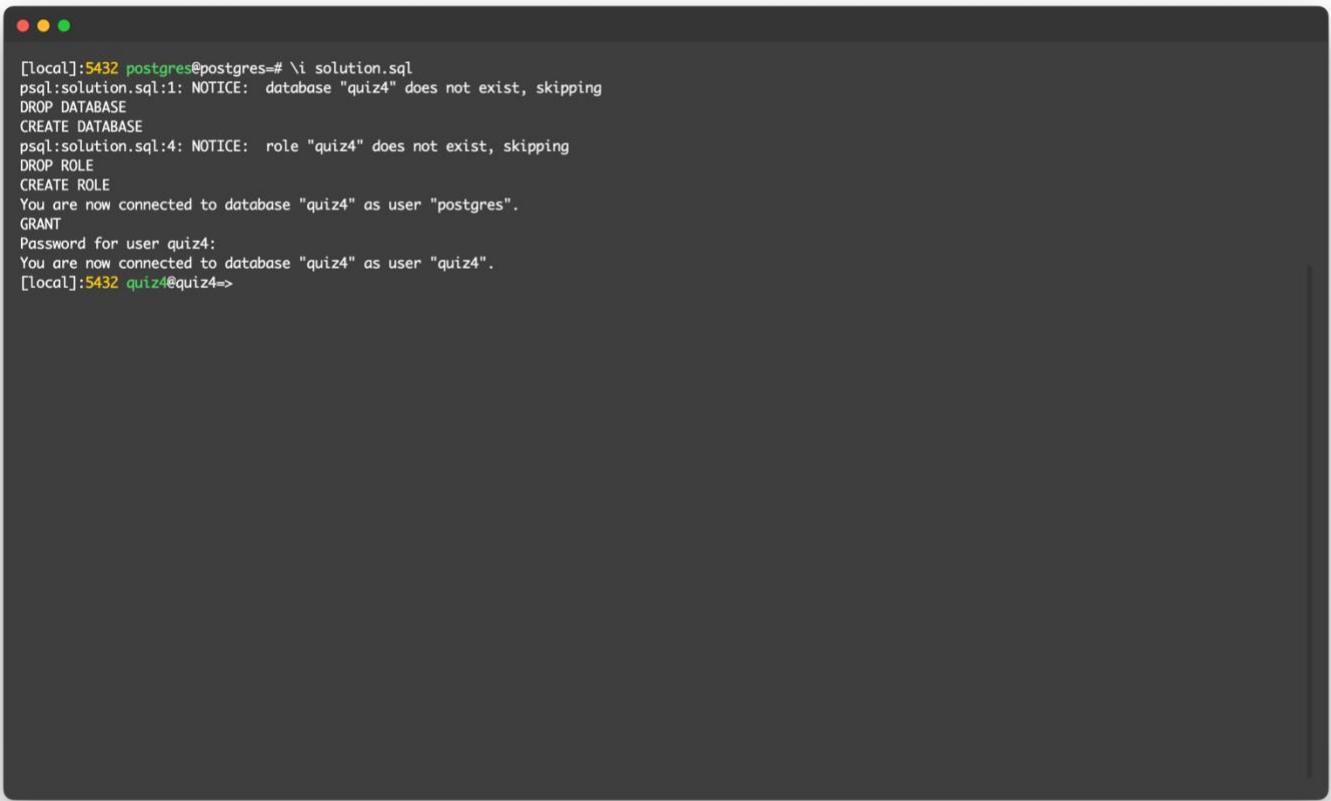


1 – creating quiz4 database



A screenshot of a terminal window with a dark background and light-colored text. The window title bar shows three red, yellow, and green circular icons. The terminal prompt is "[local]:5432 postgres@postgres=". The user runs the command "\i solution.sql". The output shows PostgreSQL attempting to drop the database "quiz4" and failing because it does not exist. It then creates the database, creates a role named "quiz4", and connects the user "quiz4" to the database. Finally, it grants all privileges to the user "quiz4". The command "\i" is used to execute the SQL file "solution.sql". The output is as follows:

```
[local]:5432 postgres@postgres=# \i solution.sql
psql:solution.sql:1: NOTICE:  database "quiz4" does not exist, skipping
DROP DATABASE
CREATE DATABASE
psql:solution.sql:4: NOTICE:  role "quiz4" does not exist, skipping
DROP ROLE
CREATE ROLE
You are now connected to database "quiz4" as user "postgres".
GRANT
Password for user quiz4:
You are now connected to database "quiz4" as user "quiz4".
[local]:5432 quiz4@quiz4=>
```

1 – creating tables

The screenshot shows the pgAdmin 4 interface. The left pane is the Object Explorer, displaying a database named 'quiz'. Under 'Tables (2)', there are entries for 'author' and 'book'. The right pane is the Query Editor, showing a SQL script for creating these tables. The script includes grants, table definitions for 'book' and 'author', and an 'INSERT INTO' statement for 'book'. The status bar at the bottom indicates 'Query returned successfully in 32 msec.'

```
6
7 v \c quiz4 postgres
8 GRANT ALL PRIVILEGES ON SCHEMA public TO quiz4;
9 v \c quiz4 quiz4
10
11 DROP TABLE IF EXISTS book;
12 v CREATE TABLE book (
13     id serial PRIMARY KEY,
14     title text NOT NULL,
15     isbn text NOT NULL,
16     pages integer NOT NULL, -- how many pages in the book
17     pub_date date NOT NULL DEFAULT NOW() -- automatically inserted by PostgreSQL
18 );
19
20 DROP TABLE IF EXISTS author;
