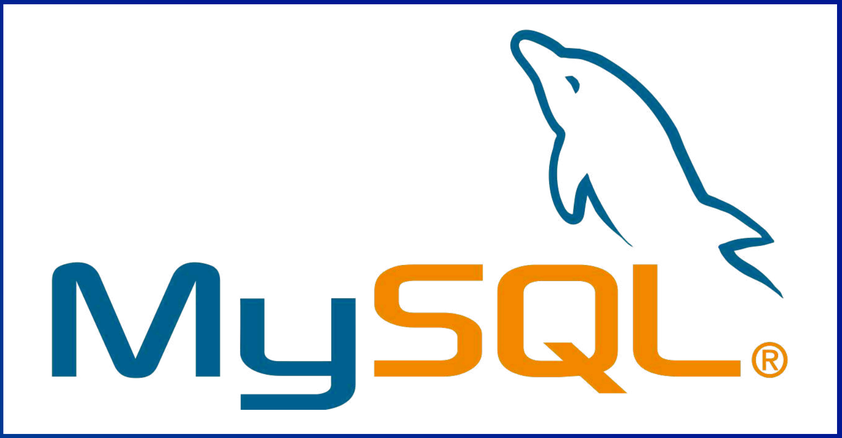
09. More on Backend + Database

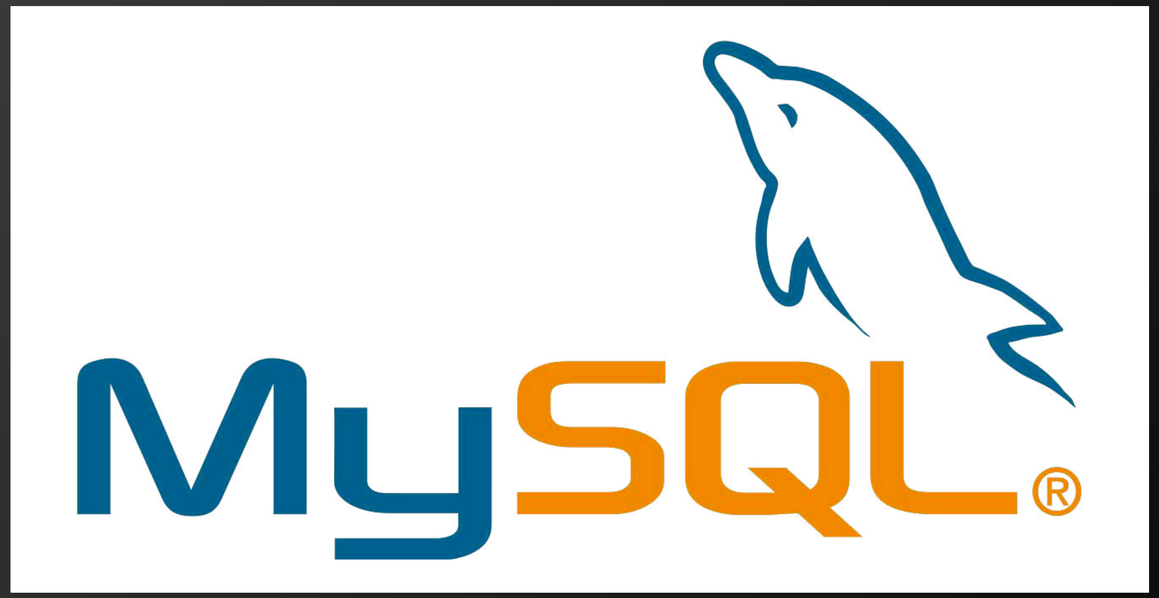


Ric Huang / NTUEE

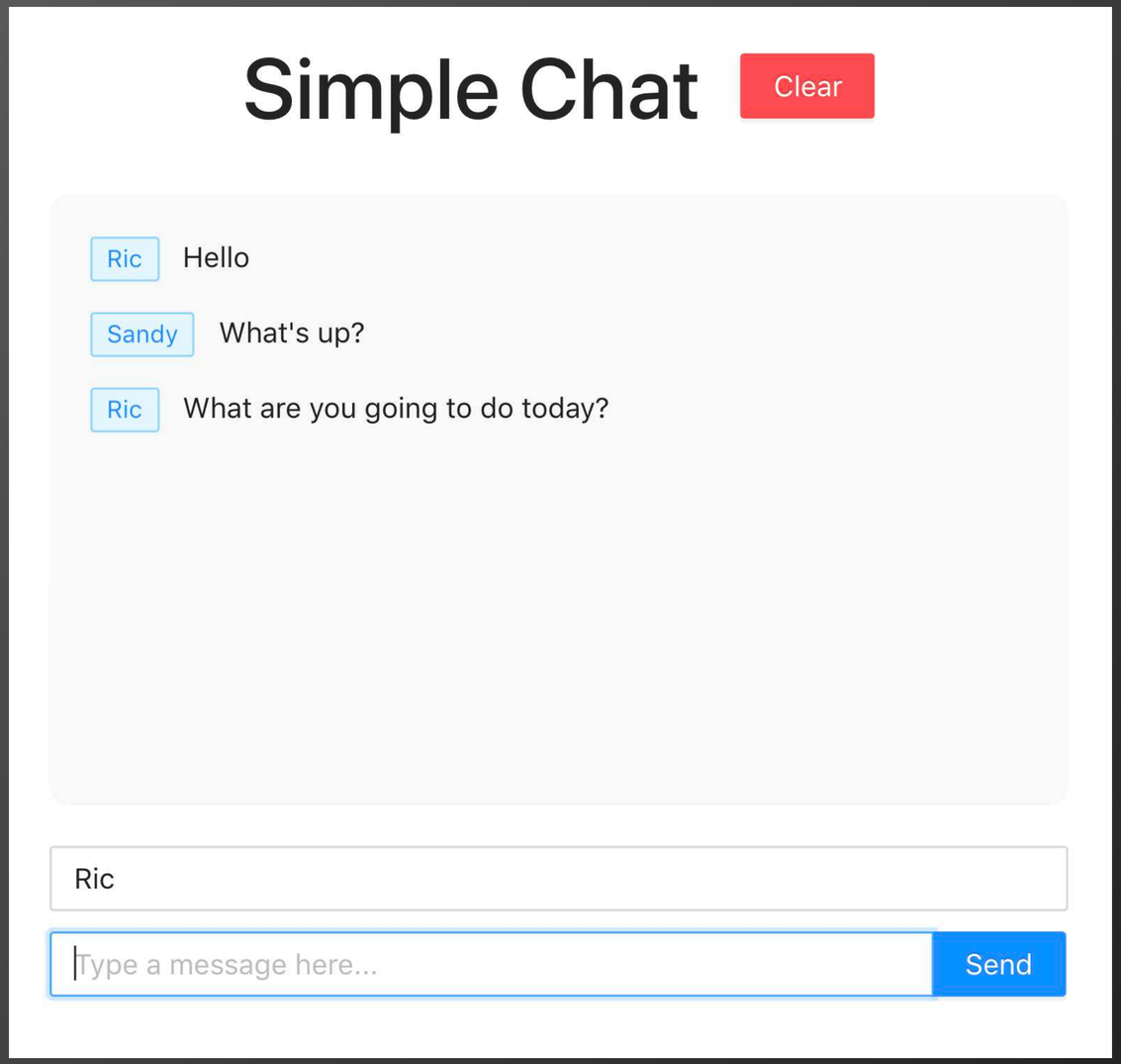
(EE 3035) Web Programming

© 2020 - Ric Huang ALL RIGHTS RESERVED

今天的重點...



Think: 一個簡單的 chat app...



- 在 "遊戲"、"對話" 類的 web application, 會需要有 "即時性的互動"、"主動通知" 的功能
- 目前學到的後端技術，可以做到嗎？還差什麼？

