

# Database & REST Service & Postman

---

## Database & REST Service & Postman

Database(CRUD Operation)

MySQL + (MySQL Workbench)

NoSQL - MongoDB/Cassandra + Compass

SQL vs. NoSQL\*\* (interview common question)

Graph Database

JDBC(Java Database Connectivity)

Connecting database using Java and Database client

Connecting to database in SpringBoot

REST API(End-Point)

URI, URL, URN (Unique Resource)

URL Structure

Path Variables **and** Request Parameter

Path Variable examples

Request Parameter (there is a length limit, SD:"TinyURL")

HTTP Methods and Status Code

REST Resource Naming Guide/Convention

Design

Exercise (super useful)

Reference

GraphQL (RoR tutorial)

**RPC** (Remote Procedure Call - with **Http/AMQP**) → Web APIs;

**REST**

GraphQL (vs.Rest)

Postman\*

Public APIs

API Collection

Environments

Question: **what is endpoint(API)** ?

## Database(CRUD Operation)

---

### MySQL + (MySQL Workbench)

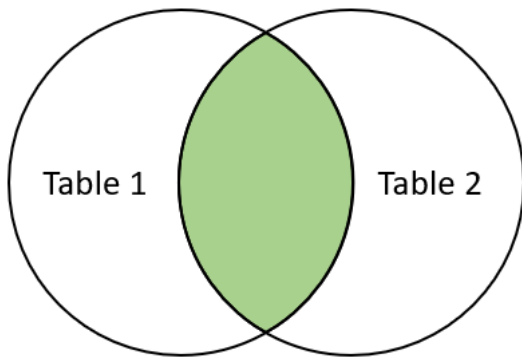
```
1  --Created DB Schema
2  CREATE SCHEMA `ChuvaTest` DEFAULT CHARACTER SET utf8 ;
3
4  --Created DB Table
```

```

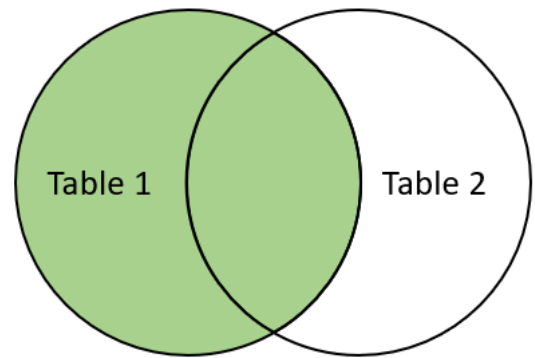
5 CREATE TABLE `Chuwa`.`User` (
6   `id` INT NOT NULL,
7   `firstName` VARCHAR(100) NULL,
8   `lastName` VARCHAR(100) NULL,
9   `Address` VARCHAR(200) NULL,
10  PRIMARY KEY (`id`));
11
12 --Insert
13 INSERT INTO `Chuwa`.`User` (`id`, `firstName`, `lastName`, `Address`) VALUES ('1',
14   'Charles', 'C', '123 Fake Address');
15
16 --Select
17 SELECT * FROM Chuwa.User;

