SELECT column1, column2, ...

FROM table1

JOIN table2 ON table1.column = table2.column

WHERE condition

ORDER BY column1 [ASC | DESC], column2 [ASC | DESC], ...

COUNT, SUM, AVG, MIN, MAX, GROUP\_CONCAT

**Second largest salary**

SELECT MAX(salary) AS second\_largest\_salary

FROM employees

WHERE salary < (SELECT MAX(salary) FROM employees);

SELECT \* FROM employees WHERE salary = (

SELECT MAX(salary)

FROM employees

WHERE salary < (

SELECT MAX(salary)

FROM employees

)

);

1NF – eliminate duplicate columns

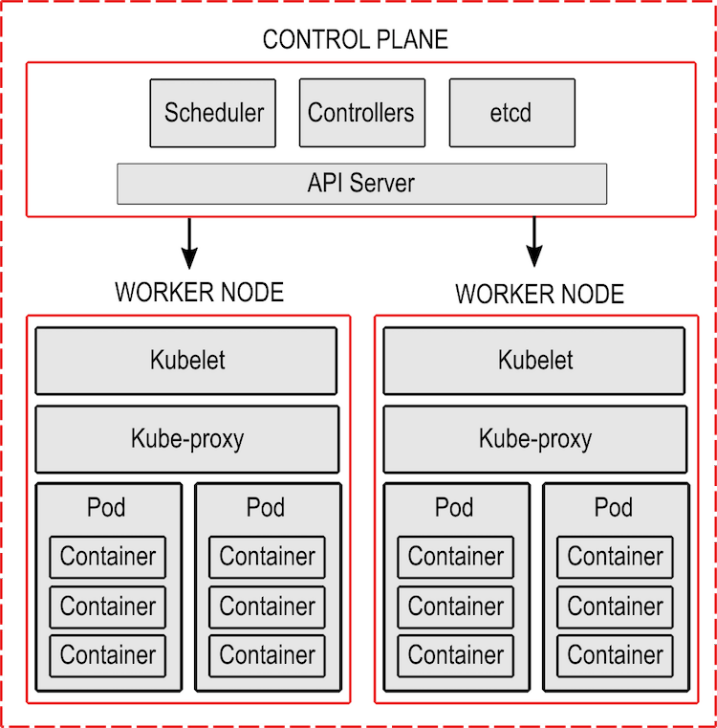
2NF – remove subsets of data

3NF – remove columns not dependent on primary key

4NF – remove multi valued dependencies

5NF – address join dependencies

**Kubernetes**



Services for networking

* ClusterIP (No external access)
* NodePort (external access)
* Externalname
* Loadbalancer (to internet)

Deployment – Ensure correct pod count

**Pod**

apiVersion: v1

kind: Pod

metadata:

name: my-pod

labels:

app: my-app

spec:

containers:

- name: my-container

image: nginx:latest

Rolling updates

Rollbacks

**Probes**

* Livenes
* Ready
* Startup

spec:

containers:

- name: my-container

image: my-image:latest

ports:

- containerPort: 8080

livenessProbe:

httpGet:

path: /healthz

port: 8080

initialDelaySeconds: 15

periodSeconds: 10

readinessProbe:

httpGet:

path: /readyz

port: 8080

initialDelaySeconds: 20

periodSeconds: 10

**Docker**

# Use an official Node.js runtime as a base image

FROM node:14

# Set environment variables

ENV NODE\_ENV=production

ENV PORT=3000

# Set the working directory in the container

WORKDIR /app

# Copy package.json and package-lock.json to the container

COPY package\*.json ./

# Install dependencies

RUN npm install

# Copy the rest of the application code to the container

ADD folder/path.

# Expose the port the app runs on

EXPOSE $PORT

# Set the entry point of the container

ENTRYPOINT ["node", "app.js"]

# Set default command arguments

CMD ["--port", "3000"]

version: '3.8'

services:

frontend:

build: ./frontend

ports:

- "80:80"

depends\_on:

- backend

backend:

build: ./backend

ports:

- "8000:8000"

**AWS**

* Users | Groups | Roles | Policies | Permissions
* MFA | Identity Federation | Access Analyzer

**AWS Lambda**

Function (Code to process)

Trigger (invoke lambda)

Event (Json)

