**Week 1: Core JDBC & Essential Patterns**

**Day 1: JDBC Setup & Driver Basics**

* **Theory**:
  + JDBC architecture (Driver, Connection, Statement, ResultSet)
  + Driver types (Type 4 preferred)
  + Modern driver loading (JDBC 4.0+ ServiceLoader vs legacy Class.forName())
* **Lab**:
  + Set up PostgreSQL + H2 (embedded) databases.
  + Write a "Hello JDBC" program to test connections to both.

**Day 2: Connections & Simple Queries**

* **Theory**:
  + DriverManager vs DataSource
  + Connection strings (URL formats for MySQL/PostgreSQL/H2)
* **Lab**:
  + Implement DriverManager.getConnection().
  + **Anti-Pattern Demo**: Leak connections by forgetting close().

**Day 3: PreparedStatement & Security**

* **Theory**:
  + SQL injection (show Statement vulnerability)
  + Parameterized queries (?, setInt(), setString())
* **Lab**:
  + Write a login method vulnerable to ' OR 1=1 -- attack.
  + Fix it with PreparedStatement.

**Day 4: ResultSet & Data Mapping**

* **Theory**:
  + Forward-only vs scrollable ResultSet
  + Mapping ResultSet to Java objects (DTOs)
* **Lab**:
  + Build a UserMapper class converting ResultSet → User object.
  + Implement pagination with LIMIT/OFFSET.

**Day 5: Transactions**

* **Theory**:
  + ACID properties, setAutoCommit(false), commit()/rollback()
  + Isolation levels (READ\_COMMITTED, SERIALIZABLE)
* **Lab**:
  + Simulate a bank transfer with rollback on failure.
  + **Debug**: Force a deadlock and handle it with retries.

**Week 2: Advanced JDBC for Production**

**Day 6: Connection Pooling (HikariCP)**

* **Theory**:
  + Why pooling? (TCP overhead, performance)
  + HikariCP configuration (maximumPoolSize, leakDetectionThreshold)
* **Lab**:
  + Set up HikariCP standalone (your provided code).
  + Benchmark pooled vs non-pooled connections under load.

**Day 7: Batch Processing**

* **Theory**:
  + addBatch(), executeBatch()
  + Batch size optimization (memory vs latency)
* **Lab**:
  + Insert 10K records: Compare single vs batch (measure with System.nanoTime()).
  + **Advanced**: Batch updates (not just inserts).

**Day 8: Stored Procedures & CallableStatement**

* **Theory**:
  + IN/OUT parameters, {call proc\_name(?, ?)} syntax
* **Lab**:
  + Create a PostgreSQL stored procedure (e.g., calculate\_discount).
  + Call it from JDBC with registerOutParameter().

**Day 9: Metadata & Dynamic SQL**

* **Theory**:
  + DatabaseMetaData (schema inspection)
  + ResultSetMetaData (dynamic column handling)
* **Lab**:
  + Build a "DB Explorer" that lists all tables + columns.
  + Generate dynamic WHERE clauses safely.

**Day 10: Advanced ResultSets**

* **Theory**:
  + Scrollable (TYPE\_SCROLL\_INSENSITIVE)
  + Updatable (CONCUR\_UPDATABLE)
* **Lab**:
  + Update a user’s email directly through ResultSet.
  + **Challenge**: Delete a row via ResultSet.deleteRow().

**Day 11: RowSets (CachedRowSet, WebRowSet)**

* **Theory**:
  + Disconnected data, offline editing
* **Lab**:
  + Load data into CachedRowSet, modify offline, sync back.

**Day 12: Exception Handling & SQLState**

* **Theory**:
  + SQLException subtypes (SQLTimeoutException, BatchUpdateException)
  + SQLState codes (40P01 = deadlock)
* **Lab**:
  + Write retry logic for deadlocks (use SQLState).

**Day 13: Performance Tuning**

* **Theory**:
  + Fetch size (setFetchSize(1000) for large queries)
  + Connection pool tuning (validation queries, timeouts)
* **Lab**:
  + Benchmark fetchSize=10 vs fetchSize=1000 on a 50K-row query.

**Day 14: Final Project**

* **Build an Inventory System**:
  + Requirements:
    1. HikariCP connection pooling.
    2. Batch import of products from CSV.
    3. Transactional stock updates (rollback on failure).
    4. Admin panel with updatable ResultSet.
    5. Dynamic filtering using ResultSetMetaData.