National University of Computer and Emerging Sciences

Laboratory Manuals

for

Database Systems Lab

(CL -2005)

Course Instructor	Ms. Aleena Ahmad
Lab Instructor	Seemab Ayub
Section	BCS-4E
Semester	Spring 2025

Department of Computer Science
FAST-NU, Lahore, Pakistan

Lab Manual 08

SQL

SQL tutorial gives unique learning on Structured Query Language and it helps to make practice on SQL commands which provides immediate results. SQL is a language of database, it includes database creation, deletion, fetching rows and modifying rows etc. SQL is an ANSI (American National Standards Institute) standard, but there are many different versions of the SQL language.

Why SQL?

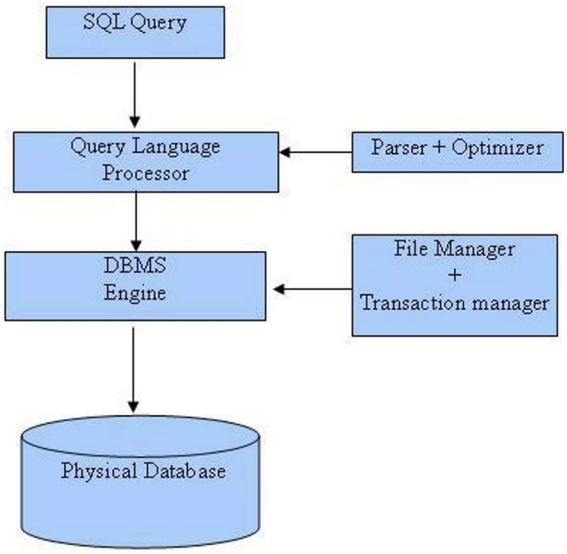
- Allows users to access data in relational database management systems.
- Allows users to describe the data.
- Allows users to define the data in the database and manipulate that data.
- Allows embedding within other languages using SQL modules, libraries & pre-compilers.
- Allows users to create and drop databases and tables.
- Allows users to create views, stored procedure, functions in a database.
- Allows users to set permissions on tables, procedures and views

SQL Process

When you are executing an SQL command for any RDBMS, the system determines the best way to carry out your request and SQL engine figures out how to interpret the task.

There are various components included in the process. These components are Query Dispatcher, Optimization Engines, Classic Query Engine and SQL Query Engine, etc. Classic query engine handles all non-SQL queries, but SQL query engine won't handle logical files.

Following is a simple diagram showing SQL Architecture:



1. Introduction to React

React is a JavaScript library used to build user interfaces. It allows developers to create reusable UI components that manage their own state and update efficiently in response to user interactions.

---

2. What are Components?

Components are the building blocks of a React application. They are JavaScript functions or classes that return React elements (JSX).

Example Functional Component:

```
function Welcome(props) {
  return <h1>Hello, {props.name}</h1>;
}
```

---

3. JSX - JavaScript XML

JSX is a syntax extension for JavaScript. It looks like HTML and is used to describe what the UI should look like.

Example:

```
const element = <h1>Hello, world!</h1>;
```

JSX allows embedding expressions, assigning classes, and more.

---

4. Props (Properties)

Props are arguments passed into components. They allow data to be passed from parent to child components.

Example:

```
function Greet(props) {
  return <p>Good morning, {props.name}</p>;
```

```
}
```

Use props in a parent component:

```
<Greet name="Alice" />
```

---

5. State and Lifecycle

State is a built-in object that stores property values that belong to a component. When state changes, the component re-renders.

Class Component Example:

```
class Counter extends React.Component {
  constructor(props) {
    super(props);
    this.state = { count: 0 };
  }

  render() {
    return <h1>{this.state.count}</h1>;
  }
}
```

In functional components, use `useState`.

---

6. Functional Components vs Class Components

- **Functional Components**: Simpler syntax, use Hooks for state and lifecycle.
- **Class Components**: More complex, use `this.state` and lifecycle methods like `componentDidMount`.

---

7. Hooks: useState and useEffect

Hooks are special functions in React 16.8+ that let you use state and other features without writing a class.

useState Example:

```
import { useState } from 'react';


function Counter() {
  const [count, setCount] = useState(0);


  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>Click me</button>
    </div>
  );
}
```

useEffect Example:

```
import { useEffect } from 'react';


useEffect(() => {
  console.log('Component did mount');
}, []); // Empty array means run once on mount
```

---

8. Conditional Rendering

React allows rendering different UI elements based on conditions.

