 2024-05-02 |
| 4 | Maddie | Keefer | madds | dancing in spain | 2022-05-13 | dancing in... | [null] |
| 5 | Amy | Rein | amy.tamy | artsy fartsy work | 2021-03-12 | artsy fart... | [null] |
| 6 | Amy | Rein | amy.tamy | little runner girl | 2024-05-30 | little run... | [null] |
| 7 | Katherine | Rein | khaki | Field Trip Time | 2021-12-03 | Field Trip... | 2021-06-23 |
| 8 | Katherine | Rein | khaki | First Day of School Post! | 2022-12-15 | First Day ... | 2023-03-18 |
| 9 | Katherine | Rein | khaki | Finally done with this | 2023-11-12 | Finally do... | [null] |

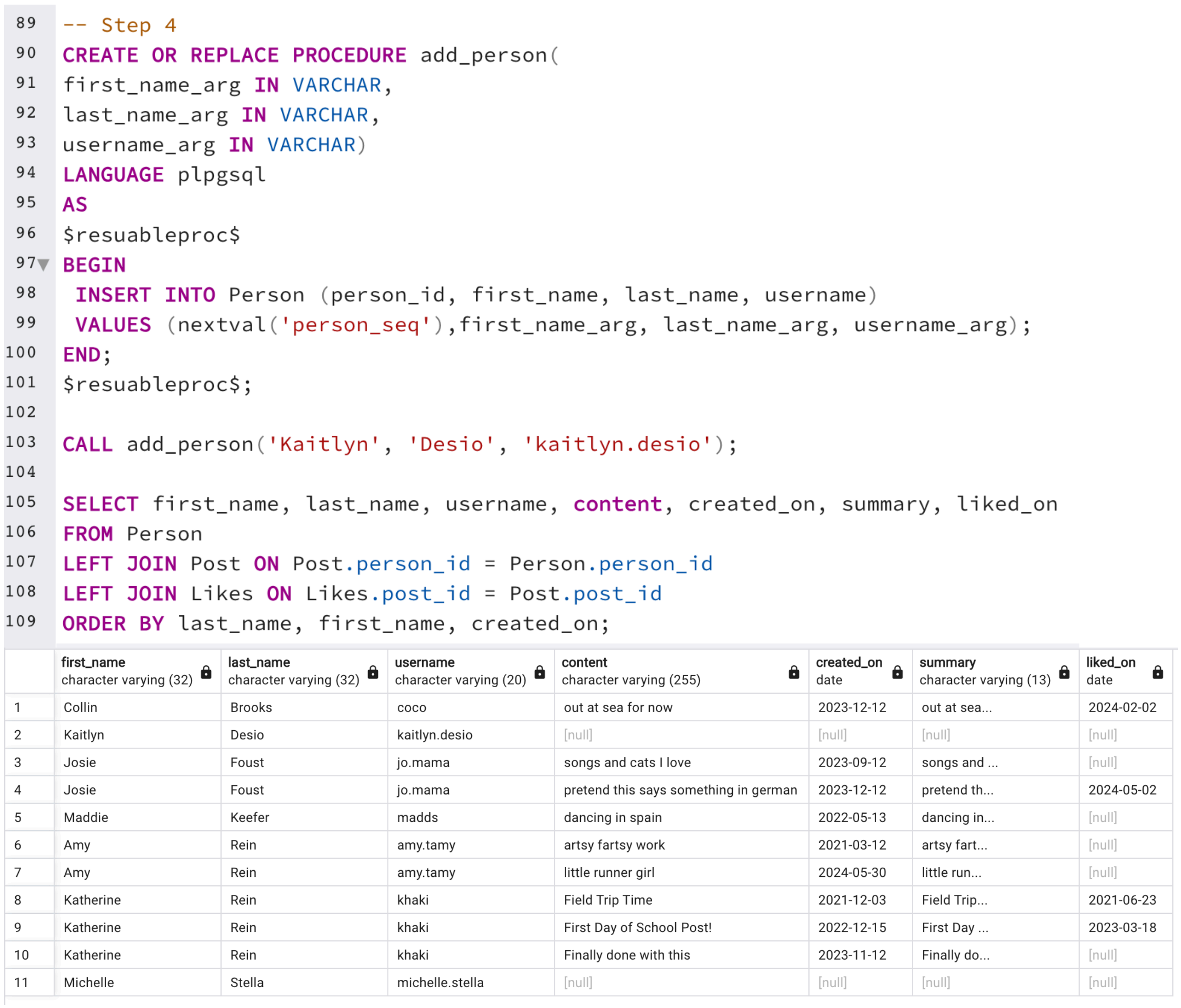
3. *Create Hardcoded Procedure* – Create a stored procedure named “add_michelle_stella” which has no parameters and adds a person named “Michelle Stella” to the Person table. Execute the stored procedure, and list out the rows in the Person table to show that Michelle Stella has been added.



```sql
-- Step 3
CREATE OR REPLACE PROCEDURE add_michelle_stella ()
AS
$proc$
 BEGIN
 INSERT INTO Person (person_id, first_name, last_name, username)
 VALUES (nextval('person_seq'), 'Michelle', 'Stella', 'michelle.stella');
 END;
$proc$ LANGUAGE plpgsql;

CALL add_michelle_stella ();

SELECT first_name, last_name, username, content, created_on, summary, liked_on
FROM Person
LEFT JOIN Post ON Post.person_id = Person.person_id
LEFT JOIN Likes ON Likes.post_id = Post.post_id
ORDER BY last_name, first_name, created_on;
```

