CS-401 MODERN PROGRAMMING PRACTICES

	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday
OOAD / RDBMS / OOP	**AM: Lesson 1:** *The OO Paradigm for Building Software Solutions* **PM: Lab 1**	**AM: Lesson 2:** *Associations among Objects and Classes* [Lab 1 due 10 AM] **PM: Lab 1 solutions, Lab 2**	**AM: Lesson 3:** *Inheritance and Composition* [Lab 2 due 10 AM] **PM: Lab 2 solutions, Lab 3**	**AM: Lesson 4**: *Interaction Diagrams* [Lab 3 due 10 AM] **PM: Lab 3 solutions, Lab 4**	**AM: Lesson 5**: *Inheritance and Abstraction* [Lab 4 due 10 AM] **PM: Lab 4 solutions, Lab 5**	**AM: Lesson 6:** *Relational Model, View & Normalization* [Lab 5 due 10 AM] **Lab 5 solutions**
	AM: Lesson 7: *SQL (DML&DDL)* **PM: Lab 6, 7**	**AM: Lesson 8**: *Index, SQL Injection, JDBC application & Intro to Maven* [Lab 6 due 10 AM] [Lab 7 due 5 PM] **PM: Lab 6 solutions, Lab 8**	**AM:** Review for Midterm exam **Lab 7 & 8 solutions** **PM:** *Study for Midterm*	**MIDTERM EXAM**	**AM: Lesson 9**: *Interfaces in Java 8 and the Object Superclass* **PM: Lab 9**	**AM: Lesson 10**: *Functional Programming in Java* [Lab 9 due 10 AM] **Lab 9 solutions**
OOP	**AM: Lesson 11**: ***The Stream API*** **PM: Lab 10**	**AM: Lesson 11**: ***The Stream API*** [Lab 10 due 10 AM] **PM: Lab 10 solutions, Lab**	**AM: Lesson 12** *Best Programming Practices with Java 8* [Lab 11 due 10 AM] **PM: Lab 11 solutions, Lab**	**AM: Lesson 13** *Generic Programming* **PM: Lab 12 solutions, Lab 13**	**AM**: Review for Final exam **Lab 13 solutions**	**Final Exam**
	AM: Java Swing **PM: Course Project**	**Course Project**	**Course Project**	**AM:** *Project Presentation – Connect whole with parts*		

Lesson 8

Index, View, SQL Injection, JDBC

Dr. Bright Gee Varghese

Wholeness

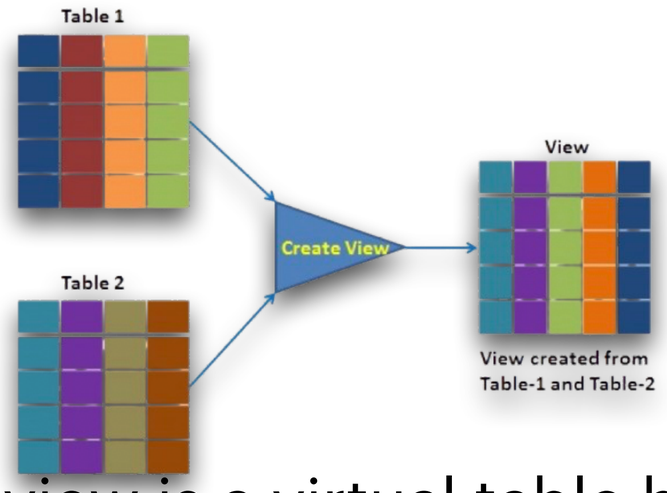
Java provides convenient tools for accessing data stored in a database. The relationship between stored data and an executing program parallels the relationship between awareness and its interaction with the world; that interaction is most successful and rewarding if awareness is broad (corresponding to a well-designed program) and is well integrated with the laws of nature,

with the ways of manifest existence (JDBC).

Science & Technology of Consciousness: TM is a simple, effortless mental technique that can be used by anyone, no matter what their lifestyle. It promotes spontaneous fulfillment of desires, by bringing the desires of the individual into accord with Natural Law, without the individual having to know the underlying mechanism.

Outline

- View
- Index
- SQL injection
- JDBC Application Architecture
- Maven/Gradle



View

- A view is a virtual table based on the result-set of an SQL statement.
- A view contains rows and columns, just like a real table. The fields in a view are fields from one or more real tables in the database.
- Unlike a physical table, a view does not store data itself; instead, it dynamically retrieves data from the underlying tables whenever it is queried.
- The contents of a view are defined as a query on one or more base relations.
- A View is defined as a query on one or more base tables or views.

students			
student_id	**name**	**major**	**advisor_id**
1	Alice Brown	CS	101
2	Bob Smith	Math	102
3	Carol White	CS	101

professors		
professor_id	**prof_name**	**department**
101	Dr. Johnson	CS
102	Dr. Lee	Math

studentAdvisors				
student_id	**student_name**	**major**	**advisor_name**	**advisor_department**
1	Alice Brown	CS	Dr. Johnson	CS
2	Bob Smith	Math	Dr. Lee	Math
3	Carol White	CS	Dr. Johnson	CS

```
CREATE VIEW StudentAdvisors AS
SELECT
   s.student_id,
   s.name AS student_name,
   s.major,
   p.prof_name AS advisor_name,
   p.department AS advisor_department
FROM
   Students s
   JOIN Professors p ON s.advisor_id = p.professor_id;
```

View

Purpose of Views

- It provides a powerful and flexible security mechanism by hiding parts of the database from certain users. Users are not aware of the existence of any attributes or tuples that are missing from the view.
- It permits users to access data in a way that is customized to their needs, so that the same data can be seen by different users in different ways, at the same time.
- It can simplify complex operations on the base relations.

