JAY ARRE TALOSIG
CTFDMBSL - COM-231
Mrs. JENSEN A. SANTILLAN

Activity

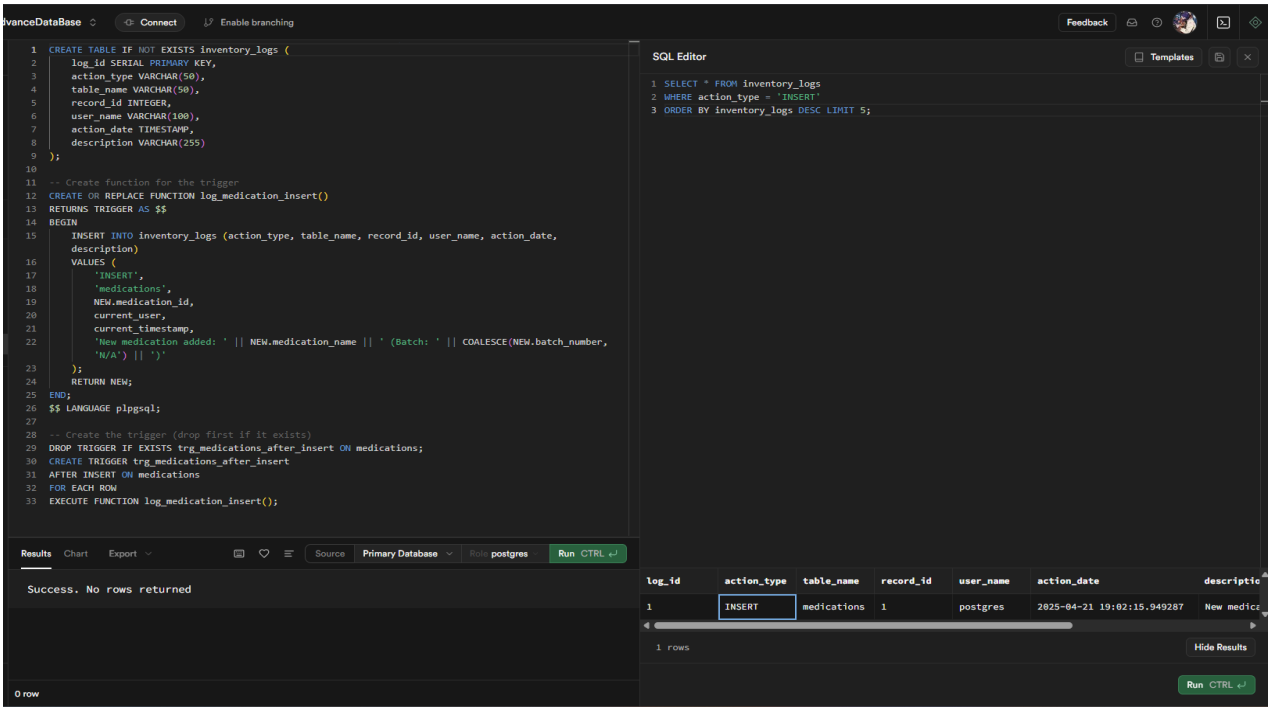
Practical Exercises

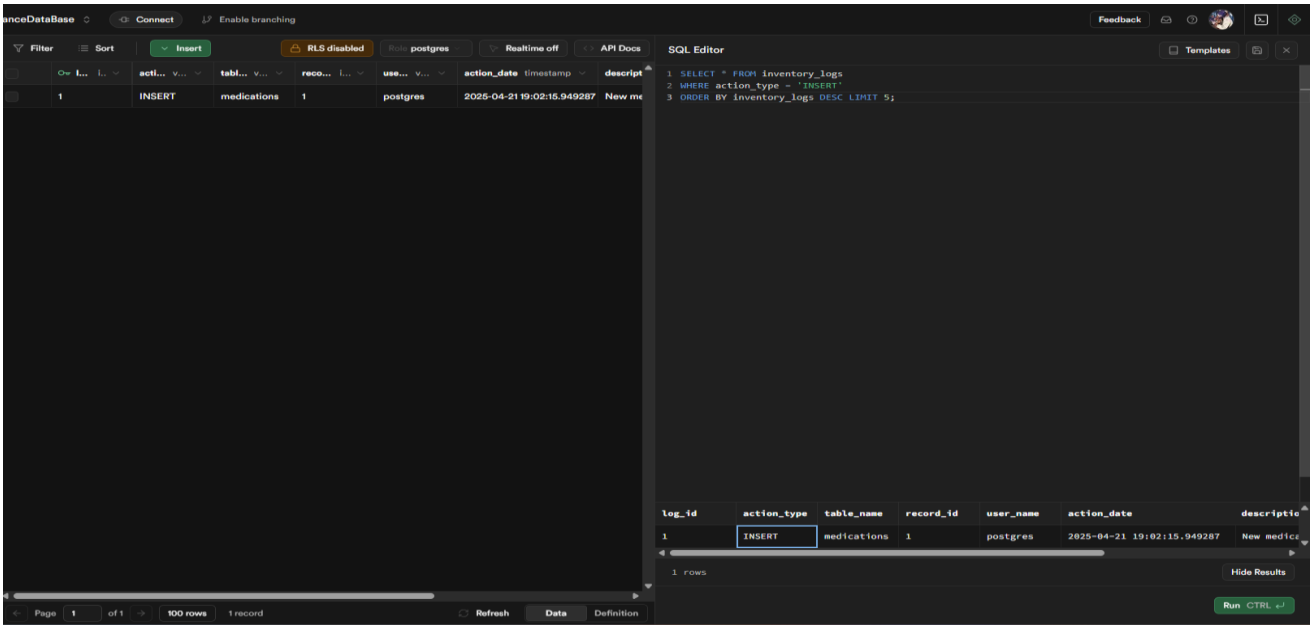
1. Create an AFTER INSERT trigger that logs every new