| | first_name character varying (32) | last_name character varying (32) | username character varying (20) | content character varying (255) | created_on date | summary character varying (13) | liked_on date |
|---|---|---|---|---|---|---|---|
| 1 | Collin | Brooks | coco | out at sea for now | 2023-12-12 | out at sea... | 2024-02-02 |
| 2 | Josie | Foust | jo.mama | songs and cats I love | 2023-09-12 | songs and ... | [null] |
| 3 | Josie | Foust | jo.mama | pretend this says something in german | 2023-12-12 | pretend th... | 2024-05-02 |
| 4 | Maddie | Keefer | madds | dancing in spain | 2022-05-13 | dancing in... | [null] |
| 5 | Amy | Rein | amy.tamy | artsy fartsy work | 2021-03-12 | artsy fart... | [null] |
| 6 | Amy | Rein | amy.tamy | little runner girl | 2024-05-30 | little run... | [null] |
| 7 | Katherine | Rein | khaki | Field Trip Time | 2021-12-03 | Field Trip... | 2021-06-23 |
| 8 | Katherine | Rein | khaki | First Day of School Post! | 2022-12-15 | First Day ... | 2023-03-18 |
| 9 | Katherine | Rein | khaki | Finally done with this | 2023-11-12 | Finally do... | [null] |
| 10 | Michelle | Stella | michelle.stella | [null] | [null] | [null] | [null] |

4. *Create Reusable Procedure* – Create a reusable stored procedure named "add_person" that uses parameters and allows you to insert any new person into the Person table. Execute the stored procedure with a person of your choosing, then list out the Person table to show that the person was added to the table.

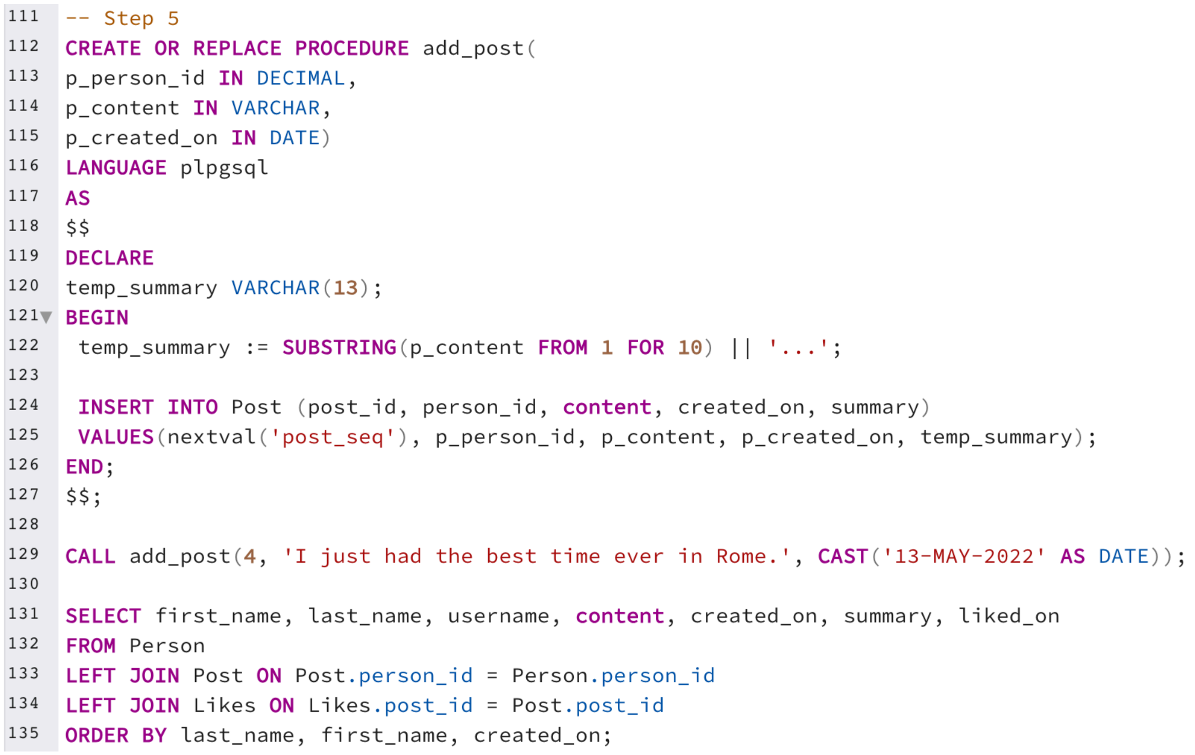


```sql
-- Step 4
CREATE OR REPLACE PROCEDURE add_person(
first_name_arg IN VARCHAR,
last_name_arg IN VARCHAR,
username_arg IN VARCHAR)
LANGUAGE plpgsql
AS
$resuableproc$
BEGIN
 INSERT INTO Person (person_id, first_name, last_name, username)
 VALUES (nextval('person_seq'),first_name_arg, last_name_arg, username_arg);
END;
$resuableproc$;

CALL add_person('Kaitlyn', 'Desio', 'kaitlyn.desio');

SELECT first_name, last_name, username, content, created_on, summary, liked_on
FROM Person
LEFT JOIN Post ON Post.person_id = Person.person_id
LEFT JOIN Likes ON Likes.post_id = Post.post_id
ORDER BY last_name, first_name, created_on;
```