Server 主動通知 Client 端

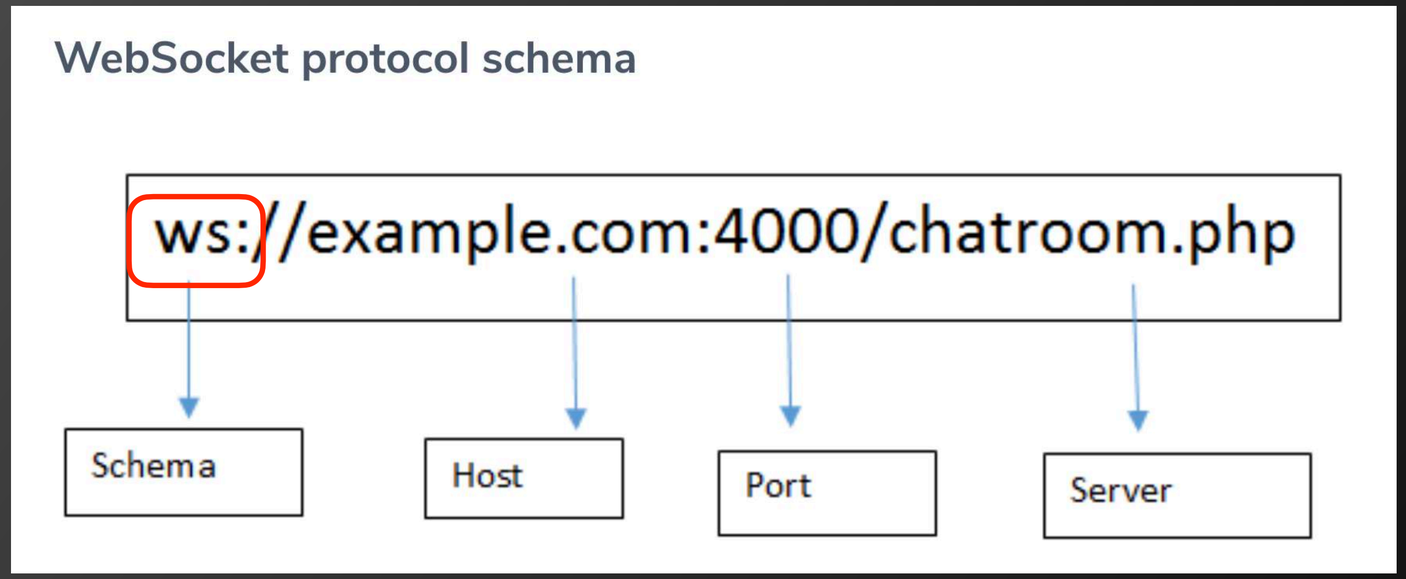
- 之前的例子都是用 HTTP 或是相對應的 middleware 來處理 client / server 之間的溝通
- HTTP 的 server 是很懶惰的，只有當 client 端傳送 CRUD 的請求，他才會對此 cleint 作出對應的動作
- 想像我們在遊戲或是社群的場景，A client 對 server 做出 CRUD, 然後 B client 應該要即時知道這個變化，但如果 server 不主動通知 B 的話，除非 B 定時去詢問 server, 否則他就不可能知道 => 這十分沒有效率

WebSocket

- WebSocket is the communication protocol which provides bidirectional communication (duplex) between the Client and the Server over a TCP connection.
- WebSocket remains open all the time so they allow the real-time data transfer. When client triggers the request to the server, server does not close the connection on receiving the response. It rather persists and waits for client or server itself to terminate the request.

HTTP vs. WebSocket

HTTP

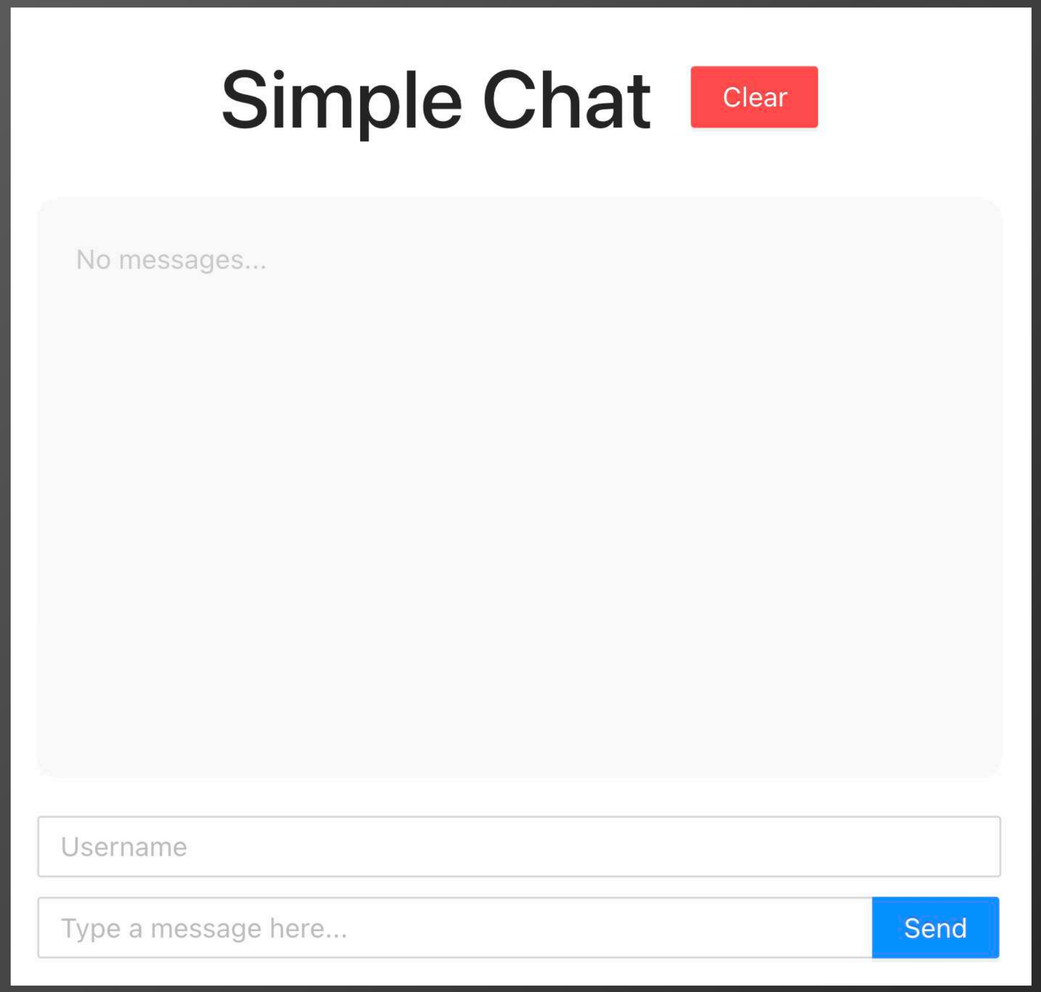


WebSocket

Real–time Duplex Communication (FYI)

- WebSocket offers a full real-time duplex communication.
- FYI, there are many so-called duplex communication techniques between server and client, such as: (ref)
  - Polling
  - Long Polling
  - Streaming
  - Postback and AJAX
  - HTML5

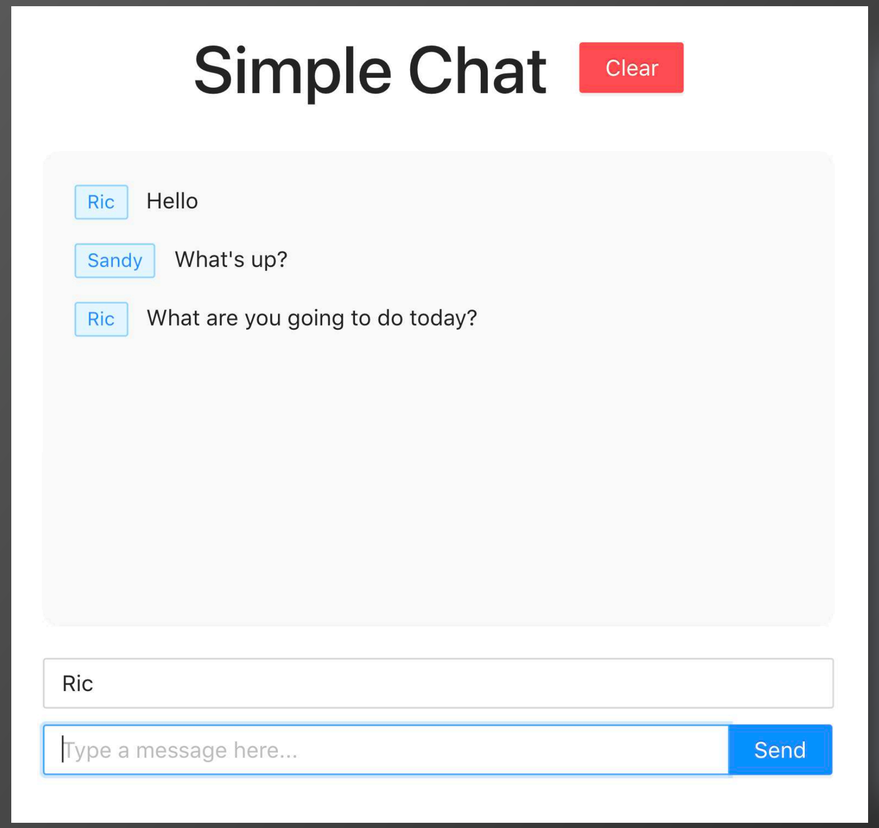
Preparing for the Simple Chat App



- 我們接下來將會用 "Simple Chat" 這個 app 來學習如何串接 React + WebSocket + Express + Mongoose
- 如同先前的範例，先創建一個 simpleChat project, 然後用 create-react-app 建立 frontend, 再到 Ceiba 下載 chat-starter-src.tgz, 在 frontend 目錄底下解壓縮 (應該會取代 src 目錄)
- 安裝 "yarn add antd --save"
- yarn start 後應該會看到 simple chat 的靜態網頁 (如左圖)