entry in a "Logs" table.

AFTER INSERT Trigger

This trigger automatically logs every new medication added to inventory.





Explanation:

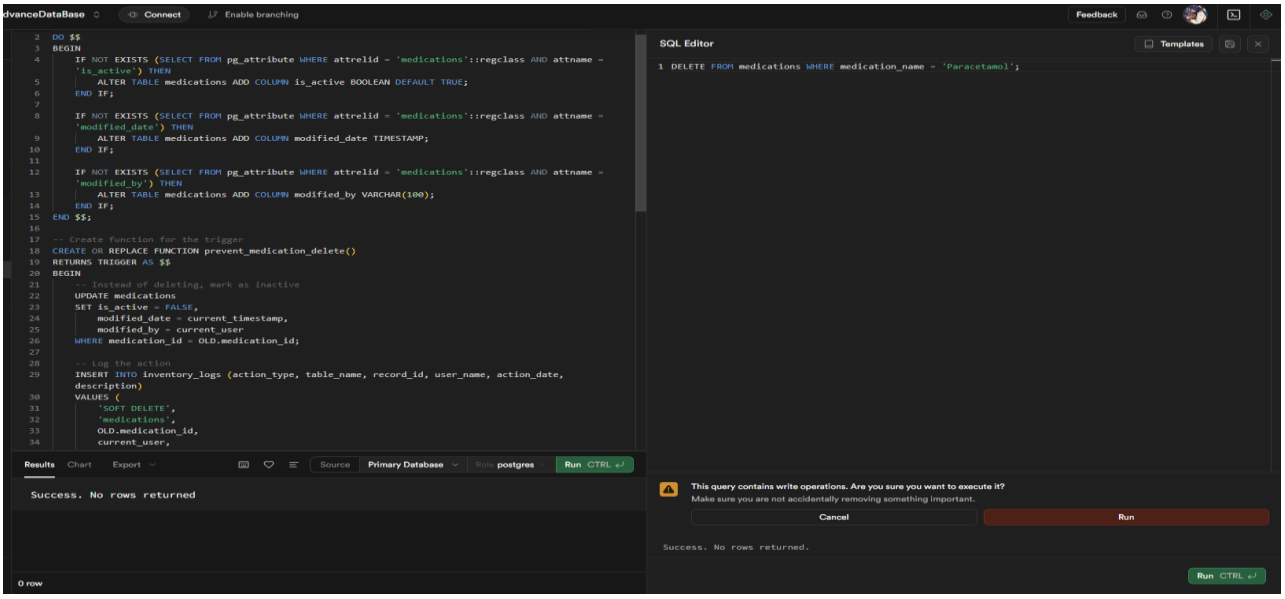
- I create a log table to store information about database actions
- The trigger function runs after each insert operation
- It captures details like who made the change, when it happened and what was added
- The NEW keyword refers to the newly inserted record's data

