PostgreSQL
Notes

by Anchal

Database Fundamentals

What is a Database?

A structured system for storing, managing, and retrieving data efficiently.

💾

PostgreSQL Essentials🐘

PostgreSQL Overview

PostgreSQL, or Postgres, is a powerful open-source relational database system known for advanced features and modern design.

🚀

psql Command-Line💻

- `\?` - Help
- `\l` - List databases
- `\psql --help` - `psql` help

Connecting to a Database🔗

Two ways:

1. `psql -h localhost -p 5432 -U username database_name`
2. Within `psql` shell: `\l`, `\c database_name`

Danger Zone☠️

```sql
DROP DATABASE name
```

⚠️ Irreversible! Deletes all data and tables. Never use on production databases.

Creating Tables📄

Table Creation

Use `CREATE TABLE`. Define columns, data types, and constraints.

```sql
CREATE TABLE person (
    id BIGSERIAL NOT NULL PRIMARY KEY,
    first_name VARCHAR(50) NOT NULL,
    last_name VARCHAR(50) NOT NULL,
    gender VARCHAR(6) NOT NULL,
```

```
    date_of_birth DATE NOT NULL
);
```

Describing Tables

`\d` or `\d table_name` to describe tables.

Data Manipulation💼

Inserting Data

Use `INSERT INTO` for data insertion:

```
INSERT INTO person (first_name, last_name, gender, date_of_birth)
VALUES
    (1, 'John', 'Doe', 'MALE', DATE '1990-05-15'), (2, 'Jane', 'Smith', 'FEMALE',
    DATE '1988-03-22'), (3, 'Michael', 'Johnson', 'MALE', DATE '1995-09-10'),
    (4, 'Emily', 'Brown', 'FEMALE', DATE '1992-07-03'), (5, 'David', 'Lee',
    'MALE', DATE '1987-11-28'), (6, 'Olivia', 'Garcia', 'FEMALE', DATE '1998-
    02-19'), (7, 'Daniel', 'Martinez', 'MALE', DATE '1986-12-14'), (8, 'Sophia',
    'Anderson', 'FEMALE', DATE '1991-04-27'), (9, 'Ethan', 'Taylor', 'MALE',
    DATE '1996-06-05'), (10, 'Ava', 'Clark', 'FEMALE', DATE '1989-08-08');
```

Querying Data📝

Selecting Data

Use `SELECT`:

```
SELECT * FROM person;
SELECT first_name, last_name FROM person;
```

Output:

id	first_name	last_name	gender	date_of_birth
1	John	Doe	MALE	1990-05-15
2	Jane	Smith	FEMALE	1988-03-22
3	Michael	Johnson	MALE	1995-09-10
4	Emily	Brown	FEMALE	1992-07-03
5	David	Lee	MALE	1987-11-28
6	Olivia	Garcia	FEMALE	1998-02-19
7	Daniel	Martinez	MALE	1986-12-14

id	first_name	last_name	gender	date_of_birth
8	Sophia	Anderson	FEMALE	1991-04-27
9	Ethan	Taylor	MALE	1996-06-05
10	Ava	Clark	FEMALE	1989-08-08

Sorting Data

- Ascending: `SELECT * FROM person ORDER BY last_name`
- Descending: `SELECT * FROM person ORDER BY last_name DESC`

DATA TYPES

Data Type	Description
`bigint` `integer`	Signed eight-byte integer
`smallint`	Signed four-byte integer Signed two-byte integer
`numeric(p,s)` `real`	Exact numeric with `p` digits, `s` of them Four-byte floating-point
`double precision`	Eight-byte floating-point Exact numeric with `p`
`decimal(p,s)` `text`	digits, `s` of them Variable-length
`char(n)` `varchar(n)`	character string Fixed-length character
`date` `time`	string Variable-length character string
`timestamp`	Date (year, month, day) Time of day (no
`timestamptz`	time zone) Date and time (no time
`interval` `boolean`	zone) Date and time with time zone
`enum` `uuid` `json`	Time interval True/False values
`jsonb` `bytea` `point`	Enumerated (enum) types Universally
`line` `lseg` `box` `path`	unique identifier JSON data type Binary
`polygon`	JSON data type Binary data Point on a
	plane Infinite line on a plane Line
	segment on a plane Rectangular box on
	a plane Closed path on a plane Polygon
	on a plane