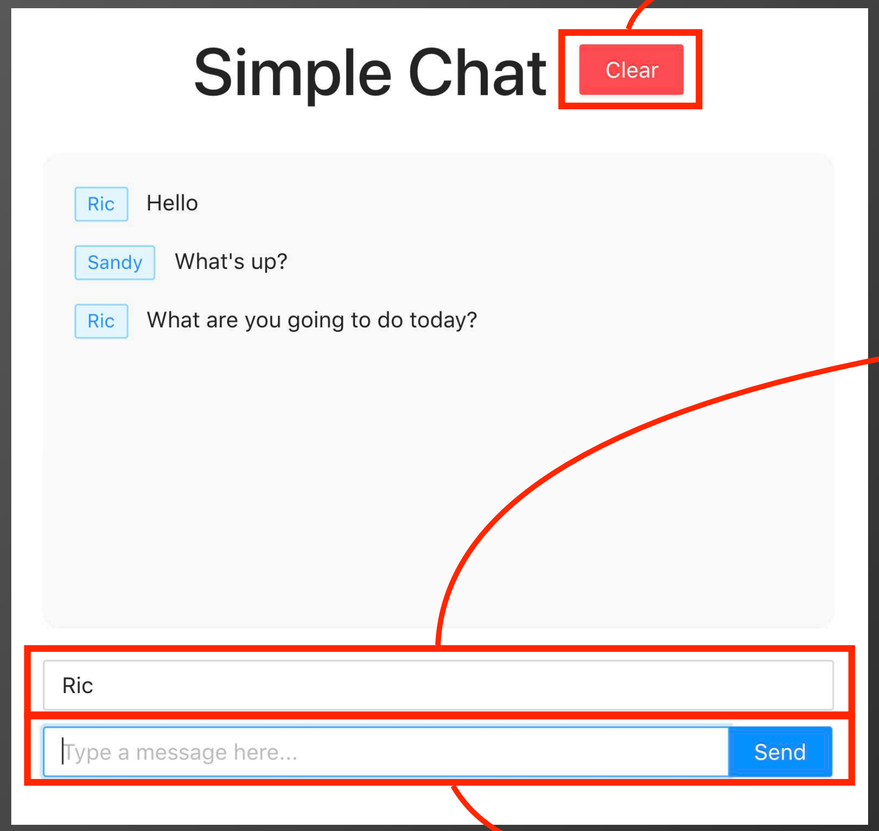
Simple Chat App。Frontend

- 前端畫面 (App.js)：使用 Ant Design 的 UI library (ref)




```
import { Button, Input } from 'antd'
...
  <div className="App">
    <div className="App-title">
      <h1>Simple Chat</h1>
      <Button> Clear </Button>
   </div>
    <div className="App-messages" />
    <Input />
    <Input.Search />
  </div>
```

Frontend ◦ antd (ref)

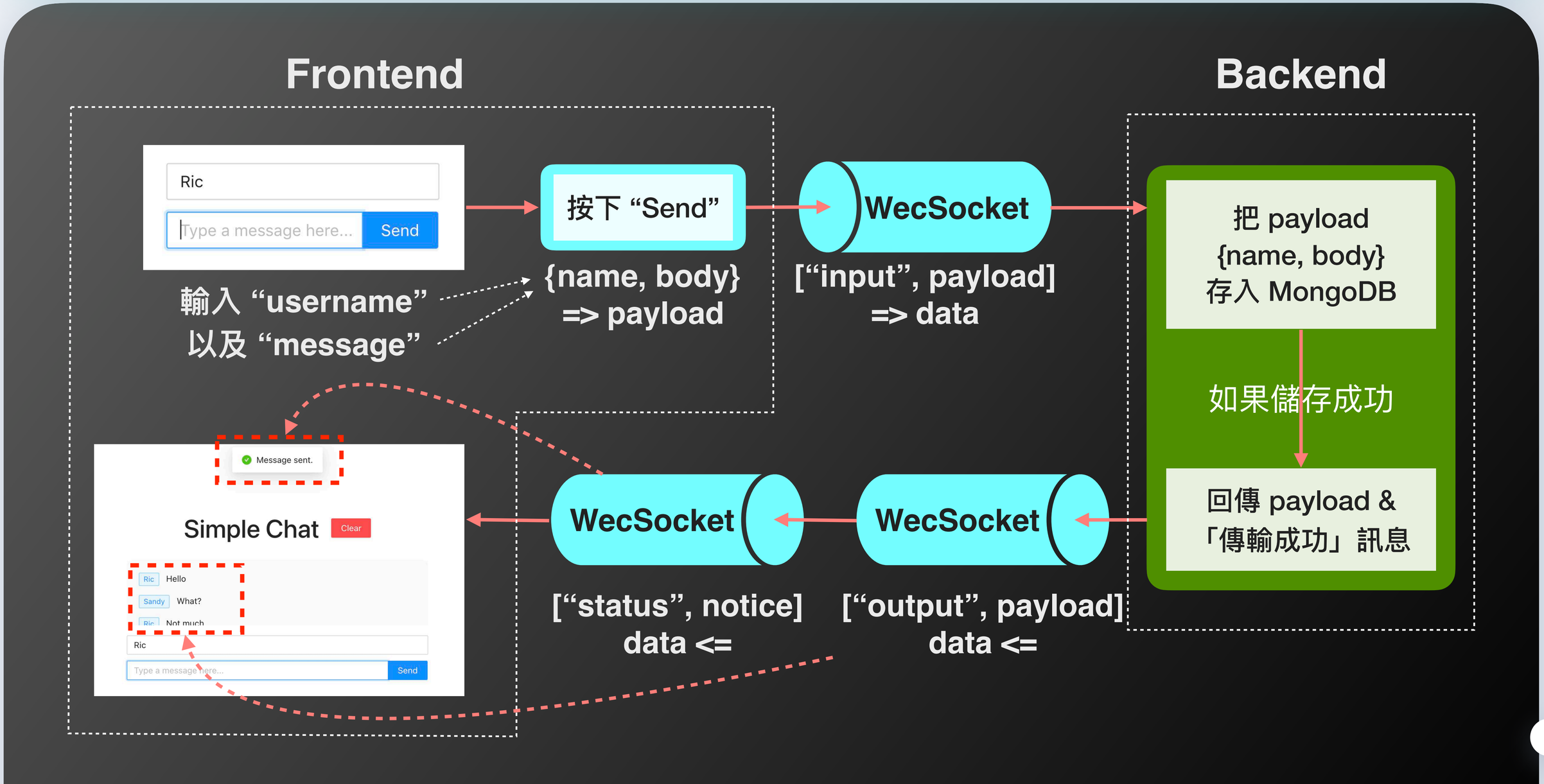



```
<Button type="primary" danger >
  Clear
</Button>
```

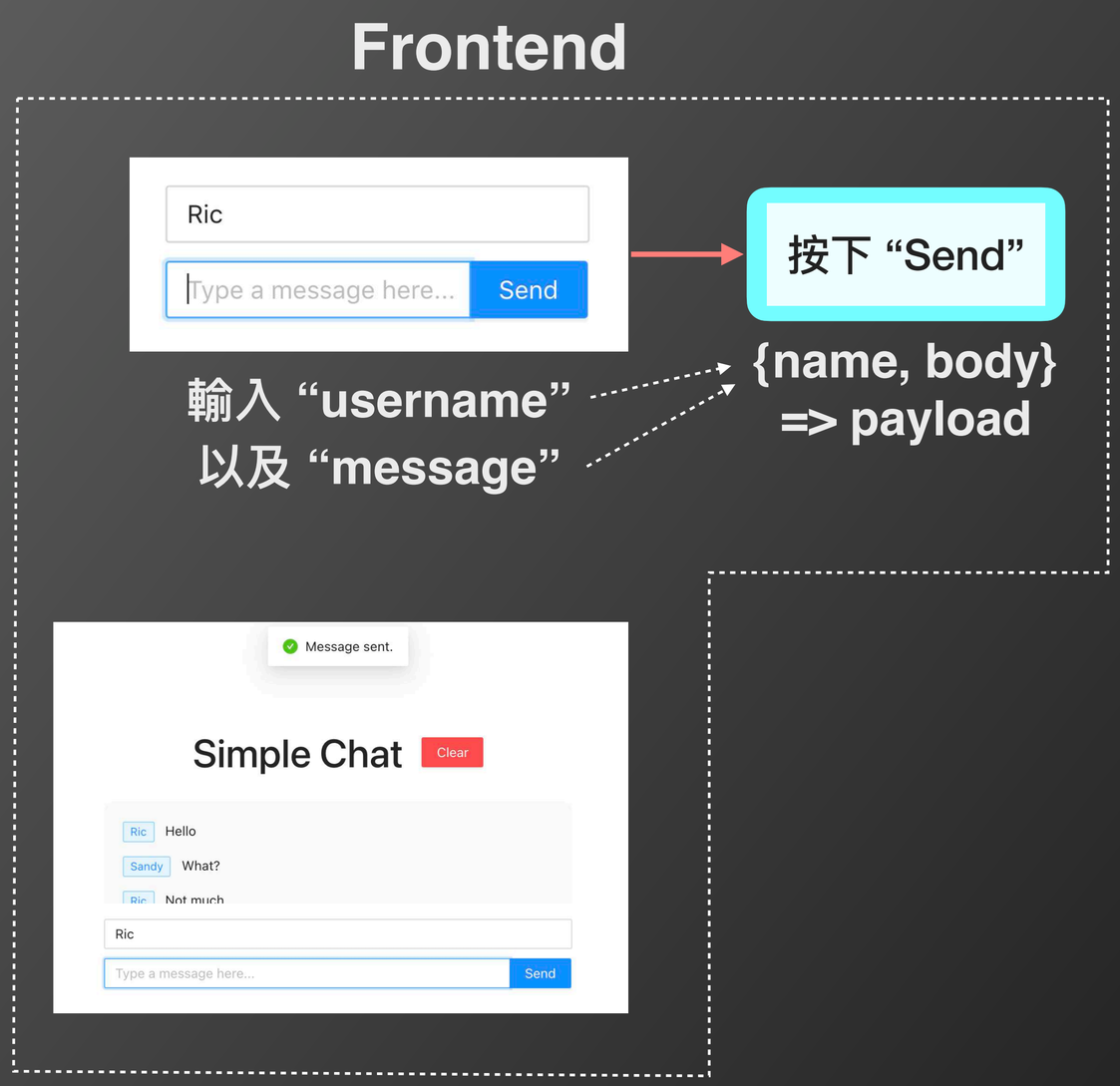
```
<Input
 placeholder="Username"
 style={{ marginBottom: 10 }}
></Input>
```

```
<Input.Search
enterButton="Send"
 placeholder="Type a message here..."
></Input.Search>
```

