MySQL Database & Template Engine

9

Fundamental Web Programming

Asst. Prof. Manop Phankokkruad, Ph.D.
School of Information Technology
King Mongkut's Institute of Technology Ladkrabang

KMITL

Outline

1. Database Integration
2. MySQL database
3. CRUD Operations in MySQL
4. Template Engine

Database Integration

Database integration combines data from diverse sources to create a consolidated version. These sources include databases, the cloud, data warehouses, virtual databases, files, and more. Database integration makes data accessible to multiple stakeholders and client applications without reducing data quality.

Database Integration

- MySQL
- MongoDB
- Oracle
- PostgreSQL
- SQL Server
- SQLite
- Cassandra
- Couchbase
- CouchDB
- LevelDB
- Neo4j
- Redis
- Elasticsearch

MySQL database

2

MySQL is a Relational Database Management system which is free Open-Source Software Under GNU License. It is also supported by Oracle Company . It is fast , scalable, easy to use database management System. MySQL support many operating system like Windows, Linux, MacOS etc.

MySQL is a Structured Query Language which is used to manipulate, manage and retrieve data with the help of various Queries.

KMITL

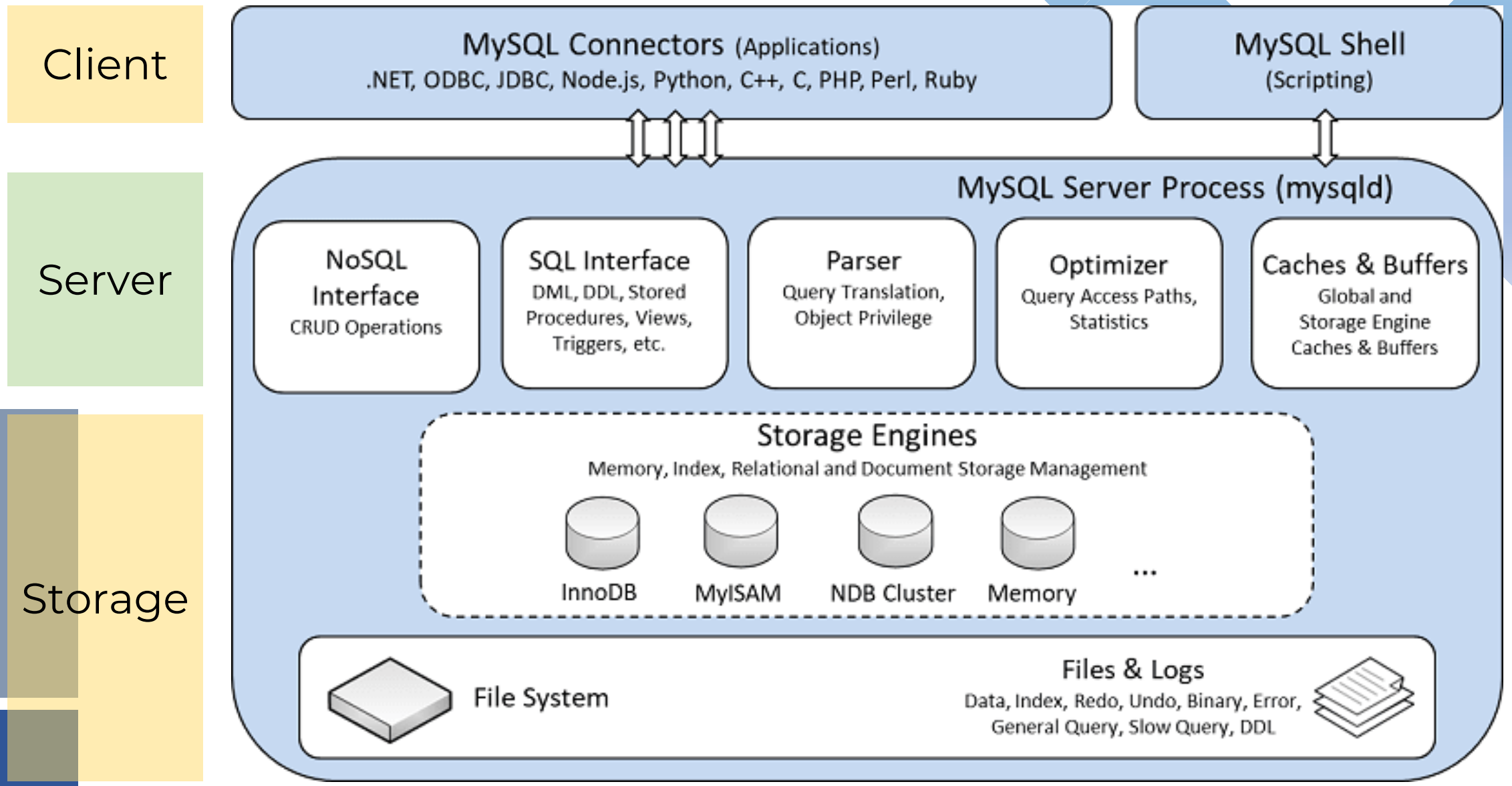
MySQL Architecture

Architecture of MySQL describes the relation among the different components of MySQL System. MySQL follow Client-Server Architecture. It is designed so that end user that is Clients can access the resources from Computer that is server using various networking services. The Architecture of MYSQL contain following major layer's :

- Client Layer
- Server Layer
- Storage Layer

KMITL

MySQL Architecture



KMITL

MySQL Architecture : Client Layer

Client Layer is the topmost layer in the diagram. The Client give request instructions to the Serve with the help of Client Layer .The Client make request through Command Prompt or through GUI screen by using valid MySQL commands and expressions. If the Expressions and commands are valid then the output is obtained on the screen. Some important services of client layer are :

- Connection Handling.
- Authentication.
- Security.

KMITL

MySQL Architecture : Server Layer

Server Layer is responsible for all logical functionalities of relational database management system of MySQL. When the Client give request instructions to the Server and the server gives the output as soon as the instruction is matched. The various subcomponents of MySQL server are:

- Thread Handling
- Parser
- Optimizer
- Query Cache

KMITL

MySQL Architecture : Storage Layer