```

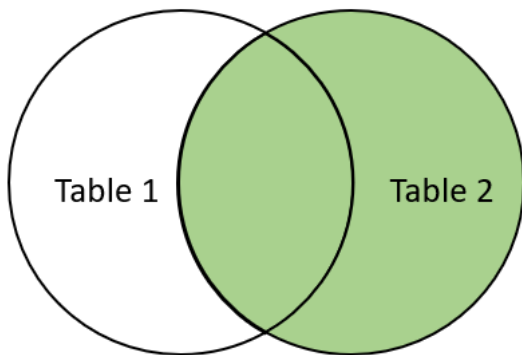
4 types of Joins



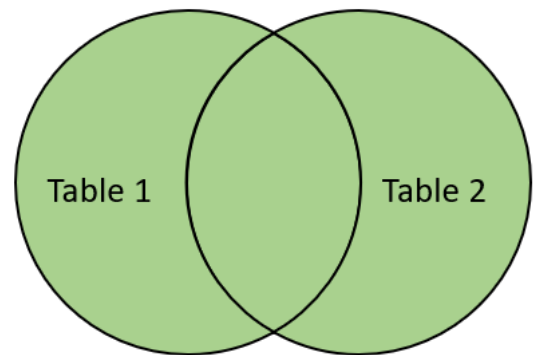
**INNER JOIN**



**LEFT JOIN**



**RIGHT JOIN**



**CROSS JOIN**

## NoSQL - MongoDB/Cassandra + Compass

```
1 { //e.g. Employee collection/table in JSON - Format
2   _id: ,
3   Emp_ID: "10025AE336"
4   Personal_details:{
5     First_Name: "Radhika",
6     Last_Name: "Sharma",
7     Date_Of_Birth: "1995-09-26"
8     Maiden_name: "C" //No need to alter DB/Table structure
9     Address: {
10      city: "Hyderabad",
11      Area: "Madapur",
12      State: "Telangana"
13    }
14  },
15  Contact: {
16    e-mail: "radhika_sharma.123@gmail.com",
17    phone: "9848022338"
18  },
19  Address: {
20    city: "Hyderabad",
21    Area: "Madapur",
22    State: "Telangana",
23  }
24 }
25
26 --GrahQL
27 { //e.g. Employee collection/table in JSON - Format
28   _id: ,
29   Emp_ID: "10025AE336"
30   Personal_details:{
31     First_Name: "Radhika",
32     Last_Name: "Sharma",
33   }
34 }
```

```
1 --Creaet DB
2 use my_database
3
4 --show list of DBs
5 show dbs
6
```

```

7  --delete DB
8  db.dropDatabase()
9  { "dropped": "my_database", "ok": 1}
10
11
12  --create collection (aka Table)
13  db.createCollection(collection_name, options)
14  db.createCollection(<collection_name>, { capped: <boolean>,
15                                     autoIndexId: <boolean>,
16                                     size: <number>,
17                                     max: <number>,
18                                     storageEngine: <document>,
19                                     validator: <document>,
20                                     validationLevel: <string>,
21                                     validationAction: <string>,
22                                     indexOptionDefaults: <document>,
23                                     viewOn: <string>,
24                                     pipeline: <pipeline>,
25                                     collation: <document>,
26                                     writeConcern: <document>} )
27
28  e.g.
29  db.createCollection("MyCollection")
30
31  --this will automatically create `movie` collection
32  db.movie.insert({"name": "Avengers: Endgame"})
33

```

[More Syntax learning](#)

## SQL vs. NoSQL\*\* (interview common question)

SQL	NoSQL
<b>RELATIONAL</b> DATABASE MANAGEMENT SYSTEM (RDB-MS)	Non-relational or <b>distributed</b> database system.
These databases have fixed or <b>static</b> or predefined <b>schema</b> (有网格的本子)	They have <b>dynamic schema</b> (无网格的本子)
These databases are <b>NOT</b> suited for <b>hierarchical</b> data storage.	These databases are best suited for <b>hierarchical data</b> storage. (JSON style)
These databases are best suited for <b>complex queries</b>	These databases are <b>NOT so good</b> for complex queries(很痛苦)
<b>Vertically Scalable</b> (扩容单台服务器. e.g 32g -> 128G) is easier fo SQL	<b>Horizontally Scalable</b> (多台服务器扩容) (e.g. 一台--> 多台)
Follows <b>ACID</b> property	Follows <b>CAP</b> (consistency, availability, partition tolerance)
<b>Examples:</b> <u>MySQL</u> , <u>PostgreSQL</u> , Oracle, MS-SQL Server, <u>SQLite</u> (In memory DB) etc.	<b>Examples:</b> <u>MongoDB</u> , HBase, Neo4j, <u>Cassandra</u> etc.

Question: **In your project, which one did you use ? And why ?**

- **QPS** (Query Per Second) (e.g. 10k QPS)