Data Type	Description
circle inet cidr	Circle on a plane
macaddr bit(n)	IPv4 or IPv6 host address
bit varying(n)	IPv4 or IPv6 network address
tsvector tsquery	MAC address
hstore oid pg_lsn	Fixed-length bit string
uuid money xml	Variable-length bit string
interval	Text search vector (full-text search)
oidvector	Text search query (full-text search)
	Key-value store
	Object identifier (system use)
	Log sequence number
	Universally unique identifier
	Currency amount
	XML data
	Time interval
	Array of object identifiers

Distinct Values

```
SELECT DISTINCT gender FROM person;
```

Output:

gender
MALE
FEMALE

Filtering Data

```
SELECT * FROM person WHERE gender = 'FEMALE';
```

Output:

id	first_name	last_name	gender	date_of_birth
2	Jane	Smith	FEMALE	1988-03-22
4	Emily	Brown	FEMALE	1992-07-03
6	Olivia	Garcia	FEMALE	1998-02-19
8	Sophia	Anderson	FEMALE	1991-04-27
10	Ava	Clark	FEMALE	1989-08-08

Comparison Operators

Use `<`, `>`, `=`, `<=`, and `>=`.

Limiting Results

```
SELECT * FROM person LIMIT 5;
SELECT * FROM person OFFSET 5 LIMIT 5;
SELECT * FROM person OFFSET 5 FETCH 5 ROW ONLY;
```

Advanced Queries🔍

IN Clause

Filter with `IN`:

```
SELECT * FROM person WHERE last_name IN ('Doe', 'Smith', 'Anderson');
```

Output:

id	first_name	last_name	gender	date_of_birth
1	John	Doe	MALE	1990-05-15
2	Jane	Smith	FEMALE	1988-03-22
8	Sophia	Anderson	FEMALE	1991-04-27

BETWEEN Clause

```
SELECT * FROM person
WHERE date_of_birth BETWEEN DATE '1990-01-01' AND '1995-12-31';
```

Output:

id	first_name	last_name	gender	date_of_birth
1	John	Doe	MALE	1990-05-15
4	Emily	Brown	FEMALE	1992-07-03
8	Sophia	Anderson	FEMALE	1991-04-27

LIKE Operator

The `LIKE` operator allows you to perform pattern matching within strings.

- `_` matches any single character.
- `%p%` matches any string containing 'p'.
- `ILIKE` can be used for case-insensitive matching.

Examples:

- Match names starting with 'J':

```
SELECT * FROM person WHERE first_name LIKE 'J%';
```

Output:

id	first_name	last_name	gender	date_of_birth
1	John	Doe	MALE	1990-05-15
2	Jane	Smith		

| FEMALE | 1988-03-22 |

- Match names containing 'an':

```
SELECT * FROM person WHERE last_name LIKE '%an%';
```

- Case-insensitive match for 'john':

```
SELECT * FROM person WHERE first_name ILIKE 'john';
```

GROUP BY Clause

The `GROUP BY`clause is used to group rows with similar values in specified columns, often used with aggregate functions.

Example:

```
SELECT country_of_birth, COUNT(*) FROM person GROUP BY country_of_birth;
```

Output:

country_of_birth	count
USA	3
Canada	2
UK	2
France	1
Germany	2

GROUP BY HAVING Clause

The `GROUP BY`clause can be combined with `HAVING`to filter grouped results based on aggregate functions.

Example:

```
SELECT country_of_birth, COUNT(*) FROM person GROUP BY country_of_birth HAVING
COUNT(*) > 1;
```

Output:

country_of_birth	count
USA	3 2 2
Canada	2
UK	
Germany	

Useful Aggregate Functions

Sample Car Table🚗

Let's create a sample `car` table with 10 values:

id	make	model	price
1	Ford	Mustang	50000
2	Honda	Civic	25000
3	Ford	Fusion	30000
4	Toyota	Camry	28000
5	Honda	Accord	32000
6	Toyota	Corolla	22000
7	Ford	Focus	27000
8	Honda	Fit	18000
9	Toyota	RAV4	35000
10	Ford	Escape	31000

