Database Configuration Files

1. **pom.xml (Maven Configuration)**

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
  XML declaration specifying version 1.0 and UTF-8 encoding
  This is the standard XML header for Maven project files
-->
<project xmlns="http://maven.apache.org/POM/4.0.0"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
      https://maven.apache.org/xsd/maven-4.0.0.xsd">
  <!--
    Project root element with Maven namespace declarations
    xmlns="http://maven.apache.org/POM/4.0.0" - Maven POM namespace
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" - XML Schema instance namespace
    xsi:schemaLocation - Location of XML Schema for validation
  -->

  <modelVersion>4.0.0</modelVersion>
  <!--
    Maven POM model version
    This specifies the version of the POM format being used
  -->

  <parent>
    <groupId>org.springframework.boot</groupId>
    <!--
```

```
    Spring Boot parent group ID
    This identifies the Spring Boot organization
  -->
  <artifactId>spring-boot-starter-parent</artifactId>
  <!--
    Spring Boot starter parent artifact ID
    This provides default configuration and dependency management
  -->
  <version>3.2.5</version>
  <!--
    Spring Boot version
    This specifies the version of Spring Boot to use
  -->
  <relativePath/>
  <!--
    Relative path to parent POM
    Empty means parent is in the repository
  -->
</parent>

<groupId>com.example</groupId>
<!--
  Project group ID
  This identifies the organization or group that owns the project
  Follows reverse domain naming convention
-->
<artifactId>pams</artifactId>
<!--
```

```
    Project artifact ID
    This is the unique identifier for the project
    PAMS = Patient Appointment Management System
-->
<version>0.0.1-SNAPSHOT</version>
<!--
    Project version
    SNAPSHOT indicates this is a development version
-->
<name>pams</name>
<!--
    Project name
    This is the human-readable name of the project
-->
<description>Patient Appointment Management System</description>
<!--
    Project description
    This provides a brief description of what the project does
-->


<properties>
    <!--
        Maven properties section
        This defines custom properties that can be used throughout the POM
    -->
    <java.version>17</java.version>
    <!--
        Java version property
```

```
    This specifies the Java version to use for compilation and runtime
    Java 17 is a Long Term Support (LTS) version
  -->
</properties>

<dependencies>
  <!--
    Dependencies section
    This lists all the libraries and frameworks the project depends on
  -->

  <!-- Spring Boot Web Starter -->
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <!--
      Spring Boot organization group ID
      This identifies the Spring Boot organization
    -->
    <artifactId>spring-boot-starter-web</artifactId>
    <!--
      Spring Boot web starter artifact ID
      This includes all dependencies needed for web applications:
      - Spring MVC for web controllers
      - Embedded Tomcat server
      - Jackson for JSON processing
      - Spring Boot auto-configuration
    -->
  </dependency>
```

```
<!-- Spring Boot Thymeleaf Starter -->
<dependency>
  <groupId>org.springframework.boot</groupId>
  <!--
    Spring Boot organization group ID
    This identifies the Spring Boot organization
  -->
  <artifactId>spring-boot-starter-thymeleaf</artifactId>
  <!--
    Spring Boot Thymeleaf starter artifact ID
    This includes all dependencies needed for Thymeleaf templating:
    - Thymeleaf template engine
    - Thymeleaf Spring integration
    - Thymeleaf Spring Security integration
    - Thymeleaf Spring Data integration
  -->
</dependency>

