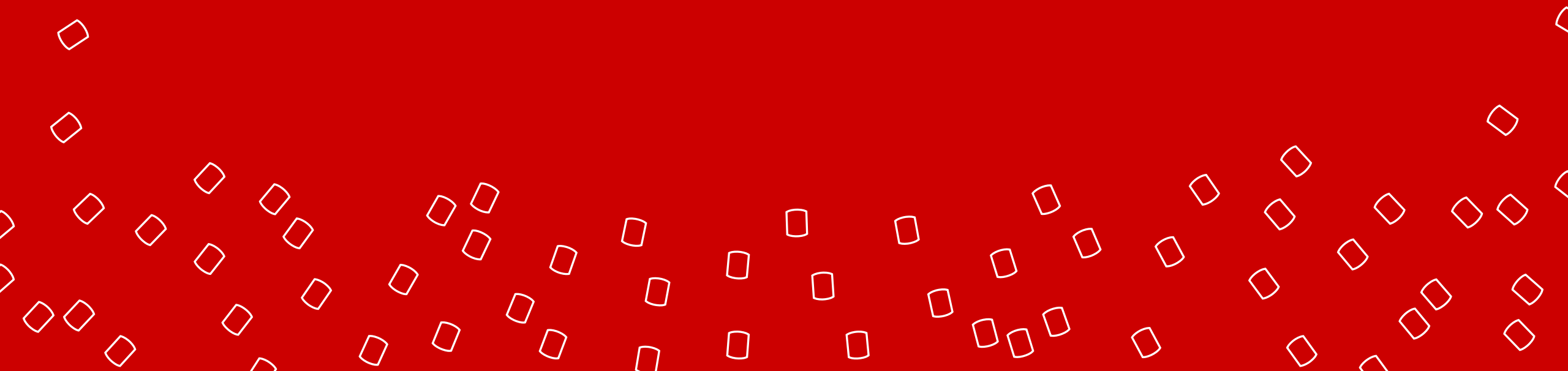


# Leveraging AI as a PostgreSQL DBA



| About me

# Grant Fritchey

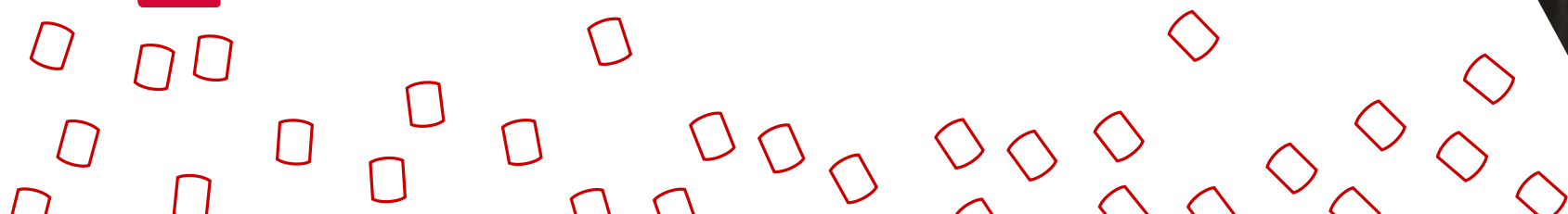
## DevOps Advocate

Microsoft PostgreSQL MVP  
AWS Community Builder

 @gfritchey

 Grant.Fritchey@Red-Gate.com

 scarydba.com



## Session Goals



- Learn mechanisms to ensure good AI results
- Understand how to put AI to work day-to-day
- Discover ways AI helps administer PostgreSQL

## Absolutely Not Goals



- Discussion of implementation of AI extensions
- Development of AI within your environment



## | You Have To Understand...



- I'm just a DBA
- But I want to help

## | Starting Line

- All About That Prompt
- Trust but Verify
- Wrenches, Screwdrivers & Hammers, Oh My