Use a View

Syntax

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

Use a View

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

Create a view called clientNoWithName that provides only the essential fields from the base relation, Client:

- clientNo
- fName
- lName

Also, sort the results by clientNo so the list is easier to navigate.

Client

clientNo	fName	lName	telNo	prefType	maxRent	eMail
CR56	Aline	Stewart	0141-848-1825	Flat	350	astewart@hotmail.com
CR62	Mary	Tregear	01224-196720	Flat	600	maryt@hotmail.co.uk
CR74	Mike	Ritchie	01475-392178	House	750	mritchie01@yahoo.co.uk
CR76	John	Kay	0207-774-5632	Flat	425	john.kay@gmail.com

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

Use a View

Create a view called clientNoWithName that provides only the essential fields:

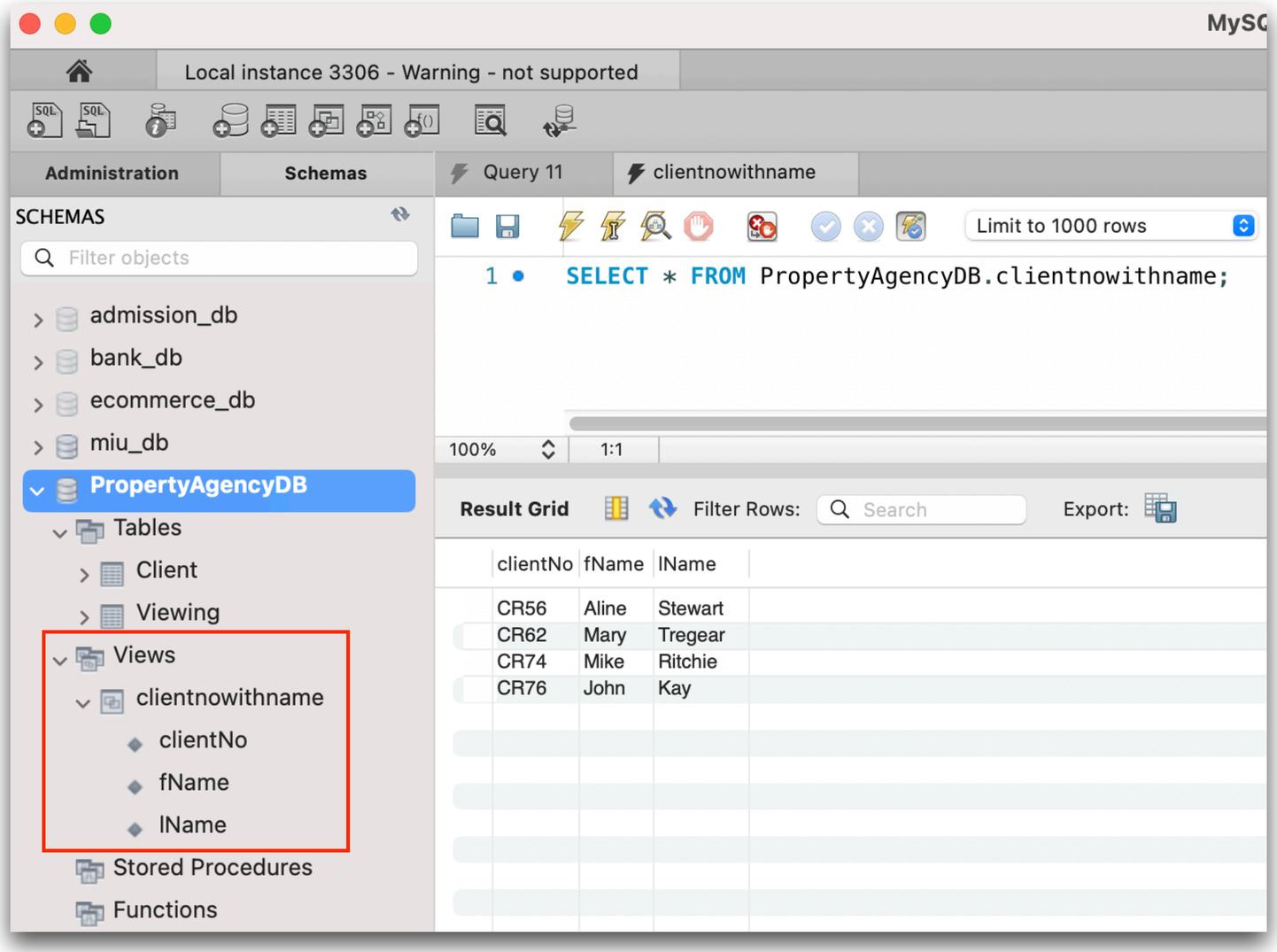
- clientNo
- fName
- lName

Also, sort the results by clientNo so the list is easier to navigate.

```
MySQL  localhost:3306 ssl  propertyagencydb  SQL  CREATE VIEW clientNoWithName AS
                                                  SELECT clientNo, fName, lName
                                                  FROM client
                                                  ORDER BY clientNo;
Query OK, 0 rows affected (0.0093 sec)
MySQL  localhost:3306 ssl  propertyagencydb  SQL  SELECT *
                                                  FROM clientNoWithName;
+----------+-------+---------+
| clientNo | fName | lName   |
+----------+-------+---------+
| CR56     | Aline | Stewart |
| CR62     | Mary  | Tregear |
| CR74     | Mike  | Ritchie |
| CR76     | John  | Kay     |
+----------+-------+---------+
```

Use a View

MySQL Workbench



View
Update

- Views are read-only by default in many databases (like MySQL) if:
  - They involve JOINs
  - They use GROUP BY, DISTINCT, LIMIT, or subqueries
- So you usually cannot do INSERT, UPDATE, or DELETE on a complex view.