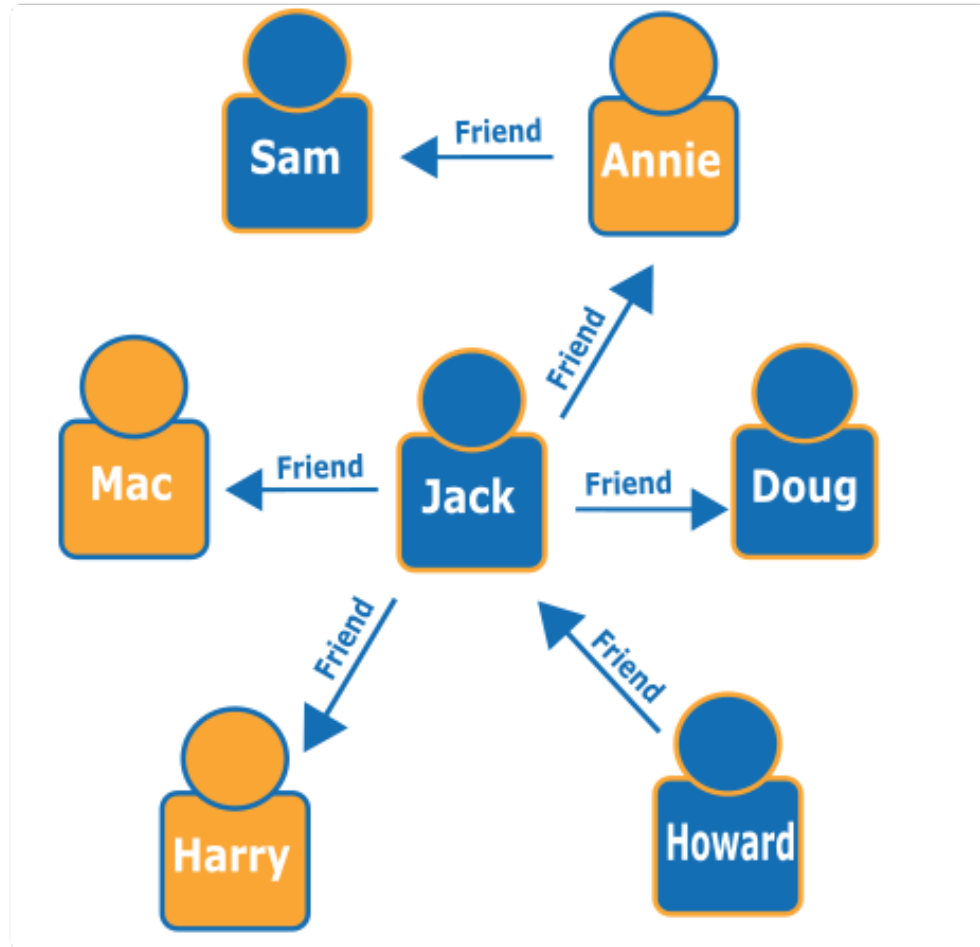
**No-SQL becomes more and more popular.**

## Graph Database

Key concept: Node + Edge

- **Neo4j**
- TigerGraph
- Amazon Neptune

(DFS + BFS)

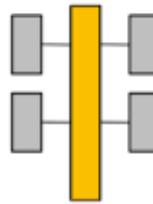


## SQL Database

### Relational

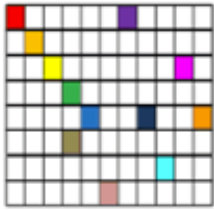


### Analytical (OLAP)



## NoSQL Database

### Column-Family



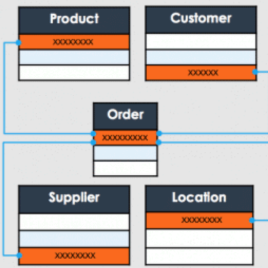
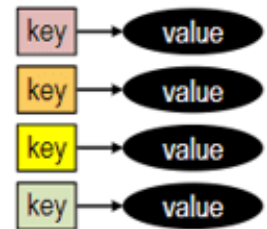
### Graph



### Document



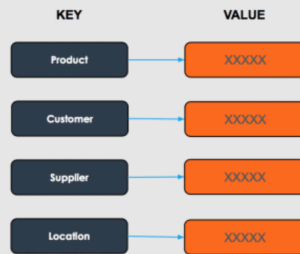
### Key-Value



## Relational Database

### COMPLEX, SLOW, TABLE JOINS REQUIRED

- Rigid schema
- High performance for transactions
- Poor performance for deep analytics



## Key-Value Database

### MULTIPLE SCANS OF MASSIVE TABLE REQUIRED

- High fluid schema/no schema
- High performance for simple transactions
- Poor performance for deep analytics



## Graph Database

### PRE-CONNECTED BUSINESS ENTITIES – NO JOINS NEEDED

- Flexible schema
- High performance for complex transactions
- High performance for deep analytics