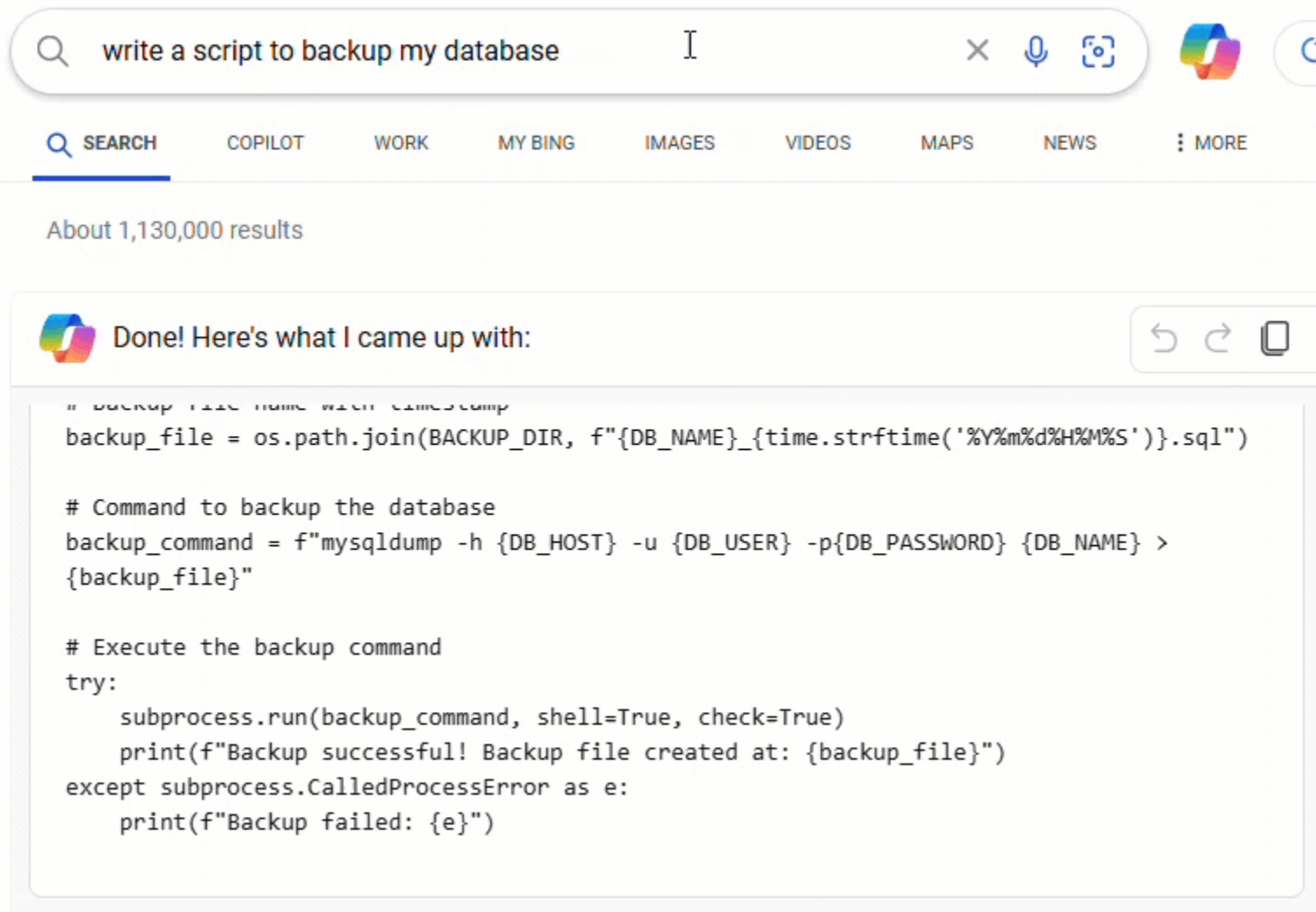


## | All About That Prompt

- Clarity, clarity, clarity
- Use appropriate terms
- More detail is better
- Context matters
- Stop digging, start over



# Bad Demo



The image shows a Bing search interface. The search bar contains the text "write a script to backup my database". Below the search bar, there are navigation links for "SEARCH", "COPILOT", "WORK", "MY BING", "IMAGES", "VIDEOS", "MAPS", "NEWS", and "MORE". The search results show "About 1,130,000 results". A snippet from a result is displayed, starting with "Done! Here's what I came up with:". The snippet contains a Python script for backing up a MySQL database. The script defines a backup file path, constructs a backup command using environment variables for database host, user, password, and name, and then uses the subprocess module to execute the command. It includes try/except blocks for error handling.

```
# backup file name with timestamp
backup_file = os.path.join(BACKUP_DIR, f"{DB_NAME}_{time.strftime('%Y%m%d%H%M%S')}.sql")

# Command to backup the database
backup_command = f"mysqldump -h {DB_HOST} -u {DB_USER} -p{DB_PASSWORD} {DB_NAME} >
{backup_file}"

# Execute the backup command
try:
    subprocess.run(backup_command, shell=True, check=True)
    print(f"Backup successful! Backup file created at: {backup_file}")
except subprocess.CalledProcessError as e:
    print(f"Backup failed: {e}")
```



## | Trust But Verify

- Old training data
- Bad training data
- Hallucinations
- Security!



## DeleteMeNow

- Can I get a detailed explanation of the native postgresql function called deletemenow?
- I meant the deletemenow built-in function created by Ryan Booz





## Use **Smart Assistant** to ask anything

### Trending



LA Earthquake Rattles Southern California

News



Olympic Gold Rush

Sports

### For You



The Best Way to Say "Thank You. Hire Me."