View

Update

```
 MySQL  localhost:3306 ssl  propertyagencydb  SQL  UPDATE clientNoWithName
                                                   SET fName = 'Alan'
                                                   WHERE clientNo = 'CR56';
Query OK, 1 row affected (0.0042 sec)

Rows matched: 1  Changed: 1  Warnings: 0
 MySQL  localhost:3306 ssl  propertyagencydb  SQL  SELECT * FROM clientNoWithName;
+----------+-------+---------+
| clientNo | fName | lName   |
+----------+-------+---------+
| CR76     | John  | Kay     |
| CR74     | Mike  | Ritchie |
| CR62     | Mary  | Tregear |
| CR56     | Alan  | Stewart |
+----------+-------+---------+
4 rows in set (0.0040 sec)
 MySQL  localhost:3306 ssl  propertyagencydb  SQL  SELECT * FROM client;
+----------+-------+---------+---------------+----------+---------+----------------------+
| clientNo | fName | lName   | telNo         | prefType | maxRent | eMail                |
+----------+-------+---------+---------------+----------+---------+----------------------+
| CR56     | Alan  | Stewart | 0141-848-1825 | Flat     |     350 | astewart@hotmail.com |
| CR62     | Mary  | Tregear | 01224-196720  | Flat     |     600 | maryt@hotmail.co.uk  |
| CR74     | Mike  | Ritchie | 01475-392178  | House    |     750 | mritchie01@yahoo.co.uk |
| CR76     | John  | Kay     | 0207-774-5632 | Flat     |     425 | john.kay@gmail.com   |
+----------+-------+---------+---------------+----------+---------+----------------------+
4 rows in set (0.0016 sec)
```

Outline

- View
- Index
- SQL injection
- JDBC Application Architecture
- Maven/Gradle

Index

- An index is a database object that improves the speed of data retrieval operations on a table.
- Like an index in a book, it helps you quickly locate specific data without scanning the entire table.

- Purpose
  - Speeds up SELECT queries and WHERE clause operations.

  - Slows down INSERT, UPDATE, and DELETE operations (due to index maintenance).

Index

How does an Index work?

- Without Index:
  - The database performs a full table scan to find matching rows (slow for large tables).
- With Index:
  - The database uses the index to quickly locate rows (like a binary search).

Index

Types

- Single-Column Index: Index on one column.
  - CREATE INDEX idx_lastname ON Employees(last_name);
- Composite Index: Index on multiple columns.
  - CREATE INDEX idx_name_department ON Employees(first_name, department_id);
- Unique Index: Ensures all values in the indexed column(s) are unique.
  - CREATE UNIQUE INDEX idx_email ON Employees(email);

Index

Syntax

Creates an index on a table. Duplicate values are allowed:

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

Creates a unique index on a table. Duplicate values are not allowed:

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

Index

Performance Evaluation – Step-by-Step

1. Create a table.
2. Populate data.
3. Verify the data.
4. Query without an index and analyze it.
5. Create index.
6. Query with an index and analyze it.

Index

Performance Evaluation – Step-by-Step

1. Create a table

id	name	email	age	city

```
MySQL  localhost:3306 ssl  user_db  SQL
[redacted]
CREATE TABLE users (
id INT AUTO_INCREMENT PRIMARY KEY,
name VARCHAR(100),
email VARCHAR(100),
age INT,
city VARCHAR(100)
);
```

Index

Performance Evaluation – Step-by-Step

2. Populate data

```
MySQL  localhost:3306 ssl  user_db  SQL  SET SESSION cte_max_recursion_depth = 1000000;
Query OK, 0 rows affected (0.0025 sec)
MySQL  localhost:3306 ssl  user_db  SQL  INSERT INTO users (name, email, age, city)
[redacted]
WITH RECURSIVE numbers AS (
SELECT 1 AS id
UNION ALL
SELECT id + 1 FROM numbers WHERE id < 1000000
)
SELECT
CONCAT('User', id),
CONCAT('user', id, '@example.com'),
FLOOR(18 + (RAND() * 60)), -- Age between 18 and 78
IF(RAND() < 0.5, 'New York', 'Los Angeles')
FROM numbers;
Query OK, 1000000 rows affected (7.6825 sec)
```

Generates numbers from 1 to 1,000,000 dynamically within SQL using recursion

Generates 1,000,000 rows of realistic user data (name, email, age, city)

```
MySQL  localhost:3306 ssl  SQL  [redacted]
WITH RECURSIVE myNumbers AS(
SELECT 0 AS student_id
UNION
SELECT student_id+1 from myNumbers
WHERE student_id < 5
)
SELECT * FROM myNumbers;
+------------+
| student_id |
+------------+
|          0 |
|          1 |
|          2 |
|          3 |
|          4 |
|          5 |
+------------+
6 rows in set (0.0020 sec)
```

