PostgreSQL: Beyond The Basics

Lorna Mitchell, IBM
https://speakerdeck.com/lornajane

Meet PostgreSQL

- https://www.postgresql.org/
- Traditional RDBMS
- Open source, strong community
- Powerful, flexible, stable ... need I say more?

Schema

Schema Design

What data to store? How to read/write it?

- Normalising data
- Using appropriate keys
- Picking data types

Data Types

For your everyday needs

- **Text:** `char` or `text`
- **Number:** `int/bigint` or `real`
- **Boolean:** `bool`
- **Date and Time:** `date`, `time` or `timestamptz`

Data Types

PostgreSQL has data types to suit more data needs:

- UUID data type to handle unique identifiers
- JSON and JSONB for working with JSON data

Data Types: UUID

Use a UUID as a primary key: (`CREATE EXTENSION pgcrypto`)

```
CREATE TABLE products (
  product_id uuid primary key default gen_random_uuid(),
  display_name text
);

INSERT INTO products (display_name)
  VALUES ('Jumper') RETURNING product_id;
```

```
              product_id                | display_name
----------------------------------------+--------------
73089ae3-c0a9-4c0a-8287-e0f6ec41a200 | Jumper
```

RETURNING Keyword

Look at that insert statement again

```
INSERT INTO products (display_name)
  VALUES ('Jumper') RETURNING product_id;
```

The `RETURNING` keyword allows us to retrieve a field in one step

Data Types: JSONB

Add a column to the table to hold attributes

```
ALTER TABLE products ADD COLUMN attrs jsonb;
```

Add some data

```
INSERT INTO products (display_name, attrs) VALUES
('Dress', '{"length": {"value": 61, "units":"inch"},
    "pockets":true, "colour":"teal"}');
```

Data Types: JSONB

We can use the JSON in our `WHERE` clause

```
SELECT display_name AS product, attrs->>'colour' AS colour
    FROM products
    WHERE attrs->>'pockets' = 'true';
```

```
 product | colour
---------+--------
 Cardi   | red
 Dress   | teal
 Jeans   | indigo
(3 rows)
```

Indexes

Examples might be:

- Primary key ensuring uniqueness
- Some other unique key
- Indexes facilitating fast lookup on one or more columns
- Indexes that use expressions

Indexes: Primary key

Primary keys are always unique

```
CREATE TABLE employees (
    id serial primary key,
    name text
    );
```

The `serial` data type is numeric and incrementing

Indexes: Expressions

Use an expression if you'll use one when fetching data

```
CREATE TABLE employees (
    id serial primary key,
    name text
    );

CREATE INDEX name_idx
    ON employees (lower(name));
```

Indexes: Composite keys

Indexes can cover multiple columns, e.g. find new hires by department

```
CREATE TABLE employees (
    id serial primary key,
    name text,
    department_id int,
    join_date date);

CREATE INDEX join_dept_idx
    ON employees (department_id, join_date DESC);
```

SQL

Joining a Table to Itself

```
CREATE TABLE employees (
    id serial primary key,
    name text,
    department_id int,
    manager_id int,
    join_date date);
```

```
 id |  name   | department_id | join_date  | manager_id
----+---------+---------------+------------+------------
  1 | Anna    |             3 | 2017-01-03 |
  2 | Beth    |             3 | 2014-01-03 |          1
  3 | Charlie |             2 | 2014-05-03 |          1
```

Joining a Table To Itself

Join a table to itself, using aliases and a left join

```
SELECT e.id AS employee_id,
    e.name AS employee, m.name AS manager
FROM employees e
LEFT JOIN employees m ON (e.manager_id = m.id);
```

```
 employee_id | employee | manager
-------------+----------+---------
           1 | Anna     |
           2 | Beth     | Anna
           3 | Charlie  | Anna
```

Aggregate Functions

... we're going to need a bigger dataset!

The Pagila database is a sample of DVD rental data
https://wiki.postgresql.org/wiki/Sample_Databases

Aggregate Functions

Aggregates: `COUNT()`, `SUM()` etc

`GROUP BY` to define subtotal boundaries

How many films with "DINOSAUR" in the name are in each store?

```
SELECT i.store_id, f.title, count(*) AS total
FROM inventory i INNER JOIN film f ON i.film_id = f.film_id
WHERE f.title LIKE '%DINOSAUR%'
GROUP BY f.title, i.store_id;
```

Aggregate Functions

The `HAVING` keyword - filter by aggregate result

Films where there are more than 7 copies in the inventory?

```
SELECT f.title, count(*) AS total
FROM inventory i
INNER JOIN film f ON i.film_id = f.film_id
GROUP BY f.title
HAVING count(*) > 7;
```

Common Table Expressions

CTE for short

Declare extra statements to re-use later

Syntax:

```
WITH meaningfulname AS
  (subquery goes here joining whatever)
SELECT .... FROM meaningfulname ...
```

Common Table Expressions

```
SELECT a.address_id a.address, ci.city, co.country
FROM address a JOIN city ci USING (city_id)
JOIN country co USING (country_id);
```

Result:

```
addres_id |       address        |    city      |  country
----------+----------------------+--------------+-----------
        1 | 47 MySakila Drive    | Lethbridge   | Canada
        2 | 28 MySQL Boulevard   | Woodridge    | Australia
        3 | 23 Workhaven Lane    | Lethbridge   | Canada
        4 | 1913 Hanoi Way       | Sasebo       | Japan
        5 | 692 Joliet Street    | Athenai      | Greece
        6 | 53 Idfu Parkway      | Nantou       | Taiwan
```

Common Table Expressions

```
WITH addr AS ( SELECT a.address_id a.address, ci.city, co.country
    FROM address a JOIN city ci USING (city_id)
    JOIN country co USING (country_id)
SELECT c.first_name, a.country FROM customer c
    JOIN addr a USING (address_id);
```

Result:

```
 first_name  |                 country
-------------+-----------------------------------
 MARY        | Japan
 PATRICIA    | United States
 LINDA       | Greece
 ELIZABETH   | Taiwan
```

Window Functions

Window functions allow us to calculate aggregate values while still returning the individual rows.

e.g. A list of films plus:

- their running time
- the average running time for this rating of film
- the average for all films

Window Functions

```
SELECT title, rating, length,
    ROUND(AVG(length) OVER (PARTITION BY rating)) AS avg_rating,
    ROUND(AVG(length) OVER ()) AS avg_all
    FROM film ORDER BY title;
```

Result:

```
        title         | rating | length | avg_rating | avg_all
----------------------+--------+--------+------------+---------
 ACADEMY DINOSAUR     | PG     |     86 |        112 |     115
 ACE GOLDFINGER       | G      |     48 |        111 |     115
 AFFAIR PREJUDICE     | G      |    117 |        111 |     115
 AGENT TRUMAN         | PG     |    169 |        112 |     115
 AIRPLANE SIERRA      | PG-13  |     62 |        120 |     115
```

PostgreSQL Tips and Tricks

- Best in-shell help I've ever seen (type `\h [something]`)
- Try adding `NULLS FIRST|LAST` to your `ORDER BY`
- Fabulous support for geographic data http://postgis.net/

See also: https://github.com/dhamaniasad/awesome-postgres

PostgreSQL

Resources

- PostgreSQL: https://www.postgresql.org/
- Get a hosted version from http://ibm.com/cloud

Contact me:

- Twitter: `@lornajane`
- My blog: https://lornajane.net