Writing



Don't Get Cited for Bad Citations

Education

Browse more Assistants and AI models!

Smart

GPT-4o

Research

Genius

Creative

More

Ask anything...





## | Wrenches, Screwdrivers and Hammers, Oh My

- Which tools to use?
- Back to our goals, just a DBA
- Basically, Chat
  - Microsoft CoPilot
  - Google Gemini
  - OpenAI ChatGPT
  - Perplexity
  - You.com
  - Brave.com



# Putting AI To Work as a DBA

- Focused learning
- Designing databases
- Generating data
- Creating test scripts
- Writing code
- Query Optimization
- Code Evaluation





## | Focused Learning

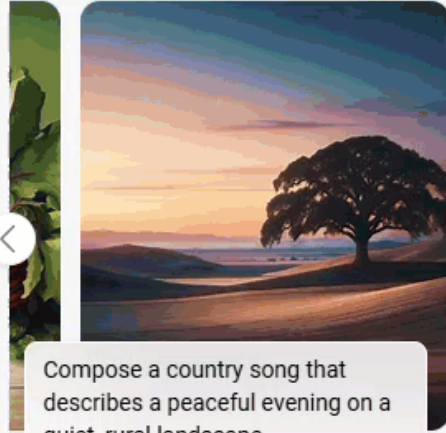
- How does PostgreSQL enforce referential integrity?





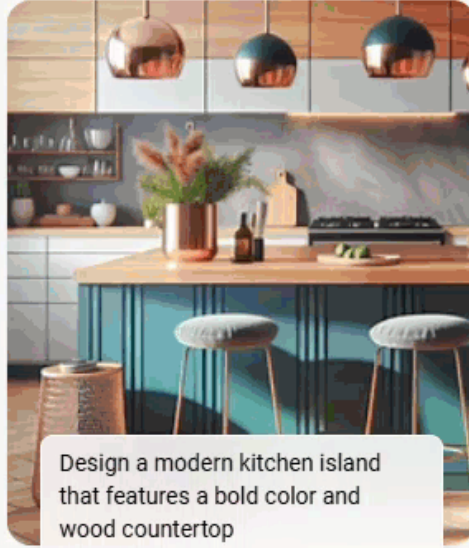


Your everyday AI companion



Compose a country song that describes a peaceful evening on a quiet, rural landscape

[Suno](#) [Terms](#) | [Privacy](#)



Design a modern kitchen island that features a bold color and wood countertop



Compose a pop song about an astronaut returning to Earth

[Suno](#) [Terms](#) | [Privacy](#)

Copilot uses AI. Check for mistakes. [Terms](#) | [Privacy](#) | [FAQs](#) [Try Copilot Pro](#)

Choose a conversation style

More  
Creative

More  
Balanced

More  
Precise



How does PostgreSQL enforce referential integrity?



## Database Design

- Create a set of postgresql tables including primary and foreign keys for amateur radios and radio operators where the relationship between the two is many to many.



# Hello, Grant

## How can I help you today?

Explain the following code step-by-step in detail



Create a splashy watercolor image



Explain how something works like an engineer



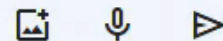
Design a custom cake for a good friend



Humans review some saved chats to improve Google AI. To stop this for future chats, turn off Gemini Apps Activity. If this setting is on, don't enter info you wouldn't want reviewed or used. [How it works](#)

[Manage Activity](#) [Dismiss](#)

Create a set of postgresql tables including primary and foreign keys for amateur radios and radio operators where the relationship between the two is many to many.