Anchor member

Recursive member

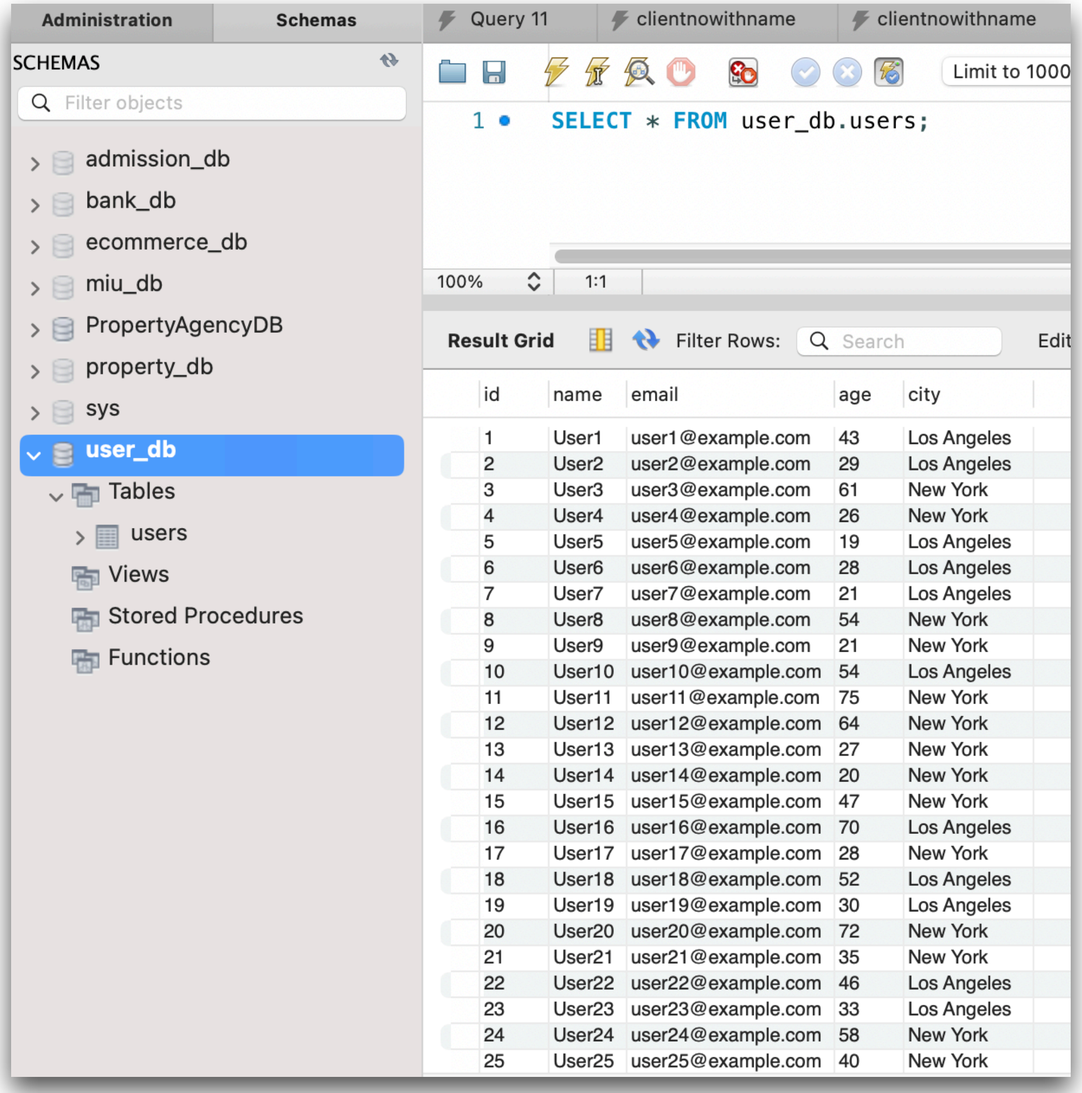
```sql
WITH RECURSIVE employee_hierarchy AS (
    -- Anchor query: Start with Ankit (employee_id = 1)
    SELECT employee_id, employee_name, manager_id, age
    FROM employees
    WHERE employee_id = 1

    UNION ALL

    -- Recursive query: Join the employees table with itself to get the employees reporting to each manager
    SELECT e.employee_id, e.employee_name, e.manager_id, e.age
    FROM employees e
    INNER JOIN employee_hierarchy eh ON e.manager_id = eh.employee_id
)
SELECT * FROM employee_hierarchy;
```

Ref:
https://www.geeksforgeeks.org/sql/recursive-join-in-sql/

Index

Performance Evaluation – Step-by-Step

3. Verify the data



Index

Performance Evaluation – Step-by-Step

4. Query without an index and analyze it.

```
1 row in set (0.0234 sec)
 MySQL  localhost:3306 ssl  user_db  SQL  EXPLAIN ANALYZE
                                          SELECT * FROM users WHERE email = 'user500000@example.com';
+-----------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------+
| EXPLAIN                                                                                  |
+-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------+
| -> Filter: (users.email = 'user500000@example.com')  (cost=100747 rows=99553) (actual time=201..386 rows=1 loops=1)
    -> Table scan on users  (cost=100747 rows=995531) (actual time=0.11..281 rows=1e+6 loops=1)
 |
+-----------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------+
1 row in set (0.3897 sec)
```

1. Table scan on users
MySQL is scanning all 1,000,000 rows (notice: rows=1e+6) because it has no index on email.
The cost=100747 is MySQL's estimated internal cost to do this full scan.
actual time=0.11..281: Real time taken: 0.11 ms to start returning rows, 281ms to complete.
2. Filter: users.email = 'user500000@example.com'
After scanning every row, MySQL filters to find the row with that specific email.
It estimates about 99553 rows might match (rows=99553), which is just a guess (no stats for unindexed columns).
actual time=201..386: Real time taken: 201 ms to start returning rows, 386 ms to complete.
In reality, only 1 row matched (rows=1) and was returned in ~386 ms total.