21 v CREATE TABLE author (
22     id serial PRIMARY KEY,
23     fname text NOT NULL,
24     lname text NOT NULL
25 );
26
27 -- Add your insert statements here
28
29 -- 1
30
31 v INSERT INTO book (title, isbn, pages, pub_date)
Data Output Messages Notifications
```

NOTICE: table "book" does not exist, skipping
NOTICE: table "author" does not exist, skipping
CREATE TABLE

Total rows: 0 of 0 | Query complete 00:00:00.032 | Ln 25, Col 3

1 – showing inserted books

The screenshot shows the pgAdmin 4 interface. The left sidebar is titled "Object Explorer" and lists various database objects under "quiz4" schema, including Schemas, Tables, Functions, and Views. The main area is titled "Query History" and contains the following SQL code:

```
1 ~ SELECT *
2   FROM book;
```

Below the query, the "Data Output" tab is selected, displaying a table with the results of the query:

	id [PK] integer	title text	isbn text	pages integer	pub_date date
1	1	Database Processing	123456789	670	2023-09-12
2	2	A+ Computer Repair	987654321	505	2022-10-11
3	3	C++ How to Program	997755331	644	2021-11-29
4	4	Intro to Computer Science	554632231	893	2021-01-01
5	5	Learn PostgreSQL 16	667785343	430	2023-12-07

At the bottom of the interface, status messages are displayed: "Total rows: 5 of 5" and "Query complete 00:00:00.071".

1 – showings inserted authors

The screenshot shows the pgAdmin 4 interface. The left pane is the Object Explorer, displaying a tree structure of database objects under the schema 'public'. The 'Tables (2)' node is expanded, showing 'author' and 'book'. The 'author' table is selected. The right pane contains a query editor window with the following content:

```
1 v SELECT *
2   FROM author;
```

Below the query editor is a Data Output pane showing the results of the query:

	id [PK] integer	fname text	lname text
1	1	David	Kroenke
2	2	Sally	Gates
3	3	Jorge	Martinez
4	4	George	Kalos
5	5	Randall	Smith

At the bottom of the pgAdmin window, status messages indicate "Total rows: 5 of 5" and "Query complete 00:00:00.038".