## NoSQL vs SQL

	Funct. Req.								Non-Funct. Req.										Techniques																				
	Scan Queries	ACID Transactions	Conditional Writes	Joins	Sorting	Filter Queries	Full-Text Search	Analytics	Data Scalability	Write Scalability	Read Scalability	Elasticity	Consistency	Write Latency	Read Latency	Write Throughput	Read Availability	Write Availability	Durability	Range-Sharding	Hash-Sharding	Entity-Group Sharding	Consistent Hashing	Shared-Disk	Transaction Protocol	Sync. Replication	Async. Replication	Primary Copy	Update Anywhere	Logging	Update-in-Place	Caching	In-Memory Storage	Append-Only Storage	Global Indexing	Local Indexing	Query Planning	Analytics Framework	Materialized Views
MongoDB	x		x		x	x	x	x	x	x	x		x	x	x	x	x		x	x	x					x	x		x			x	x			x	x	x	
Redis	x	x	x								x		x	x	x	x	x		x								x	x		x		x							
HBase	x		x		x				x	x	x	x	x	x		x			x	x						x		x		x		x		x					
Riak							x	x	x	x	x	x		x	x	x	x	x	x	x	x		x				x		x	x	x	x	x		x	x		x	
Cassandra	x		x		x		x	x	x	x	x	x	x	x		x	x	x	x	x	x						x		x	x	x	x		x	x	x			x
MySQL	x	x	x	x	x	x	x	x			x		x						x					x	x		x	x		x	x	x				x	x		

Table 1: A direct comparison of functional requirements, non-functional requirements and techniques among **MongoDB**, **Redis**, **HBase**, **Riak**, **Cassandra** and **MySQL** according to our NoSQL Toolbox.

### Recap Reasons for SQL:

- **Structured data**
- **Strict schema**
- **Relational data**
- Need for **complex joins**
- Transactions
- Clear patterns for scaling (?)
- More established: developers, community, code, tools, etc
- **Lookups by index** are very fast;

### Recap Reasons for NoSQL: (easier for Scale)

- Semi-structured data
- **Dynamic or flexible schema(implicit schema)**
- Non-relational data
- No need for complex joins
- **Store many TB (or PB) of data**
- Very data intensive workload
- **Very high throughput for IOPS (QPS)**



# JDBC(Java Database Connectivity)

## DataSource:

- Driver
- URL,
- Username,
- Password

跟手动连接数据库和获取数据的步骤差不多，只是变成了Java语句。

## Connecting database using Java and Database client

The screenshot shows a Java IDE with a class named `JDBC` and a database client configuration window. The code defines static variables for driver, URL, username, and password, and a method to connect to the database. The configuration window shows the same settings for a MySQL driver.

```
import java.sql.*;

/**
 * @author b1go
 * @date 6/14/22 11:48 PM
 */
public class JDBC {
    private static final String DRIVER = "com.mysql.jdbc.Driver";
    private static final String URL = "jdbc:mysql://localhost:3361/EMP";
    private static final String USERNAME = "chuwa_geek";
    private static final String PASSWORD = "chuwa_yyds";

    public Employee getEmployeeById(int id) throws Exception {
        Employee employee = new Employee();

        Connection conn = null;
        Statement stmt = null;
        ResultSet rs = null;

        try {
            // 1, load Driver
            Class.forName(DRIVER);
            // 2, connect to Database;
            conn = DriverManager.getConnection(URL, USERNAME, PASSWORD);
            // 3, define sql statement
            String sql = "SELECT * FROM emp WHERE ID = " + id;
            // 4, create a statement object
        }
    }
}
```

The configuration window shows the following settings:

- Name: @localhost
- Comment:
- Connection type: default
- Driver: MySQL
- Host: localhost
- Port: 3306
- Authentication: User & Password
- User: springstudent
- Password: <hidden>
- Save: Forever
- Database:
- URL: jdbc:mysql://localhost:3306
- Test Connection: MySQL 8.0.22

Arrows indicate the mapping between code variables and configuration fields:

- `DRIVER` points to the Driver field.
- `URL` points to the URL field.
- `USERNAME` points to the User field.
- `PASSWORD` points to the Password field.

## Basic flow:

1. load Driver
2. connect to Database using username, password and url
3. define SQL statement
4. use stmt object to execute sql statement and use stmt object to execute sql statement;
5. get ResultSet's data and set them to java object(employee)
6. close connections and other resource.