Index

Performance Evaluation – Step-by-Step

5. Create an index.

```
MySQL  localhost:3306 ssl  user_db  SQL  CREATE INDEX idx_users_email ON users(email);
Query OK, 0 rows affected (1.4158 sec)

Records: 0  Duplicates: 0  Warnings: 0
 MySQL  localhost:3306 ssl  user_db  SQL  SHOW INDEX FROM users;
+-------+------------+----------------+--------------+-------------+-----------+-------------+----------+--------+------
+------------+---------+---------------+---------+------------+
| Table | Non_unique | Key_name        | Seq_in_index | Column_name | Collation | Cardinality | Sub_part | Packed | Null |
Index_type | Comment | Index_comment | Visible | Expression |
+-------+------------+----------------+--------------+-------------+-----------+-------------+----------+--------+------
+------------+---------+---------------+---------+------------+
| users |          0 | PRIMARY         |            1 | id          | A         |      995531 |     NULL | NULL   |      | BTREE
|         |               | YES     | NULL       |
| users |          1 | idx_users_email |            1 | email       | A         |      995531 |     NULL | NULL   | YES  | BTREE
|         |               | YES     | NULL       |
+-------+------------+----------------+--------------+-------------+-----------+-------------+----------+--------+------
+------------+---------+---------------+---------+------------+
2 rows in set (0.0025 sec)
```

Index

Performance Evaluation – Step-by-Step

6. Query with an index and analyze it.

```
 MySQL  localhost:3306 ssl  user_db  SQL  EXPLAIN ANALYZE
                                          SELECT * FROM users WHERE email = 'user500000@example.com';
+------------------------------------------------------------------------------------------------------------------------
-----------------+
| EXPLAIN
|
+------------------------------------------------------------------------------------------------------------------------
-----------------+
| -> Index lookup on users using idx_users_email (email = 'user500000@example.com')  (cost=0.357 rows=1) (actual time=0.0557..0.0577
rows=1 loops=1)
 |
+------------------------------------------------------------------------------------------------------------------------
-----------------+
1 row in set (0.0022 sec)
```

Index

Drop an Index

Syntax

```
DROP INDEX index_name
ON table_name;
```

Index

Drop an Index

```
DROP INDEX index_name

ON table_name;
```

```
MySQL localhost:3306 ssl user_db SQL SHOW INDEXES FROM users;
+-------+------------+-----------------+--------------+-------------+-----------+-------------+----------+--------+------+------------+---------+---------------+---------+------------+
| Table | Non_unique | Key_name        | Seq_in_index | Column_name | Collation | Cardinality | Sub_part | Packed | Null | Index_type | Comment | Index_comment | Visible | Expression |
+-------+------------+-----------------+--------------+-------------+-----------+-------------+----------+--------+------+------------+---------+---------------+---------+------------+
| users |          0 | PRIMARY         |            1 | id          | A         |      995531 |     NULL | NULL   |      | BTREE      |         |               | YES     | NULL       |
| users |          1 | idx_users_email |            1 | email       | A         |      995531 |     NULL | NULL   | YES  | BTREE      |         |               | YES     | NULL       |
+-------+------------+-----------------+--------------+-------------+-----------+-------------+----------+--------+------+------------+---------+---------------+---------+------------+
2 rows in set (0.0069 sec)
MySQL localhost:3306 ssl user_db SQL DROP INDEX idx_users_email
                                     ON users;
Query OK, 0 rows affected (0.0644 sec)

Records: 0  Duplicates: 0  Warnings: 0
MySQL localhost:3306 ssl user_db SQL SHOW INDEXES FROM users;
+-------+------------+----------+--------------+-------------+-----------+-------------+----------+--------+------+------------+---------+---------------+---------+------------+
| Table | Non_unique | Key_name | Seq_in_index | Column_name | Collation | Cardinality | Sub_part | Packed | Null | Index_type | Comment | Index_comment | Visible | Expression |
+-------+------------+----------+--------------+-------------+-----------+-------------+----------+--------+------+------------+---------+---------------+---------+------------+
| users |          0 | PRIMARY  |            1 | id          | A         |      995531 |     NULL | NULL   |      | BTREE      |         |               | YES     | NULL       |
+-------+------------+----------+--------------+-------------+-----------+-------------+----------+--------+------+------------+---------+---------------+---------+------------+
1 row in set (0.0015 sec)
MySQL localhost:3306 ssl user_db SQL
```

Index

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

```
CREATE INDEX RentInd ON PropertyForRent (city, rent);
```

then an index called RentInd is created for the PropertyForRent table. Entries will be in alphabetical order by city and then by rent within each city.

Index

When to use index?

1. Use index when
    1. Columns are frequently used in WHERE clause.
    2. Columns are used in JOIN conditions.
    3. Columns are used in ORDER BY or GROUP BY clauses.
2. Avoid Indexes when
    1. Tables are small.
    2. Columns are frequently updated (high write operations).

Index

Index Drawbacks

- Increased Storage:
  - Indexes consume additional disk space.
- Write Performance Impact:
  - INSERT, UPDATE, and DELETE operations are slower because indexes must be updated.
- Maintenance Overhead:
  - Indexes need to be rebuilt or reorganized periodically.

Outline

- View
- Index
- SQL injection
- JDBC Application Architecture
- Maven/Gradle

SQL Injection

- It is a code injection technique that might destroy your database.

miu_users

userId	username	password
1	user1	password1
2	user2	password2

```
MySQL  localhost:3306 ssl  user_db  SQL   SELECT *
                                          FROM miu_users
                                          WHERE username = 'user1' AND password = 'p1' OR 1=1;
```

userId	username	password
1	user1	password1
2	user2	password2

(username = 'user1' AND password = 'p1') OR (1=1)

SQL Injection

A login form with username and password

```
String url = "jdbc:mysql://localhost:3306/user_db";
String user = "root";
String password = “password";

String inputUsername = "user1"; //request.getParameter("username");
String inputPassword = "password1"; //request.getParameter("password");
try(Connection conn = DriverManager.getConnection(url, user, password)) {
    String query = (
            "SELECT * FROM miu_users " +
            "WHERE username = '%s' AND password = '%s'"
    ).formatted(inputUsername, inputPassword);
    Statement stmt = conn.createStatement();
    ResultSet rs = stmt.executeQuery(query);
    while(rs.next()) {
        System.out.println(
                rs.getString("username") +
                        " " + rs.getString("password")
        );
    }
} catch (Exception e) {
    throw new RuntimeException(e);
}
```

SQL Injection

How to inject?