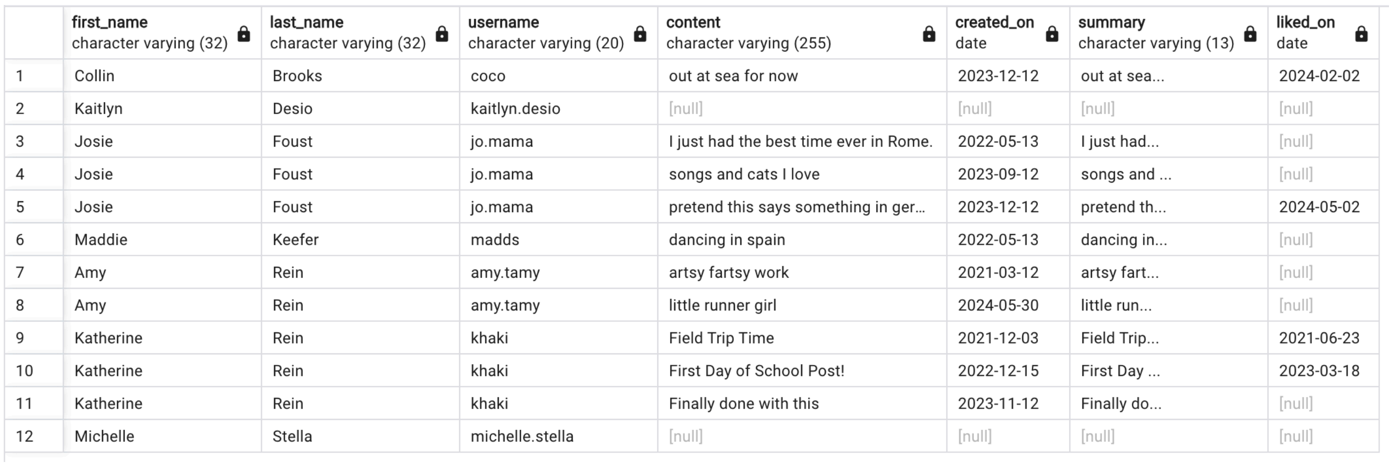
| | first_name character varying (32) | last_name character varying (32) | username character varying (20) | content character varying (255) | created_on date | summary character varying (13) | liked_on date |
|---|---|---|---|---|---|---|---|
| 1 | Collin | Brooks | coco | out at sea for now | 2023-12-12 | out at sea... | 2024-02-02 |
| 2 | Kaitlyn | Desio | kaitlyn.desio | [null] | [null] | [null] | [null] |
| 3 | Josie | Foust | jo.mama | songs and cats I love | 2023-09-12 | songs and ... | [null] |
| 4 | Josie | Foust | jo.mama | pretend this says something in german | 2023-12-12 | pretend th... | 2024-05-02 |
| 5 | Maddie | Keefer | madds | dancing in spain | 2022-05-13 | dancing in... | [null] |
| 6 | Amy | Rein | amy.tamy | artsy fartsy work | 2021-03-12 | artsy fart... | [null] |
| 7 | Amy | Rein | amy.tamy | little runner girl | 2024-05-30 | little run... | [null] |
| 8 | Katherine | Rein | khaki | Field Trip Time | 2021-12-03 | Field Trip... | 2021-06-23 |
| 9 | Katherine | Rein | khaki | First Day of School Post! | 2022-12-15 | First Day ... | 2023-03-18 |
| 10 | Katherine | Rein | khaki | Finally done with this | 2023-11-12 | Finally do... | [null] |
| 11 | Michelle | Stella | michelle.stella | [null] | [null] | [null] | [null] |

5. *Create Deriving Procedure* – Create a reusable stored procedure named "add_post" that uses parameters and allows you to insert any new post into the Post table. Instead of passing in the summary as a parameter, derive the summary from the content,

storing the derivation temporarily in a variable (which is then used as part of the insert statement). Recall that the summary field stores the first 10 characters of the content followed by "…". Execute the stored procedure to add a post of your choosing, then list out the Post table to show that the addition succeeded.

