PostgreSQL

Advanced Postgres

Surendra Panpaliya

Agenda

- Data Integrity and Constraints
- Primary and foreign keys
- Unique and check constraints
- Default values and nullability

Agenda

- User Management and Security
- Creating and managing roles
- Granting and revoking permissions
- Best practices for database security


SSL
Data Integrity

Data Integrity

Refers to

Accuracy

Consistency

of Data stored

in a Database

Data Integrity

MAINTAINED THROUGH

VARIOUS MECHANISMS

CONSTRAINTS

Data Integrity

Ensure

Data entered

into Database

Correct

Consistent

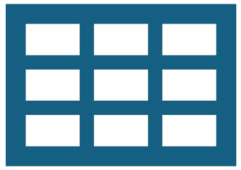
Reliable

Types of Data Integrity

Surendra Panpaliya

Entity Integrity

Ensures that



each row in a table

is uniquely identifiable.

Entity Integrity

Achieved using

Primary Keys

which are unique identifiers

for table records

Referential Integrity

Ensures that

relationships between tables

remain consistent.

Referential Integrity

Enforced through

Foreign Keys

Ensure that

a value in one table

Matches a value in another table.

Domain Integrity

Ensures that

data entered into a column

is of the correct type,

format, and range.

Domain Integrity

Achieved through

Data Types,

Check Constraints,

Unique Constraints

Default Values.

User-Defined Integrity

Specific business rules

that do not fit

into the other categories.

User-Defined Integrity

Enforced through

Triggers and

Stored Procedures.

Constraints

Surendra Panpaliya

Constraints

Rules

enforced

on data columns

on a table

Constraints

Used to ensure

Accuracy

Reliability

of the data in

the database.

Constraints

Critical

To Maintain

Consistent

Valid

Accurate data

Primary and foreign keys

Surendra Panpaliya

Primary and foreign keys

Essential for

Maintaining

Data integrity

Establishing

relationships between

different entities

Primary Key Example

```
-- Creating a table for patients with a primary key

CREATE TABLE patients (
    patient_id SERIAL PRIMARY KEY,
    first_name VARCHAR(50),
    last_name VARCHAR(50),
    dob DATE,
    gender CHAR(1),
    -- other columns
);
```

Foreign Key Examples

Example 1: One-to-Many Relationship

```
-- Creating a table for appointments with a foreign key referencing patients

CREATE TABLE appointments (
    appointment_id SERIAL PRIMARY KEY,
    patient_id INT REFERENCES patients(patient_id),
    appointment_date DATE,
    doctor_id INT,
    -- other columns
);
```

Foreign Key Examples

Example 2: Many-to-Many Relationship

```
-- Creating a table for medications prescribed to patients
CREATE TABLE medications (
    medication_id SERIAL PRIMARY KEY,
    name VARCHAR(100),
    dosage VARCHAR(50),
    -- other columns
);
```

Foreign Key Examples

```
-- Creating a table to track medications prescribed to patients
CREATE TABLE patient_medications (
    prescription_id SERIAL PRIMARY KEY,
    patient_id INT REFERENCES patients(patient_id),
    medication_id INT REFERENCES medications(medication_id),
    prescription_date DATE,
    -- other columns
);
```

Foreign Key Data Type

Serial Data Type

Used for

auto-incrementing

integer values,

commonly used

for primary keys.

Foreign Key Data Type

References

Declares a foreign key constraint

References

Ensuring that values in

Referencing column

appointments.patient_id,

patient_medications.patient_id,

patient_medications.medication_id

References

Exist in	
Referenced column	
patients.patient_id,	
medications.medication_id	

Unique and check constraints

Play a crucial role in

Ensuring data accuracy

Enforcing specific

Rules or Conditions

Unique Constraint on Patient Identifier

Create a table

for patients with

unique constraint

on a patient identifier

Unique Constraint on Patient Identifier

```
CREATE TABLE patients (
    patient_id SERIAL PRIMARY KEY,
    patient_identifier VARCHAR(50) UNIQUE,
    first_name VARCHAR(50),
    last_name VARCHAR(50),
    dob DATE,
    gender CHAR(1),
    );
```

2. Check Constraint on Vital Signs

Creating a table

for recording

vital signs with

Check constraint

2. Check Constraint on Vital Signs

```
CREATE TABLE vital_signs (
    vital_signs_id SERIAL PRIMARY KEY,
    patient_id INT REFERENCES patients(patient_id),
    measurement_date DATE,
```

Check Constraint Examples

temperature NUMERIC,

systolic_bp INT,

diastolic_bp INT,

pulse_rate INT,

Check Constraint Examples

```
CONSTRAINT valid_temperature

CHECK (temperature >= 34 AND

temperature <= 42),

-- Valid temperature range in Celsius
```

Check Constraint Examples

```
CONSTRAINT valid_bp CHECK (systolic_bp >= 70 AND
systolic_bp <= 250 AND diastolic_bp >= 40 AND
diastolic_bp <= 150),
 -- Valid blood pressure range
```

Check Constraint Examples

```
CONSTRAINT valid_pulse_rate CHECK (pulse_rate >= 40

AND pulse_rate <= 200)

);

 -- Valid pulse rate range
```