Example:

```
function Greeting(props) {
  if (props.isLoggedIn) {
    return <h1>Welcome back!</h1>;
  } else {
    return <h1>Please sign in.</h1>;
  }
}
```

---

9. List Rendering and Keys

Use `map()` to render lists. Keys help React identify which items changed.

Example:

```
const items = ['Apple', 'Banana', 'Cherry'];

const listItems = items.map((item, index) =>
  <li key={index}>{item}</li>
);
```

---

10. Component Composition

Components can be nested inside others. This allows for complex UIs built from simple parts.

Example:

```
function App() {
  return (
```

```
    <div>
      <Header />
      <Content />
      <Footer />
    </div>
  );
}
```

---

11. Summary and Best Practices

- Use functional components with Hooks.
- Keep components small and focused.
- Pass data via props and manage local state properly.
- Use keys in lists.
- Use JSX for templating.
- Separate concerns and reuse components.

---

1. Introduction to Routing in React

When you use React to build an app, your application typically runs as a **single-page application** (SPA). This means the entire app is loaded on one page, and navigation between different views (like "Home" or "Profile" pages) doesn't cause the whole page to reload. **Routing** is a technique that helps switch between different views in your app without refreshing the page.

React Router is a tool that helps you add routing to your React app.

- **Routing** is like the map of your app, guiding the user from one page or view to another.
- It enables **navigation** between different "pages" (components) of your app.

---

2. Setting Up React Router

To begin using React Router, you need to install it first. This is done using **npm**, which stands for Node Package Manager.

npm is a tool that helps you download and manage third-party code libraries. React Router is one of these libraries.

Install React Router

In your project folder, open the terminal (a command line interface) and type:

```
npm install react-router-dom
```

This command tells **npm** to download and install React Router into your app. You will now be able to use it in your code.

Using React Router in Your App

Now, you need to **set up** React Router in your app to make sure it works.

In your main JavaScript file (usually `index.js` or `main.jsx`), you need to **wrap** your entire app inside a `BrowserRouter` component.

```
import { BrowserRouter } from 'react-router-dom';
import ReactDOM from 'react-dom';
import App from './App';
const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <BrowserRouter>
    <App />
  </BrowserRouter>
);
```

Explanation:

- **`BrowserRouter`** is a special component that makes React Router work. It enables navigation between different views in your app.
- **`ReactDOM.createRoot()`** is a function that starts up your React app and makes it appear on the webpage.
- **`document.getElementById('root')`** is how React knows where to display the app in the HTML file. It's like telling React, "Place the app inside this section of the webpage."

---

3. Creating Routes and Links

Now, you can define different **routes** (pages) inside your app. For this, you'll use the `Routes` and `Route` components from React Router.

Setting Up Routes

In your `App.js` file:

```
import { Routes, Route, Link } from 'react-router-dom';
import Home from './Home';
import SignIn from './SignIn';
import SignUp from './SignUp';


function App() {
  return (
    <div>
      <nav>
        <Link to="/">Home</Link>
        <Link to="/signin">Sign In</Link>
        <Link to="/signup">Sign Up</Link>
```

```
      </nav>


      <Routes>
        <Route path="/" element={<Home />} />
        <Route path="/signin" element={<SignIn />} />
        <Route path="/signup" element={<SignUp />} />
      </Routes>
    </div>
  );
}
```

Explanation of Concepts:

- **`<Link>`**: This component works like an anchor (`<a>`) tag in HTML. It allows users to click on links to navigate to different parts of your app.
    - `to="/"` means that when you click the link, it takes the user to the homepage (`/`).
- **`<Routes>`**: This component is like a container that holds all the different **routes** of your app. It checks the current URL and displays the correct component based on the path.
- **`<Route>`**: This component is used to define a specific **route** (page) in your app. Each route has a `path` (URL) and an `element` (the component that gets displayed).
    - For example, `path="/signin"` means that when the URL is `/signin`, the `SignIn` component is displayed.

---

4. What is Authentication?

Authentication is the process of verifying the identity of a user. It's like checking the ID of someone to make sure they are who they say they are.

In web applications, **authentication** is usually done using **user credentials** like:

- **Username**
- **Email**
- **Password**

The app compares the credentials entered by the user with stored data to confirm the user's identity. If the credentials match, the user is **authenticated**, and they can access certain pages or features in the app.

---

5. Creating Sign In and Sign Up Forms

Forms are the way users input data (like email and password). To create a **Sign In** and **Sign Up** form, you use HTML input fields inside React components.