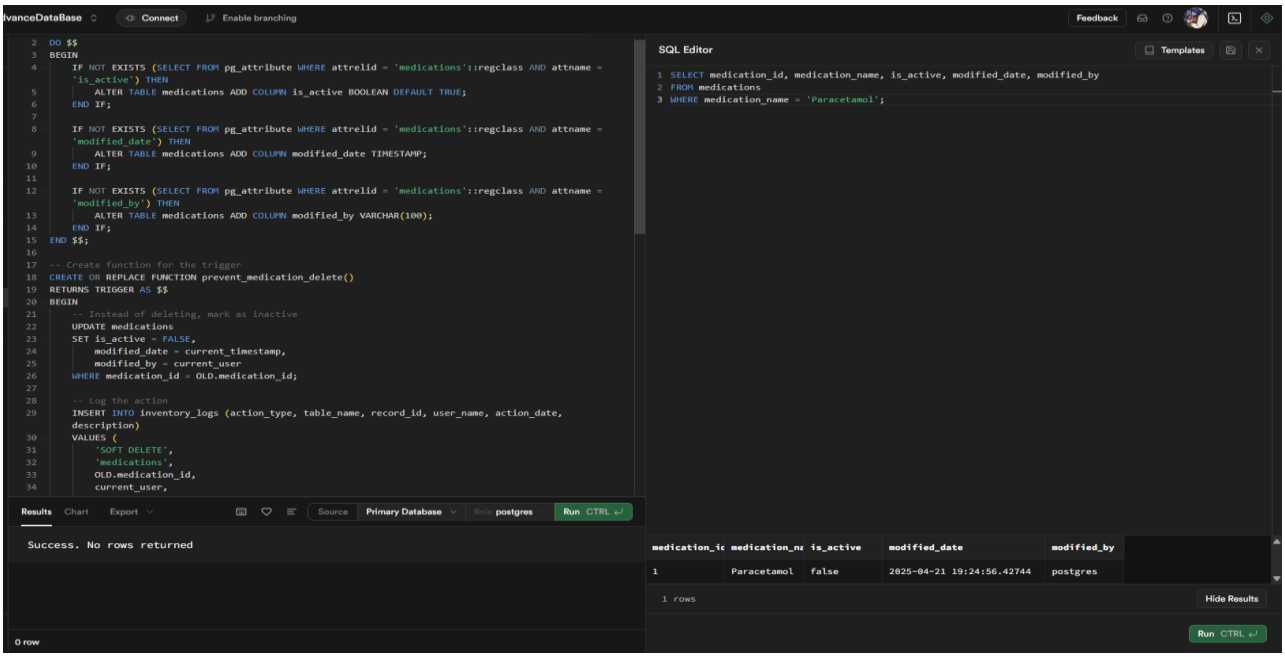
2. Develop an INSTEAD OF DELETE trigger that prevents

record deletion.

INSTEAD OF DELETE Trigger

This trigger prevents deletion of records and instead marks them as inactive:


dvanceDataBase
Connect
Enable branching
Feedback
DO $$
BEGIN
IF NOT EXISTS (SELECT FROM pg_attribute WHERE attrelid = 'medications'::regclass AND attname = 'is_active') THEN
ALTER TABLE medications ADD COLUMN is_active BOOLEAN DEFAULT TRUE;
END IF;
IF NOT EXISTS (SELECT FROM pg_attribute WHERE attrelid = 'medications'::regclass AND attname = 'modified_date') THEN
ALTER TABLE medications ADD COLUMN modified_date TIMESTAMP;
END IF;
IF NOT EXISTS (SELECT FROM pg_attribute WHERE attrelid = 'medications'::regclass AND attname = 'modified_by') THEN
ALTER TABLE medications ADD COLUMN modified_by VARCHAR(100);
END IF;
END $$;
-- Create function for the trigger
CREATE OR REPLACE FUNCTION prevent_medication_delete()
RETURNS TRIGGER AS $$
BEGIN
-- Instead of deleting, mark as inactive
UPDATE medications
SET is_active = FALSE,
modified_date = current_timestamp,
modified_by = current_user
WHERE medication_id = OLD.medication_id;
-- Log the action
INSERT INTO inventory_logs (action_type, table_name, record_id, user_name, action_date, description)
VALUES (
'SOFT DELETE',
'medications',
OLD.medication_id,
current_user,
Results
Chart
Export
Source
Primary Database
Role postgres
Run CTRL
Success. No rows returned
0 row
SQL Editor
Templates
1 DELETE FROM medications WHERE medication_name = 'Paracetamol';
This query contains write operations. Are you sure you want to execute it?
Make sure you are not accidentally removing something important.
Cancel
Run
Success. No rows returned.
Run CTRL