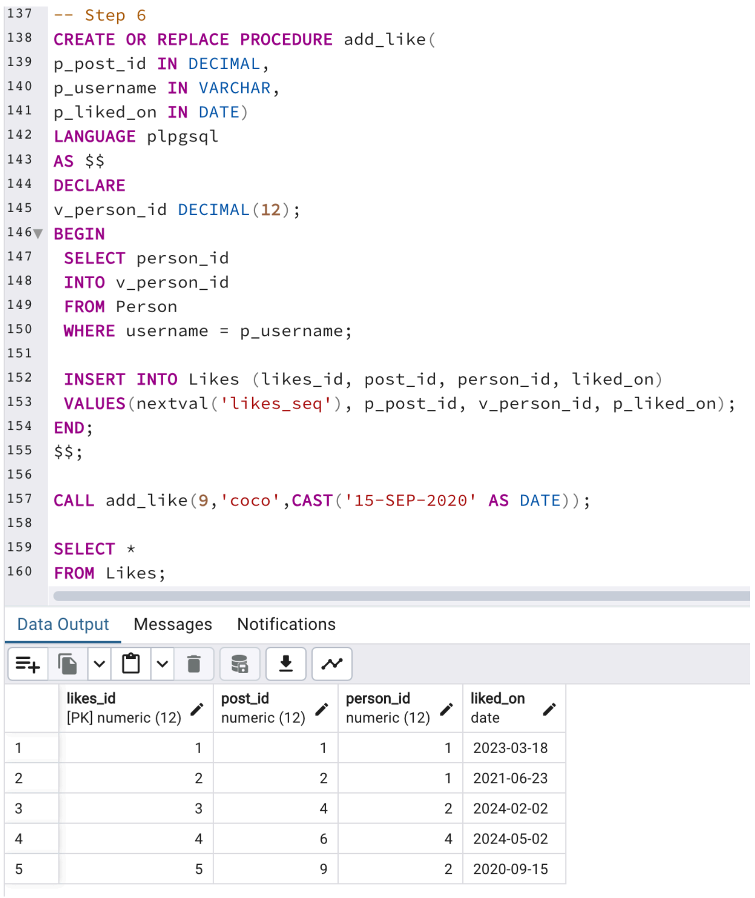


```sql
-- Step 5
CREATE OR REPLACE PROCEDURE add_post(
p_person_id IN DECIMAL,
p_content IN VARCHAR,
p_created_on IN DATE)
LANGUAGE plpgsql
AS
$$
DECLARE
temp_summary VARCHAR(13);
BEGIN
 temp_summary := SUBSTRING(p_content FROM 1 FOR 10) || '...';

 INSERT INTO Post (post_id, person_id, content, created_on, summary)
 VALUES(nextval('post_seq'), p_person_id, p_content, p_created_on, temp_summary);
END;
$$;

CALL add_post(4, 'I just had the best time ever in Rome.', CAST('13-MAY-2022' AS DATE));

SELECT first_name, last_name, username, content, created_on, summary, liked_on
FROM Person
LEFT JOIN Post ON Post.person_id = Person.person_id
LEFT JOIN Likes ON Likes.post_id = Post.post_id
ORDER BY last_name, first_name, created_on;
```



| | first_name character varying (32) | last_name character varying (32) | username character varying (20) | content character varying (255) | created_on date | summary character varying (13) | liked_on date |
|---|---|---|---|---|---|---|---|
| 1 | Collin | Brooks | coco | out at sea for now | 2023-12-12 | out at sea... | 2024-02-02 |
| 2 | Kaitlyn | Desio | kaitlyn.desio | [null] | [null] | [null] | [null] |
| 3 | Josie | Foust | jo.mama | I just had the best time ever in Rome. | 2022-05-13 | I just had... | [null] |
| 4 | Josie | Foust | jo.mama | songs and cats I love | 2023-09-12 | songs and ... | [null] |
| 5 | Josie | Foust | jo.mama | pretend this says something in ger... | 2023-12-12 | pretend th... | 2024-05-02 |
| 6 | Maddie | Keefer | madds | dancing in spain | 2022-05-13 | dancing in... | [null] |
| 7 | Amy | Rein | amy.tamy | artsy fartsy work | 2021-03-12 | artsy fart... | [null] |
| 8 | Amy | Rein | amy.tamy | little runner girl | 2024-05-30 | little run... | [null] |
| 9 | Katherine | Rein | khaki | Field Trip Time | 2021-12-03 | Field Trip... | 2021-06-23 |
| 10 | Katherine | Rein | khaki | First Day of School Post! | 2022-12-15 | First Day ... | 2023-03-18 |
| 11 | Katherine | Rein | khaki | Finally done with this | 2023-11-12 | Finally do... | [null] |
| 12 | Michelle | Stella | michelle.stella | [null] | [null] | [null] | [null] |

6. *Create Lookup Procedure* – Create a reusable stored procedure named "add_like" that uses parameters and allows you to insert any new "like". Rather than passing in the person_id value as a parameter to identify which person is liking which post, pass in the username of the person. The stored procedure should then lookup the person_id and store it in a variable to be used in the insert statement. Execute the procedure to add a "like" of your choosing, then list out the Like table to show the addition succeeded.



```
-- Step 6
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 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(9,'coco',CAST('15-SEP-2020' AS DATE));

SELECT *
FROM Likes;
```

