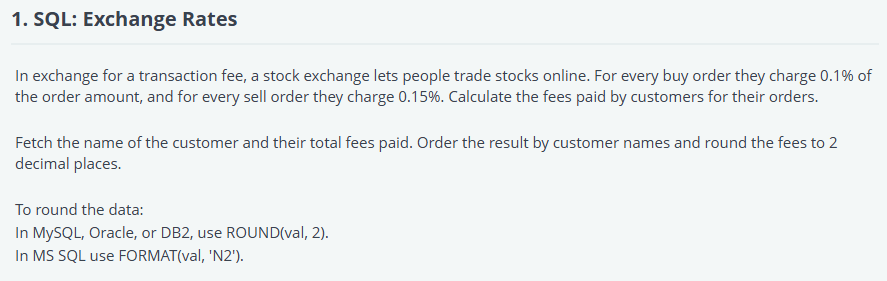
# SQL: Exchange Rates



/\*

Enter your query below.

Please append a semicolon ";" at the end of the query

\*/

SELECT

    c.customer\_name,

    ROUND(SUM(

        CASE

            WHEN o.order\_type = 'Buy' THEN o.order\_amount \* 0.001  -- 0.1% fee for Buy orders

            WHEN o.order\_type = 'Sell' THEN o.order\_amount \* 0.0015 -- 0.15% fee for Sell orders

        END

    ), 2) AS total\_fees

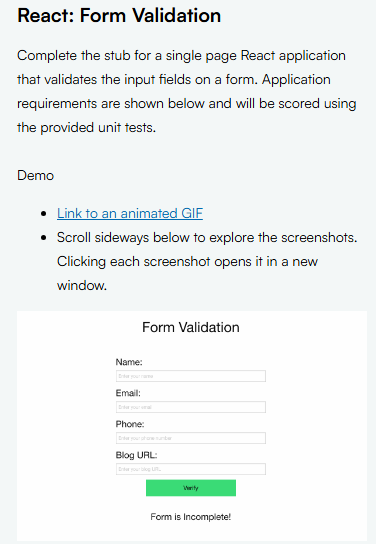
FROM customers c

JOIN orders o ON c.id = o.customer\_id

GROUP BY c.customer\_name

ORDER BY c.customer\_name;

# React: Form Validation



## src/App.js

import React, { useState } from 'react';

import Form from './components/Form';

import Message from './components/Message';

const App = () => {

  const [formStatus, setFormStatus] = useState('Incomplete');

  const handleFormValidation = (isValid) => {

    if (isValid) {

      setFormStatus('Complete');

    } else {

      setFormStatus('Incomplete');

    }

  };

  return (

    <div>

      <Form onValidate={handleFormValidation} />

      <Message formStatus={formStatus} />

    </div>

  );

};

export default App;

## src/components/Form.js

import React, { useState } from 'react';

const Form = ({ onValidate }) => {

const [name, setName] = useState('');

const [email, setEmail] = useState('');

const [phone, setPhone] = useState('');

const [url, setUrl] = useState('');

const validateName = (name) => /^[A-Za-z]{3,30}$/.test(name);

const validateEmail = (email) => /^[a-zA-Z0-9.\_%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$/.test(email);

const validatePhone = (phone) => /^[2-9]{1}[0-9]{9}$/.test(phone);

const validateUrl = (url) => /^(https?:\/\/)?(www\.)?[a-zA-Z0-9-]+\.[a-z]{2,6}$/.test(url);

const validateForm = () => {

const isValid = validateName(name) && validateEmail(email) && validatePhone(phone) && validateUrl(url);

onValidate(isValid);

};

const handleSubmit = (e) => {

e.preventDefault();

validateForm();