Storage Layer: This Storage Engine Layer of MySQL Architecture make it's unique and most preferable for developers. In MySQL server, for different situations and requirement's different types of storage engines are used which are InnoDB, MyISAM, NDB, Memory etc. These storage engines are used as pluggable storage engines where tables created by user are plugged with them.

KMITL

CRUD Operations in MySQL

3

MySQL uses SQL to store the data in the form of RDBMS. SQL is the most popular language for adding, accessing, and managing content in a database. It is most noted for its quick processing, proven reliability, ease, and flexibility of use.

MySQL provides a set of some basic but most essential operations that will help user to easily interact with the MySQL database and these operations are known as CRUD operations.

The model must be able to **Create**, **Read**, **Update**, and **Delete** resources.

KMITL

Node.JS & MySQL : Connection

Install MySQL package

- To access a MySQL database with Node.JS, user need a MySQL package. This will use the **mysql** package, downloaded from NPM.
- To download and install the **mysql** package, open the Command Terminal and execute the following:

```
$ npm install mysql
```

Node.JS & MySQL : Connection

Connecting to MySQL

```
var mysql = require('mysql');
```

- Define a connection parameters & options

```
var con = mysql.createConnection({
  host: "localhost",
  user: "yourusername",
  password: "yourpassword"
});
```

KMITL

Node.JS & MySQL : Connection

When establishing a connection, you can set the following options:

- **host**: The hostname of the database you are connecting to. (Default: localhost)
- **port**: The port number to connect to. (Default: 3306)
- **user**: The MySQL user to authenticate as.
- **password**: The password of that MySQL user.
- **database**: Name of the database to use for this connection (Optional).

KMITL

Node.JS & MySQL : Connection

Create a Connection with MySQL database

```
con.connect(function(err) {
  if (err) throw err;
  console.log("Connected!");
});
```

End the connection when done

```
con.end((err) => {
if (err)
   console.error('Error closing MySQL connection:', err);
else
   console.log('Connection closed'); });
```

KMITL

Node.JS & MySQL : Creating a Database

Create a database using SQL command

```
con.connect(function(err) {
  if (err) throw err;
  console.log("Connected!");
  con.query("CREATE DATABASE mydb", function (err, result) {
    if (err) throw err;
    console.log("Database created");
  });
});
```