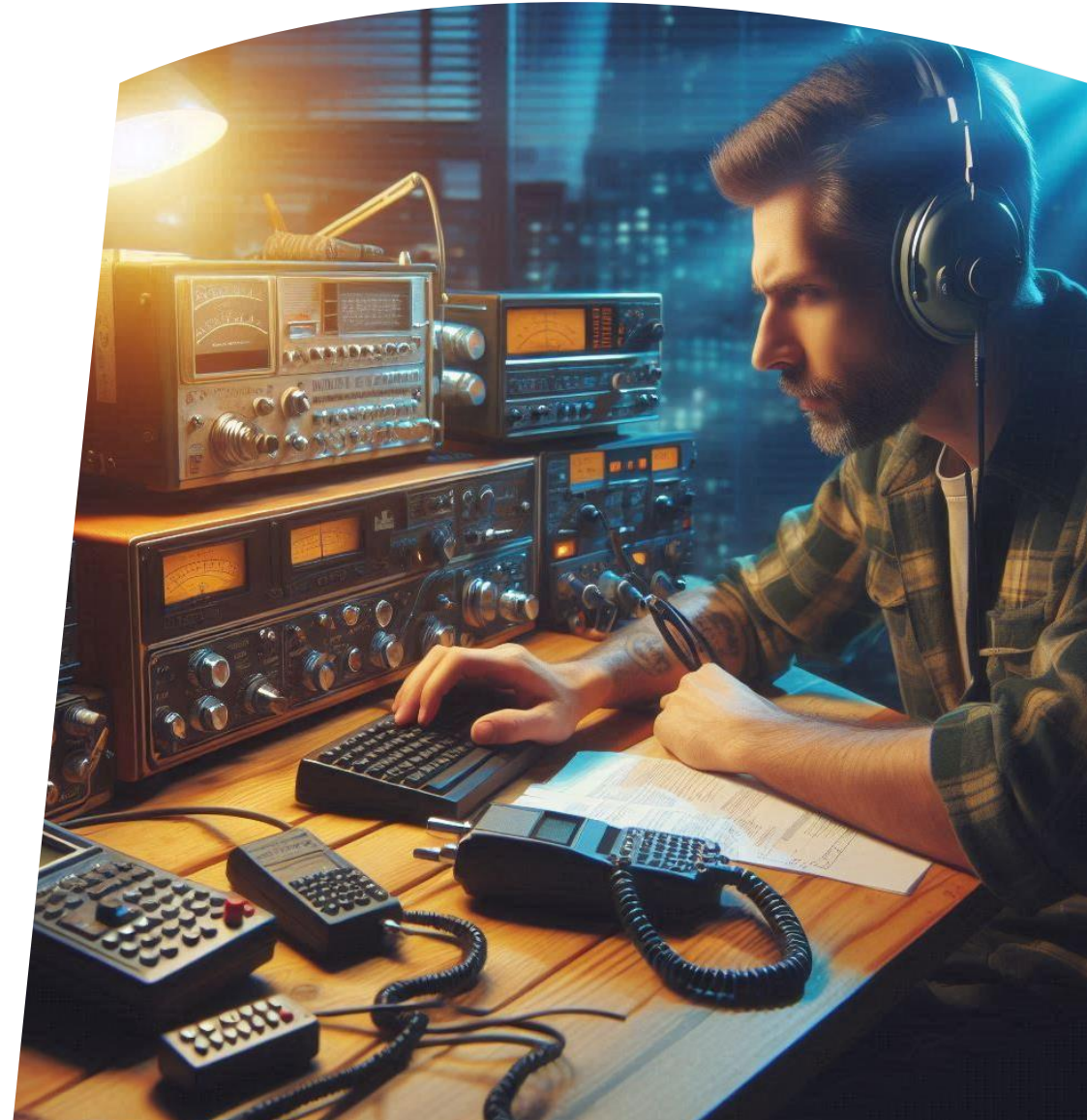




```
CREATE TABLE radios (  
    radio_id SERIAL PRIMARY KEY,  
    callsign TEXT UNIQUE,  
    model TEXT,  
    manufacturer TEXT,  
    year_made INTEGER,  
    frequency_range TEXT,  
    power_output INTEGER  
);  
  
CREATE TABLE operators (  
    operator_id SERIAL PRIMARY KEY,  
    callsign TEXT UNIQUE,  
    first_name TEXT,  
    last_name TEXT,  
    license_class TEXT,  
    city TEXT,  
    state TEXT,  
    country TEXT  
);  
  
CREATE TABLE radio_operators (  
    radio_operator_id SERIAL PRIMARY KEY,  
    radio_id INTEGER REFERENCES radios(radio_id),  
    operator_id INTEGER REFERENCES operators(operator_id)  
);
```

## | Generating Data

- Using those tables, can you generate full inserts for sample data based on actual amateur radios and operators.



```
);

-- Junction Table to establish the many-to-many relationship
CREATE TABLE operator_radios (
  operator_id INT NOT NULL,      -- Foreign key referencing operators
  radio_id INT NOT NULL,        -- Foreign key referencing radios
  assigned_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP, -- Timestamp when the operator was assigned
  PRIMARY KEY (operator_id, radio_id), -- Composite primary key
  FOREIGN KEY (operator_id) REFERENCES operators(id) ON DELETE CASCADE, -- Foreign key constraint
  FOREIGN KEY (radio_id) REFERENCES radios(id) ON DELETE CASCADE -- Foreign key constraint
);
```

### Explanation:

- ``radios`` Table: Stores details about each radio, like model, brand, and frequency band.
- ``operators`` Table: Stores details about each radio operator, including their name, license number, and country.
- ``operator_radios`` Table: A junction table with composite primary keys (``operator_id`` and ``radio_id``) to represent the many-to-many relationship between radios and operators. Foreign key constraints are added to ensure data integrity.