Duplex message communication process



Simple Chat App ◦ Frontend Design



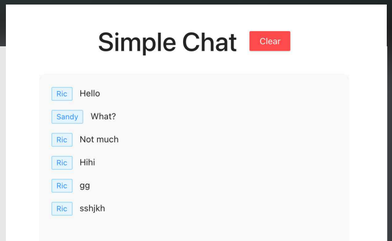
- Define a "useChat" hook to handle chat related states and methods
  - messages: to store the message as an array of { username, textBody }
  - status: for the server to inform the status of message passing
  - sendMessage(): send { username, textBody }

Frontend Design ◦ Define "useChat" Hook

```
import { useState } from "react";
const useChat = () => {
  const [messages, setMessages] = useState([]);
  const [status, setStatus] = useState({});
  const sendMessage = (msg) => { console.log(msg); }
  return {
    status,
    messages,
    sendMessage
  };
};

export default useChat;
```

Frontend Design ◦ Use "useChat" Hook in App.js



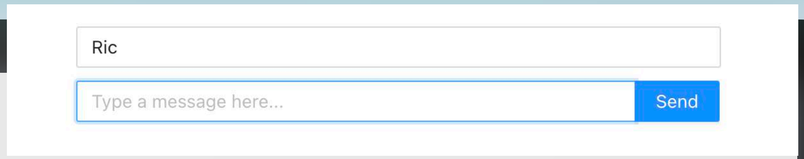

```
import { useState } from 'react'
import { Button, Input, Tag } from 'antd'
import useChat from './useChat'
function App() {
  const { status, messages, sendMessage } = useChat()
  const [username, setUsername] = useState('')
  const [body, setBody] = useState('')  // textBody
 return (
    <div className="App">
      <div className="App-title"> ... </div>
      <div className="App-messages">
        {messages.length === 0 ? (
          <p style={{ color: '#ccc' }}> No messages... </p>
        ) : (
          messages.map(({ name, body }, i) => (
            <p className="App-message" key={i}>
              <Tag color="blue">{name}</Tag> {body}
            </p>
          ))
        )}
      </div>
```

Define states and methods

Initial or when cleared

Print each message: { name, textBody }

Frontend Design ◦ Use "useChat" Hook in App.js




```
      <Input
        placeholder="Username"
        value={username}
        onChange={(e) => setUsername(e.target.value)}
        style={{ marginBottom: 10 }}
      ></Input>
      <Input.Search
        value={body}
        onChange={(e) => setBody(e.target.value)}
        enterButton="Send"
        placeholder="Type a message here..."
        onSearch={(msg) => {
          sendMessage({ name: username, body: msg })
          setBody('')
        }}
      ></Input.Search>
    </div>
  )
}
```

Save and store the username

Save and store the textBody

When “Send”, call sendMessage()

yarn start again.

Do you see
{ username, body }
printed on console?

建立 Client/Server WebSocket 連線

- Client side — `new WebSocket(url)`

```
const client = new WebSocket('ws://localhost:4000')
```

- Note: On client side, you can't require or import 'ws'. If you do, you will receive the following error:

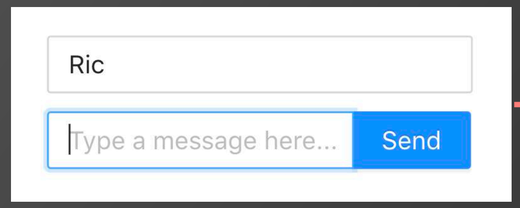
Error: ws does not work in the browser. Browser clients must use the native WebSocket object

- Server side — `new WebSocket.Server({server: serverObj})`
  - Do "yarn init -y" & "yarn add express mongoose dotenv-defaults nodemon ws --dev"

```
const app = express();
const server = http.createServer(app);
const wss = new WebSocket.Server({ server });
```

Frontend ◦ WebSocket / sendMessage()

- To generalize data sending from client side, we define a general "sendData(data)" in addition to "sendMessage(msg)"



輸入 “username”
以及 “message”

按下 “Send”

{name, body}
=> payload

WecSocket

[“input”, payload]
=> data

```
<Input.Search
  onSearch={(msg) => {
    sendMessage({
      name: username, body: msg })
></Input.Search>
```

```
const sendData = async (data) => {
   await client.send(
            JSON.stringify(data));
 };
```

```
const sendMessage = (payload) => {
  sendData(["input", payload]);
};
```

Sending Internet Data as "Byte String"

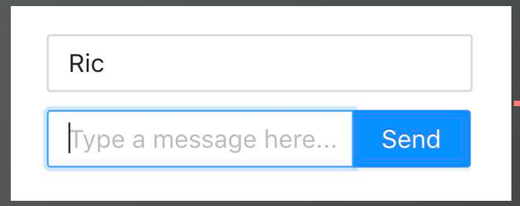
- Note: data sent thru internet is treated as "byte string"
- JSON.stringify(obj) // converting object to string

```
JSON.stringify({});                    // '{}'
JSON.stringify(true);                  // 'true'
JSON.stringify('foo');                 // '"foo"'
JSON.stringify([1, 'false', false]); // '[1,"false",false]'
JSON.stringify({ x: 5 });              // '{"x":5}'
```

- JSON.parse(str[, reviver]) // converting string to object

```
JSON.parse('{}');                      // {}
JSON.parse('true');                    // true
JSON.parse('"foo"');                   // "foo"
JSON.parse('[1, 5, "false"]');         // [1, 5, "false"]
JSON.parse('{"x":5}');                 // { x: 5 }
```

Sending Internet Data as "Byte String" (Overview)



輸入 “username”
以及 “message”

按下 “Send”

{name, body}
=> payload

WecSocket

[“input”, payload]
=> data

把 payload
{name, body}
存入 MongoDB

Client

```
<Input.Search
  onSearch={(msg) => {
    sendMessage({
      name: username, body: msg })
></Input.Search>
```

```
const sendMessage = (payload) => {
  sendData(["input", payload]);
};
```

```
const sendData = async (data) => {
  await client.send(
        JSON.stringify(data));
};
```

Server

```
const wssConnect = ws => {
  ws.onmessage = async (byteString) => {
    const { data } = byteString
    const [task, payload] = JSON.parse(data)
    switch (task) {
      case 'input': {
        const { name, body } = payload
        const message
          = new Message({ name, body })
        try { await message.save();
        } catch (e) { throw new Error
          ("Message DB save error: " + e);
        }
        sendData(['output', [payload]])
      break
```

Backend。server.js

- 建立 backend project: "cd backend & yarn init -y"
- 安裝必要套件："yarn add express mongoose dotenv-defaults nodemon ws --dev"
- 設定好 .env 以及 .env-defaults
- 從 Ceiba 下載 server-init.tgz 到 backend, 然後解壓縮
- 在 package.json 加上：

```
"type": "module",
"scripts": {
  "server": "nodemon server.js"
},
```

Message MongoDB Schema

- 如前所述，MongoDB 的儲存格式就像是 JSON format 一樣，因此，在 JS 裡頭就用 object 的格式來定義即可

```
import mongoose from 'mongoose';
const Schema = mongoose.Schema
// Creating a schema, sort of like working with an ORM
const MessageSchema = new Schema({
  name: {
    type: String,
    required: [true, 'Name field is required.']
  },
  body: {
    type: String,
    required: [true, 'Body field is required.']
  }
})
// Creating a table within database with the defined schema
const Message = mongoose.model('message', MessageSchema)
// Exporting table for querying and mutating
export default Message;
```