If the hacker enters user1' -- as the username, the query becomes:

```
String inputUsername = "user1' -- ";
```

```
SELECT * FROM miu_users WHERE username = 'user1' -- ' AND password = ''
```

The -- comments out the rest of the query, bypassing the password check.

user1 password1

SQL Injection

Risks

- Data Breach:
  - Attackers can access sensitive data.
- Data Manipulation:
  - Attackers can modify or delete data.
- System Compromise:
  - Attackers can execute administrative operations on the database.
- Reputation Damage:
  - Loss of trust from users and stakeholders.

SQL Injection

How to prevent it?

- Use PreparedStatement instead of Statement.
- Validate and sanitize user inputs.
- Use stored procedures.
- Implement least privilege access for database users.
- Use ORM frameworks (e.g., Hibernate) that abstract SQL queries.

PreparedStatement

- It is an object that represents a precompiled SQL statement.
- A SQL statement is precompiled and stored in a PreparedStatement object. This object can then be used to efficiently execute this statement multiple times.

PreparedStatement
Sample Code

```
String updateSQL =
        "UPDATE compro_users SET email = ? WHERE fName = ? AND lName = ?";

try (PreparedStatement ps = connection.prepareStatement(updateSQL)) {
    // Set parameters in order
    ps.setString(1, "jane.updated@gmail.com"); // new email
    ps.setString(2, "Jane");                   // match first name
    ps.setString(3, "Smith");                  // match last name

int rowsUpdated = ps.executeUpdate();     // execute the update
    System.out.println("Rows updated: " + rowsUpdated);
}
```

PreparedStatement

How does it prevent SQL injection?

- Prepared statements enforce the separation between templated SQL and user-supplied input.
- Prepared statements always treat client-supplied data as content of a parameter and never as a part of an SQL statement.

```
String updateSQL =
        "UPDATE compro_users SET email = ? WHERE fName = ? AND lName = ?";

try (PreparedStatement ps = connection.prepareStatement(updateSQL)) {
    // Set parameters in order
    ps.setString(1, "jane.updated@gmail.com"); // new email
    ps.setString(2, "Jane");                   // match first name
    ps.setString(3, "Smith");                  // match last name
    //…
}
```

Ref:
https://docs.oracle.com/javase/tutorial/jdbc/basics/prepared.html

PreparedStatement

How does it prevent SQL injection?

```
    String inputUsername = "user1' -- "; //request.getParameter("username");
        String inputPassword = ""; //request.getParameter("password");
//        String inputUsername = "user1"; //request.getParameter("username");
//        String inputPassword = "password1"; //request.getParameter("password");
        try(Connection conn = DriverManager.getConnection(url, user, password)) {
            String query = (
                    "SELECT * FROM miu_users " +
                            "WHERE username = ? AND password = ?"
            );
            try(PreparedStatement preparedStatement = conn.prepareStatement(query)) {
                preparedStatement.setString(1, inputUsername);
                preparedStatement.setString(2, inputPassword);
                ResultSet rs = preparedStatement.executeQuery();
                while(rs.next()) {
                    System.out.println(
                            rs.getString("username") +
                                    " " + rs.getString("password")
                    );
                }
            } catch (Exception e) {
                e.printStackTrace();
            }
        } catch (Exception e) {
            throw new RuntimeException(e);
        }
    }
```

java.sql.SQLException: Access denied for user 'user1' --'@'localhost' (using password: YES)

PreparedStatement

Benefits

- Security:
  - Prevents SQL injection by parameterizing inputs.
- Performance:
  - Precompiled queries reduce database overhead.
- Readability:
  - Separates SQL logic from data inputs, making code cleaner.
- Reusability:
  - The same PreparedStatement can be reused with different parameters.

PreparedStatement

Sample Code for Reusability

```
// Insert data using PreparedStatement and batch
String insertSQL = "INSERT INTO compro_users (fName, lName, email) VALUES (?, ?, ?)";
try (PreparedStatement ps = connection.prepareStatement(insertSQL)) {
    ps.setString(1, "John");
    ps.setString(2, "Doe");
    ps.setString(3, "johndoe@gmail.com");
    ps.addBatch();

    ps.setString(1, "Jane");
    ps.setString(2, "Smith");
    ps.setString(3, "janesmith@gmail.com");
    ps.addBatch();

    int[] insertResults = ps.executeBatch();
    System.out.println("Rows inserted: " + insertResults.length);
}
```

Statement

No Reusability

```
Statement stmt = connection.createStatement();

stmt.addBatch("INSERT INTO compro_users (fName, lName, email) VALUES ('John', 'Doe', 'johndoe@gmail.com')");
stmt.addBatch("INSERT INTO compro_users (fName, lName, email) VALUES ('Jane', 'Smith', 'janesmith@gmail.com')");

int[] results = stmt.executeBatch();
```

Outline

- View
- Index
- SQL injection
- JDBC Application Architecture
- Maven/Gradle

Gradle
Build Tool

- Gradle is the open source build system of choice for Java, Android, and Kotlin developers. From mobile apps to microservices, from small startups to big enterprises, it helps teams deliver better software, faster.