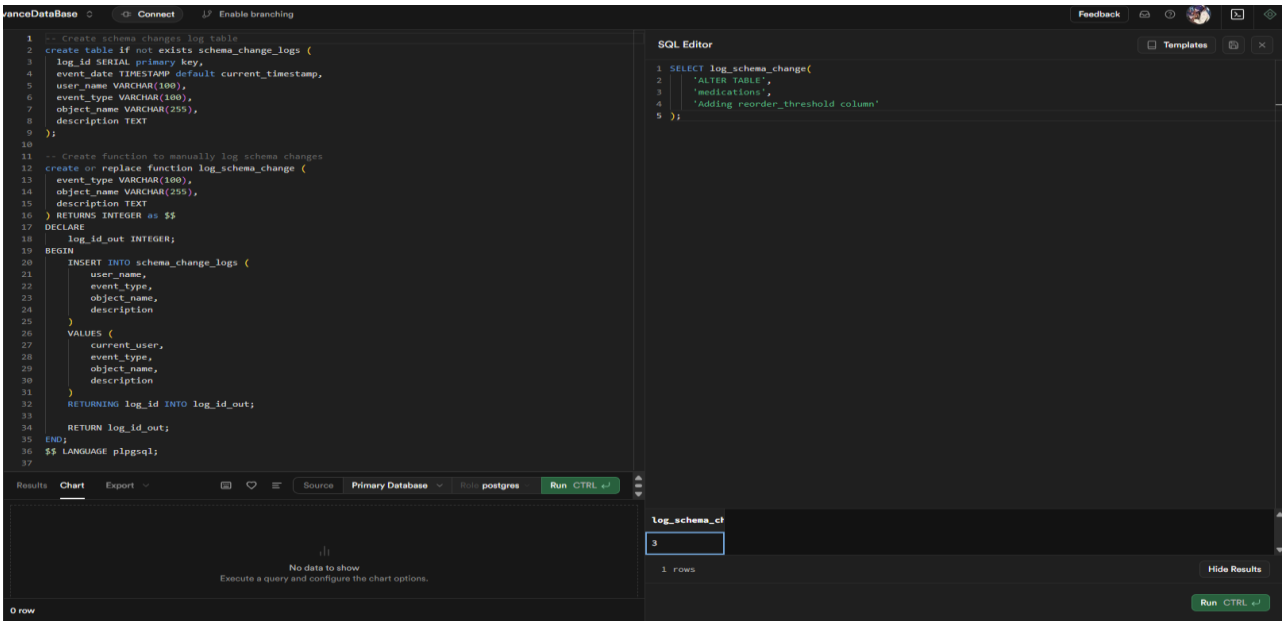

dvanceDataBase
Connect
Enable branching
Feedback
DO $$
BEGIN
IF NOT EXISTS (SELECT FROM pg_attribute WHERE attrelid = 'medications'::regclass AND attname = 'is_active') THEN
ALTER TABLE medications ADD COLUMN is_active BOOLEAN DEFAULT TRUE;
END IF;
IF NOT EXISTS (SELECT FROM pg_attribute WHERE attrelid = 'medications'::regclass AND attname = 'modified_date') THEN
ALTER TABLE medications ADD COLUMN modified_date TIMESTAMP;
END IF;
IF NOT EXISTS (SELECT FROM pg_attribute WHERE attrelid = 'medications'::regclass AND attname = 'modified_by') THEN
ALTER TABLE medications ADD COLUMN modified_by VARCHAR(100);
END IF;
END $$;
-- Create function for the trigger
CREATE OR REPLACE FUNCTION prevent_medication_delete()
RETURNS TRIGGER AS $$
BEGIN
-- Instead of deleting, mark as inactive
UPDATE medications
SET is_active = FALSE,
modified_date = current_timestamp,
modified_by = current_user
WHERE medication_id = OLD.medication_id;
-- Log the action
INSERT INTO inventory_logs (action_type, table_name, record_id, user_name, action_date, description)
VALUES (
'SOFT DELETE',
'medications',
OLD.medication_id,
current_user,
Results
Chart
Export
Source
Primary Database
Role postgres
Run CTRL
Success. No rows returned
0 row
SQL Editor
Templates
1 SELECT medication_id, medication_name, is_active, modified_date, modified_by
2 FROM medications
3 WHERE medication_name = 'Paracetamol';
medication_id
medication_name
is_active
modified_date
modified_by
1
Paracetamol
false
2025-04-21 19:24:56.42744
postgres
1 rows
Hide Results
Run CTRL