Backend ◦ server.js

```
const app = express()
const server = http.createServer(app)
const wss = new WebSocket.Server({ server })
...
db.once('open', () => {
  console.log('MongoDB connected!')

  wss.on('connection', wssConnect)

  const PORT = process.env.port || 4000

  server.listen(PORT, () => {
    console.log(`Listening on http://localhost:${PORT}`)
  })
})
```

Bind wssConnect() to the wss.on('connection') event

Backend ◦ server.js ◦ wssConnect() (1/3)

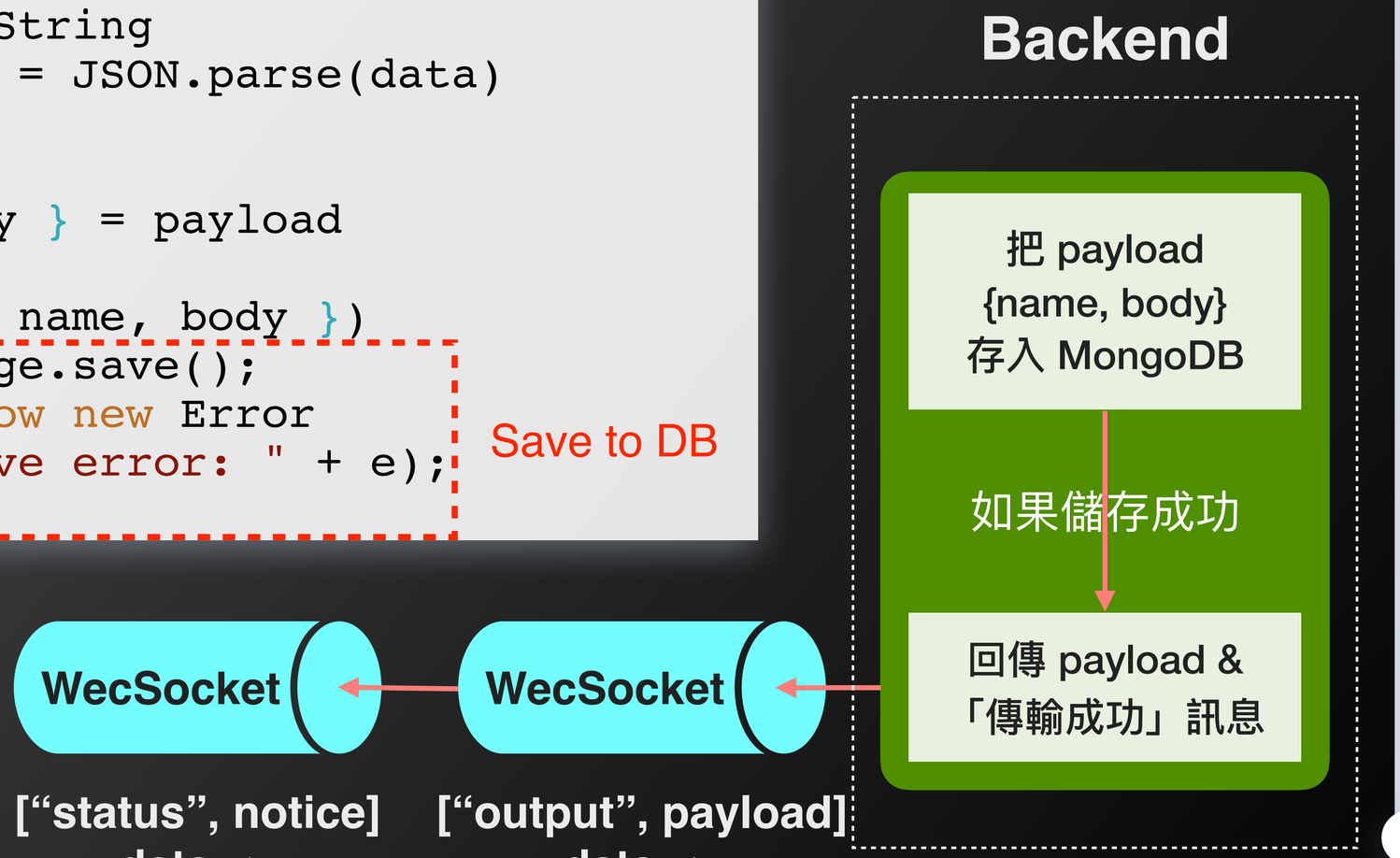
```
const wssConnect = ws => {
  const sendData = (data) => {
    ws.send(JSON.stringify(data))
  }
  const sendStatus = (payload) => {
   sendData(['status', payload])
  }
```

Some helper functions

Backend ◦ server.js ◦ wssConnect() (2/3)

```
const wssConnect = ws => {
  ... // sendData(), sendStatus()
  ws.onmessage = async (byteString) => {
    const { data } = byteString
    const [task, payload] = JSON.parse(data)
    switch (task) {
      case 'input': {
        const { name, body } = payload
        const message
          = new Message({ name, body })
        try { await message.save();
        } catch (e) { throw new Error
          ("Message DB save error: " + e);
        }
```

Save to DB

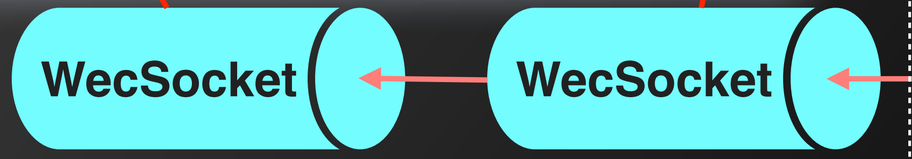


Backend ◦ server.js ◦ wssConnect() (3/3)

```
const wssConnect = ws => {
  ...
  ws.onmessage = async (byteString) => {
    ...
    switch (task) {
      case 'input': {
        ... // save to DB
        sendData(['output', [payload]])
        sendStatus({
          type: 'success',
          msg: 'Message sent.'
        })
        break
      }
      default: break
    }
  }
}
```

Sent to client

Notification message



["status", notice]
data <=

["output", payload]
data <=

Frontend • Receiving Data

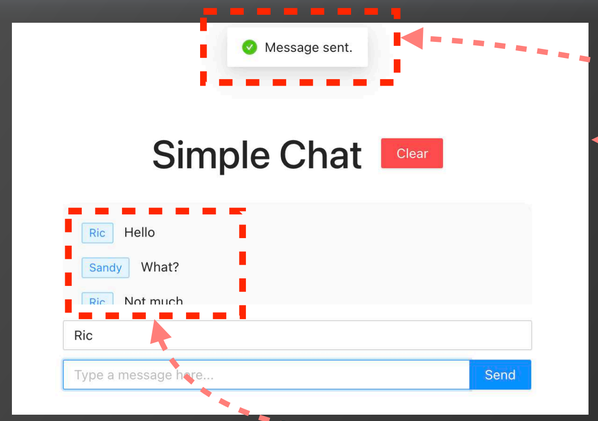
useChat.js

```
const useChat = () => {
  ... // define messages, status
  client.onmessage = (byteString) => {
    const { data } = byteString;
    const [task, payload] = JSON.parse(data);
    switch (task) {
      case "output": {
        setMessages(() =>
        [...messages, ...payload]); break; }
      default: break;
    }
  }
  ... // define sendData(), sendMessage()
}
```

Received from server

App.js

```
import useChat from './useChat'
function App() {
  ...
  return (
    ...
    <div className="App-messages">
      {messages.length === 0 ? (...) : (
        messages.map(({name,body},i) => (
          <p key={i}>
            <Tag color="blue">
            {name}</Tag> {body}
          </p>)))}
    </div>
```



WecSocket

WecSocket

回傳 payload &
「傳輸成功」訊息

[“status”, notice]
data <=

[“output”, payload]
data <=

Frontend • Displaying Status

useChat.js

```
const useChat = () => {
  ...
  client.onmessage = (byteString) => {
  ...
    switch (task) {
      case "output": { ... }
      case "status": {
        setStatus(payload); break; }
      default: break;
    }
  }
  ...
}
```

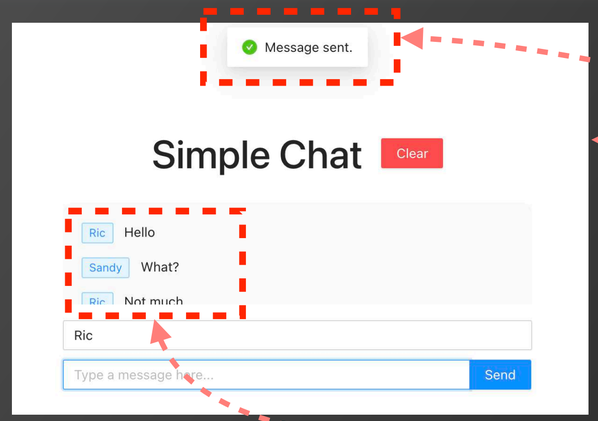
Received from server

App.js

```
function App() {
  const displayStatus = (payload) => {
    if (payload.msg) {
      const { type, msg } = payload
      const content = {
        content: msg, duration: 0.5 }
      switch (type) {
        case 'success':
          message.success(content)
          break
        case 'error':
        default:
          message.error(content)
          break
  }}}
  useEffect(() => {
    displayStatus(status)}, [status])
```

antd component

每次 re-render 後



WecSocket

WecSocket

回傳 payload &
「傳輸成功」訊息

[“status”, notice]
data <=

[“output”, payload]
data <=

Have you successfully seen the chat?