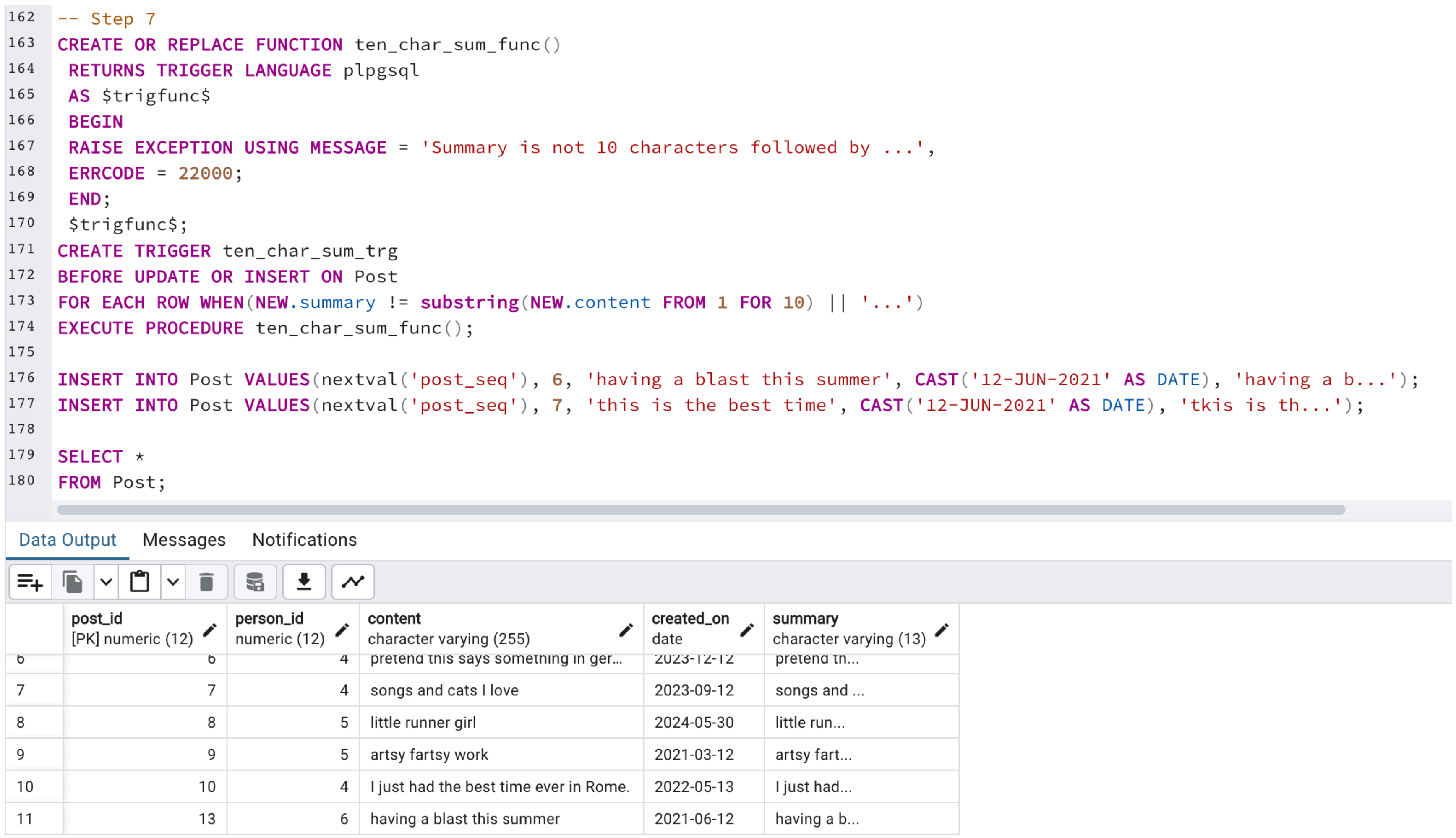
Data Output | Messages | Notifications

| | likes_id [PK] numeric (12) | post_id numeric (12) | person_id numeric (12) | liked_on date |
|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 2023-03-18 |
| 2 | 2 | 2 | 1 | 2021-06-23 |
| 3 | 3 | 4 | 2 | 2024-02-02 |
| 4 | 4 | 6 | 4 | 2024-05-02 |
| 5 | 5 | 9 | 2 | 2020-09-15 |

## ***Section Two – Triggers***

7. *Single Table Validation Trigger* – One practical use of a trigger is validation within a single table (that is, the validation can be performed by using columns in the table being modified). Create a trigger that validates that the summary is being inserted correctly, that is, that the summary is actually the first 10 characters of the content followed by "…". The trigger should reject an insert that does not have a valid summary value. Verify the trigger works by issuing two insert commands – one with a correct summary, and one with an incorrect summary. List out the Post table after the inserts to show one insert was blocked and the

other succeeded.

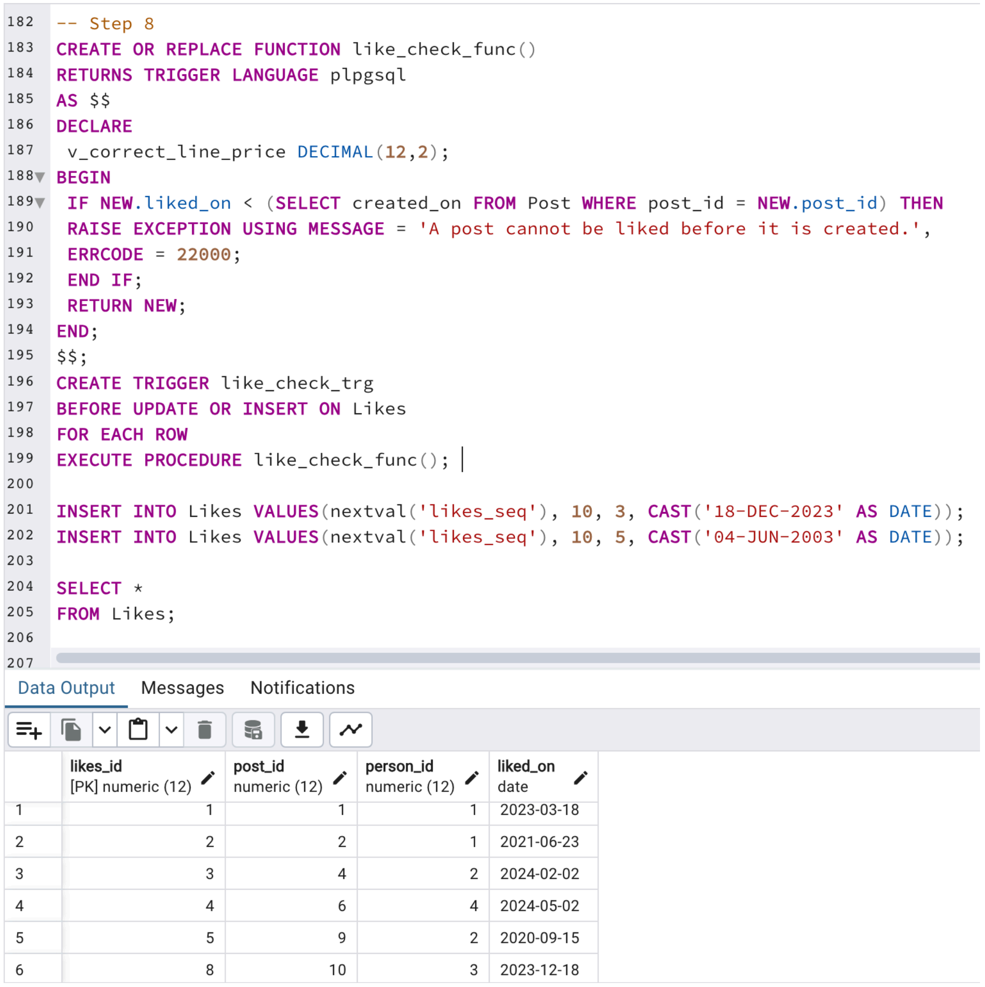


```
-- Step 7
CREATE OR REPLACE FUNCTION ten_char_sum_func()
 RETURNS TRIGGER LANGUAGE plpgsql
 AS $trigfunc$
 BEGIN
 RAISE EXCEPTION USING MESSAGE = 'Summary is not 10 characters followed by ...',
 ERRCODE = 22000;
 END;
 $trigfunc$;
CREATE TRIGGER ten_char_sum_trg
BEFORE UPDATE OR INSERT ON Post
FOR EACH ROW WHEN(NEW.summary != substring(NEW.content FROM 1 FOR 10) || '...')
EXECUTE PROCEDURE ten_char_sum_func();

INSERT INTO Post VALUES(nextval('post_seq'), 6, 'having a blast this summer', CAST('12-JUN-2021' AS DATE), 'having a b...');
INSERT INTO Post VALUES(nextval('post_seq'), 7, 'this is the best time', CAST('12-JUN-2021' AS DATE), 'tkis is th...');

SELECT *
FROM Post;
```