<!-- Spring Boot Data JPA Starter -->
<dependency>
  <groupId>org.springframework.boot</groupId>
  <!--
    Spring Boot organization group ID
    This identifies the Spring Boot organization
  -->
  <artifactId>spring-boot-starter-data-jpa</artifactId>
  <!--
```

Spring Boot Data JPA starter artifact ID

This includes all dependencies needed for JPA and database access:

- Spring Data JPA for repository abstraction

- Hibernate as the JPA implementation

- Spring ORM for object-relational mapping

- Spring Transaction for transaction management

-->

</dependency>

<!-- MySQL Connector -->

<dependency>

<groupId>mysql</groupId>

<!--

MySQL organization group ID

This identifies the MySQL organization

-->

<artifactId>mysql-connector-java</artifactId>

<!--

MySQL Connector artifact ID

This is the JDBC driver for MySQL database

Allows Java applications to connect to MySQL databases

-->

<scope>runtime</scope>

<!--

Runtime scope

This means the dependency is only needed at runtime, not during compilation

The JDBC driver is loaded dynamically at runtime

```
    -->
</dependency>

<!-- Jackson Databind -->
<dependency>
    <groupId>com.fasterxml.jackson.core</groupId>
    <!--
        Jackson organization group ID
        This identifies the Jackson organization
    -->
    <artifactId>jackson-databind</artifactId>
    <!--
        Jackson Databind artifact ID
        This provides JSON serialization and deserialization:
        - Object to JSON conversion
        - JSON to Object conversion
        - Custom serializers and deserializers
        - Annotation-based configuration
    -->
</dependency>

<!-- Spring Boot Test Starter -->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <!--
        Spring Boot organization group ID
        This identifies the Spring Boot organization
    -->
```

```
    <artifactId>spring-boot-starter-test</artifactId>
    <!--
      Spring Boot test starter artifact ID
      This includes all dependencies needed for testing:
      - JUnit 5 for unit testing
      - Mockito for mocking
      - AssertJ for assertions
      - Spring Test for integration testing
    -->
    <scope>test</scope>
    <!--
      Test scope
      This means the dependency is only available during testing
      It won't be included in the final application package
    -->
  </dependency>
</dependencies>

<build>
  <!--
    Build configuration section
    This defines how Maven should build the project
  -->
  <plugins>
    <!--
      Plugins section
      This lists Maven plugins to use during the build process
    -->
```

```
        <!-- Spring Boot Maven Plugin -->
        <plugin>
          <groupId>org.springframework.boot</groupId>
          <!--
            Spring Boot organization group ID
            This identifies the Spring Boot organization
          -->
          <artifactId>spring-boot-maven-plugin</artifactId>
          <!--
            Spring Boot Maven plugin artifact ID
            This plugin provides Spring Boot specific build functionality:
            - Creates executable JAR files
            - Runs the application
            - Repackages dependencies
            - Provides Spring Boot specific goals
          -->
        </plugin>
      </plugins>
    </build>
</project>
```

2. application.properties (Application Configuration

```
#
# Spring Boot Application Configuration
# This file contains all the configuration properties for the PAMS application
# Properties are loaded automatically by Spring Boot
#
```

```
#
==========================================================================
==========
# DATABASE CONFIGURATION
#
==========================================================================
==========


# MySQL Database Connection URL
spring.datasource.url=jdbc:mysql://localhost:3306/pams_db?useSSL=false&all
owPublicKeyRetrieval=true&serverTimezone=UTC
#
# spring.datasource.url - Database connection URL
# jdbc:mysql:// - JDBC protocol for MySQL
# localhost:3306 - MySQL server host and port (default MySQL port is 3306)
# pams_db - Database name (Patient Appointment Management System
Database)
# useSSL=false - Disable SSL connection (for local development)
# allowPublicKeyRetrieval=true - Allow public key retrieval (for MySQL 8.0+)
# serverTimezone=UTC - Set server timezone to UTC to avoid timezone issues
#


# MySQL Database Username
spring.datasource.username=root
#
# spring.datasource.username - Database username
# root - Default MySQL superuser account
# In production, use a dedicated database user with limited privileges
#
```

```
# MySQL Database Password
spring.datasource.password=password
#
# spring.datasource.password - Database password
# password - Default password for root user
# In production, use a strong, unique password
# Consider using environment variables or encrypted properties
#


#
================================================================
==========
# JPA/HIBERNATE CONFIGURATION
#
================================================================
==========


