PostgreSQL Cheat Sheet for Clinic Appointment Management System

This cheat sheet provides a quick reference for common PostgreSQL commands and concepts relevant to your database project.

1. Database Design and Table Creation

Creating a Database

```
CREATE DATABASE clinic_db;
```

Creating Tables

```
CREATE TABLE table_name (
    column1_name data_type PRIMARY KEY,
    column2_name data_type NOT NULL UNIQUE,
    column3_name data_type CHECK (condition),
    column4_name data_type,
    -- ...
    FOREIGN KEY (column_name) REFERENCES other_table(other_column)
);

-- Example: Patients Table
CREATE TABLE Patients (
    patient_id SERIAL PRIMARY KEY, -- SERIAL for auto-incrementing integer primary key
    first_name VARCHAR(50) NOT NULL,
    last_name VARCHAR(50) NOT NULL,
    date_of_birth DATE,
    phone_number VARCHAR(15) UNIQUE,
    email VARCHAR(100) UNIQUE CHECK (email LIKE '%@%.%')
);

-- Example: Doctors Table
CREATE TABLE Doctors (
    doctor_id SERIAL PRIMARY KEY,
    first_name VARCHAR(50) NOT NULL,
    last_name VARCHAR(50) NOT NULL,
    specialty_id INT, -- Will be a foreign key to Specialties table
```

```
    phone_number VARCHAR(15) UNIQUE,
    email VARCHAR(100) UNIQUE
);

-- Example: Foreign Key Relationship
-- After creating Specialties table:
CREATE TABLE Appointments (
    appointment_id SERIAL PRIMARY KEY,
    patient_id INT NOT NULL,
    doctor_id INT NOT NULL,
    appointment_date DATE NOT NULL,
    appointment_time TIME NOT NULL,
    status VARCHAR(20) DEFAULT 'Scheduled',
    FOREIGN KEY (patient_id) REFERENCES Patients(patient_id),
    FOREIGN KEY (doctor_id) REFERENCES Doctors(doctor_id)
);
```

Data Types (Common Examples)

- SERIAL: Auto-incrementing integer (for primary keys)
- INT / INTEGER: Whole numbers
- VARCHAR(n): Variable-length string, max n characters
- TEXT: Variable-length string, no limit
- DATE: Date (YYYY-MM-DD)
- TIME: Time (HH:MI:SS)
- TIMESTAMP: Date and time
- BOOLEAN: True/False
- DECIMAL(p, s) / NUMERIC(p, s): Exact numeric, p total digits, s digits after decimal
- MONEY: Currency amount

Constraints

- PRIMARY KEY: Unique identifier for each row, cannot be NULL.
- FOREIGN KEY: Establishes a link between two tables.
- NOT NULL: Ensures a column cannot have a NULL value.
- UNIQUE: Ensures all values in a column are different.
- CHECK (condition): Ensures all values in a column satisfy a specific condition.
- DEFAULT value: Sets a default value for a column if none is specified.

2. Data Operations

Inserting Data

```
INSERT INTO table_name (column1, column2, ...)
VALUES (value1, value2, ...);

-- Example:
INSERT INTO Patients (first_name, last_name, date_of_birth, phone_number, email)
VALUES ('John', 'Doe', '1985-03-15', '123-456-7890', 'john.doe@example.com');
```

Updating Data

```
UPDATE table_name
SET column1 = new_value1, column2 = new_value2, ...
WHERE condition;

-- Example:
UPDATE Patients
SET phone_number = '987-654-3210'
WHERE patient_id = 1;
```

Deleting Data

```
DELETE FROM table_name
WHERE condition;

-- Example:
DELETE FROM Appointments
WHERE appointment_id = 5;
```

Transaction Control (COMMIT and ROLLBACK)

```
BEGIN; -- or START TRANSACTION;

-- Perform multiple SQL statements
INSERT INTO Invoices (invoice_id, patient_id, amount) VALUES (101, 1, 150.00);
UPDATE Patients SET balance = balance + 150.00 WHERE patient_id = 1;

-- If all operations are successful:
COMMIT;
```