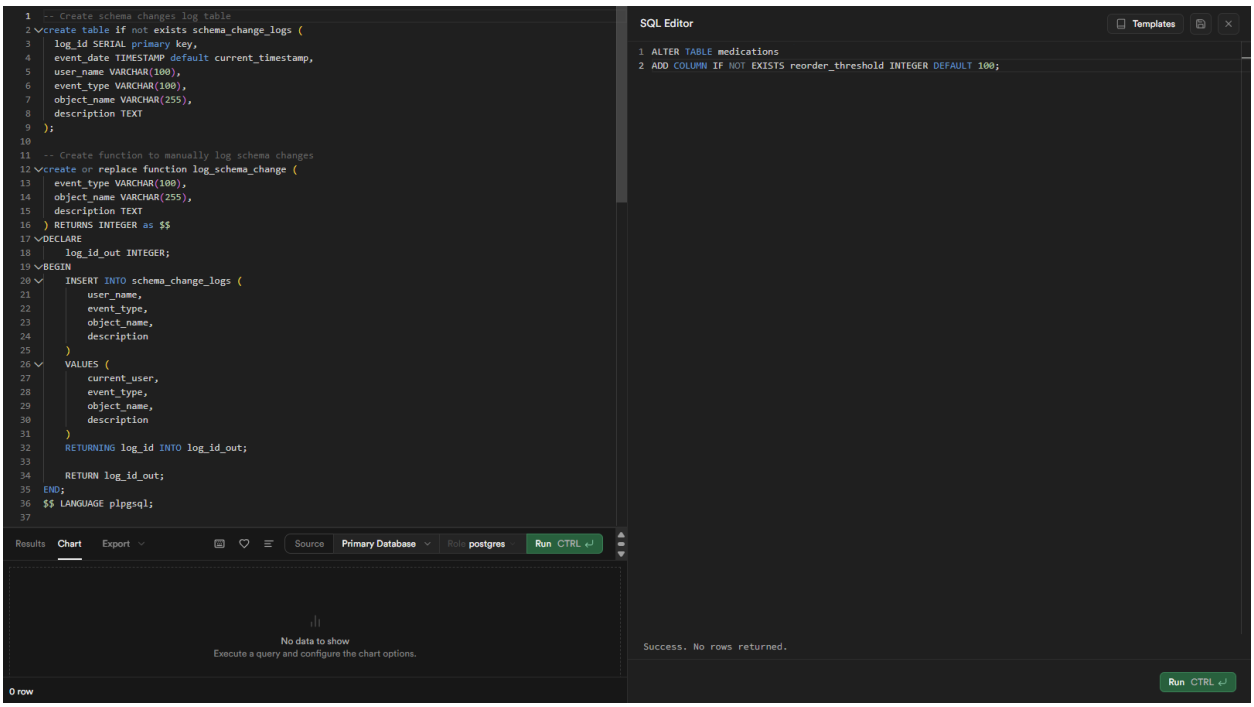
Explanation:

- The trigger activates BEFORE DELETE operations
- Instead of allowing the delete, it updates the record to mark it inactive
- It logs this action to maintain an audit trail

3. Design a DDL trigger to capture schema modifications.

DDL Trigger for Schema Changes


vanceDataBase
Connect
Enable branching
Feedback
-- Create schema changes log table
create table if not exists schema_change_logs (
log_id SERIAL primary key,
event_date TIMESTAMP default current_timestamp,
user_name VARCHAR(100),
event_type VARCHAR(100),
object_name VARCHAR(255),
description TEXT
);
-- Create function to manually log schema changes
create or replace function log_schema_change (
event_type VARCHAR(100),
object_name VARCHAR(255),
description TEXT
) RETURNS INTEGER as $$
DECLARE
log_id_out INTEGER;
BEGIN
INSERT INTO schema_change_logs (
user_name,
event_type,
object_name,
description
)
VALUES (
current_user,
event_type,
object_name,
description
)
RETURNING log_id INTO log_id_out;
RETURN log_id_out;
END;
$$ LANGUAGE plpgsql;
Results
Chart
Export
Source
Primary Database
Role postgres
Run CTRL
No data to show
Execute a query and configure the chart options.
0 row
SQL Editor
Templates
SELECT log_schema_change(
'ALTER TABLE',
'medications',
'Adding reorder_threshold column'
);
log_schema_ch
3
1 rows
Hide Results
Run CTRL


-- Create schema changes log table
create table if not exists schema_change_logs (
log_id SERIAL primary key,
event_date TIMESTAMP default current_timestamp,
user_name VARCHAR(100),
event_type VARCHAR(100),
object_name VARCHAR(255),
description TEXT
);
-- Create function to manually log schema changes
create or replace function log_schema_change (
event_type VARCHAR(100),
object_name VARCHAR(255),
description TEXT
) RETURNS INTEGER as $$
DECLARE
log_id_out INTEGER;
BEGIN
INSERT INTO schema_change_logs (
user_name,
event_type,
object_name,
description
)
VALUES (
current_user,
event_type,
object_name,
description
)
RETURNING log_id INTO log_id_out;
RETURN log_id_out;
END;
$$ LANGUAGE plpgsql;
Results
Chart
Export
Source
Primary Database
Role postgres
Run CTRL
No data to show
Execute a query and configure the chart options.
0 row
SQL Editor
Templates
ALTER TABLE medications
ADD COLUMN IF NOT EXISTS reorder_threshold INTEGER DEFAULT 100;
Success. No rows returned.
Run CTRL