**EC2**

On demand | Reserved | Spot instance | Dedicated hosts

Traceid - assigned to a single request or transaction

Spanid - unique identifier assigned to each operation

Correlationid - identifier used to correlate related events or logs

**ES6**

|  |  |  |
| --- | --- | --- |
| const | string template | classes |
| block scope | destructuring | modules export |
| arrow function | promises |  |

function delay(ms) {

return new **Promise**(resolve => setTimeout(resolve, ms));

}

**async** function asyncExample() {

**await** delay(2000);

console.log('After 2 seconds');

}

asyncExample();

**React – Class component**

import React, { Component } from 'react';

class MyComponent extends Component {

constructor(props) {

super(props);

this.state = {

count: 0

};

this.handleClick = this.handleClick.bind(this);

}

handleClick() {

this.setState(prevState => ({

count: prevState.count + 1

}));

}

render() {

return (

<div>

<h1>Count: {this.state.count}</h1>

<button onClick={this.handleClick}>Increment</button>

</div>

);

}

}

export default MyComponent;

Mounting:

constructor(props)

componentDidMount()

Updating:

shouldComponentUpdate(nextProps, nextState)

componentDidUpdate(prevProps, prevState)

Unmounting:

componentWillUnmount()

Error Handling:

componentDidCatch(error, info)

**React – Fuction component**

import React from 'react';

// Functional component definition

const MyComponent = () => {

return (

<div>

<h1>Hello, React!</h1>

<p>This is a functional component.</p>

</div>

);

};

export default MyComponent;

**Hooks** (hook into react features such as state/lifecycle)

**useState** – const [color, setColor] = useState(“red”)

**useEffect –** perform side effect

useEffect( ()=> {setTimeout( ())} , [])

**useContext**

* Manage state globally
* Use together with useState

**useReducer**

* **useReducer(reducer, initialstate)**

import React, { useReducer } from 'react';

const reducer = (state, action) => {

}

const [state, dispatch] = useReducer(reducer, { count: 0 });

button onClick={() => dispatch({ type: 'increment' })}>Increment</button>

**useRef** (create a mutable ref object whose **current** property can hold a reference to a DOM element or any other mutable value.)

import React, { useRef } from 'react';

const MyComponent = () => {

const inputRef = useRef(null);

const focusInput = () => {

inputRef.current.focus();

};

return (

<div>

<input ref={inputRef} type="text" />

<button onClick={focusInput}>Focus Input</button>

</div>

);

};

export default MyComponent;

**Redux**

Single source of truth

State – read only

- Appln global state

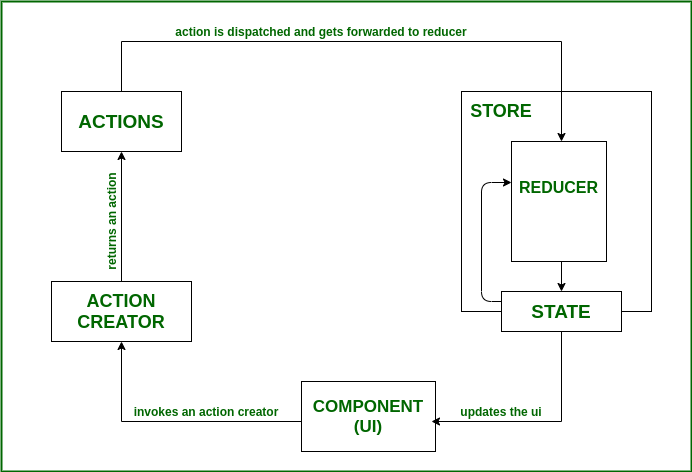
- Never modify directly

- Action dispatch

Action - {type: <> , payload:<>}

Reducer

Changes made with pure functions



Cetralising application state

Single store per application

For creating UI

Business in reducer

Easy state tnsfrmtn bw component

Can store application state

**Route**

import React from 'react';

import { **BrowserRouter as Router, Route, Switch** } from 'react-router-dom';

import Home from './components/Home';

import About from './components/About';

import NotFound from './components/NotFound';

const App = () => {

return (

**<Router>**

**<Switch>**

<Route exact path="/" component={Home} />

<Route path="/about" component={About} />

<Route component={NotFound} />

</Switch>

</Router>

);

};

<Route path="/users/:id" component={UserDetails} />

<Link to="/about">About</Link>

export default App;

BrowserRouter

Route (dynamic routes with parameters)

Switch

Link

Redirect

Programmatic naviation

const history = useHistory();

history.push('/about');