2 – creating stored procedure

The screenshot shows the pgAdmin 4 interface with the following details:

- Object Explorer:** On the left, it lists database objects under the schema "quiz4.quiz4@quiz4". The "Procedures" node is expanded, showing a single entry: "insert_book(N p_title text,)".
- Query Editor:** The main area contains the SQL code for the stored procedure:

```
48
49 v CREATE OR REPLACE PROCEDURE insert_book(
50   p_title TEXT,
51   p_isbn TEXT,
52   p_pages INTEGER
53 )
54 AS $$
55 BEGIN
56   IF p_title IS NULL THEN
57     RAISE EXCEPTION 'title parameter cannot be empty';
58   END IF;
59
60   IF p_isbn IS NULL THEN
61     RAISE EXCEPTION 'isbn parameter cannot be empty';
62   END IF;
63
64   IF p_pages IS NULL THEN
65     RAISE EXCEPTION 'pages parameter cannot be empty';
66   END IF;
67
68   INSERT INTO book (title, isbn, pages)
69   VALUES (p_title, p_isbn, p_pages);
70
71 $$
72 LANGUAGE plpgsql;
73
74 CALL insert_book('Testing Book', '448833992', 700);
```
- Data Output:** Below the query editor, it says "CREATE PROCEDURE".
- Messages:** It displays the message "Query returned successfully in 35 msec."
- Notifications:** This tab is currently inactive.
- Status Bar:** At the bottom right, it shows "Total rows: 0 of 0" and "Query complete 00:00:00.035".
- Bottom Right:** It also shows "Ln 71, Col 21".

2 – calling stored procedure

The screenshot shows the pgAdmin 4 interface with the following details:

- Object Explorer:** On the left, under the "Schemas (1)" section, the "public" schema is expanded, showing "Procedures (1)". A specific procedure, "insert_book", is selected.
- Query Editor:** The main area contains the following SQL code:

```
62    IF p_pages IS NULL THEN
63        RAISE EXCEPTION 'pages parameter cannot be empty';
64    END IF;
65
66    INSERT INTO book (title, isbn, pages)
67        VALUES (p_title, p_isbn, p_pages);
68
69    $ LANGUAGE plpgsql;
70
71    CALL insert_book('Testing Book', '448833992', 700);
72
73    -- 3
74
75    CREATE OR REPLACE FUNCTION enter_author(
76        p_fname TEXT,
77        p_lname TEXT
78    )
79    RETURNS INT
80    AS $$$
81    DECLARE
82        r_author_id INT;
83    BEGIN
84        IF p_fname IS NULL THEN
85            RAISE EXCEPTION 'fname parameter cannot be empty';
86        END IF;
87    END;
88
```
- Data Output:** Below the code, it says "Query returned successfully in 23 msec."
- Status Bar:** At the bottom right, it shows "Total rows: 0 of 0 | Query complete 00:00:00.023 | Ln 73, Col 52".

2 – showing stored procedure result

The screenshot shows the pgAdmin 4 interface. In the Object Explorer, under the 'public' schema, a procedure named 'insert_book' is selected. In the Query tab, the following SQL code is run:

```
1 ~ SELECT *
2   FROM book;
```

The Data Output tab displays the results of the query:

	id [PK] integer	title text	isbn text	pages integer	pub_date date
1	1	Database Processing	123456789	670	2023-09-12
2	2	A+ Computer Repair	987654321	505	2022-10-11
3	3	C++ How to Program	997755331	644	2021-11-29
4	4	Intro to Computer Science	554632231	893	2021-01-01
5	5	Learn PostgreSQL 16	667785343	430	2023-12-07
6	6	Testing Book	448833992	700	2024-04-10

Total rows: 6 of 6 Query complete 00:00:00.036 Ln 2, Col 11

2 – showing stored procedure validation

The screenshot shows the pgAdmin 4 interface with the following details:

- Object Explorer:** On the left, it shows the database schema structure under "Schemas (1) / public".
- Query Editor:** The main area contains a PL/pgSQL script. The last two lines of the script attempt to call a function that expects non-null parameters:

```
73 CALL insert_book('Testing Book', '448833992', 700);
CALL insert_book(NULL, '423232332', 200);
```