Example: Sign Up Form

```
import { useState } from 'react';


function SignUp() {
  const [email, setEmail] = useState('');
  const [password, setPassword] = useState('');


  const handleSubmit = (e) => {
    e.preventDefault();
    alert(`Signed up with Email: ${email}`);
  };
```

```
  return (
    <form onSubmit={handleSubmit}>
      <h2>Sign Up</h2>
      <input
        type="email"
        placeholder="Email"
        value={email}
        onChange={(e) => setEmail(e.target.value)}
      />
      <input
        type="password"
        placeholder="Password"
        value={password}
        onChange={(e) => setPassword(e.target.value)}
      />
      <button type="submit">Sign Up</button>
    </form>
  );
}
```

Explanation of Key Concepts:

- **`useState`**: This is a React function (hook) used to **store** and **track** values like the email and password typed by the user.
    - `const [email, setEmail] = useState('')`: This creates a state variable `email` to store the value typed by the user.
    - `setEmail(e.target.value)` updates the `email` state every time the user types.
- **onChange**: This is an event handler. It runs every time the user types into an input field. It updates the state to reflect the new value the user enters.
- **`handleSubmit`**: This function is triggered when the form is submitted. The `e.preventDefault()` prevents the form from reloading the page, which is the default behavior for forms. It also displays the email entered by the user.

---

6. Managing Form Data with useState

React makes forms easier to manage by connecting the form fields to **state**.

Here's a basic example of connecting an input field to state:

```
const [value, setValue] = useState('');


<input value={value} onChange={(e) => setValue(e.target.value)} />
```

Explanation:

- **`value={value}`**: The input field shows the current value stored in the `value` state.
- **`onChange={(e) => setValue(e.target.value)}`**: Every time the user types something, the `setValue` function updates the state with the new value.

This setup ensures that React is **in control** of the form field's data. This is called a **controlled component**.

---

7. Simulating Authentication with Local State

In a real app, you would check a backend server to verify the user's credentials. For simplicity, we'll simulate this by using local state.

```
import { useState } from 'react';


function App() {
  const [isAuthenticated, setIsAuthenticated] = useState(false);


  return (
    <div>
      {isAuthenticated ? <p>Welcome!</p> : <p>Please log in</p>}
      <button onClick={() =>
setIsAuthenticated(true)}>Login</button>
    </div>
  );
}
```

Explanation:

- **`isAuthenticated`**: This state variable holds whether the user is logged in (`true`) or not (`false`).

- **`setIsAuthenticated(true)`**: When the button is clicked, it updates the state to `true`, which simulates the user logging in.

- The **conditional rendering** (`{isAuthenticated ? <p>Welcome!</p> : <p>Please log in</p>}`) changes what is displayed based on the authentication status.

---

8. Navigation after Login

After successful login, you usually want to navigate the user to a different page. To do this, you can use the `useNavigate` hook from React Router.

```
import { useNavigate } from 'react-router-dom';


const navigate = useNavigate();

navigate('/');
```

Explanation:

- **`useNavigate()`**: This is a React Router hook that allows you to programmatically navigate between pages in your app.

- **`navigate('/')`**: This tells React Router to go to the homepage (`/`), typically after a successful login.

---

9. Protecting Routes (Concept)

Sometimes, you want to restrict access to certain pages for users who are not logged in. This is called **route protection**.

You can achieve this by checking whether the user is authenticated before allowing them to see certain pages.

```
function ProtectedRoute({ isAuthenticated, children }) {
  if (!isAuthenticated) {
    return <Navigate to="/signin" />;
  }
  return children;
}
```

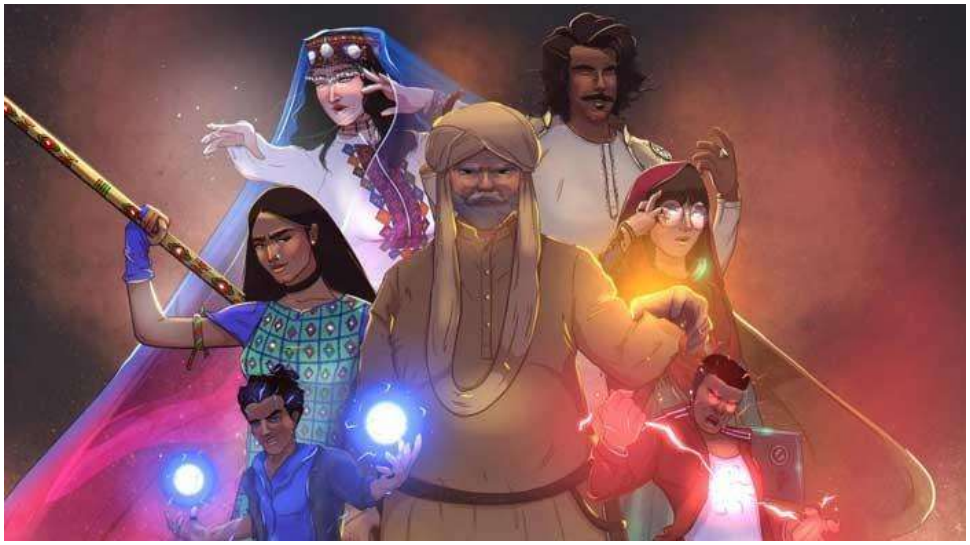
Explanation:

- **`ProtectedRoute`**: This is a custom component that checks if the user is authenticated.
- **`Navigate`**: If the user is not authenticated, the `Navigate` component redirects them to the login page (`/signin`).
- **`children`**: This represents the protected content that only authenticated users can access.

---

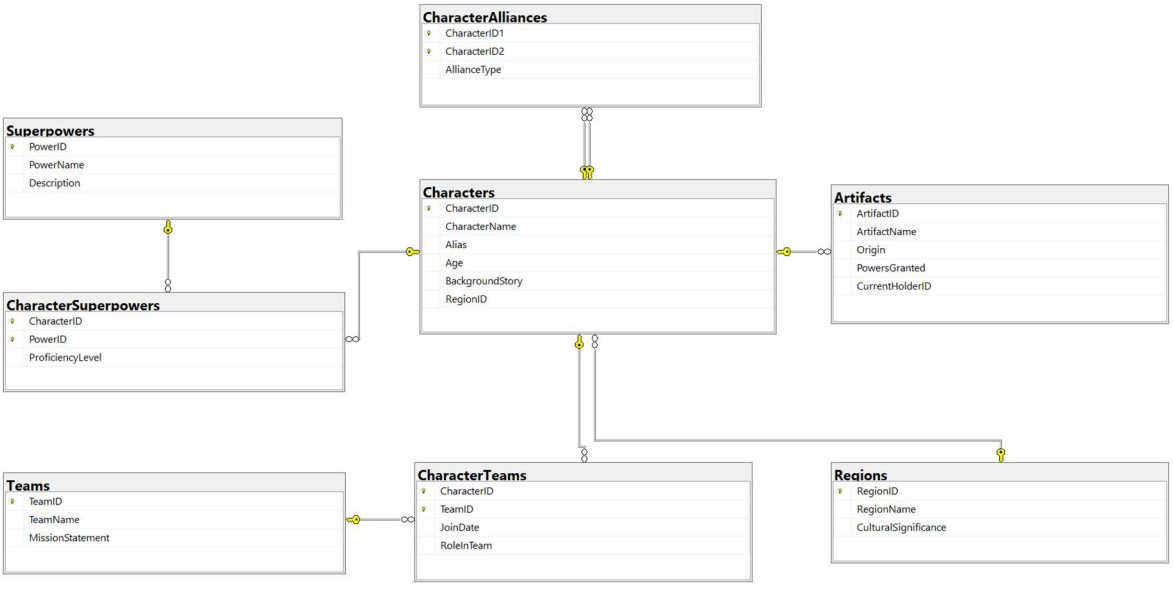
10. Summary and Best Practices

- **Use `react-router-dom`** to manage navigation between different parts of your app.
- **Use `useState`** to manage form data and authentication status.
- **Simulate login** with state for learning purposes before integrating with a real authentication service.
- **Use `Navigate`** to protect routes and ensure only authenticated users can access certain pages.



Happy learning and querying!

Download sql file provided with this manual. Run that file and a schema with following details will be created:



In Lab Exercises

Justice League Pakistan: Heroes Unite

In a world where Pakistani superheroes come together to fight for justice, users are heroes joining a secret league to access exclusive missions, resources, and training. The login/signup system is their gateway to becoming part of this heroic team..

Write a few queries to getting started with this lab work

1. List all alliances and superpowers count for each alliance.
2. List all teams having more than one Artifact owner.
3. Name the character who has more than one artifact and more than average count of superpowers

Also help designing few react components supporting following functional requirements

1. A hero should be able to sign up on localhost through a signup form
2. A hero should be able to sign in on localhost through a signin form
3. A hero should be able to get his alliance info on Home Page
4. A hero should be able to list his super powers on home Page

5. A hero should be able to see his teams on the home page.

Submission Guidelines

---

1. submit following files strictly following the naming convention: l231234.sql
2. zip of src folders from frontend and backend

Best of Luck! Happy Querying