To improve:

1. 檢查 username, textBody 是否為空值
2. Username 輸入之後自動跳到 textBody input
3. 一開始就顯示 DB 之前的對話記錄
4. Clear message 還沒做

檢查 username, textBody 是否為空值

- 在 <Input.Search onSearch> 的時候檢查 —

```
<Input.Search
  ...
  onSearch={(msg) => {
    if (!msg || !username) {
      displayStatus({
        type: 'error',
        msg: 'Please enter a username and a message body.'
      })
      return
    }

    sendMessage({ name: username, body: msg })
    setBody('')
  }}
></Input.Search>
```

Username 輸入之後自動跳到 textBody input

- How? By "useRef()"

1. 先宣告一個 useRef() hook for textBody

```
const bodyRef = useRef(null)
```

2. 把這個 bodyRef 指定給 <Input.Search ref>

```
<Input.Search
    ref={bodyRef}
```

3. 當 <Input> 輸入 "Enter" 時，把 focus 移到 bodyRef

```
<Input
  onKeyDown={(e) => {
    if (e.key === 'Enter') {
      bodyRef.current.focus()
  }}}>
```

一開始就顯示 DB 之前的對話記錄

- 在 server.js 的 wssConnect() 加個 action

```
const wssConnect = ws => {
  ... // sendData(), sendMessage()
  Message.find().sort({ created_at: -1 }).limit(100)
    .exec((err, res) => {
      if (err) throw err
      // initialize app with existing messages
      sendData(['init', res])
    })
}
```

- 在 useChat 裡頭加個 "init" 的 task

```
const useChat = () => {
  client.onmessage = (byteString) => {
    switch (task) {
      case "init": {
        setMessages(() => payload);
        break;
  }}}
}
```

Clear message

- 在 useChat 裡頭加個 "clearMessages()" 的 method

```
const useChat = () => {
  const clearMessages = () => {
    sendData(["clear"]);
  };
  return { status, messages, sendMessage, clearMessages };
}
```

- 在 App.js 裡頭 handle <Button onClick>

```
function App() {
  const {status, messages, sendMessage, clearMessages}= useChat()
  return (...
        <Button type="primary" danger onClick={clearMessages}>
          Clear
        </Button>
    ...)
}
```

Clear message (continued)

- 在 ws.onmessage 裡頭加個 “clear" 的 task

```
case 'clear': {
  Message.deleteMany({}, () => {
    sendData(['cleared'])
    sendStatus({ type: ‘info', msg: 'Message cache cleared.'})
  })
  break
}
```

- 在 useChat 裡頭加個 "cleared" 的 task

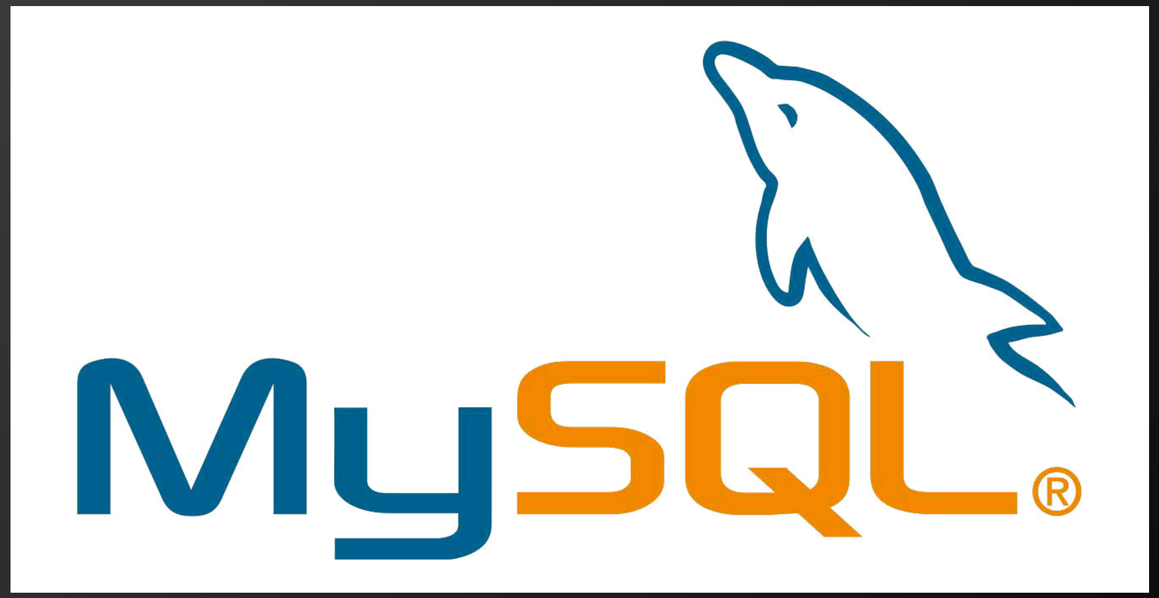
```
client.onmessage = (byteString) => {
  switch (task) {
    case "cleared": {
      setMessages([]);
      break;
  }}
}
```

當然，上述簡單的 chat app,
還有許多還可以進化的地方，包含：

1. 一對多 or 多對多的廣播(broadcasting)機制
2. 建立登入的機制，區分不同使用者的資料
3. 使用第三方的 API 服務 (e.g. Google, FB)
4. 更有效率的 DB 存取/搜尋服務 (e.g. GraphQL)

我們會在接下來的單元一一跟大家介紹

今天的重點...



MySQL 練習

- 先確定你的 localhost 有 mysql running
  - 在 terminal 下 'mysql' 指令試試看
- If not, you need to install mysql server
  - For Mac: https://www.ioa.tw/macOS/MySQL.html (tested and worked!)
  - For other OS, please ask Google.
- Optionally, you can install a SQL client tool
  - For Mac, 推薦 Sequal Pro: https://sequelpro.com/download
  - Others, try MySQL Workbench: https://dev.mysql.com/downloads/workbench/

從 Ceiba 下載：final_exam.csv & final_exam 2.csv

final_exam

Name	Subject	Score
Amy	Math	30
Ben	Math	90
Calvin	Math	80
David	Math	100
Amy	English	90
David	English	100
Calvin	English	40
Ben	English	80
Amy	Chinese	80
David	Chinese	70

RDBMS 的特性

- Key Words: Case Insensitive (但通常用大寫)
- Column Name & Value: Case Sensitive
- Constraints
  - Primary key (PK): uniquely specifies a tuple within a table
  - Foreign key (FK): a field in a relational table that matches the primary key column of another table
  - Index(ing): to speed up the access of data
- What are the PKs and FKs in the tables of the studied case?

把 final_exam.csv 讀進 DB

1. 首先，連接到 localhost 的 DB server
   - Default port: 3306
2. 建立一個 database，並指定使用它
   - CREATE DATABASE testDB;
   - USE testDB;
3. 建立一個 table
   - CREATE TABLE final_exam (Name VARCHAR(255), Subject VARCHAR(255), Score INT);
4. 把資料讀進去
   - INSERT INTO final_exam (Name, Subject, Score) VALUES ('Amy', 'Math', 30);
   - INSERT INTO final_exam (Name, Subject) VALUES ('Ben', 'Math');

也可以直接用 SQL Client
來建立 DB & 讀進 table

SQL Data Types

- Int
- Float
- Varchar
- Text
- Date
- 不同 DB 有些微的不同

UPDATE Table

- 修改符合條件的資料：
  - UPDATE final_exam SET Score=90 WHERE Name='Ben' AND Subject='Math';
- 修改所有資料：
  - UPDATE final_exam SET Score=0;
  - UPDATE final_exam SET Score=0 WHERE 1=1;

DELETE Data from Table

- 刪除符合條件的資料：
  - DELETE FROM final_exam WHERE Name='Amy' AND Subject='Math';
- 刪除所有資料
  - DELETE FROM final_exam;
  - DELETE FROM final_exam WHERE 1=1;

用 SELECT 來 Query Data (command ref)

```
SELECT select_expr
  FROM table
  WHERE constraints
  GROUP BY grouping method (e.g. col_name)
  ORDER BY ordering
  LIMIT row_count
  ;
```