| | post_id<br>[PK] numeric (12) | person_id<br>numeric (12) | content<br>character varying (255) | created_on<br>date | summary<br>character varying (13) |
|---|---|---|---|---|---|
| 6 | 6 | 4 | pretend this says something in ger... | 2023-12-12 | pretend th... |
| 7 | 7 | 4 | songs and cats I love | 2023-09-12 | songs and ... |
| 8 | 8 | 5 | little runner girl | 2024-05-30 | little run... |
| 9 | 9 | 5 | artsy fartsy work | 2021-03-12 | artsy fart... |
| 10 | 10 | 4 | I just had the best time ever in Rome. | 2022-05-13 | I just had... |
| 11 | 13 | 6 | having a blast this summer | 2021-06-12 | having a b... |

8. *Cross-Table Validation Trigger* – Another practical use of a trigger is cross-table validation (that is, the validation needs columns from at least one table external to the table being updated). Create a trigger that blocks a “like” from being inserted if its “liked_on” date is before the post’s “created_on” date. Verify the trigger works by inserting two “likes” – one that passes this

validation, and one that does not. List out the Likes table after the inserts to show one insert was blocked and the other succeeded.

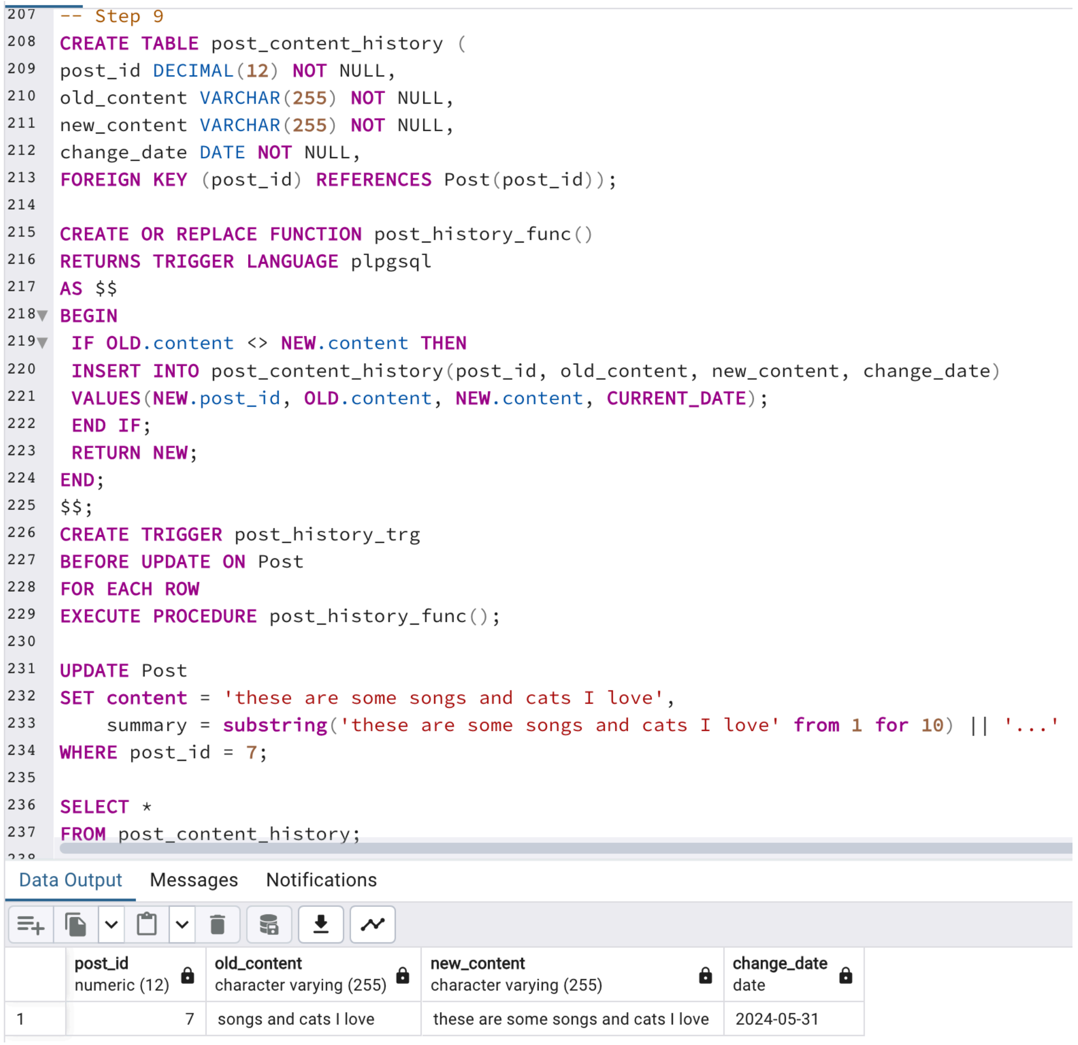


```
-- Step 8
CREATE OR REPLACE FUNCTION like_check_func()
RETURNS TRIGGER LANGUAGE plpgsql
AS $$
DECLARE
 v_correct_line_price DECIMAL(12,2);
BEGIN
 IF NEW.liked_on < (SELECT created_on FROM Post WHERE post_id = NEW.post_id) THEN
 RAISE EXCEPTION USING MESSAGE = 'A post cannot be liked before it is created.',
 ERRCODE = 22000;
 END IF;
 RETURN NEW;
END;
$$;
CREATE TRIGGER like_check_trg
BEFORE UPDATE OR INSERT ON Likes
FOR EACH ROW
EXECUTE PROCEDURE like_check_func();

INSERT INTO Likes VALUES(nextval('likes_seq'), 10, 3, CAST('18-DEC-2023' AS DATE));
INSERT INTO Likes VALUES(nextval('likes_seq'), 10, 5, CAST('04-JUN-2003' AS DATE));

SELECT *
FROM Likes;
```