Check Constraint Examples

CHECK CONSTRAINTS

VALID_TEMPERATURE

VALID_BP

VALID_PULSE_RATE

Check Constraint

Ensure

Recorded vital signs

Within

Medically

Acceptable ranges

Check Constraint

Constraints help

Maintain Data integrity

by preventing

Insertion of

Unrealistic or

Erroneous values

Default values and nullability

CONSTRAINTS ARE IMPORTANT

FOR MANAGING

DATA CONSISTENCY

ENSURING

COMPLETENESS OF INFORMATION

1. Default Value for Admission Status

```
CREATE TABLE admissions (
    admission_id SERIAL PRIMARY KEY,
    patient_id INT REFERENCES patients(patient_id),
    admission_date DATE,
    discharge_date DATE,
    admission_status VARCHAR(50) DEFAULT 'Admitted',
    -- other columns
);
```

2. Default Value for Prescription Frequency

```
CREATE TABLE medications (
    medication_id SERIAL PRIMARY KEY,
    name VARCHAR(100),
    dosage VARCHAR(50),
    frequency VARCHAR(50) DEFAULT 'As needed',
);
```

Nullability Constraints Examples

```
CREATE TABLE patients (
    patient_id SERIAL PRIMARY KEY,
    first_name VARCHAR(50),
    last_name VARCHAR(50),
    dob DATE,
    gender CHAR(1),
    allergies VARCHAR(200) NULL,  -- Nullable column for allergies
);
```

Non-Nullable Column for Diagnosis

```
CREATE TABLE diagnoses (
    diagnosis_id SERIAL PRIMARY KEY,
    patient_id INT REFERENCES patients(patient_id),
    diagnosis_date DATE,
    diagnosis_details TEXT NOT NULL,
    -- other columns
);
```

User Management and Security

Surendra Panpaliya

User Management and Security

CRITICAL TO PROTECT

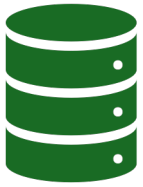
SENSITIVE PATIENT INFORMATION

MAINTAIN

REGULATORY COMPLIANCE

User Management Examples

Create



New database user

for healthcare

application

User Management Examples

CREATE USER healthcare_app_user

WITH PASSWORD 'secure_password’;

Grant Privileges

Grant necessary

privileges

to the healthcare

application user

Grant Privileges

GRANT SELECT, INSERT, UPDATE, DELETE

ON ALL TABLES IN SCHEMA

public TO healthcare_app_user;

Revoking Privileges

Revoke

unnecessary privileges

from the healthcare

application user

Revoking Privileges

```
REVOKE DELETE ON sensitive_table
FROM healthcare_app_user;
```

Security Best Practices Examples

Surendra Panpaliya

Encrypting Data at Rest

EXAMPLE OF ENCRYPTING SENSITIVE COLUMNS

Encrypting Data at Rest

```
CREATE TABLE patient_records (

record_id SERIAL PRIMARY KEY,

patient_id INT REFERENCES patients(patient_id),

medical_history BYTEA ENCRYPTED,

other columns

);
```

Role-Based Access Control (RBAC)

Create roles for different levels of access

CREATE ROLE doctor;

CREATE ROLE nurse;

Role-Based Access Control (RBAC)

GRANT SELECT, INSERT ON appointments TO doctor;

GRANT SELECT, INSERT, UPDATE, DELETE ON patient_records TO nurse;

Auditing and Logging

Enable logging for database activities

ALTER DATABASE my_healthcare_db

SET log_statement = 'all';

Creating and Managing roles

MANAGING ROLES

INVOLVES CREATING ROLES

WITH SPECIFIC PRIVILEGES

Creating and managing roles

Ensure

Data security

Access control

Creating Roles

Create roles for different healthcare personnel

```
CREATE ROLE doctor;

CREATE ROLE nurse;
```

Create roles for different healthcare personnel

Doctor, Nurse

Admin roles are created

to differentiate between

different types of

healthcare personnel

Creating Roles

CREATE ROLE admin

WITH LOGIN PASSWORD

'secure_password’;

Creating Roles

Admin role

Created with

Login password

for administrative access

Granting and revoking permissions

Surendra Panpaliya

Granting Privileges to Roles

Grant necessary privileges to roles

GRANT SELECT, INSERT, UPDATE, DELETE ON appointments TO doctor;

GRANT SELECT, INSERT, UPDATE, DELETE ON patient_records TO nurse;

Granting Privileges to Roles

Grant necessary privileges to roles

GRANT ALL PRIVILEGES ON patient_data TO admin;

Revoking Privileges

Revoke unnecessary privileges from roles

REVOKE DELETE ON sensitive_table FROM nurse;

Assign roles to database users

```
GRANT doctor TO healthcare_doctor_user;
GRANT nurse TO healthcare_nurse_user;
GRANT admin TO healthcare_admin_user;
```

Best practices for database security

Surendra Panpaliya

Best practices for database security

- Database security
- Paramount
- To protect
- Sensitive
- Patient information

Use Strong Authentication and Authorization

Ensure strong password policies and use encrypted connections (SSL/TLS):