This design allows an operator to have multiple radios, and each radio can be operated by multiple operators. The ``ON DELETE CASCADE`` in foreign key constraints ensures that if a record in the ``operators`` or ``radios`` table is deleted, the corresponding records in the ``operator_radios`` table are also deleted.



- [How to insert data?](#)
- [Query many-to-many relationship?](#)

Using those tables, can you generate full inserts for sample data based on actual amateur radios and operators.





```
*<LearningPostgres> Console
*<LearningPostgres> Console X

/ /

◉ INSERT INTO radios (callsign, model, manufacturer, year_made, fr
('W1ABC', 'TS-450S', 'Kenwood', 1998, '1.8-30MHz', 100),
('N2XX', 'IC-7300', 'Icom', 2017, '1.8-54MHz', 100),
('K3LR', 'FT-2000X', 'Yaesu', 2009, '1.8-54MHz', 150),
('W4KYV', 'FT-817ND', 'Yaesu', 2004, '1.8-30MHz', 5W);

◉ INSERT INTO operators (callsign, first_name, last_name, license_
('W1ABC', 'John', 'Doe', 'Extra', 'Boston', 'MA', 'USA'),
('N2XX', 'Jane', 'Smith', 'General', 'New York', 'NY', 'USA'),
('K3LR', 'Michael', 'Johnson', 'Technician', 'Philadelphia', 'PA
('W4KYV', 'Emily', 'Davis', 'Ham', 'Atlanta', 'GA', 'USA');

◉ INSERT INTO radio_operators (radio_id, operator_id) VALUES
((SELECT radio_id FROM radios WHERE callsign = 'W1ABC'), (SELECT
((SELECT radio_id FROM radios WHERE callsign = 'N2XX'), (SELECT
((SELECT radio_id FROM radios WHERE callsign = 'K3LR'), (SELECT
((SELECT radio_id FROM radios WHERE callsign = 'W4KYV'), (SELECT
```

## | Creating Test Scripts

- Now create a script to test the database to determine if all the other scripts have functioned correctly on the postgresql database.



Now create a script to test the database to determine if all the other scripts have functioned correctly on the postgresql database.

Cancel

Save

## ✿ Answer

Certainly! I'll create a PostgreSQL script that tests the database to ensure all the previous scripts have functioned correctly. This script will check for the existence of tables, verify the data insertions, and test some relationships. Here's a comprehensive test script:

sql

```
-- Test Script for Amateur Radio Database

-- Start transaction (so we can rollback any changes made by the test)
BEGIN;

-- Function to log test results
CREATE OR REPLACE FUNCTION log_test(test_name TEXT, result BOOLEAN) RETURNS
VOID AS $$
BEGIN
    IF result THEN
        RAISE NOTICE '% - PASSED', test_name;
    ELSE
        RAISE EXCEPTION '% - FAILED', test_name;
    END IF;
END;
```