**Notice** `try... catch...catch...finally`

Catch 范围从小到大。finally关闭已打开的资源。

```
1 public class JDBC {
```

```

1 public class JDBC {
2     private static final String DRIVER = "com.mysql.jdbc.Driver";
3     private static final String URL = "jdbc:mysql://localhost:3361/EMP";
4     private static final String USERNAME = "chuwa_geek";
5     private static final String PASSWORD = "chuwa_yyds";
6
7     public Employee getEmployeeById(int id) throws Exception {
8         Employee employee = new Employee();
9
10        Connection conn = null;
11        Statement stmt = null;
12        ResultSet rs = null;
13
14        try {
15            // 1, load Driver
16            Class.forName(DRIVER);
17            // 2, connect to Database;
18            conn = DriverManager.getConnection(URL, USERNAME, PASSWORD);
19            // 3, define sql statement
20            String sql = "SELECT * FROM emp WHERE ID = " + id;
21            // 4, create a statement object
22            stmt = conn.createStatement();
23            // 5, use stmt object to execute sql statement;
24            rs = stmt.executeQuery(sql); // the result is return to ResultSet
25
26            while(rs.next()) {
27                // 6, get ResultSet's data to java object(employee)
28                employee.setId(rs.getInt("id"));
29                employee.setName(rs.getString("name"));
30            }
31
32            // 7, close conections and other resource.
33            rs.close();
34            stmt.close();
35            conn.close();
36
37            return employee;
38        } catch (SQLException e) {
39            e.printStackTrace();
40        } catch (Exception e) {
41            e.printStackTrace();
42        } finally {
43            if (rs != null) {
44                rs.close();
45                rs = null;
46            }
47
48            if (stmt != null) {

```

```

47         if (stmt != null) {
48             stmt.close();
49             stmt = null;
50         }
51         if (conn != null) {
52             conn.close();
53             conn = null;
54         }
55     }
56
57     return null;
58 }
59 }

```

## Connecting to database in SpringBoot

The screenshot shows the Spring Boot IDE with the `application-dev.yml` file open. The database configuration is as follows:

```

spring:
  datasource:
    url: jdbc:mysql://localhost:3306/mall
    username: springstudent
    password: springstudent
  druid:
    initial-size: 5 #连接池初始化大小
    min-idle: 10 #最小空闲连接数
    max-active: 20 #最大连接数
    web-stat-filter:
      exclusions: "*.js,*.gif,*.jpg,*.p
    stat-view-servlet: #访问监控网页的登录
      login-username: druid
      login-password: druid
  data:
    elasticsearch:
      repositories:
        enabled: true

```

The Data Sources configuration window is also open, showing the following details for the `mall@localhost` data source:

- Name: `mall@localhost`
- Comment: (empty)
- General tab selected
- Connection type: `default` Driver: `MySQL`
- Host: `localhost` Port: `3306`
- Authentication: `User & Password`
- User: `springstudent`
- Password: `<hidden>`
- Database: `mall`
- URL: `jdbc:mysql://localhost:3306/mall`

```

1 List<User> users = new ArrayLsit<>();
2 user1 <- database;
3 users.add(user1)

```

## REST API(End-Point)

"Java Interface"

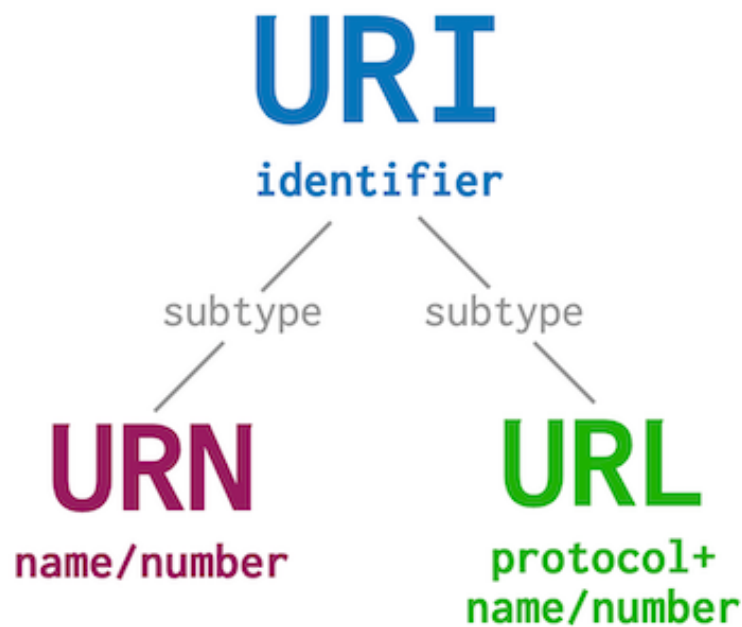
End-Point --> End to End Point

URL URL URL (Unique Resource)