Node.JS & MySQL : Creating a Table

The CREATE TABLE statement is used to create a table in MySQL.

```
con.connect(function(err) {
  if (err) throw err;
  console.log("Connected!");
  var sql = "CREATE TABLE customers (name VARCHAR(255), address VARCHAR(255))";
  con.query(sql, function (err, result) {
    if (err) throw err;
    console.log("Table created");
  });
});
```

KMITL

CRUD Operations

Create operation : User can start adding data in them. The INSERT INTO statement is used to add new records to a MySQL table.

```
const addUser = (user) => {
  const query = 'INSERT INTO users SET ?';
  connection.query(query, user, (err, result) => {
    if (err) throw err;
    console.log('User added:', result.insertId);
  });
};
```

KMITL

CRUD Operations

Read operation : SELECT statement is used to select data from one or more tables.

```
const getUsers = () => {
  const query = 'SELECT * FROM users';
  connection.query(query, (err, results) => {
    if (err) throw err;
    console.log('Users:', results);
  });
};
```

KMITL

CRUD Operations

Update operation : UPDATE statement is used to update existing records in a table. The following examples update the record in the table.

```
const updateUser = (id, user) => {
  const query = 'UPDATE users SET ? WHERE id = ?';
  connection.query(query, [user, id], (err, result) => {
    if (err) throw err;
    console.log('User updated:', result.affectedRows);
  });
};
```

KMITL

Execute Queries : DELETE

Delete operation : DELETE statement is used to delete records from a table. The following examples delete the record in the table.

```
const deleteUser = (id) => {
  const query = 'DELETE FROM users WHERE id = ?';
  connection.query(query, id, (err, result) => {
    if (err) throw err;
    console.log('User deleted:', result.affectedRows);
  });
};
```

KMITL

Template engine

4

Template engine is a software tool that combines templates with data to create documents or programs. They are commonly used to build web applications and render server-side data. Template engine converts variables to values and changes the template to HTML files to send to the client.

Popular template engines such as

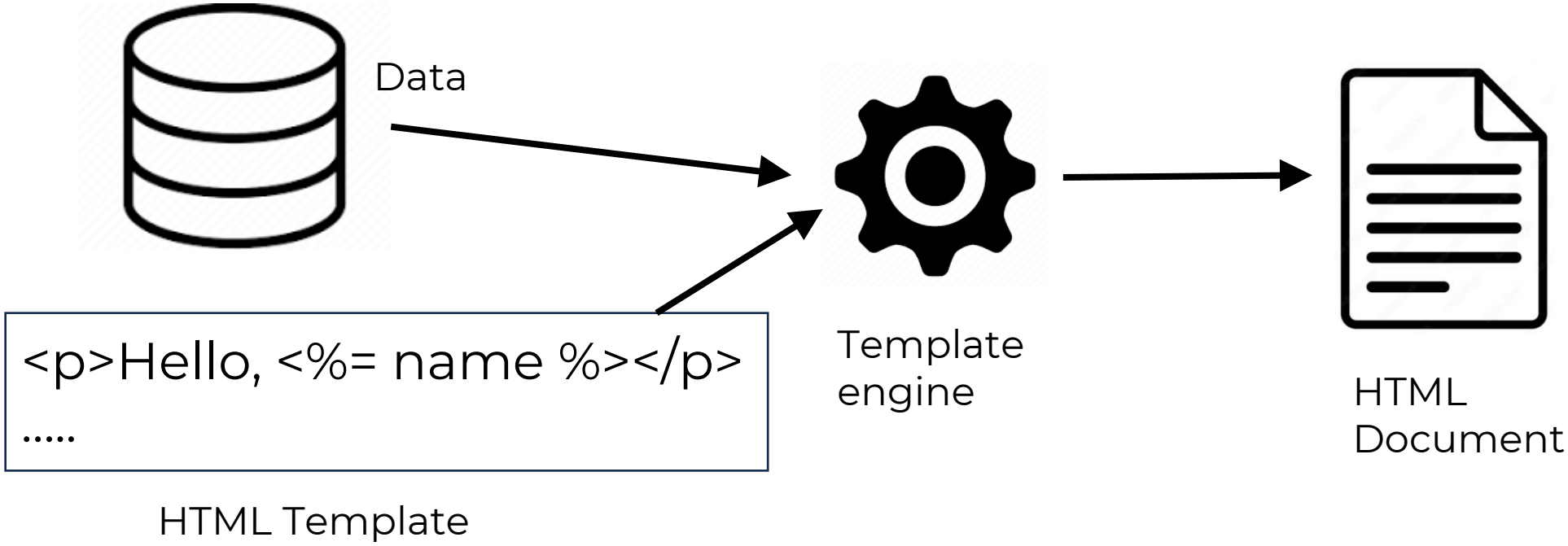
EJS , Jade, Pug, Mustache, HandlebarsJS, Blade.