⊕ Ask follow-up

Pro



```
-- 1. Check if tables exist
```



```
* <LearningPostgres> Console
* <LearningPostgres> Script-2 X

-- Check data integrity
SELECT * FROM radios WHERE callsign IS NULL;
SELECT * FROM operators WHERE callsign IS NULL;
SELECT * FROM radio_operators WHERE radio_id IS NULL OR operator_id IS NULL;

-- Verify foreign key relationships
SELECT * FROM radio_operators WHERE radio_id NOT IN (SELECT radio_id FROM radios);
SELECT * FROM radio_operators WHERE operator_id NOT IN (SELECT operator_id FROM op

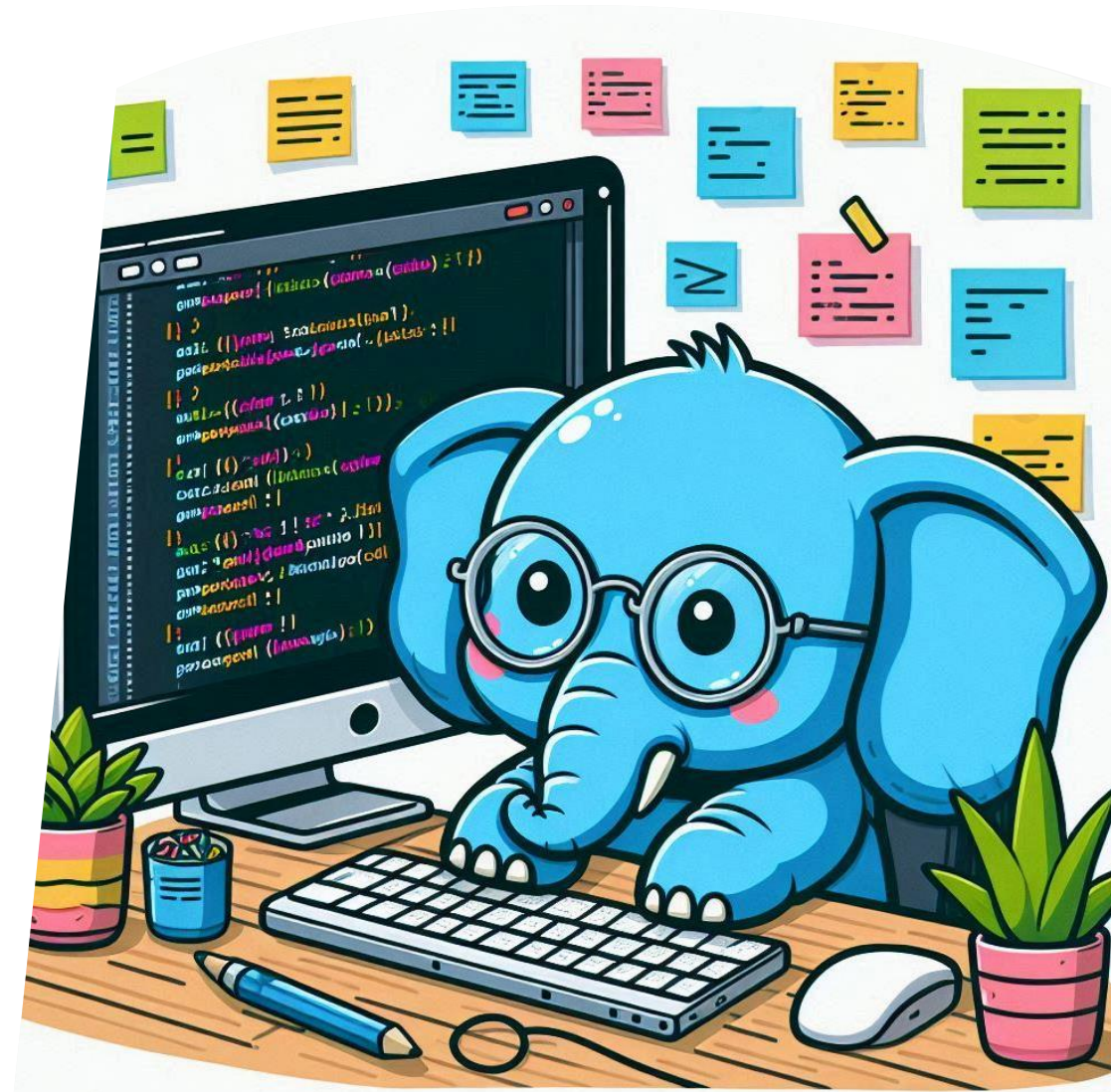
-- Test constraints
INSERT INTO radios (callsign) VALUES ('W1ABC'); -- Should fail if callsign is UNIQ
INSERT INTO operators (callsign) VALUES ('N2XX'); -- Should fail if callsign is UN

-- Test common queries
```

Name	Value
Updated Rows	4
Query	INSERT INTO radio_operators (radio_id, operator_id) VALUES ((SELECT radio_id FROM radios WHERE callsign = 'W1ABC'), (SELECT operator_id FROM operators WHERE callsign = 'W1ABC')), ((SELECT radio_id FROM radios WHERE callsign = 'N2XX'), (SELECT operator_id FROM operators WHERE callsign = 'N2XX')), ((SELECT radio_id FROM radios WHERE callsign = 'K3LR'), (SELECT operator_id FROM operators WHERE callsign = 'K3LR')), ((SELECT radio_id FROM radios WHERE callsign = 'W4KYV'), (SELECT operator_id FROM operators WHERE callsign = 'W4KYV'))
Start time	Mon Aug 26 13:33:10 CDT 2024
Finish time	Mon Aug 26 13:33:10 CDT 2024

## Writing Code

- Create a PostgreSQL function to retrieve operators that own a particular radio. Function input parameter should be radio\_id. Output will be the radio model and brand as well as the operator first and last name and their call sign.



- **Constraints:** Check if any defined constraints (e.g., `UNIQUE`, `NOT NULL`) are being enforced.
- **Queries:** Test common queries that might be used to retrieve or analyze data.