- Messages Tab:** Below the editor, the "Messages" tab is selected, displaying the following error message:

```
ERROR: title parameter cannot be empty
CONTEXT: PL/pgSQL function insert_book(text,text,integer) line 4 at RAISE
```
- Status Bar:** At the bottom, the status bar shows "Total rows: 0 of 0" and "Query complete 00:00:00.044".
- Bottom Right:** The text "Ln 74, Col 1" indicates the specific location of the error.

3 – creating function

The screenshot shows the pgAdmin 4 interface with the following details:

- Object Explorer:** On the left, it lists various database objects under the schema `public`, including `Aggregates`, `Collations`, `Domains`, `FTS Configurations`, `FTS Dictionaries`, `FTS Parsers`, `FTS Templates`, `Foreign Tables`, `Functions`, `Materialized Views`, `Operators`, `Procedures`, `Sequences`, `Tables`, `Trigger Functions`, `Types`, and `Views`.
- Query Editor:** The main area contains the SQL code for creating a function:

```
76 -- 3
77
78 CREATE OR REPLACE FUNCTION enter_author(
79     p_fname TEXT,
80     p_lname TEXT
81 )
82 RETURNS INT
83 AS $$$
84 DECLARE
85     r_author_id INT;
86 BEGIN
87     IF p_fname IS NULL THEN
88         RAISE EXCEPTION 'fname parameter cannot be empty';
89     END IF;
90
91     IF p_lname IS NULL THEN
92         RAISE EXCEPTION 'lname parameter cannot be empty';
93     END IF;
94
95     INSERT INTO author (fname, lname)
96     VALUES (p_fname, p_lname)
97     RETURNING id INTO r_author_id;
98
99     RETURN r_author_id;
100 END;
101 $$ LANGUAGE plpgsql;
102
103 SELECT *
```
- Data Output:** Below the query editor, it says "Query returned successfully in 26 msec."
- Status Bar:** At the bottom, it shows "Total rows: 0 of 0" and "Query complete 00:00:00.026".
- Bottom Right:** It indicates "Ln 101, Col 21".

3 – showing function return value

The screenshot shows the pgAdmin 4 interface with the following details:

- Object Explorer:** On the left, it lists various database objects under the schema "quiz4/quiz4@quiz4". These include Schemas (1), Functions (1) (with one named "enter_author(p_fname text)", Tables (2) (with "author" and "book"), and Procedures (1) (with one named "insert_book(IN p_title text, I...").
- Query Tab:** The main area contains a SQL script. The function definition for "enter_author" is shown, which takes a parameter "p_fname" of type text and returns an integer. It includes logic to check if "p_lname" is NULL and raise an exception if it is. The function also inserts into the "author" table and returns the generated ID. The script then calls this function with parameters 'Andres' and 'Hung'. Finally, it creates a table "author_logs" with columns "id_old" (INT), "fname_old" (TEXT), and "lname_old" (TEXT).

```
89
90
91     IF p_lname IS NULL THEN
92         RAISE EXCEPTION 'lname parameter cannot be empty';
93     END IF;
94
95     INSERT INTO author (fname, lname)
96     VALUES (p_fname, p_lname)
97     RETURNING id INTO r_author_id;
98
99     RETURN r_author_id;
100
101    END;
102    $$ LANGUAGE plpgsql;
103
104    SELECT *
105    FROM enter_author('Andres', 'Hung');
106
107    -- 4
108
109    CREATE TABLE author_logs(
110        id_old INT,
111        fname_old TEXT,
112        lname_old TEXT
113    );
```
- Data Output Tab:** Below the query results, there is a table titled "enter_author" with one row. The column "integer" has the value 6.
- Status Bar:** At the bottom, it shows "Total rows: 1 of 1" and "Query complete 00:00:00.026".
- Bottom Right:** "Ln 103, Col 1"