# JPA Database Platform
spring.jpa.database-platform=org.hibernate.dialect.MySQLDialect
#
# spring.jpa.database-platform - JPA database platform
# org.hibernate.dialect.MySQLDialect - Hibernate dialect for MySQL
# This tells Hibernate how to generate SQL for MySQL database
# Different databases have different SQL syntax and features
#


# JPA DDL Auto Strategy
spring.jpa.hibernate.ddl-auto=update
```

#

 # spring.jpa.hibernate.ddl-auto - Hibernate DDL (Data Definition Language) auto strategy

 # update - Update the database schema based on entity changes

 # Other options:

 # - create: Drop and create tables on startup (destroys existing data)

 # - create-drop: Create tables on startup, drop on shutdown

 # - validate: Only validate schema, don't make changes

 # - none: Don't perform any schema operations

 # update is safe for development as it preserves existing data

#

 # JPA Show SQL

spring.jpa.show-sql=true

#

 # spring.jpa.show-sql - Show SQL statements in console

 # true - Display all SQL statements executed by Hibernate

 # Useful for debugging and understanding what queries are being generated

 # Set to false in production for performance and security

#

 # JPA Properties

spring.jpa.properties.hibernate.format_sql=true

#

 # spring.jpa.properties.hibernate.format_sql - Format SQL output

 # true - Format SQL statements for better readability

 # Makes the SQL output easier to read and debug

#

```
#
================================================================================
# SQL INITIALIZATION CONFIGURATION
#
================================================================================


# SQL Initialization Mode
spring.sql.init.mode=always
#
# spring.sql.init.mode - SQL initialization mode
# always - Always run SQL initialization scripts
# Other options:
#   - never: Never run initialization scripts
#   - embedded: Only run for embedded databases (H2, HSQLDB)
# always ensures that schema.sql and data.sql are executed on startup
#


# Deferred Datasource Initialization
spring.jpa.defer-datasource-initialization=true
#
# spring.jpa.defer-datasource-initialization - Defer datasource initialization
# true - Wait for JPA to initialize before running SQL scripts
# This ensures that tables are created before data is inserted
# Prevents errors when inserting data into non-existent tables
#
```

```
#
==========
# THYMELEAF CONFIGURATION
#
==========


# Thymeleaf Cache
spring.thymeleaf.cache=false
#
# spring.thymeleaf.cache - Thymeleaf template caching
# false - Disable template caching
# Useful for development as changes to templates are reflected immediately
# Set to true in production for better performance
#


#
==========
# SERVER CONFIGURATION
#
==========


# Server Port
server.port=8082
#
# server.port - HTTP server port
# 8082 - Port number for the web server
```

```
# Default Spring Boot port is 8080
# Using 8082 to avoid conflicts with other applications
# Access the application at http://localhost:8082
#

#
================================================================
==========
# LOGGING CONFIGURATION
#
================================================================
==========

# Logging Level for Root Logger
logging.level.root=INFO
#
# logging.level.root - Root logger level
# INFO - Log informational messages and above
# Other levels (from lowest to highest):
#   - TRACE: Very detailed information
#   - DEBUG: Detailed information for debugging
#   - INFO: General information
#   - WARN: Warning messages
#   - ERROR: Error messages
# INFO provides a good balance of information without too much noise
#