-- If an error occurs or you want to undo:
ROLLBACK;

3. Data Retrieval and Queries

Basic Select

SELECT column1, column2 FROM table_name;
SELECT * FROM table_name; -- Select all columns

-- Example:
SELECT first_name, last_name FROM Patients;
SELECT * FROM Doctors;

Filtering Data (WHERE)

SELECT column1, column2 FROM table_name
WHERE condition;

-- Operators: =, <>, <, >, <=, >=, LIKE, ILIKE (case-insensitive LIKE), IN, BETWEEN, IS NULL, IS NOT NULL, AND, OR, NOT

-- Example:
SELECT * FROM Appointments
WHERE appointment_date = '2025-07-08';

SELECT * FROM Patients
WHERE date_of_birth > '2000-01-01';

SELECT * FROM Doctors
WHERE specialty_id IN (1, 3);

Ordering Results (ORDER BY)

SELECT column1, column2 FROM table_name
ORDER BY column_name ASC|DESC;

-- Example:
SELECT * FROM Doctors

```
ORDER BY last_name ASC;

SELECT * FROM Appointments
ORDER BY appointment_date DESC, appointment_time ASC;
```

Limiting Results (`LIMIT` and `OFFSET`)

```
SELECT * FROM table_name
LIMIT number_of_rows OFFSET starting_row;

-- Example:
SELECT * FROM Patients
LIMIT 10 OFFSET 0; -- First 10 patients
```

Aggregate Functions

- `COUNT(column_name)`: Counts rows. `COUNT(*)` counts all rows.
- `SUM(column_name)`: Calculates the sum of a numeric column.
- `AVG(column_name)`: Calculates the average of a numeric column.
- MIN(column_name): Finds the minimum value.
- MAX(column_name): Finds the maximum value.

```
SELECT COUNT(*) FROM Patients;
SELECT AVG(amount) FROM Invoices;
```

Grouping Data (GROUP BY and HAVING)

```
SELECT column1, aggregate_function(column2)
FROM table_name
GROUP BY column1
HAVING condition_on_aggregate_function; -- Filter groups

-- Example: Count appointments per doctor
SELECT doctor_id, COUNT(appointment_id) AS total_appointments
FROM Appointments
GROUP BY doctor_id;

-- Example: Patients with more than 3 appointments
SELECT patient_id, COUNT(appointment_id) AS num_appointments
FROM Appointments
GROUP BY patient_id
```

```
HAVING COUNT(appointment_id) > 3;
```

Joining Tables (JOIN)

- INNER JOIN: Returns rows when there is a match in both tables.
- LEFT JOIN (or LEFT OUTER JOIN): Returns all rows from the left table, and the matched rows from the right table. If no match, NULLs from the right table.
- RIGHT JOIN (or RIGHT OUTER JOIN): Returns all rows from the right table, and the matched rows from the left table. If no match, NULLs from the left table.
- FULL JOIN (or FULL OUTER JOIN): Returns all rows when there is a match in one of the tables.

```
SELECT P.first_name, P.last_name, D.first_name, D.last_name, A.appointment_date
FROM Patients AS P
INNER JOIN Appointments AS A ON P.patient_id = A.patient_id
INNER JOIN Doctors AS D ON A.doctor_id = D.doctor_id;

-- Example: Doctors with no appointments (using LEFT JOIN)
SELECT D.first_name, D.last_name
FROM Doctors AS D
LEFT JOIN Appointments AS A ON D.doctor_id = A.doctor_id
WHERE A.appointment_id IS NULL;
```

Subqueries

A query nested inside another query.

```
SELECT first_name, last_name
FROM Patients
WHERE patient_id IN (SELECT patient_id FROM Appointments WHERE
appointment_date = CURRENT_DATE);

-- Correlated Subquery Example (less common for simple cases, but powerful)
SELECT D.first_name, D.last_name
FROM Doctors D
WHERE NOT EXISTS (SELECT 1 FROM Appointments A WHERE A.doctor_id =
D.doctor_id AND A.appointment_date = CURRENT_DATE);
```