### Proposed Test Script

#### SQL

```
-- Check data integrity
SELECT * FROM radios WHERE callsign IS NULL;
SELECT * FROM operators WHERE callsign IS NULL;
SELECT * FROM radio_operators WHERE radio_id IS NULL OR operator_id IS NULL;

-- Verify foreign key relationships
SELECT * FROM radio_operators WHERE radio_id NOT IN (SELECT radio_id FROM radios);
SELECT * FROM radio_operators WHERE operator_id NOT IN (SELECT operator_id FROM operators);

-- Test constraints
INSERT INTO radios (callsign) VALUES ('W1ABC'); -- Should fail if callsign is unique
INSERT INTO operators (callsign) VALUES ('N2XX'); -- Should fail if callsign is unique

-- Test common queries
SELECT COUNT(*) FROM radios;
SELECT COUNT(*) FROM operators;
SELECT COUNT(*) FROM radio_operators;

SELECT r.callsign, o.first_name, o.last_name
FROM radios r
JOIN radio_operators ro ON r.radio_id = ro.radio_id
JOIN operators o ON ro.operator_id = o.operator_id;
```

create a postgresql function to retrieve operators that own a particular radio. Function input parameter should be `radio_id`. Output will be the radio model and brand as well as the operator first and last name and their call sign.





```
CREATE FUNCTION get_radio_owners(radio_id INT)
RETURNS TABLE (
    model TEXT,
    brand TEXT,
    first_name TEXT,
    last_name TEXT,
    callsign TEXT
)
AS $$
BEGIN
    RETURN QUERY
    SELECT r.model,
           r.manufacturer AS brand,
           o.first_name,
           o.last_name,
           o.callsign
    FROM radios r
    JOIN radio_operators ro ON r.radio_id = ro.radio_id
    JOIN operators o ON ro.operator_id = o.operator_id
    WHERE r.radio_id = $1;
END;
$$
LANGUAGE plpgsql;

SELECT * FROM get_radio_owners(7);
```

## Code Evaluation

- Evaluate this PostgreSQL function for compliance with best practices:



Evaluate this PostgreSQL function for compliance with best practices: CREATE FUNCTION

```
get_radio_owners(radio_id INT)
```

```
RETURNS TABLE (
```

```
  model TEXT,
```

```
  brand TEXT,
```

```
  first_name TEXT,
```

```
  last_name TEXT,
```

```
  callsign TEXT
```

```
)
```

```
AS $$
```

```
BEGIN
```

```
  RETURN QUERY
```

```
  SELECT r.model,
```

```
         r.manufacturer AS brand,
```

```
         o.first_name,
```

```
         o.last_name,
```

```
         o.callsign
```

```
  FROM radios r
```

☰ Focus + Attach

Pro





## | Query Optimization

- Using the existing data structures, can the following query be modified to enhance performance?

```
SELECT ar.model, ro2.callsign FROM
public.amateur_radios AS ar JOIN (
SELECT * FROM
public.operator_radio_association
AS ora ) AS ora2 ON ar.radio_id =
ora2.radio_id JOIN ( SELECT * FROM
public.radio_operators AS ro) AS ro2
ON ora2.operator_id IN ( SELECT
ro.operator_id FROM
public.radio_operators AS ro )
WHERE 'General' =
LEFT(ro2.license_class,7);
```



Qu

Using the existing data structures, can the following query be modified to enhance performance?

```
SELECT
  ar.model,
  ro2.callsign
FROM
  public.amateur_radios AS ar
JOIN (
  SELECT
    *
  FROM
    public.operator_radio_association AS ora
) AS ora2 ON
  ar.radio_id = ora2.radio_id
JOIN (
  SELECT
    *
  FROM
    public.radio_operators AS ro) AS ro2
ON
```

Cancel

Save

# | Results

## A-Z QUERY PLAN

Hash Join (cost=39.10..95.88 rows=2040 width=156) (actual time=0.093..0.100 rows=9 loops=1)

## A-Z QUERY PLAN

Unique (cost=29.81..29.93 rows=16 width=156) (actual time=0.083..0.086 rows=2 loops=1)



## Manual Edits

```
SELECT
ar.model,
ro.callsign
FROM
public.amateur_radios AS ar
JOIN
public.operator_radio_association AS ora ON ar.radio_id = ora.radio_id
JOIN
public.radio_operators AS ro ON ora.operator_id = ro.operator_id
WHERE
ro.license_class = 'General';
```

A-Z QUERY PLAN

Nested Loop (cost=4.38..29.49 rows=16 width=156) (actual time=0.055..0.059 rows=2 loops=1)

## And After The Indexes

```
CREATE INDEX idx_radio_id ON  
public.operator_radio_association (radio_id) ;  
CREATE INDEX idx_operator_id ON  
public.operator_radio_association (operator_id) ;  
CREATE INDEX idx_license_class ON  
public.radio_operators (license_class) ;
```