# Logging Level for Hibernate
logging.level.org.hibernate.SQL=DEBUG
```

#

 # logging.level.org.hibernate.SQL - Hibernate SQL logging level

 # DEBUG - Log all SQL statements

 # This works in conjunction with spring.jpa.show-sql=true

 # Provides detailed SQL logging for debugging database operations

#

 # Logging Level for Hibernate Parameters

logging.level.org.hibernate.type.descriptor.sql.BasicBinder=TRACE

#

 # logging.level.org.hibernate.type.descriptor.sql.BasicBinder - Hibernate parameter logging

 # TRACE - Log SQL parameter values

 # This shows the actual values being passed to SQL statements

 # Very useful for debugging but can be verbose

#

#
==========

 # APPLICATION SPECIFIC CONFIGURATION

#
==========

 # Application Name

spring.application.name=pams

#

 # spring.application.name - Application name

 # pams - Patient Appointment Management System

 # Used for logging, monitoring, and service discovery

#

#
==========

 # SECURITY CONFIGURATION (Optional)

#
==========

 # CSRF Protection (if using Spring Security)

 # spring.security.csrf.enabled=true

#

 # spring.security.csrf.enabled - CSRF protection

 # true - Enable Cross-Site Request Forgery protection

 # Commented out as this project doesn't use Spring Security

 # Uncomment if you add Spring Security dependency

#

#
==========

 # DEVELOPMENT CONFIGURATION

#
==========

 # Development Profile

```
spring.profiles.active=dev
#
# spring.profiles.active - Active Spring profiles
# dev - Development profile
# Profiles allow different configurations for different environments
# Other common profiles: test, prod, staging
#


#
================================================================
==========
# ERROR HANDLING CONFIGURATION
#
================================================================
==========


# Error Page Configuration
server.error.include-stacktrace=always
#
# server.error.include-stacktrace - Include stack trace in error pages
# always - Always include stack trace
# Useful for development debugging
# Set to never in production for security
#


# Error Page Path
server.error.path=/error
#
# server.error.path - Error page path
```

 # /error - Path for error page

 # Spring Boot will redirect to this path when errors occur

3. schema.sql (Database Schema)

--

-- PAMS Database Schema

-- This file contains the database schema for the Patient Appointment Management System

-- It creates tables and inserts initial data

-- Executed automatically by Spring Boot on application startup

--

--
==========

-- DATABASE CREATION (Optional - usually done manually)

--
==========

-- CREATE DATABASE IF NOT EXISTS pams_db;

--

-- CREATE DATABASE IF NOT EXISTS - Create database if it doesn't exist

-- pams_db - Database name for Patient Appointment Management System

-- Commented out as database is usually created manually

-- Uncomment if you want to create database automatically

--

-- USE pams_db;

--

```
-- USE - Select the database to use
-- pams_db - Database name
-- Commented out as database selection is handled by connection URL
-- Uncomment if you want to explicitly select database
--


--
================================================================
==========
-- TABLE CREATION
--
================================================================
==========


-- Admin Table
CREATE TABLE IF NOT EXISTS Admin (
  admin_id INT AUTO_INCREMENT PRIMARY KEY,
  --
  -- admin_id - Primary key for admin table
  -- INT - Integer data type
  -- AUTO_INCREMENT - Automatically increment value for each new record
  -- PRIMARY KEY - Unique identifier for each admin record
  --


  name VARCHAR(100) NOT NULL,
  --
  -- name - Admin's full name
  -- VARCHAR(100) - Variable character string with maximum 100 characters
  -- NOT NULL - This field cannot be empty
```

```
    --

    email VARCHAR(100) UNIQUE NOT NULL,
    --
    -- email - Admin's email address
    -- VARCHAR(100) - Variable character string with maximum 100 characters
    -- UNIQUE - Each email must be unique across all admin records
    -- NOT NULL - This field cannot be empty
    --

    role VARCHAR(50) DEFAULT 'ADMIN',
    --
    -- role - Admin's role in the system
    -- VARCHAR(50) - Variable character string with maximum 50 characters
    -- DEFAULT 'ADMIN' - Default value is 'ADMIN' if not specified
    --

    password VARCHAR(255) NOT NULL
    --
    -- password - Admin's password (should be hashed)
    -- VARCHAR(255) - Variable character string with maximum 255 characters
    -- NOT NULL - This field cannot be empty
    -- Note: In production, passwords should be hashed using BCrypt or similar
    --
);