3 – showing function result

The screenshot shows the pgAdmin 4 interface. The left pane is the Object Explorer, displaying various database objects like Schemas, Tables, Functions, and Procedures. The right pane shows the Query Editor with the following SQL code:

```
1 ~ SELECT *
2 FROM author;
```

The Data Output tab displays the results of the query:

	id [PK] integer	fname text	lname text
1	1	David	Kroenke
2	2	Sally	Gates
3	3	Jorge	Martinez
4	4	George	Kalos
5	5	Randall	Smith
6	6	Andres	Hung

Total rows: 6 of 6 Query complete 00:00:00.058 Ln 2, Col 12

4 – author_logs table

The screenshot shows the pgAdmin 4 interface with the following details:

- Object Explorer:** On the left, under "Tables (3)", the "author_logs" table is selected.
- Query Editor:** The main area contains the following SQL code:

```
END;
$$ LANGUAGE plpgsql;

SELECT *
FROM enter_author('Andres', 'Hung');

-- 4

CREATE TABLE author_logs(
    id_old INT,
    fname_old TEXT,
    lname_old TEXT
);

-- 5

-- trigger function
CREATE OR REPLACE FUNCTION log_author_delete_update()
RETURNS TRIGGER
AS $$
BEGIN
    INSERT INTO author_logs (id_old, fname_old, lname_old)
    VALUES (OLD.id, OLD.fname, OLD.lname);
END
$$;
```
- Data Output:** Below the query editor, it says "Query returned successfully in 40 msec."
- Status Bar:** At the bottom right, it shows "Total rows: 1 of 1" and "Query complete 00:00:00.040".
- Bottom Right:** "Ln 108, Col 1"

5 – creating trigger function

The screenshot shows the pgAdmin 4 interface with the following details:

- Object Explorer:** On the left, under "Schemas (1) / public / Trigger Functions (1)", the trigger function "log_author_delete_update" is selected.
- Query Editor:** The main area contains the SQL code for creating the trigger function:

```
113 -- S
114 -- trigger function
115
116 -- trigger function
117 CREATE OR REPLACE FUNCTION log_author_delete_update()
118 RETURNS TRIGGER
119 AS $$
120 BEGIN
121     INSERT INTO author_logs (id_old, fname_old, lname_old)
122         VALUES (OLD.id, OLD.fname, OLD.lname);
123
124     -- TG_OP or trigger operation can check which operation is being used
125     IF TG_OP = 'DELETE' THEN
126         -- need to return OLD record for delete operations so that it deletes
127         RETURN OLD;
128     ELSEIF TG_OP = 'UPDATE' THEN
129         RETURN NEW;
130     END IF;
131 END;
132 $$ LANGUAGE plpgsql;
133
134 DROP TRIGGER IF EXISTS trg_author_delete_update
135 ON author;
```
- Data Output:** Shows "CREATE FUNCTION".
- Messages:** Shows "Query returned successfully in 39 msec."
- Notifications:** None.
- Status Bar:** Total rows: 1 of 1 | Query complete 00:00:00.039 | Ln 132, Col 21

5 – creating trigger on authors table

The screenshot shows the pgAdmin 4 interface with the following details:

- Object Explorer:** On the left, it lists various database objects under the schema "public". These include Aggregates, Collations, Domains, FTS Configurations, FTS Dictionaries, FTS Parsers, FTS Templates, Foreign Tables, Functions (with one entry: enter_author(p_fname text)), Materialized Views, Operators, Procedures (with one entry: insert_book(IN p_title text, I)), Sequences, Tables (with three entries: author, book, author_logs), Trigger Functions (with one entry: log_author_delete_update()), Types, and Views.
- Dashboard:** At the top, there are tabs for Properties, SQL, Statistics, Dependencies, Dependents, Processes, and a file named "solution.sql*".
- Query Pad:** The main area contains a query window titled "Query History". The code listed is:

```
127      RETURN OLD;
128      ELSEIF TG_OP = 'UPDATE' THEN
129          RETURN NEW;
130      END IF;
131
132      $$ LANGUAGE plpgsql;
133
134      DROP TRIGGER IF EXISTS trg_author_delete_update
135      ON author;
136
137      -- trigger
138      CREATE TRIGGER trg_author_delete_update
139      BEFORE DELETE OR UPDATE
140      ON author
141      FOR EACH ROW
142      EXECUTE PROCEDURE log_author_delete_update();
143
144      -- testing
145      UPDATE author
146      SET fname = 'Daniel'
147      WHERE id = 1;
148
149      DELETE FROM author
```