SQL Editor

Templates

```
1 SELECT * FROM schema_change_logs
2 ORDER BY log_id DESC LIMIT 5;
```

log_id	event_date	user_name	event_type	object_name	description
3	2025-04-21 19:33:05.	postgres	ALTER TABLE	medications	Adding reorder_threshold column
2	2025-04-21 19:32:36.	postgres	ALTER TABLE	medications	Adding reorder_threshold column
1	2025-04-21 19:31:54.	postgres	ALTER TABLE	medications	Adding reorder_threshold column

3 rows

Hide Results

SQL Editor

Templates

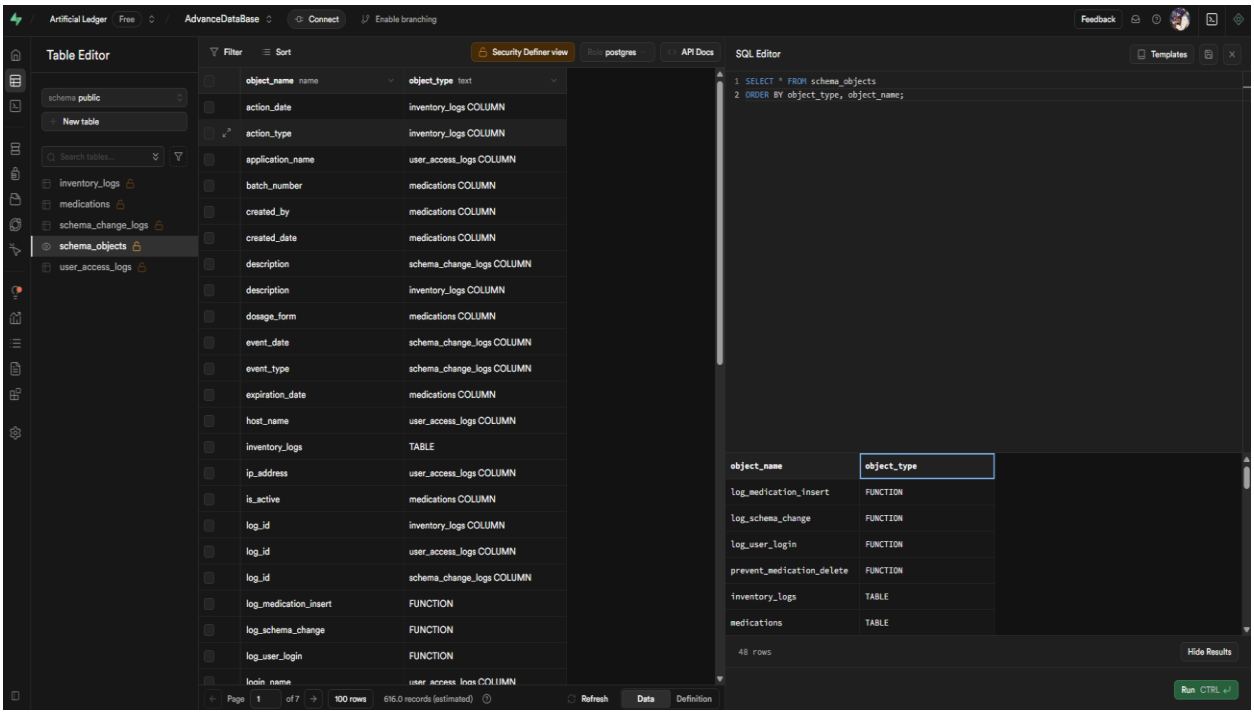
```
1 SELECT * FROM schema_objects
2 ORDER BY object_type, object_name;
```

object_name	object_type
log_medication_insert	FUNCTION
log_schema_change	FUNCTION
log_user_login	FUNCTION
prevent_medication_delete	FUNCTION
inventory_logs	TABLE
medications	TABLE