Ref:

https://dpeuniversity.gradle.com/app/courses/012de84f-fcd3-45d4-9c4c-284382eb3f3f

Gradle

Build Tool

```
bright~$brew install gradle
==> Downloading https://formulae.brew.sh/api/formula.jws.json
==> Downloading https://formulae.brew.sh/api/cask.jws.json
...

bright~$gradle -v

------------------------------------------------------------
Gradle 8.14.2
------------------------------------------------------------

Build time:    2025-06-05 13:32:01 UTC
Revision:      30db2a3bdfffa9f8b40e798095675f9dab990a9a

Kotlin:        2.0.21
Groovy:        3.0.24
Ant:           Apache Ant(TM) version 1.10.15 compiled on August 25 2024
Launcher JVM:  23.0.1 (Oracle Corporation 23.0.1+11-39)
Daemon JVM:    /Library/Java/JavaVirtualMachines/jdk-23.jdk/Contents/Home (no JDK specified, using current Java home)
OS:            Mac OS X 16.0 x86_64

bright~$
```

Maven
Build Tool

- Maven is an open-source build automation and project management tool widely used for Java applications.
- As a build automation tool, it automates the source code compilation and dependency management, assembles binary codes into packages, and executes test scripts.
- Maven translates and packages your source code so that it becomes an executable application.

Maven

Project Object Model (POM)

- A POM is the basement of the Maven framework.
- It's a type of XML file that accommodates data from your project and the configuration details.
- It includes the project, group ID, POM model version, artifact ID (project ID), and version. The project is the key element of your XML file. Group ID means the ID of the group to which your project belongs. Here, the version informs you about the number of your project releases.

pom.xml (dbdemo101)

```xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instar
         xsi:schemaLocation="http://maven.apache.org/POM/4
    <modelVersion>4.0.0</modelVersion>

    <groupId>com.bright</groupId>
    <artifactId>dbdemo101</artifactId>
    <version>1.0-SNAPSHOT</version>

    <properties>
        <maven.compiler.source>24</maven.compiler.source>
        <maven.compiler.target>24</maven.compiler.target>
```

Maven

Create a Jar with all Dependencies

Executable JAR

To create an executable uber JAR, one simply needs to set the main class that serves as the application entry point:

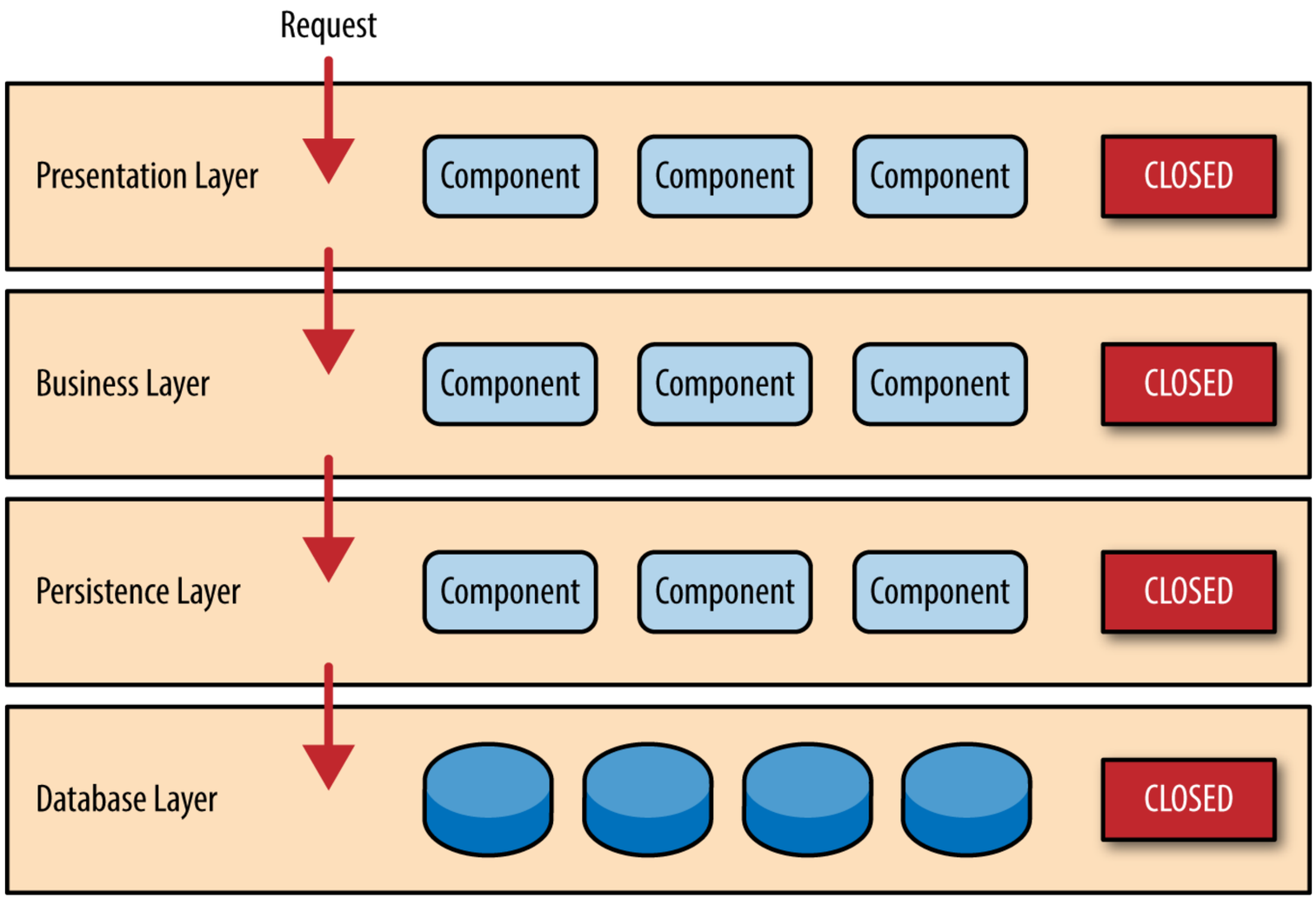
```
<project>
  ...
  <build>
    <plugins>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-shade-plugin</artifactId>
        <version>3.6.0</version>
        <executions>
          <execution>
            <phase>package</phase>
            <goals>
              <goal>shade</goal>
            </goals>
            <configuration>
              <transformers>
                <transformer implementation="org.apache.maven.plugins.shade.resource.ManifestResourceTransformer">
                  <mainClass>org.sonatype.haven.HavenCli</mainClass>
                </transformer>
              </transformers>
            </configuration>
          </execution>
        </executions>
      </plugin>
    </plugins>
  </build>
  ...
</project>
```