Below the query window, there are tabs for Data Output, Messages, and Notifications. The "Messages" tab is selected, showing the message "Query returned successfully in 40 msec."

At the bottom of the pgAdmin window, status bars indicate "Total rows: 1 of 1" and "Query complete 00:00:00.040".

5 – testing trigger update

The screenshot shows the pgAdmin 4 interface. The left pane is the Object Explorer, displaying the schema structure of the 'public' schema, including tables like 'author', 'book', and 'author_logs', and various database objects like triggers, functions, and types. The right pane is the Query Editor, showing a SQL script for creating and testing a trigger. The script includes logic for updates, a trigger creation, and test queries to update and delete rows in the 'author' table.

```
128     ELSEIF TG_OP = 'UPDATE' THEN
129         RETURN NEW;
130     END IF;
131
132     $$ LANGUAGE plpgsql;
133
134     DROP TRIGGER IF EXISTS trg_author_delete_update
135     ON author;
136
137     -- trigger
138     CREATE TRIGGER trg_author_delete_update
139     BEFORE DELETE OR UPDATE
140     ON author
141     FOR EACH ROW
142     EXECUTE PROCEDURE log_author_delete_update();
143
144     -- testing
145     UPDATE author
146     SET fname = 'Daniel'
147     WHERE id = 1;
148
149     DELETE FROM author
150     WHERE id = 2;
```

Query returned successfully in 48 msec.
Total rows: 1 of 1 | Query complete 00:00:00.048 | Ln 145, Col 1

5 – trigger update author results

The screenshot shows the pgAdmin 4 interface. The left pane is the Object Explorer, displaying the database schema. The 'Tables' section contains the 'author' table, which has columns: id [PK] integer, fname text, lname text. The table data is as follows:

	id [PK] integer	fname text	lname text
1	2	Sally	Gates
2	3	Jorge	Martinez
3	4	George	Kalos
4	5	Randall	Smith
5	6	Andres	Hung
6	1	Daniel	Kroenke

The right pane shows the Query History tab with the following SQL code:

```
1 v SELECT *
2   FROM author;
3
4 v SELECT *
5   FROM author_logs;
```

The Data Output tab displays the same table data as the table above.

5 – trigger update author_logs results

The screenshot shows the pgAdmin 4 interface. The left sidebar is the Object Explorer, displaying a tree view of database objects. The current node selected is 'Tables (3)'. Under 'Tables (3)', there are three entries: 'author', 'book', and 'author_logs'. The 'author' table has several child nodes: 'Columns', 'Constraints', 'Indexes', 'RLS Policies', 'Rules', 'Triggers (1)', and 'trg_author_delete_update'. The 'Triggers (1)' node is currently expanded, showing a single trigger named 'log_author_delete_update'. The main workspace contains a query editor with the following content:

```
1 ✓ SELECT *
2   FROM author;
3
4 ✓ SELECT *
5   FROM author_logs;
```

Below the query editor is a Data Output pane showing the results of the last query:

	id_old	fname_old	lname_old
1	1	David	Kroenke

At the bottom of the screen, the status bar displays: 'Total rows: 1 of 1 Query complete 00:00:00.061 Ln 4, Col 1'.