用 SELECT 來 Query Data

- 選出所有資料：
  - SELECT * FROM final_exam;
- 選出特定欄位資料：
  - SELECT Name, Score FROM final_exam;
- 選出前5筆資料：
  - SELECT * FROM final_exam LIMIT 5; //MySQL
  - SELECT TOP 5 * FROM final_exam; //MSSQL

用 SELECT 來 Query Data

- 計算考卷總數：
  - SELECT COUNT(*) FROM final_exam;
- 計算考生總數：
  - SELECT COUNT(DISTINCT Name) FROM final_exam;
- 計算所有考卷平均分數：
  - SELECT AVG(Score) AS avg_score FROM final_exam;
- 計算個人總分：
  - SELECT Name, SUM(Score) AS sum_score FROM final_exam GROUP BY Name;

用 SELECT 來 Query Data

- 計算科目平均分數並由平均分數低到高排序：
  - SELECT Subject, AVG(Score) AS avg_score FROM final_exam GROUP BY Subject ORDER BY avg_score;
- 選出滿分或不及格的考卷並由分數高到低排序：
  - SELECT * FROM final_exam WHERE Score = 100 OR Score < 60 ORDER BY Score DESC;

用 UPDATE 來新增欄位

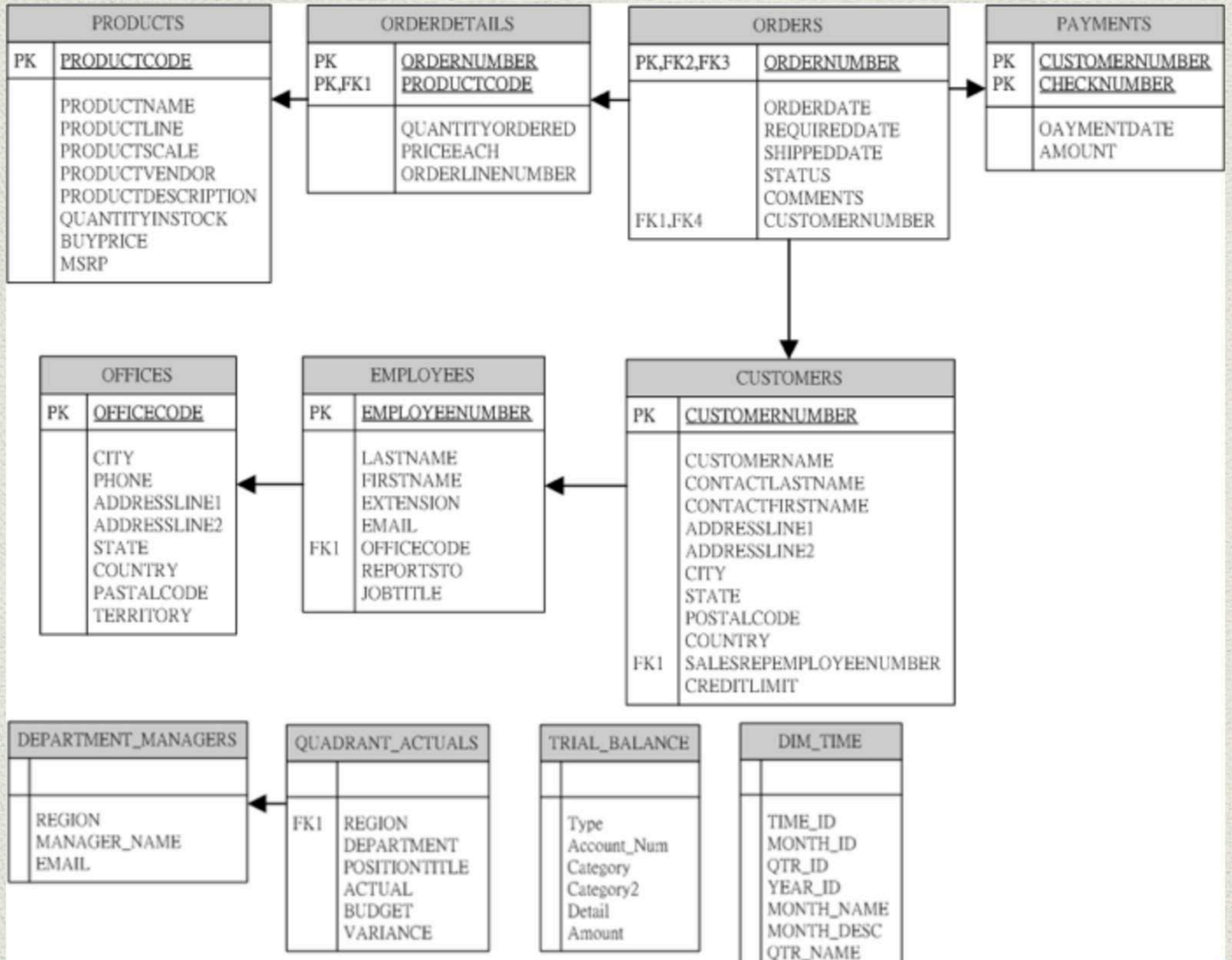
- 新增一個欄位，記錄考試成績是否及格(>= 60)，如果及格，填入 “及格”，否則填入 “不及格”
  - UPDATE final_exam SET ‘pass_or_fail’ = IF (‘Score’ >= 60, ‘及格’, ‘不及格’);

RDBMS 的 ACID 原則

- 原子性 (atomicity)：transaction 中的所有操作，要麼全部完成，要麼全部不完成。發生錯誤，會被 rollback
- 一致性 (consistency)：寫入的資料必須完全符合所有的預設規則，包含資料的精確度等等
- 隔離性 (isolation)：資料庫允許多個並發事務同時對其數據進行讀寫和修改的能力，隔離性可以防止多個事務並發執行時由於交叉執行而導致數據的不一致
- 持久性 (durability)：對資料庫所作的改變持久地保存在資料庫之中

設計一個 RDB 第一件事情就是規劃好 Schema

- Example:



RDBMS 正規化 (Normalization)

- 緣由：當相同的資訊出現在不同的 tables, 會有資料不一致的問題
  - 在課程表的資料庫裡頭，「課程名稱」同時出現在「教師」以及「教室」兩個 tables 裡面
  - 如果有一天在「教師」table 更改課名，在「教室」table 上的資料是否會被更新？
- What are the PKs and FKs in the tables of the studied case?

RDBMS 正規化 (Normalization)

- To remove structural redundancies and inconsistencies
- 1st normal form
  - Repeating attribute groups
- 2nd normal form
  - Functional dependencies
- 3rd normal form (3NF)
  - Transitive dependencies

Given a SQL table, 如何進行正規化？

- 刪除某些欄位
  - ALTER TABLE table_name DROP COLUMN col_name;
- 將不同面向的資料拆開，建立在不同的表單上
  - CREATE TABLE new_table (
    SELECT expression AS col_name,…
    FROM source_table
    WHERE constraints
    GROUP BY col_name,…
    );
  - 誰是 primary key? 誰是 foreign key?

Recall: Primary Key

- 中文：主鍵
- 每張資料表只能有一個Primary Key
  - 但Primary Key可以由多個欄位組成
- 同一張資料表內，Primary Key的值不能重複亦不能有NULL

設定/刪除 Primary Key

- 設定Primary Key：
  - ALTER TABLE final_exam ADD PRIMARY KEY(Name); //會失敗，因為Name有重複值。
  - ALTER TABLE final_exam ADD CONSTRAINT pk_final_exam PRIMARY KEY(Name, Subject);
- 刪除Primary Key
  - ALTER TABLE final_exam DROP PRIMARY KEY; //MySQL
  - ALTER TABLE final_exam DROP CONSTRAINT pk_final_exam; //MSSQL

Foreign Key

Post

id	1
user_id	1
title	A title
content	xxx....

User

id	1
name	Amy
email	example@gmail.com
age	18

id	2
user_id	1
title	New Post
content	xxx....