Ref: https://maven.apache.org/plugins/maven-shade-plugin/examples/executable-jar.html

N-Layer Architecture Overview

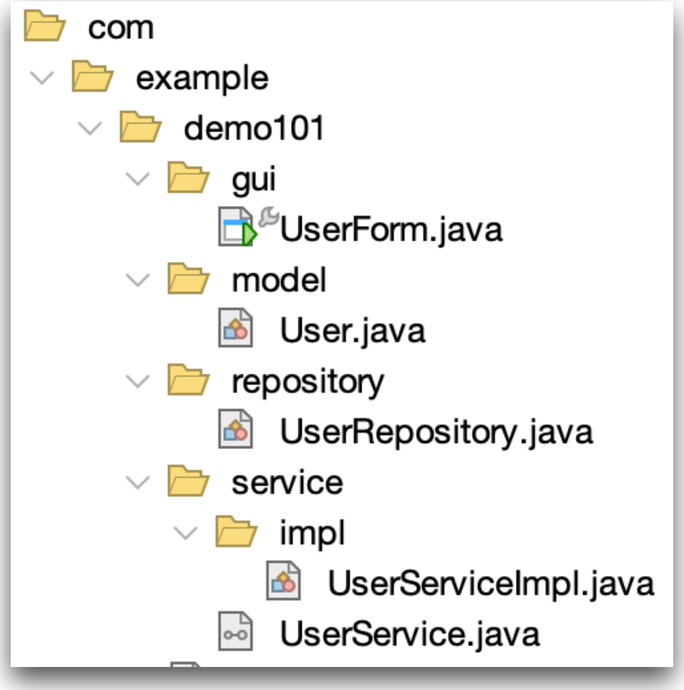
- The most common architecture pattern is the layered architecture pattern, otherwise known as the n-tier architecture pattern.
- This pattern is the de facto standard for most Java EE applications and therefore is widely known by most architects, designers, and developers.
- **Benefits**
  - Improves modularity and maintainability.
  - Enhances scalability and testability.
  - Promotes separation of concerns.

N-Layer Architecture Overview



Ref:
https://www.oreilly.com/library/view/software-architecture-patterns/9781491971437/ch01.html

N-Layer Architecture Overview



Architecture Layer	Purpose	Java Classes / Packages
Presentation Layer	UI input/output, triggers flow	`com.example.demo101.gui.UserForm`
Business Layer	Business logic, validations	`com.example.demo101.service.UserServiceUserServiceImpl`
Persistence Layer	Data access, interacts with database	`com.example.demo101.repository.UserRepository`
Database Layer	Stores data in tables	Database (e.g., MySQL/PostgreSQL)

Model

```
public class User {
    private String userId;
    private String firstName;
    private String lastName;

    public User(String firstName, String lastName) {
        this.firstName = firstName;
        this.lastName = lastName;
    }

  /...
}
```

DTO

```
public record UserDto(
        String firstName,
        String lastName
) {
}
```

Repository

```java
public class UserRepository {
    private static final String URL = "jdbc:mysql://localhost:3306/user_db";
    private final String USER = "root";
    private final String PASSWORD = "college1234";

    public void createUser(User user) {

        try (Connection connection = DriverManager.getConnection(URL, USER, PASSWORD)) {
            String insertQuery = "INSERT INTO miu_users (firstName, lastName) VALUES (?, ?)";
            PreparedStatement preparedStatement = connection.prepareStatement(insertQuery);
            preparedStatement.setString(1, user.getFirstName());
            preparedStatement.setString(2, user.getLastName());
            int noOfRowsInserted = preparedStatement.executeUpdate();
            System.out.println(noOfRowsInserted + " row(s) inserted");
        } catch (Exception exception) {
            System.out.println(exception.getMessage());
        }
    }
  //...
}
```

Service

```
public interface UserService {
    public void save(UserDto userDto);
    List<UserDto> findAll();
}
```

Service

```
public interface UserService {
    public void save(UserDto userDto);
    List<UserDto> findAll();
}
```

Main Point

JDBC provides an API for interacting with a database using SQL. To interact efficiently with a database, you typically use the database vendor's driver that allows communication between the JVM and the database. This is reminiscent of the Principle of Diving – once the initial conditions have been met, a good dive is automatic. (Here, the initial conditions are correct configuration of the data source and code to load the database driver; once the set up is right, interacting with the database is "effortless".)

Connecting the Parts of Knowledge With the Wholeness of Knowledge

1. Since Java is an OO language, it supports storage and manipulation of data within appropriate objects.
2. To work with real data effectively, Java supports interaction with external data stores (databases) through the use of various JDBC drivers, and the JDBC API.

---

3. Transcendental Consciousness: TC is the field of truth, the field of Sat. "Know that by which all else is known." -- Upanishads
4. Wholeness moving within itself: In Unity Consciousness, the final truth about life is realized in a single stroke of knowledge.