-- Patient Table
CREATE TABLE IF NOT EXISTS patient (
```

```
patient_id INT AUTO_INCREMENT PRIMARY KEY,
--
-- patient_id - Primary key for patient table
-- INT - Integer data type
-- AUTO_INCREMENT - Automatically increment value for each new record
-- PRIMARY KEY - Unique identifier for each patient record
--

name VARCHAR(100) NOT NULL,
--
-- name - Patient's full name
-- VARCHAR(100) - Variable character string with maximum 100 characters
-- NOT NULL - This field cannot be empty
--

email VARCHAR(100) UNIQUE NOT NULL,
--
-- email - Patient's email address
-- VARCHAR(100) - Variable character string with maximum 100 characters
-- UNIQUE - Each email must be unique across all patient records
-- NOT NULL - This field cannot be empty
--

phone VARCHAR(20),
--
-- phone - Patient's phone number
-- VARCHAR(20) - Variable character string with maximum 20 characters
-- NULL allowed - Phone number is optional
```

```
  --

  address TEXT,
  --
  -- address - Patient's address
  -- TEXT - Large text field for longer addresses
  -- NULL allowed - Address is optional
  --

  dob DATE,
  --
  -- dob - Date of birth
  -- DATE - Date data type (YYYY-MM-DD format)
  -- NULL allowed - Date of birth is optional
  --

  password VARCHAR(255) NOT NULL
  --
  -- password - Patient's password (should be hashed)
  -- VARCHAR(255) - Variable character string with maximum 255 characters
  -- NOT NULL - This field cannot be empty
  -- Note: In production, passwords should be hashed using BCrypt or similar
  --
);

-- Doctor Table
CREATE TABLE IF NOT EXISTS doctor (
  doctor_id INT AUTO_INCREMENT PRIMARY KEY,
```

--

-- doctor_id - Primary key for doctor table

-- INT - Integer data type

-- AUTO_INCREMENT - Automatically increment value for each new record

-- PRIMARY KEY - Unique identifier for each doctor record

--

name VARCHAR(100) NOT NULL,

--

-- name - Doctor's full name

-- VARCHAR(100) - Variable character string with maximum 100 characters

-- NOT NULL - This field cannot be empty

--

specialization VARCHAR(100) NOT NULL,

--

-- specialization - Doctor's medical specialization

-- VARCHAR(100) - Variable character string with maximum 100 characters

-- NOT NULL - This field cannot be empty

-- Examples: Cardiology, Neurology, Pediatrics, etc.

--

email VARCHAR(100) NOT NULL,

--

-- email - Doctor's email address

-- VARCHAR(100) - Variable character string with maximum 100 characters

-- NOT NULL - This field cannot be empty

--

```
  phone VARCHAR(20),
  --
  -- phone - Doctor's phone number
  -- VARCHAR(20) - Variable character string with maximum 20 characters
  -- NULL allowed - Phone number is optional
  --

  availability JSON
  --
  -- availability - Doctor's working hours and availability
  -- JSON - JSON data type for storing structured availability data
  -- NULL allowed - Availability can be set later
  -- Example: {"monday": ["09:00", "10:00", "11:00"], "tuesday": ["09:00",
"10:00"]}
  --
);