MAX

To find the maximum value in a column:

```
SELECT MAX(price) FROM car;
```

Output:

```
max
---------
50000.00
(1 row)
```

To find the maximum value per category (e.g., make):

```
SELECT make, MAX(price) FROM car GROUP BY make;
```

Output:

```
    make    |    max
------------|---------
```

```
 Ford | 50000.00  Honda |
32000.00          Toyota     |
35000.00 (3 rows)
```

MIN

To find the minimum value in a column:

```
SELECT MIN(price) FROM car;
```

Output:

```
min
---------
18000.00
(1 row)
```

To find the minimum value per category (e.g., make):

```
SELECT make, MIN(price) FROM car GROUP BY make;
```

Output:

```
     make    |      min
----------|---------
       Ford    |  27000.00   |
Honda        18000.00      |
Toyota  (3   22000.00
rows)
```

AVERAGE

To calculate the average value in a column:

```
SELECT AVG(price) FROM car;
```

Output:

```
avg
--------------------
29500.0000000000
(1 row)
```

To round the average value:

```
SELECT ROUND(AVG(price)) FROM car;
```

Output:

```
 round
-------
29500
(1 row)
```

SUM

To find the total sum of values in a column:

```
SELECT SUM(price) FROM car;
```

Output:

```
sum
---------
295000.00
(1 row)
```

To find the total sum per category (e.g., make):

```
SELECT make, SUM(price) FROM car GROUP BY make;
```

Output:

```
     make    |        sum
----------|---------
        Ford    | 138000.00
Honda        | |     75000.00
Toyota   (3          85000.00
rows)
```

Arithmetic Operators🧮

Arithmetic operators allow you to perform calculations on columns in your queries.

- For applying a percentage discount:

```
SELECT id, make, model, price, price * 0.10 AS discount FROM car;
```

- To round the calculated discount:

```
SELECT id, make, model, price, ROUND(price * 0.10, 2) AS discount FROM car;
```

- Calculating discounted price:

```
SELECT id, make, model, price, (price - (price * 0.10)) AS discounted_price FROM car;
```

Alias 🆕

You can use the `AS` keyword to assign aliases to columns or expressions in your query results.

```
SELECT id, make, model, price, price * 0.10 AS discount FROM car;
```

In this example, `price * 0.10` is given the alias "discount."

COALESCE 🙌

The `COALESCE` function returns the first non-null argument.

- To provide a default value if a column is null:

```
SELECT COALESCE(email, 'email not provided') FROM person;
```

In this example, if the `email` column is null, it returns 'email not provided.'

NULLIF 🚫

The `NULLIF` function compares two expressions and returns null if they are equal; otherwise, it returns the first expression.

- To avoid division by zero:

```
SELECT NULLIF(10 / 0, 10);
```

Here, it returns null to avoid division by zero.

- Combining `COALESCE` and `NULLIF`:

```
SELECT COALESCE(10 / NULLIF(0, 0), 0);
```

Date and Time 📅

You can work with date and time values in PostgreSQL.

- To get the current date and time:

```
SELECT NOW();
```

- To extract specific components from dates:

```
SELECT EXTRACT(YEAR FROM NOW());
SELECT EXTRACT(MONTH FROM NOW());
SELECT EXTRACT(DAY FROM NOW());
SELECT EXTRACT(DOW FROM NOW());  -- Day of the week
SELECT EXTRACT(CENTURY FROM NOW());
```

- To calculate age based on date of birth:

```
SELECT AGE(NOW(), DOB) FROM person;
```

Primary Key 🔑

A primary key is a column or set of columns that uniquely identifies each row in a table. It enforces the uniqueness of values and ensures that the column(s) cannot contain null values.

```
ALTER TABLE person ADD PRIMARY KEY (id);
```

UNIQUE Constraint

A UNIQUE constraint ensures that all values in a column or a group of columns are distinct. It en

forces the uniqueness of values but allows null values.

```
ALTER TABLE person ADD CONSTRAINT unique_email UNIQUE (email);
```

Foreign Key 🌐

A foreign key is a column or a set of columns in a table that is used to establish and enforce a link between the data in two tables. It creates a relationship between the tables.

```
ALTER TABLE orders
ADD CONSTRAINT fk_customer_id
FOREIGN KEY (customer_id)
REFERENCES customers (id);
```

Indexing 📚

Indexes are database objects used to speed up data retrieval. They are created on columns to quickly locate and access data rows.

```
CREATE INDEX idx_last_name ON person (last_name);
```

Subqueries 🔄

A subquery is a query nested within another query. It can be used to retrieve data needed for the main query or for filtering results.