CREATE USER healthcare_user WITH ENCRYPTED PASSWORD 'secure_password';

ALTER SYSTEM SET ssl = 'on';

Implement Role-Based Access Control (RBAC)

Example: Creating Roles and Granting Privileges

Create roles for different user types and grant them appropriate permissions:

Create roles for healthcare personnel

CREATE ROLE doctor;

CREATE ROLE nurse;

Grant permissions to roles

GRANT SELECT, INSERT, UPDATE ON appointments TO doctor;

GRANT SELECT, INSERT, UPDATE, DELETE ON patient_records TO nurse;

Encrypt Sensitive Data

Example: Encrypting Columns

Encrypt
sensitive data
such as
medical records
to protect confidentiality

Encrypt Sensitive Data

CREATE A TABLE WITH ENCRYPTED COLUMNS FOR PATIENT RECORDS

Encrypt Sensitive Data

```
CREATE TABLE patient_records (
record_id SERIAL PRIMARY KEY,
patient_id INT REFERENCES patients(patient_id),
medical_history BYTEA ENCRYPTED,
other columns
);
```

Regularly Update and Patch PostgreSQL

Keep PostgreSQL

Operating system

Up to date

With Security Patches

Regularly Update and Patch PostgreSQL

```
# Update PostgreSQL and OS
sudo apt-get update
sudo apt-get upgrade postgresql
```

Enable Auditing and Logging

Enable logging

To Monitor

Audit

Database activities

Enable Auditing and Logging

```
-- Configure PostgreSQL to log all statements

ALTER SYSTEM SET log_statement = 'all';
```

Implement Network Security Measures

Example: Firewall Rules

Restrict database access

through firewall rules

limit network exposure

Implement Network Security Measures

Example firewall rule to allow access from specific IP addresses

sudo ufw allow from 192.168.1.0/24 to any port 5432

Backup and Disaster Recovery

Regularly back up databases and implement disaster recovery plans:

Example: Backup using pg_dump

pg_dump -U postgres -d my_healthcare_db > backup.sql

Monitor and Detect Anomalies

Implement monitoring tools

to detect and respond

to security incidents

Example: Create a trigger

to log suspicious activities

Create a trigger to log suspicious activities

```
CREATE OR
REPLACE FUNCTION log_suspicious_activity()
RETURNS TRIGGER AS $$
```

Create a trigger to log suspicious activities

```
BEGIN
INSERT INTO security_logs (event_time, event_description)
VALUES (now(), 'Suspicious activity detected');
RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

Create a trigger to log suspicious activities

```
CREATE TRIGGER suspicious_activity_trigger
AFTER INSERT OR UPDATE OR
DELETE ON sensitive_table
FOR EACH ROW
EXECUTE FUNCTION
log_suspicious_activity();
```

What is a Trigger?

- Special kind of stored procedure
- Automatically
- executes or fires
- when certain events occur
- in a table or view

What is a Trigger?

Events can be

INSERT,

UPDATE

DELETE

operations

Trigger Uses

To enforce complex business rules

Maintain audit trails

Automatically update derived data

Ensure data integrity

Key Concepts of Triggers

Surendra Panpaliya

Trigger Function

The function that contains

the code to be executed

when the trigger fires.

Usually written in PL/pgSQL

Trigger Event

The event

that causes

the trigger to fire.

Common events are

INSERT, UPDATE, and DELETE.

Trigger Timing

Specifies whether

Trigger fires

Before or

After the event.

Can be BEFORE or AFTER.

Trigger Scope

Defines whether

the trigger fires

once per row

FOR EACH ROW

once per statement

FOR EACH STATEMENT

Educate and Train Personnel

TRAIN STAFF ON

SECURITY BEST PRACTICES

PROTOCOLS

TO PREVENT BREACHES

Educate and Train Personnel

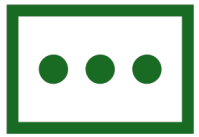
Regularly conduct

security training sessions

for healthcare staff

Educate and Train Personnel

Emphasize the importance of



Strong passwords

Data encryption

Access control

Adhere to Regulatory Compliance

ENSURE

ALL SECURITY MEASURES

ALIGN WITH

HEALTHCARE REGULATIONS

HIPAA, GDPR

Adhere to Regulatory Compliance

Regularly review

update security policies

to comply with

regulatory requirements

Adhere to Regulatory Compliance

Conduct

regular security audits

Assessments

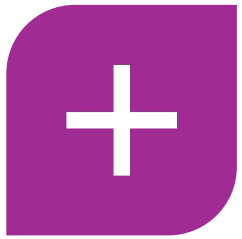
to identify vulnerabilities.

Summary and Conclusion

IMPLEMENTING

THESE BEST PRACTICES

FOR DATABASE SECURITY



HELPS

HEALTHCARE ORGANIZATIONS

Summary and Conclusion

Protect patient data,

Maintain confidentiality

Ensure compliance with

Regulatory standards

Surendra Panpaliya

Founder and CEO

GKTCS Innovations

https://www.gktcs.com