};

return (

<div className="row">

<h1 className="text-center">Form Validation</h1>

<form onSubmit={handleSubmit}>

<label>

<h3 aria-label="Name">Name:</h3>

<input

data-testid="name-input"

value={name}

onChange={(e) => setName(e.target.value)}

/>

</label>

<label>

<h3 aria-label="Email">Email:</h3>

<input

data-testid="email-input"

value={email}

onChange={(e) => setEmail(e.target.value)}

/>

</label>

<label>

<h3 aria-label="Phone">Phone:</h3>

<input

data-testid="phone-input"

value={phone}

onChange={(e) => setPhone(e.target.value)}

/>

</label>

<label>

<h3 aria-label="URL">Blog URL:</h3>

<input

data-testid="url-input"

value={url}

onChange={(e) => setUrl(e.target.value)}

/>

</label>

<div className="small-6 small-centered text-center columns">

<button

data-testid="btn"

type="submit"

className="button success expand round text-center"

>

Verify

</button>

</div>

</form>

</div>

);

};

export default Form;

## src/components/Message.js

import React from 'react';

const Message = ({ formStatus }) => {

  return (

    <div>

      <h3 className="text-center message" data-testid="message">

        {formStatus === 'Complete' ? 'Form is Complete!' : 'Form is Incomplete!'}

      </h3>

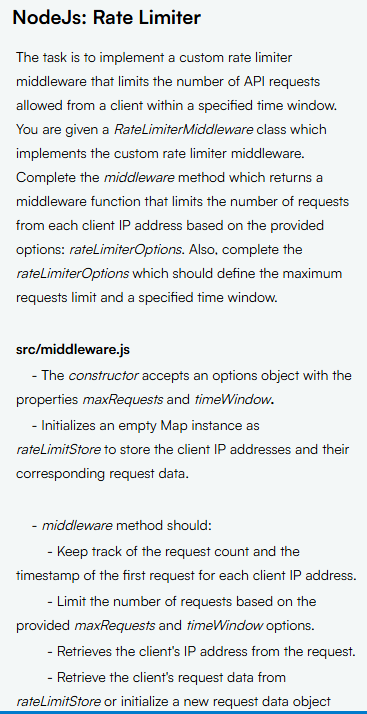
    </div>

  );

};

export default Message;

# NodeJs: Rate Limiter



## src/middleware.js

class RateLimiterMiddleware {

  constructor(options) {

    this.rateLimitStore = new Map();

    this.options = options;

  }

  middleware() {

    return (req, res, next) => {

      const clientIP = req.ip; // Retrieve client's IP address

      const currentTimestamp = Date.now();

      // Retrieve the client's request data or initialize it if it's the first request

      let requestData = this.rateLimitStore.get(clientIP) || { requestCount: 0, firstRequestTime: currentTimestamp };

      // Check if the time window has passed

      if (currentTimestamp - requestData.firstRequestTime > this.options.timeWindow) {

        // Reset request count and timestamp if the time window has expired

        requestData.requestCount = 0;

        requestData.firstRequestTime = currentTimestamp;

      }

      // Check if the request count exceeds the max allowed requests

      if (requestData.requestCount >= this.options.maxRequests) {

        res.setHeader("X-RateLimit-Remaining", 0);

        res.status(429).json({ message: "You have exceeded the rate limit. Please try again later." });

      } else {

        // Increment request count if the rate limit is not exceeded

        requestData.requestCount++;

        this.rateLimitStore.set(clientIP, requestData);

        // Calculate remaining requests

        const remainingRequests = this.options.maxRequests - requestData.requestCount;

        const resetTimestamp = requestData.firstRequestTime + this.options.timeWindow;

        res.setHeader("X-RateLimit-Limit", this.options.maxRequests); // Max requests allowed

        res.setHeader("X-RateLimit-Remaining", remainingRequests);  // Remaining requests within current time window

        res.setHeader("X-RateLimit-Reset", resetTimestamp); // Reset time for rate limit

        next(); // Proceed with the request

      }

    };

  }

  reset() {

    this.rateLimitStore.clear(); // Reset the rate limit store for all clients

  }

}

module.exports = RateLimiterMiddleware;

## src/options.js

const rateLimiterOptions = {

  maxRequests: 5,  // Maximum number of requests per client within the time window

  timeWindow: 1000 // Time window duration in milliseconds (1 second)

};

module.exports = rateLimiterOptions;