-- Appointment Table
CREATE TABLE IF NOT EXISTS appointment (
  appointment_id INT AUTO_INCREMENT PRIMARY KEY,
  --
  -- appointment_id - Primary key for appointment table
  -- INT - Integer data type
  -- AUTO_INCREMENT - Automatically increment value for each new record
  -- PRIMARY KEY - Unique identifier for each appointment record
  --
```

patient_id INT NOT NULL,

--

-- patient_id - Foreign key referencing patient table

-- INT - Integer data type

-- NOT NULL - This field cannot be empty

-- References patient.patient_id

--

doctor_id INT NOT NULL,

--

-- doctor_id - Foreign key referencing doctor table

-- INT - Integer data type

-- NOT NULL - This field cannot be empty

-- References doctor.doctor_id

--

appointment_date DATE NOT NULL,

--

-- appointment_date - Date of the appointment

-- DATE - Date data type (YYYY-MM-DD format)

-- NOT NULL - This field cannot be empty

--

time_slot TIME NOT NULL,

--

-- time_slot - Time of the appointment

-- TIME - Time data type (HH:MM:SS format)

-- NOT NULL - This field cannot be empty

--

status ENUM('BOOKED', 'CANCELED', 'COMPLETED') DEFAULT 'BOOKED',

--

-- status - Appointment status

-- ENUM - Enumeration data type with predefined values

-- 'BOOKED' - Appointment is booked and active

-- 'CANCELED' - Appointment has been canceled

-- 'COMPLETED' - Appointment has been completed

-- DEFAULT 'BOOKED' - Default value is 'BOOKED' if not specified

--

-- Foreign Key Constraints

FOREIGN KEY (patient_id) REFERENCES patient(patient_id) ON DELETE CASCADE,

--

-- FOREIGN KEY (patient_id) - Creates foreign key relationship

-- REFERENCES patient(patient_id) - References patient_id column in patient table

-- ON DELETE CASCADE - If patient is deleted, delete all their appointments

--

FOREIGN KEY (doctor_id) REFERENCES doctor(doctor_id) ON DELETE CASCADE

--

-- FOREIGN KEY (doctor_id) - Creates foreign key relationship

-- REFERENCES doctor(doctor_id) - References doctor_id column in doctor table

-- ON DELETE CASCADE - If doctor is deleted, delete all their appointments

```
    --
);


--
================================================================================
-- INDEXES FOR PERFORMANCE
--
================================================================================


-- Index on patient email for faster login lookups
CREATE INDEX IF NOT EXISTS idx_patient_email ON patient(email);
--
-- CREATE INDEX - Creates an index for faster data retrieval
-- IF NOT EXISTS - Only create if index doesn't already exist
-- idx_patient_email - Index name
-- ON patient(email) - Index on email column of patient table
-- Improves performance of login queries
--


-- Index on admin email for faster login lookups
CREATE INDEX IF NOT EXISTS idx_admin_email ON Admin(email);
--
-- CREATE INDEX - Creates an index for faster data retrieval
-- IF NOT EXISTS - Only create if index doesn't already exist
-- idx_admin_email - Index name
-- ON Admin(email) - Index on email column of Admin table
-- Improves performance of admin login queries
```

--

-- Index on appointment date for faster date-based queries

CREATE INDEX IF NOT EXISTS idx_appointment_date ON appointment(appointment_date);

--

-- CREATE INDEX - Creates an index for faster data retrieval

-- IF NOT EXISTS - Only create if index doesn't already exist

-- idx_appointment_date - Index name

-- ON appointment(appointment_date) - Index on appointment_date column

-- Improves performance of date-based appointment queries

--

-- Index on appointment status for faster status-based queries

CREATE INDEX IF NOT EXISTS idx_appointment_status ON appointment(status);

--

-- CREATE INDEX - Creates an index for faster data retrieval

-- IF NOT EXISTS - Only create if index doesn't already exist

-- idx_appointment_status - Index name

-- ON appointment(status) - Index on status column

-- Improves performance of status-based appointment queries

--

--
==========

-- INITIAL DATA INSERTION

```
--
==========


-- Insert default admin user
INSERT INTO Admin (name, email, role, password)
VALUES ('SUPERADMIN', 'admin@pams.com', 'SUPERADMIN', 'admin123')
ON DUPLICATE KEY UPDATE
  name = VALUES(name),
  role = VALUES(role),
  password = VALUES(password);
--
-- INSERT INTO Admin - Insert new admin record
-- (name, email, role, password) - Column names
-- VALUES ('SUPERADMIN', 'admin@pams.com', 'SUPERADMIN', 'admin123') - Values to insert
-- ON DUPLICATE KEY UPDATE - If email already exists, update other fields
-- name = VALUES(name) - Update name field with new value
-- role = VALUES(role) - Update role field with new value
-- password = VALUES(password) - Update password field with new value
-- This prevents errors when running the script multiple times
--