| | likes_id [PK] numeric (12) | post_id numeric (12) | person_id numeric (12) | liked_on date |
|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 2023-03-18 |
| 2 | 2 | 2 | 1 | 2021-06-23 |
| 3 | 3 | 4 | 2 | 2024-02-02 |
| 4 | 4 | 6 | 4 | 2024-05-02 |
| 5 | 5 | 9 | 2 | 2020-09-15 |
| 6 | 8 | 10 | 3 | 2023-12-18 |

9. *History Trigger* – Another practical use of trigger is to maintain a history of values as they change. Create a table named post_content_history that is used to record updates to the content of a post, then create a trigger that keeps this table up-to-

date when updates happen to post contents. Verify the trigger works by updating a post's content, then listing out the post_content_history table (which should have a record of the update).



```
-- Step 9
CREATE TABLE post_content_history (
post_id DECIMAL(12) NOT NULL,
old_content VARCHAR(255) NOT NULL,
new_content VARCHAR(255) NOT NULL,
change_date DATE NOT NULL,
FOREIGN KEY (post_id) REFERENCES Post(post_id));

CREATE OR REPLACE FUNCTION post_history_func()
RETURNS TRIGGER LANGUAGE plpgsql
AS $$
BEGIN
 IF OLD.content <> NEW.content THEN
 INSERT INTO post_content_history(post_id, old_content, new_content, change_date)
 VALUES(NEW.post_id, OLD.content, NEW.content, CURRENT_DATE);
 END IF;
 RETURN NEW;
END;
$$;
CREATE TRIGGER post_history_trg
BEFORE UPDATE ON Post
FOR EACH ROW
EXECUTE PROCEDURE post_history_func();

UPDATE Post
SET content = 'these are some songs and cats I love',
    summary = substring('these are some songs and cats I love' from 1 for 10) || '...'
WHERE post_id = 7;

SELECT *
FROM post_content_history;
```

Data Output Messages Notifications

| | post_id numeric (12) | old_content character varying (255) | new_content character varying (255) | change_date date |
|---|---|---|---|---|
| 1 | 7 | songs and cats I love | these are some songs and cats I love | 2024-05-31 |

***Section Three – Normalization***

10. *Creating Normalized Table Structure* – For this question, you create a set of normalized tables based upon the scenario given, and also identify some functional dependencies between the given fields….

   a. Identify all functional dependencies in the set of fields listed above in the spreadsheet. These can be listed in the form of:

      column1,column2,… ➔ column3, column4…

      Make sure to explain your reasoning for the functional dependency choices.

