1 — Install PostgreSQL on Windows

Option A — Official installer (recommended)

1. Download the PostgreSQL Windows installer from EnterpriseDB: run the installer and follow prompts.
    - During installation you'll set the postgres superuser password — remember it.
    - Leave default port 5432 unless you need another port.
    - The installer also installs psql and pgAdmin (GUI).

Option B — Chocolatey (if you have choco)

choco install postgresql

(Then follow instructions printed to set PATH and initialize.)

After install you should have:

- SQL Shell (psql) available in Start menu
- pgAdmin (GUI) available in Start menu

---

2 — Create database and application user (psql or pgAdmin)

Open **SQL Shell (psql)** or use **pgAdmin**. Using psql is quick:

1. Open *SQL Shell (psql)* — enter host localhost, port 5432, username postgres, password (the one you set).
2. Run these SQL commands (change names/password to your preferences):

```
-- create a database for your app

CREATE DATABASE studentdb;


-- create a non-superuser for the app

CREATE USER student_user WITH ENCRYPTED PASSWORD 'YourStrongPassword';


-- give that user privileges on the DB

GRANT ALL PRIVILEGES ON DATABASE studentdb TO student_user;
```

If you prefer pgAdmin:

- Right-click *Databases* → *Create* → *Database* → name studentdb.
- Right-click *Login/Group Roles* → *Create* → create student_user with password and give privileges on the new DB.

---

3 — Update Maven pom.xml

Add the PostgreSQL JDBC driver and keep H2 for tests only. In your pom.xml add:

```
<!-- PostgreSQL JDBC driver (runtime only) -->
<dependency>
  <groupId>org.postgresql</groupId>
  <artifactId>postgresql</artifactId>
  <scope>runtime</scope>
</dependency>

<!-- Keep H2 only for tests (optional) -->
<dependency>
  <groupId>com.h2database</groupId>
  <artifactId>h2</artifactId>
  <scope>test</scope>
</dependency>
```

Also remove duplicate spring-boot-starter-data-jpa entries if present.

Save pom.xml and run:

```
mvn clean install
```

to ensure dependencies resolve.

---

4 — Configure Spring Boot datasource

Create (or edit) src/main/resources/application.properties with these settings:

```
# Postgres datasource
spring.datasource.url=jdbc:postgresql://localhost:5432/studentdb
spring.datasource.username=student_user
spring.datasource.password=YourStrongPassword
spring.datasource.driver-class-name=org.postgresql.Driver

# Hibernate / JPA
# Use an appropriate dialect for your Postgres version
spring.jpa.database-platform=org.hibernate.dialect.PostgreSQLDialect
```

For development: 'update' is convenient. For production, use validate + migrations.

spring.jpa.hibernate.ddl-auto=update

spring.jpa.show-sql=true

spring.jpa.properties.hibernate.format_sql=true

Notes:

- spring.jpa.hibernate.ddl-auto=update will let Hibernate create tables in Postgres automatically. Use create only if you want DB reset each run.
- Use PostgreSQL10Dialect or PostgreSQL95Dialect depending on your Hibernate version; PostgreSQL10Dialect works fine for modern Postgres.

---

5 — Small code recommendation: table name casing

Postgres lowercases unquoted identifiers. In your entities you used mixed casing for one table:

- @Table(name = "Student") — this *may* cause quoting/casing mismatch. I recommend using lowercase consistent names to avoid surprises.

Change Student entity annotation to:

@Table(name = "student")

Your Course entity already uses @Table(name = "course") so that's fine. Also grade and users are already lowercase in your code — great.

Why: if Hibernate generates SQL referencing "Student" (quoted mixed-case) it will not match an unquoted student. Best to use lowercase table names.

---

6 — Identity generation / auto-increment

You're using @GeneratedValue(strategy = GenerationType.IDENTITY) for primary keys — that works correctly with Postgres (it maps to serial / identity). No change needed.

If you later import raw IDs into tables, make sure sequences are set correctly:

SELECT setval(pg_get_serial_sequence('student','id'), (SELECT MAX(id) FROM student));

---

7 — Run the app and test

1. Start PostgreSQL service (it runs as a Windows service after install).
2. Start your Spring Boot app:

mvn spring-boot:run

or run from your IDE.

3. Check logs — you should see Hibernate SQL statements and Hibernate: create table ... if tables are created. Also check Spring Boot startup for successful datasource connection.
4. Validate by hitting a REST endpoint (e.g., GET /students) or by using psql:

```
psql -U student_user -d studentdb -h localhost

# inside psql

\dt      -- list tables

SELECT count(*) FROM student;
```