-- Insert sample doctors
INSERT INTO doctor (name, specialization, email, phone, availability)
VALUES
  ('Dr. John Smith', 'Cardiology', 'john.smith@hospital.com', '+1234567890',
   '{"monday": ["09:00", "10:00", "11:00", "14:00", "15:00"], "tuesday": ["09:00", "10:00", "11:00"], "wednesday": ["09:00", "10:00", "11:00", "14:00", "15:00"],
```

"thursday": ["09:00", "10:00", "11:00"], "friday": ["09:00", "10:00", "11:00", "14:00", "15:00"]}'),

('Dr. Sarah Johnson', 'Neurology', 'sarah.johnson@hospital.com', '+1234567891',

'{"monday": ["10:00", "11:00", "14:00", "15:00"], "tuesday": ["09:00", "10:00", "11:00", "14:00", "15:00"], "wednesday": ["10:00", "11:00", "14:00", "15:00"], "thursday": ["09:00", "10:00", "11:00", "14:00", "15:00"], "friday": ["10:00", "11:00", "14:00", "15:00"]}'),

('Dr. Michael Brown', 'Pediatrics', 'michael.brown@hospital.com', '+1234567892',

'{"monday": ["09:00", "10:00", "11:00"], "tuesday": ["09:00", "10:00", "11:00", "14:00", "15:00"], "wednesday": ["09:00", "10:00", "11:00"], "thursday": ["09:00", "10:00", "11:00", "14:00", "15:00"], "friday": ["09:00", "10:00", "11:00"]}')

ON DUPLICATE KEY UPDATE

name = VALUES(name),

specialization = VALUES(specialization),

phone = VALUES(phone),

availability = VALUES(availability);

--

-- INSERT INTO doctor - Insert new doctor records

-- Multiple VALUES clauses for inserting multiple records

-- JSON availability data with working hours for each day

-- ON DUPLICATE KEY UPDATE - Update if email already exists

-- This prevents errors when running the script multiple times

--

--
==========

-- COMMENTS AND NOTES

--
==========

--

-- Database Schema Notes:

-- 1. All tables use AUTO_INCREMENT for primary keys

-- 2. Email fields are UNIQUE to prevent duplicate accounts

-- 3. Foreign key constraints ensure data integrity

-- 4. CASCADE DELETE removes related records when parent is deleted

-- 5. Indexes improve query performance

-- 6. JSON field stores doctor availability in structured format

-- 7. ENUM field restricts appointment status to valid values

-- 8. Initial data provides default admin and sample doctors

-- 9. ON DUPLICATE KEY UPDATE prevents insertion errors

-- 10. All text fields have appropriate length limits

Summary of Database Configuration Files

Key Components:

1. pom.xml - Maven configuration with:

- Spring Boot 3.2.5 with Java 17
- Web, Thymeleaf, Data JPA starters
- MySQL connector for database connectivity
- Jackson for JSON processing
- Test dependencies for unit testing

2. application.properties - Application configuration with:

- MySQL database connection settings
- JPA/Hibernate configuration
- SQL initialization settings
- Thymeleaf template caching

- **Server port configuration**
- **Logging levels for debugging**

3. **schema.sql - Database schema with:**

- **Four main tables: Admin, patient, doctor, appointment**
- **Proper foreign key relationships**
- **Indexes for performance optimization**
- **Initial data insertion**
- **JSON field for doctor availability**

Database Features:

- **MySQL 8.0+ compatible**
- **JPA/Hibernate ORM with automatic schema updates**
- **JSON storage for doctor availability**
- **Foreign key constraints for data integrity**
- **Indexes for query performance**
- **Initial data for testing and development**
- **Error prevention with ON DUPLICATE KEY UPDATE**

All files are now complete with comprehensive comments explaining every configuration option and database structure!