Post

INDEX

- 中文：索引
- 每張資料表可以有多個INDEX
- INDEX可以由多個欄位組成
- INDEX的值可以重複也可以有NULL

創造/刪除 INDEX

- 創造INDEX：
  - CREATE INDEX name_index ON final_exam (Name);
  - CREATE INDEX name_and_subject_index ON final_exam (Name, Subject);
- 刪除INDEX：
  - DROP INDEX final_exam.name_and_subject_index; //MSSQL
  - ALTER TABLE final_exam DROP INDEX name_and_subject_index; //MySQL

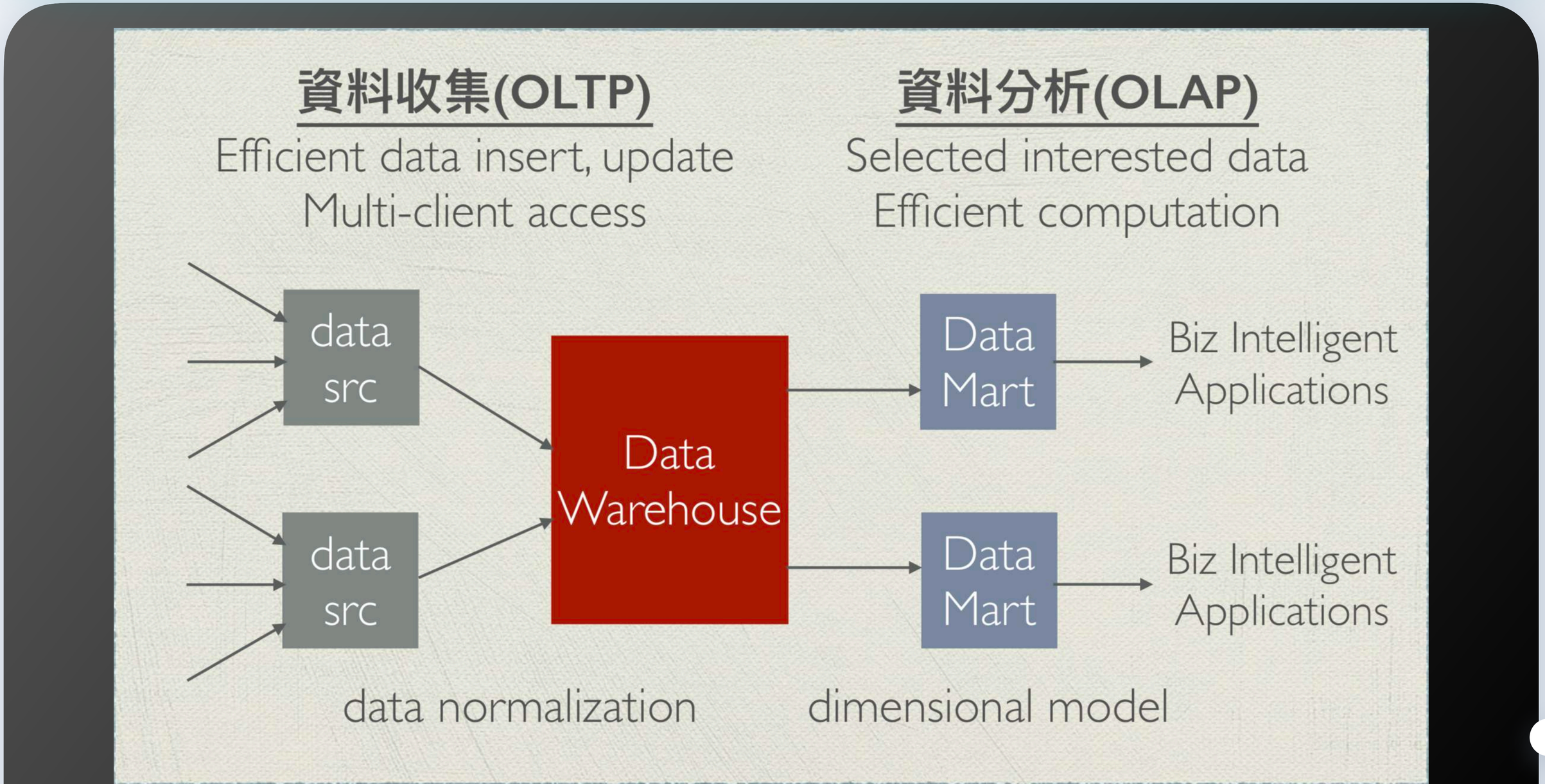
INDEX 的優缺點

- 優點：
  - 加快SELECT速度（O(n) -> O(log n)）
- 缺點：
  - 創造INDEX費時、減慢INSERT及DELETE速度
- 原理：
  - 每個INDEX都代表一個Binary Tree

UNIQUE

- 每張表可以有多個UNIQUE
- UNIQUE可以由多個欄位組成
- 同一張資料表內，UNIQUE的值不能重複但可以有NULL

Data Processing and Analysis

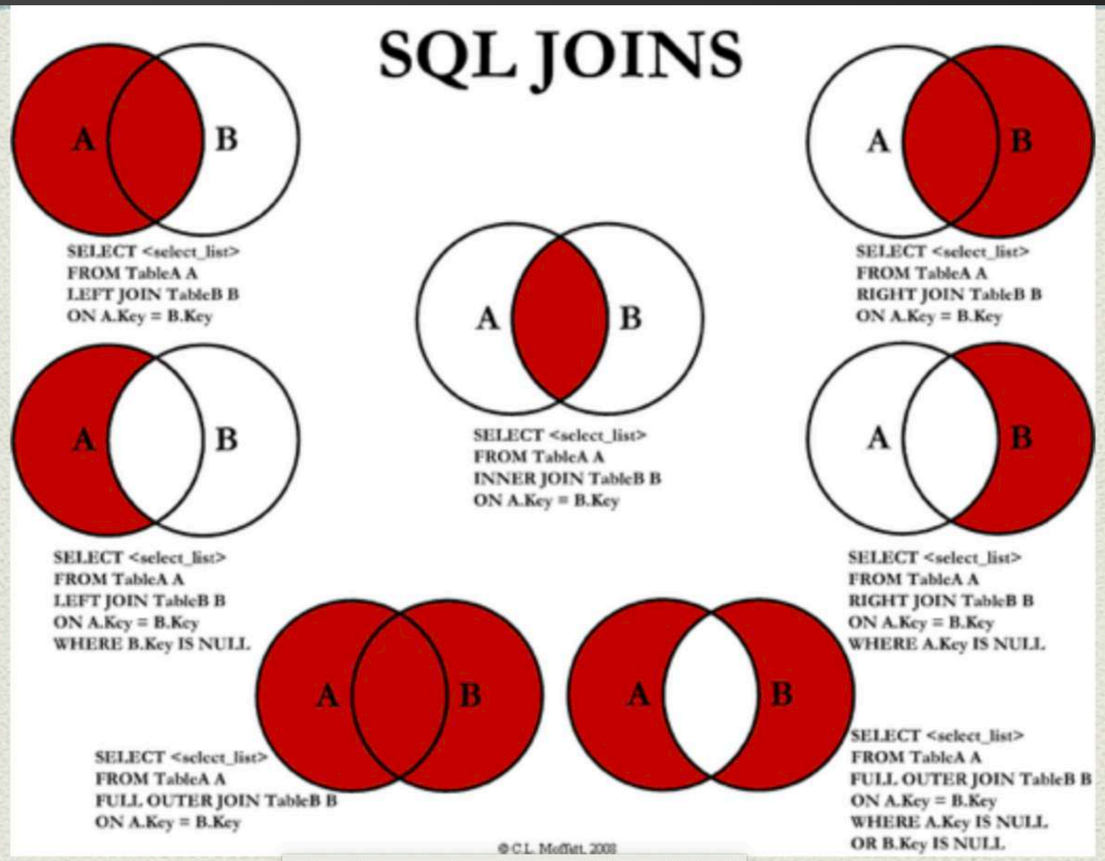


用 JOIN 產生 table for analysis

- Denormalizatation (去正規劃)
- SELECT
  m.Name, m.Subject, m.Score AS Midterm_Score,
  f.Score AS Final_Score
  FROM midterm_exam AS m
  INNER JOIN final_exam AS f
  ON m.Name=f.Name AND
  m.Subject=f.Subject;

Different Types of JOINs

- INNER JOIN
- LEFT JOIN
- RIGHT JOIN
- OUTER JOIN
- 其他：CROSS JOIN, STRAIGHT_JOIN, NATURAL XX JOIN



https://www.codeproject.com/Articles/33052/Visual-Representation-of-SQL-Joins

一些資料分析的案例

1. 某品牌服飾商想要知道他們的新增會員數為何減少？如何增加線上電商的交易量？
2. 某手機公司想要分析其在各論壇與粉絲頁的社群資料，找出社群中之意見領袖，希望能從他們來帶給其他客戶正面的影響
3. 某知名汽車廠牌推出一套汽車保養套餐，不知為何客戶青睞度很低，希望找出原因
4. 某人壽保險公司希望能利用雲端高速的數據分析，有效地抓出可能的金融詐騙行為
5. 某熱門論壇每日有上千上萬篇文章，活躍的讀者有數十萬人，想要實現一個個人化的文章推薦系統
6. 某醫美診所想要根據過去正顎銷骨手術的圖片資料，去預估新的客戶術後的長相
7. 某國家級的醫療中心收集了許多癌症病人組織切片的蛋白體質譜儀的資料，希望能透過定序、定量的分析，找出早發性病人的發病原因

結論：RDBMS/SQL 的好處

- 資料處理很穩健
- 會使用的人多，資源多易學
- 歷史悠久，長年效能優化
- 可使用 join 等功能，功能性強

結論：RDBMS/SQL 的缺點

- 即便長年下來 SQL 衍伸出很多垂直分割、讀寫分離等等等技術，但遇到某些狀況還是不適合用 SQL 處理..
  - e.g. 社群網路中的留言、按讚等資料

感謝聆聽！

© 2020 - Ric Huang ALL RIGHTS RESERVED