Example: Find all customers who have placed an order:

```
SELECT * FROM customers
WHERE id IN (SELECT DISTINCT customer_id FROM orders);
```

Joins 🔗

Joins are used to combine rows from two or more tables based on related columns between them.

INNER JOIN

An `INNER JOIN` returns only the rows that have matching values in both tables.

```
SELECT customers.name, orders.order_date
FROM customers
INNER JOIN orders ON customers.id = orders.customer_id;
```

LEFT JOIN (or LEFT OUTER JOIN)

A `LEFT JOIN` returns all rows from the left table and the matched rows from the right table. If there is no match, it returns NULL values for right table columns.

```
SELECT customers.name, orders.order_date
FROM customers
LEFT JOIN orders ON customers.id = orders.customer_id;
```

RIGHT JOIN (or RIGHT OUTER JOIN)

A `RIGHT JOIN` returns all rows from the right table and the matched rows from the left table. If there is no match, it returns NULL values for left table columns.

```
SELECT customers.name, orders.order_date
FROM customers
RIGHT JOIN orders ON customers.id = orders.customer_id;
```

FULL OUTER JOIN

A `FULL OUTER JOIN` returns all rows when there is a match in either the left or the right table. If there is no match, it returns NULL values for columns from the table without a match.

```
SELECT customers.name, orders.order_date
FROM customers
FULL OUTER JOIN orders ON customers.id = orders.customer_id;
```

Joins and Relationships🤝🔗

Creating Tables and Adding Relationship📊

```
CREATE TABLE people (
    id SERIAL PRIMARY KEY,
    first_name VARCHAR(50),
    last_name VARCHAR(50),
    gender VARCHAR(10),
    date_of_birth DATE
```

```
);

CREATE TABLE cars (
    id SERIAL PRIMARY KEY,
    make VARCHAR(50),
    model VARCHAR(50),
    price DECIMAL(10, 2),
    owner_id INT,
    FOREIGN KEY (owner_id) REFERENCES people(id),
    UNIQUE (owner_id, id)
);
```

Sample Data:

```
INSERT INTO people (first_name, last_name, gender, date_of_birth) VALUES
    ('Alice', 'Johnson', 'Female', '1990-05-15'),
    ('Bob', 'Smith', 'Male', '1985-02-10');

INSERT INTO cars (make, model, price, owner_id) VALUES
    ('Toyota', 'Camry', 25000.00, 1),
    ('Honda', 'Civic', 22000.00, 2),
    ('Ford', 'Mustang', 45000.00, 1);
```

Querying Data:

```
SELECT people.first_name, people.last_name, cars.make, cars.model
FROM people
JOIN cars ON people.id = cars.owner_id;
```

Output:

first_name	last_name	make	model
Alice	Johnson	Toyota	Camry
Bob	Smith	Honda	Civic
Alice	Johnson	Ford	Mustang

Handling Constraints and Operations🚧

DELETE Data🗑️

The DELETE statement removes rows:

```
DELETE FROM person WHERE id = 1;
```

UPDATE Data🔄

The UPDATE statement modifies data:

```
UPDATE person SET gender = 'Other' WHERE gender = 'Unknown';
```

INSERT Data 📥

INSERT adds new rows:

```
INSERT INTO person (name, age) VALUES ('John', 30);
```

Upsert and Excluded🔄📥

An upsert operation combines INSERT and UPDATE:

```
INSERT INTO person (id, name) VALUES (1, 'Alice')
    ON CONFLICT (id) DO UPDATE SET name = EXCLUDED.name;
```

Exporting to CSV📁

To export data to a CSV file:

```
\COPY (SELECT * FROM people) TO 'people.csv' WITH CSV HEADER;
```

Additional PostgreSQL Features🧩

- Serial Sequences 🔄
- PostgreSQL Extensions 🧩
- PL/v8 🧬
- UUIDs as Primary Keys 🆔