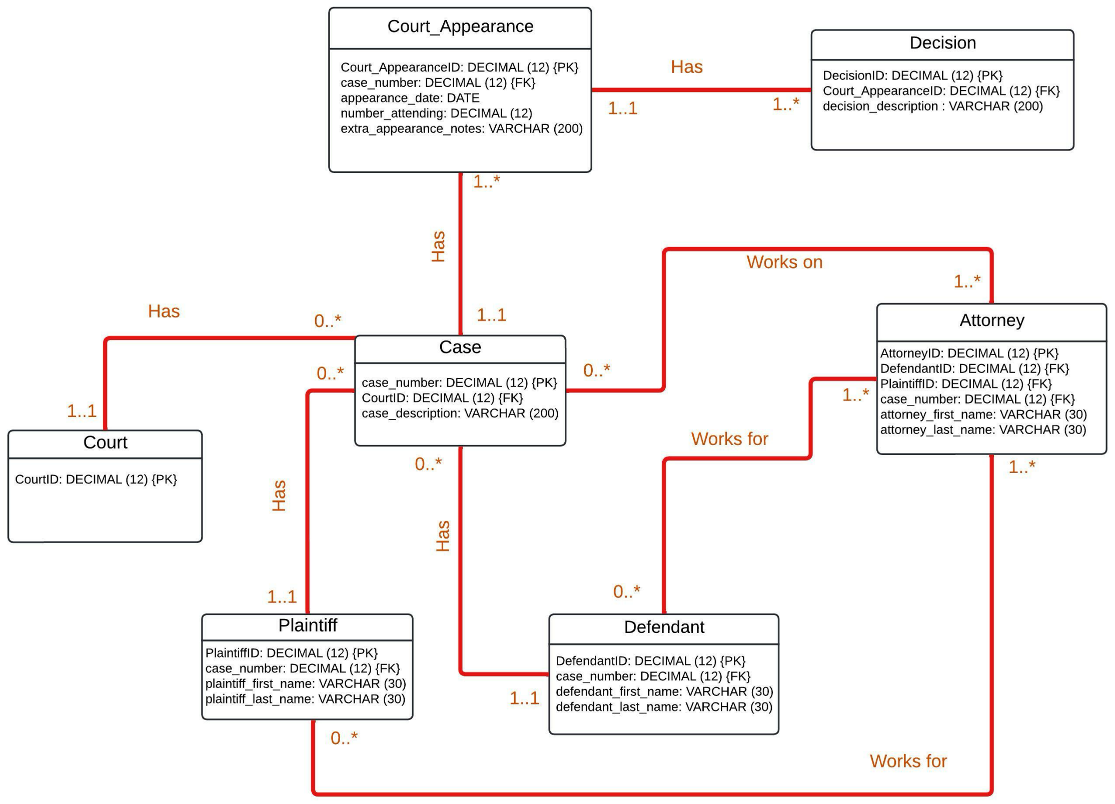
case_number ➔ case_description
case_number ➔plaintiff_first_name, plaintiff_last_name
case_number ➔ defendant_first_name, defendant_last_name
case_number ➔ courtID

case_number is a unique identifier. This means that for each value where there can only be one per case we have a functional dependency. There can only ever be one case description, plaintiff, defendant, and court per case. Therefore, the unique identifier for the case creates a functional dependency between those values and the case_number.

case_number, appearance_date ➔ number_attending, extra_appearance_notes
case_number, appearance_date ➔ attorney1_first_name, attorney1_last_name
case_number, appearance_date ➔ attorney2_first_name, attorney2_last_name
case_number, appearance_date ➔ attorney3_first_name, attorney3_last_name
case_number, appearance_date ➔ decision1_description, decision2_description

Since each case only can be heard once per day, we can find out all of the information about a certain case appearance if given the case_number and appearance_date.

   b. Suggest a set of normalized relational tables derived from how the court operates and the fields they store. Create a DBMS physical ERD representing this set of tables, which contains the entities, primary and foreign keys, attributes, relationships, and relationship constraints. You may add synthetic primary keys where needed. Make sure that the tables are normalized to BCNF, and to explain your choices.


Court_Appearance
Court_AppearanceID: DECIMAL (12) {PK}
case_number: DECIMAL (12) {FK}
appearance_date: DATE
number_attending: DECIMAL (12)
extra_appearance_notes: VARCHAR (200)
Decision
DecisionID: DECIMAL (12) {PK}
Court_AppearanceID: DECIMAL (12) {FK}
decision_description : VARCHAR (200)
Has
1..1
1..*
1..*
Has
1..1
Case
case_number: DECIMAL (12) {PK}
CourtID: DECIMAL (12) {FK}
case_description: VARCHAR (200)
Has
0..*
1..1
Court
CourtID: DECIMAL (12) {PK}
Works on
0..*
1..*
Attorney
AttorneyID: DECIMAL (12) {PK}
DefendantID: DECIMAL (12) {FK}
PlaintiffID: DECIMAL (12) {FK}
case_number: DECIMAL (12) {FK}
attorney_first_name: VARCHAR (30)
attorney_last_name: VARCHAR (30)
0..*
Has
1..1
Plaintiff
PlaintiffID: DECIMAL (12) {PK}
case_number: DECIMAL (12) {FK}
plaintiff_first_name: VARCHAR (30)
plaintiff_last_name: VARCHAR (30)
0..*
Has
1..1
Defendant
DefendantID: DECIMAL (12) {PK}
case_number: DECIMAL (12) {FK}
defendant_first_name: VARCHAR (30)
defendant_last_name: VARCHAR (30)
Works for
1..*
0..*
1..*
0..*
Works for

I chose to create a court table to generalize this for many courts if needed. The case_number is what ties pretty much everything together. The court_appearanceID is essentially case_number and appearance_date combined. This is how decision gets added to the main table. I made each decision need its own decision_id so that I could diagram it correctly. I did the same thing with the attorneys. Each table that is functionally dependent upon the case_number has its own table and with the case_number as a foreign key. The court_appearance and decision tables are representing their own partial dependencies with case_number and appearance_date. This initial set up does not have any transitive dependencies or non-candidate determinants.