KMITL

Template engine

How template engine work

1. takes a template file with variables
2. replaces the variables with actual values
3. The result is a document or program



Template engine

Benefits of Using Templating Engines:

- Enhances the productivity of a developer.
- Improves the readability, usability and maintainability of the application.
- Fastens the performance of the application.
- Maximizes client-side data processing.
- Makes it easy to reuse templates on multiple pages.
- Enables accessibility of templates from Content Delivery Networks(CDNs).

KMITL

Template engine

Embedded JavaScript (**EJS**) is a popular templating engine for Node.js that allows you to generate HTML markup with plain JavaScript. It is particularly useful for creating dynamic web pages, as it enables you to embed JavaScript logic directly within your HTML. Key Features of EJS is follows.

- **Embedded JavaScript:** Allows you to embed JavaScript logic directly within your HTML.
- **Partial Templates:** Supports partials, enabling you to reuse common template fragments (like headers and footers) across different pages.
- **Layout Support:** EJS can be used with layout managers to create consistent layouts across multiple views.

KMITL

Embedded JavaScript

- Installing EJS

```
$ npm install express ejs
```

- Set EJS as templating engine

```
const express = require('express');
const app = express();

// Set EJS as templating engine
app.set('view engine', 'ejs');
```

KMITL

Embedded JavaScript

- Create the EJS Template

home.ejs

```ejs
<!-- Home.ejs -->

<!DOCTYPE html>
<html>
<head>
    <title>Home Page</title>

    <style type="text/css" media="screen">
        body {
            background-color: skyblue;
            text-decoration-color: white;
            font-size: 7em;
        }
    </style>
</head>

<body>
    <center>This is our home page.</center>
</body>
</html>
```

KMITL

Embedded JavaScript

The page home.ejs will be displayed on requesting localhost. To add dynamic content this render method takes a second parameter which is an object.

index.js

```
app.get('/', (req, res) => {

    // The render method takes the name of the HTML
    // page to be rendered as input
    // This page should be in the views folder
    // in the root directory.
    res.render('home');

});
```

KMITL

Embedded JavaScript

- **EJS syntax**

```
<startingTag content closingTag>
```

```
<%= user.firstName %>
```

- EJS has different tags for different purposes. This start tag **<%=** is called the “escape output” tag because if the string in the content has forbidden characters like > and &, the characters will be escaped (replaced by HTML codes) in the output string.

KMITL

Embedded JavaScript

- Passing data to render #1

index.js

```
require('express')
ess()
const port = 3000
app.set('view engine', 'ejs')
const user = {
    firstName: 'Tim',
    lastName: 'Cook',
    admin: true,
}
app.get('/', (req, res) => {
    res.render('pages/index', {
        user
    })
})
app.listen(port, () => {
  console.log(`App listening at port ${port}`)
})
```

Index.ejs

```
<h1>Hi, I am <%= user.firstName  %></h1>
<% if (user.admin) { %>
    <p>Let me tell you a secret: <b>I am an admin</b></p>
<% } %>
```

KMITL

Embedded JavaScript

- Passing data to render #2

```
app.get('/', (req, res) => {

    // The render method takes the name of the HTML
    // page to be rendered as input.
    // This page should be in views folder in
    // the root directory.
    // We can pass multiple properties and values
    // as an object, here we are passing the only name
    res.render('home', { name: 'Akashdeep' });
});

const server = app.listen(4000, function () {
    console.log('listening to port 4000')
});
```

index.js

home.ejs

```
<!DOCTYPE html>
<html>
<head>
    <title>Home Page</title>
    <style type="text/css" media="screen">
        body {
            background-color: skyblue;
            text-decoration-color: white;
            font-size: 7em;
        }
    </style>
</head>

<body>
    <center>
        This is our home page.<br />
        Welcome <%=name%>, to our home page.
    </center>
</body>
</html>
```

KMITL