Common Functions

- **String Functions:**
  - UPPER(string): Converts to uppercase.
  - LOWER(string): Converts to lowercase.
  - CONCAT(string1, string2, ...): Concatenates strings.
  - LENGTH(string): Returns length of string.
  - SUBSTRING(string, start, length): Extracts a substring.
- **Date/Time Functions:**
  - CURRENT_DATE: Current date.
  - CURRENT_TIME: Current time.
  - NOW(): Current timestamp.
  - AGE(timestamp): Calculates age from a timestamp (e.g., AGE(date_of_birth)).
  - TO_CHAR(timestamp, format): Formats date/time.
    - TO_CHAR(appointment_date, 'YYYY-MM-DD HH24:MI')
    - TO_CHAR(appointment_date, 'Month DD, YYYY')
  - EXTRACT(part FROM source): Extracts parts of date/time (e.g., EXTRACT(YEAR FROM date_of_birth)).
- **Numeric Functions:**
  - ROUND(number, decimal_places)
  - CEIL(number) / FLOOR(number)

```
-- Example: Display patient names in uppercase
SELECT UPPER(first_name) AS upper_first_name, last_name FROM Patients;

-- Example: Calculate patient age (approximate)
SELECT first_name, last_name, AGE(date_of_birth) AS age FROM Patients;
-- For a more precise age in years:
SELECT first_name, last_name, EXTRACT(YEAR FROM AGE(date_of_birth)) AS
age_in_years FROM Patients;

-- Example: Format appointment date
SELECT TO_CHAR(appointment_date, 'Day, Month DD, YYYY') AS formatted_date
FROM Appointments;
```

4. Data Management Features

Views

A virtual table based on the result-set of a SQL query.

```
CREATE VIEW view_name AS
SELECT column1, column2
FROM table_name
WHERE condition;

-- Example: Today's Appointments View
CREATE VIEW TodayAppointments AS
SELECT
    A.appointment_id,
    P.first_name AS patient_first_name,
    P.last_name AS patient_last_name,
    D.first_name AS doctor_first_name,
    D.last_name AS doctor_last_name,
    A.appointment_time
FROM Appointments AS A
JOIN Patients AS P ON A.patient_id = P.patient_id
JOIN Doctors AS D ON A.doctor_id = D.doctor_id
WHERE A.appointment_date = CURRENT_DATE;

-- To use the view:
SELECT * FROM TodayAppointments;
```

Sequences

Used to generate unique numbers automatically.

```
CREATE SEQUENCE sequence_name
START WITH 1
INCREMENT BY 1
MINVALUE 1
NO MAXVALUE
CACHE 1;

-- Example: Invoice Number Sequence
CREATE SEQUENCE invoice_number_seq
START WITH 1000
INCREMENT BY 1;

-- To use the sequence (e.g., when inserting):
```

```
INSERT INTO Invoices (invoice_id, patient_id, amount)
VALUES (NEXTVAL('invoice_number_seq'), 1, 250.00);
```

Note: SERIAL and BIGSERIAL data types automatically create and use sequences for primary keys, so you might not need to manually create sequences for IDs.

Indexes

Improve the speed of data retrieval operations on a database table.

```
CREATE INDEX index_name ON table_name (column1, column2, ...);

-- Example: Index on Doctor Name
CREATE INDEX idx_doctor_name ON Doctors (last_name, first_name);

-- Example: Index on Patient Number (assuming patient_id is the "patient number")
CREATE INDEX idx_patient_id ON Patients (patient_id);
```

Other Useful Commands

Show Tables

```
\dt -- In psql terminal
```

Describe Table

```
\d table_name -- In psql terminal
```

Drop Table

```
DROP TABLE table_name;
```

Drop View

```
DROP VIEW view_name;
```

Drop Sequence

```
DROP SEQUENCE sequence_name;
```

Drop Index

DROP INDEX index_name;

This cheat sheet should provide you with a solid foundation for tackling your project's requirements in PostgreSQL. Remember to test your queries thoroughly!