48 rows

Hide Results

Run CTRL ↵

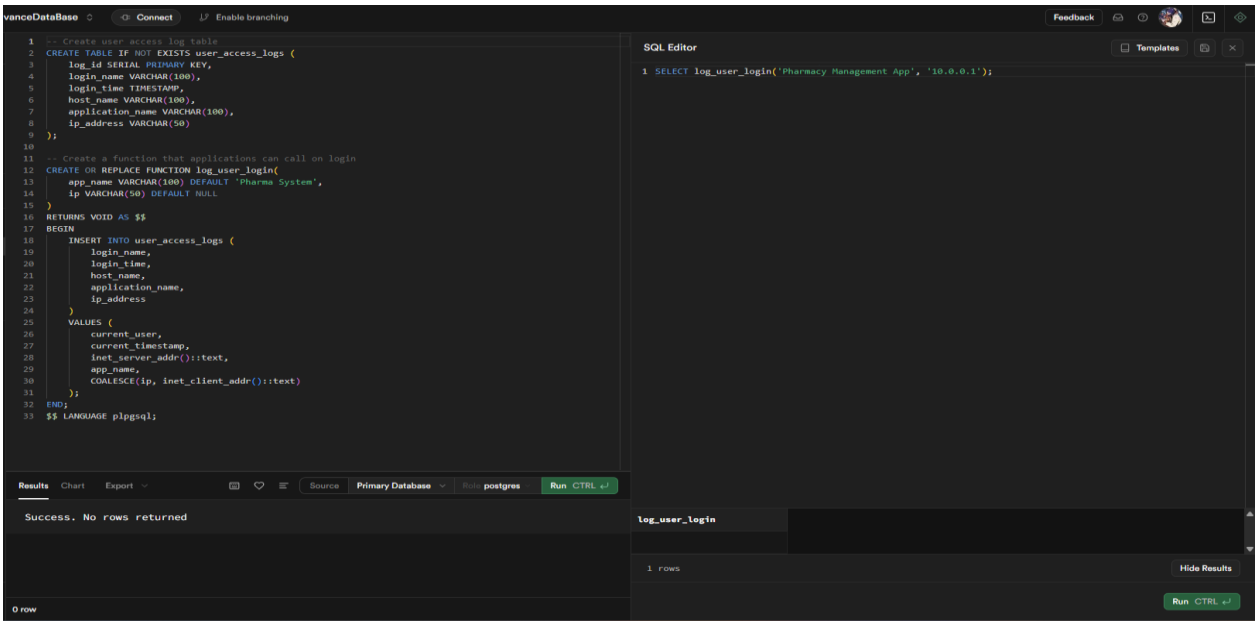


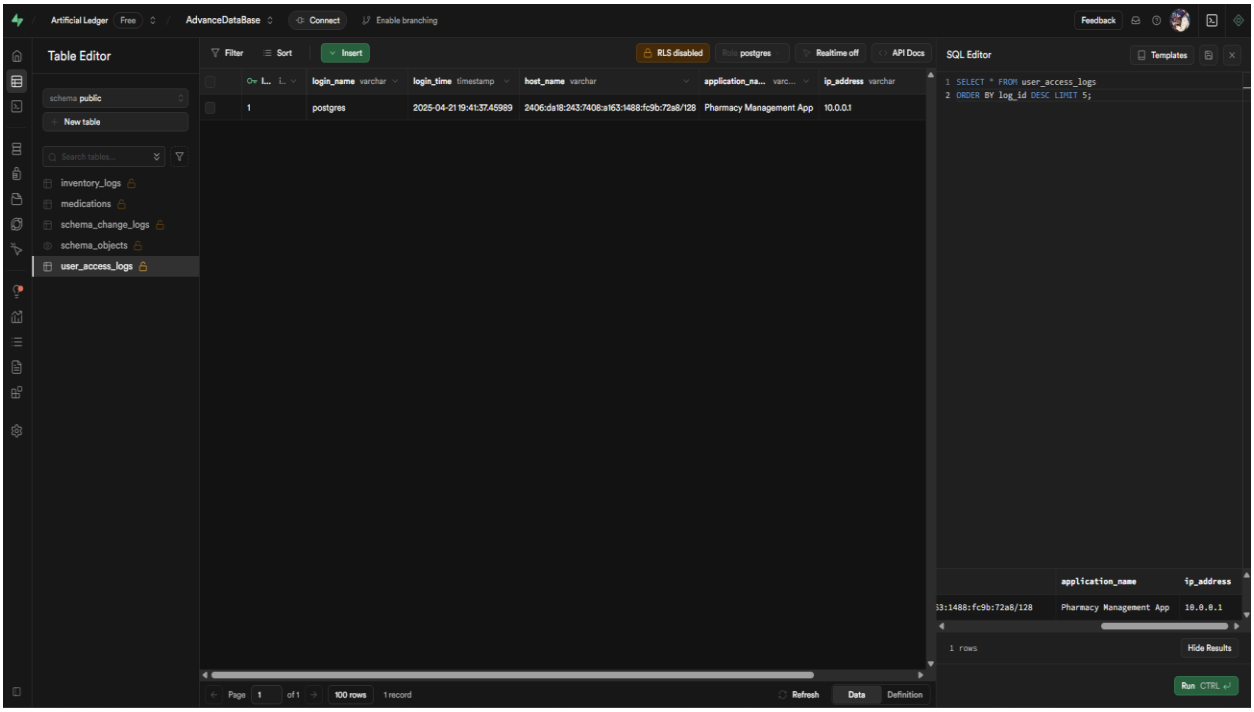
Explanation:

- This is a database-level trigger
- It fires whenever schema changes occur (CREATE, ALTER, DROP operations)
- It logs details about what changed, who made the change and when
- The `pg_event_trigger_ddl_commands()` function provides details about the change
- This helps track structural changes for compliance and troubleshooting

4. Implement a Logon trigger to track user logins.

User Logon trigger





Explanations:

- This creates a function to log user logins
- Since I can't intercept actual database logins automatically, the application needs to call this function
- It records who logged in, when, from where, and which application they used
- Certain application would call this whenever a user logs in

Q&A and Discussion

1. How can triggers enhance database security?

Triggers significantly enhance database security in pharmaceutical supply chain management by creating automated safeguards and audit mechanisms. They provide a reliable way to enforce security policies consistently across all database interactions without depending on application-level controls.

For pharmaceutical databases where regulatory compliance is critical, triggers establish a robust audit trail by automatically recording all data modifications. This is especially valuable for tracking controlled substances and ensuring compliance with regulations like HIPAA, FDA requirements, and drug pedigree laws.

The INSTEAD OF DELETE trigger we implemented demonstrates how triggers can prevent unauthorized data deletion—critical for maintaining the integrity of medication records that may be needed for years due to legal requirements. Instead of allowing records to be permanently removed, the trigger enforces a soft-delete policy by marking records inactive while preserving the complete history.

My implementation in Supabase shows how even with cloud database limitations, triggers can enforce complex validation rules and access controls beyond what standard database constraints provide. While true login triggers weren't available due to Supabase permission restrictions, the function-based approach still demonstrates how tracking mechanisms can help identify suspicious access patterns.

2. What are the limitations of triggers?

Despite their benefits, my Supabase implementation revealed several significant limitations of triggers in pharmaceutical database systems. Performance impact is a primary concern each trigger execution adds processing overhead, which can accumulate in high-transaction environments like busy pharmacies where medication dispensing must happen quickly.

The most notable limitation I encountered was with permission restrictions in Supabase. When attempting to implement a DDL trigger to track schema changes, I received a superuser privilege error, forcing me to create an alternative manual logging approach. This highlights how cloud-hosted solutions may limit certain types of triggers, requiring adaptations to your security architecture.

Triggers in Supabase also introduce hidden application logic that executes automatically without explicit calls. During testing, I noticed how this can make troubleshooting challenging since it's not immediately obvious that a trigger is affecting database operations. When issues arise in complex scenarios, tracing the problem through multiple cascading triggers becomes difficult.

Maintenance complexity increases as the system grows in Supabase. As I added more functionality to the pharmaceutical system, I had to carefully consider how new triggers might interact with existing ones. Additionally, triggers proved more difficult to test thoroughly than standard functions in the Supabase environment.

3. When should you use triggers versus other stored procedures?

My implementation in Supabase demonstrates that triggers are best used for automatic, consistent enforcement of rules that should never be bypassed. The AFTER INSERT trigger for automatic audit logging and the INSTEAD OF DELETE trigger for enforcing data integrity policies work well because these are scenarios where the action should happen automatically every time, without exception.

For scenarios requiring complex business logic or explicit control, I found functions and stored procedures in Supabase more appropriate. The manual schema logging function I implemented demonstrates this distinction it's a procedure that must be explicitly called rather than happening automatically, providing more control and visibility.

In my Supabase pharmaceutical system, I used functions for processes that would typically require stored procedures in other database systems, such as the log_schema_change and log_user_login functions. These are better suited for operations like end-of-day inventory reconciliation or processing returns of expired medications.

Through my implementation in Supabase, I found that functions offer better error handling and transaction control, critical when performing multi-step pharmaceutical operations. The hybrid approach I used triggers for automatic security enforcement and audit logging, with functions for operations requiring explicit control provides both strong security guarantees and operational flexibility within Supabase's PostgreSQL environment.