5 – testing trigger delete

The screenshot shows the pgAdmin 4 interface with the following details:

- Object Explorer:** On the left, it lists database objects under the schema `public`, including Aggregates, Collations, Domains, FTS Configurations, FTS Dictionaries, FTS Parsers, FTS Templates, Foreign Tables, Functions (1), Procedures (1), Sequences, Tables (3) (with `author` selected), Triggers (1) (with `trg_author_delete_update` selected), and Views.
- Dashboard:** Shows the connection information: `quiz4/quiz4@quiz4`.
- Properties:** Shows the current connection settings.
- SQL:** Shows the current session's SQL context.
- Statistics:** Shows performance metrics.
- Dependencies:** Shows dependencies between objects.
- Dependents:** Shows objects that depend on the current one.
- Processes:** Shows active processes.
- Solution:** Shows the file `solution.sql*`.
- Scratch Pad:** A temporary workspace.

Query Tab: Contains the following SQL code:

```
128     ELSEIF TG_OP = 'UPDATE' THEN
129         RETURN NEW;
130     END IF;
131
132     $$ LANGUAGE plpgsql;
133
134     DROP TRIGGER IF EXISTS trg_author_delete_update
135     ON author;
136
137     -- trigger
138     CREATE TRIGGER trg_author_delete_update
139     BEFORE DELETE OR UPDATE
140     ON author
141     FOR EACH ROW
142     EXECUTE PROCEDURE log_author_delete_update();
143
144     -- testing
145     UPDATE author
146     SET fname = 'Daniel'
147     WHERE id = 1;
148
149     DELETE FROM author
150     WHERE id = 2;
```

Data Output Tab: Shows the result of the query:

DELETE 1
Query returned successfully in 23 msec.

Total rows: 1 of 1 Query complete 00:00:00.023

✓ Query returned successfully in 23 msec. ✎
Ln 149, Col 1

5 – trigger delete author results

The screenshot shows the pgAdmin 4 interface. The left sidebar (Object Explorer) displays the database schema, including the 'author' table under 'Tables'. The 'Triggers' section for the 'author' table shows two triggers: 'trg_author_delete_update()' and 'log_author_delete_update()'. The central area contains a query editor with the following SQL code:

```
1 v SELECT *
2   FROM author;
3
4 v SELECT *
5   FROM author_logs;
```

The Data Output tab shows the results of the first query, which retrieves five rows from the 'author' table:

	id [PK] integer	fname text	lname text
1	3	Jorge	Martinez
2	4	George	Kalos
3	5	Randall	Smith
4	6	Andres	Hung
5	1	Daniel	Kroenke

The status bar at the bottom indicates "Total rows: 5 of 5" and "Query complete 00:00:00.057". A green message bar at the bottom right says "Successfully run. Total query runtime: 57 msec. 5 rows affected.".

5 – trigger delete author_logs results

The screenshot shows the pgAdmin 4 interface. The left sidebar (Object Explorer) displays the database schema:

- Publications
- Schemas (1)
 - public
 - Aggregates
 - Collations
 - Domains
 - FTS Configurations
 - FTS Dictionaries
 - FTS Parsers
 - FTS Templates
 - Foreign Tables
 - Functions (1)
 - enter_author(p_fname text)
 - Materialized Views
 - Operators
 - Procedures (1)
 - insert_book(IN p_title text, I)
 - Sequences
 - Tables (3)
 - author
 - Columns
 - Constraints
 - Indexes
 - RLS Policies
 - Rules
 - Triggers (1)
 - trg_author_delete_update()
 - log_author_delete_update()
 - author_logs
 - book
 - Trigger Functions (1)
 - log_author_delete_update()
 - Types
 - Views

The main area shows a query window with two SELECT statements:

```
1 v SELECT *  
2   FROM author;  
3  
4 v SELECT *  
5   FROM author_logs;
```

The Data Output tab shows the results of the second query:

	id_old	fname_old	lname_old
1	1	David	Kroenke
2	2	Sally	Gates

Total rows: 2 of 2 Query complete 00:00:00.031 Ln 4, Col 1