A-Z QUERY PLAN

Nested Loop (cost=1.23..10.57 rows=2 width=156) (actual time=0.037..0.043 rows=2 loops=1)

## | More Stuff We Could Do

- Please save all the scripts to a downloadable file.
- Generate a larger set of sample data
- Create the tests in markdown language





## Final Thoughts



- Focus on your prompts first
- Run the AI models against each other
- Trust but verify
- Stop digging, start over

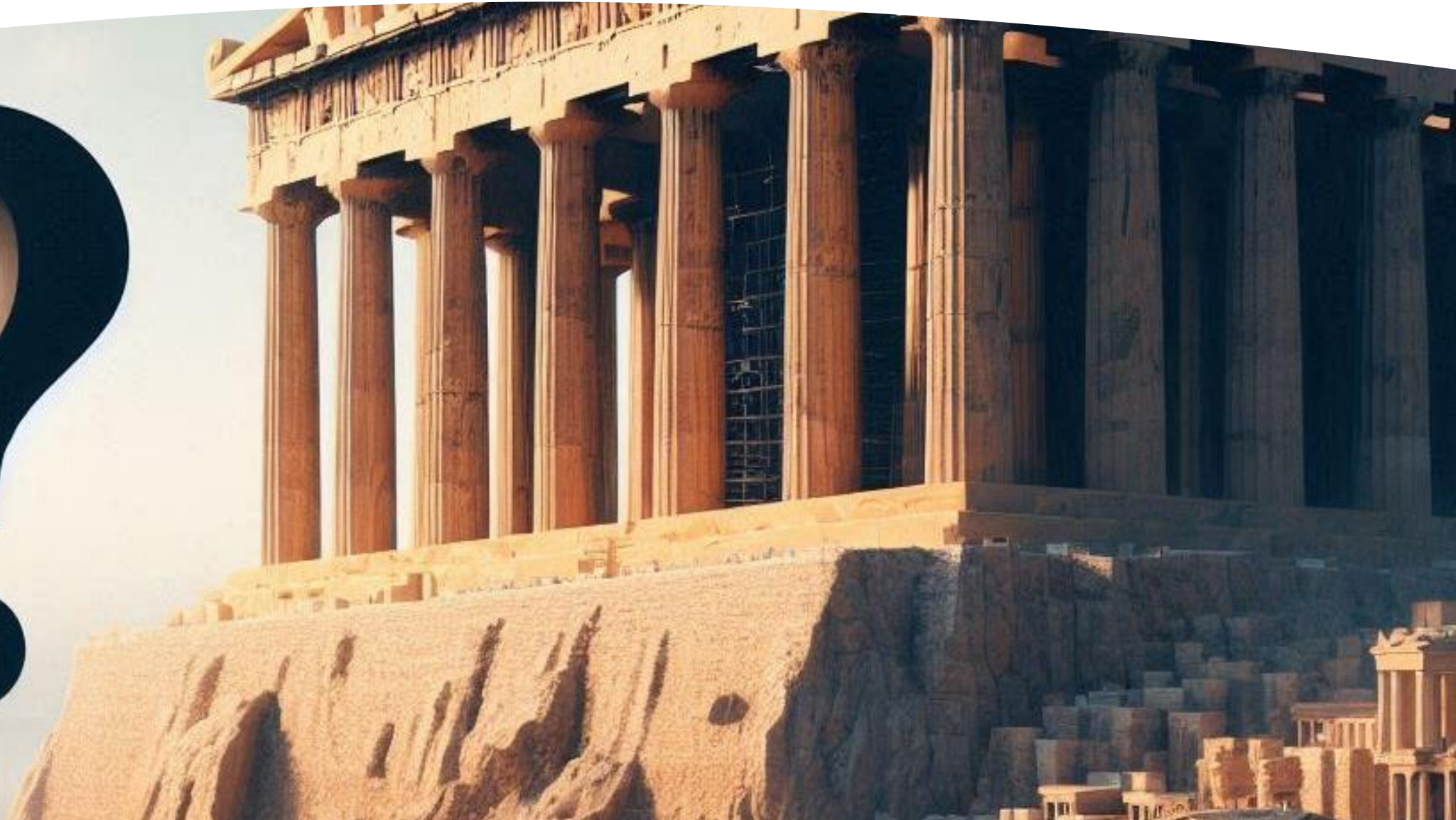
## Session Goals



- Learn mechanisms to ensure good AI results
- Understand how to put AI to work day-to-day
- Discover ways AI helps administer PostgreSQL



| Questions?





| About me

# Grant Fritchey

## DevOps Advocate

Microsoft PostgreSQL MVP  
AWS Community Builder

 @gfritchey

 Grant.Fritchey@Red-Gate.com

 scarydba.com