# URI, URL, URN (Unique Resource)

URN: **google.com**

URL: <https://google.com>



DANIEL MIESSLER 2022

## URI

- URN
- protocol + URN
  - **http://** (Web/APIs)
  - **Https://** (Secured Web/APIs)
  - **ftp://** (for file transfer)
  - **mailto://** (for email transfer)

## URL Structure

# URI/URL

## URI

## URN

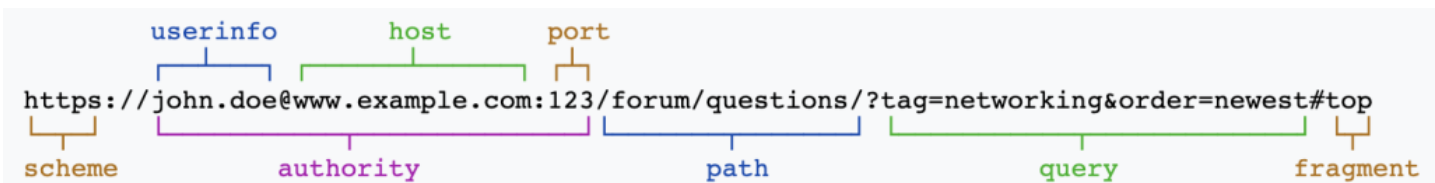
## SCHEME

## HOST

## PATH

<https://google.com/articles/articlename>

DANIEL MIESSLER 2022



## Path Variables and Request Parameter

```
1 http://localhost:8080/userapp/users/{id}/load?minAge=20&lastName=Stark;
2 http://localhost:8080/userapp/users/1234/load?minAge=20&lastName=Stark;
3
4 http://localhost:8080/userapp/users/load?userId=1234&minAge=20&lastName=Stark;
5
6 http://localhost:8080/hotel/filter?location=???&price=???&nearAiprot=?
```

Path Variable: **2**

Request Parameter: **?key1=value1&key2=value2**

Question: when do we use **path variable**? when do we use **request parameter**?

## Path Variable examples

<https://drive.google.com/drive/u/0/my-drive?fileName=Chuwa>

<https://drive.google.com:443/drive/u/1/my-drive>

<https://drive.google.com/drive/u/0/computers>

<https://drive.google.com/drive/u/0/my-drive>

<https://drive.google.com/drive/u/0/shared-with-me>

<https://drive.google.com/drive/u/0/recent>

<https://drive.google.com/drive/u/0/starred>

<https://drive.google.com/drive/u/0/starred>

<https://drive.google.com/drive/u/0/trash>

Question: **what is the path varibale in the below url?**

[https://docs.google.com/document/d/1m71\\_YP-V6E4232323-KMVNgkiLr72d6qPID1aWjI9Y28/edit](https://docs.google.com/document/d/1m71_YP-V6E4232323-KMVNgkiLr72d6qPID1aWjI9Y28/edit)

/api/v1/document/d/{id}/edit

/api/v1/document/d/{id}/view

**Request Parameter (there is a lenght limit, SD:"TinyURL")**

<https://www.google.com/search?q=chuwa&oq=chuwa&aqs=chrome..69i57j69i60l4.1261j0j4&sourceid=chrome&ie=UTF-8>

## HTTP Methods and Status Code

HTTP Verb/Type	CRUD	Entire Collection (e.g. /customers)	Specific Item (e.g. /customers/{id})
POST	<b>Create</b> & Save	201 (Created), 'Location' header with link to /customers/{id} containing new ID.	<b>404 (Not Found)</b> , 409 (Conflict) if resource already exists..
GET	Read	200 (OK), list of customers. Use pagination, sorting and filtering to navigate big lists.	200 (OK), single customer. 404 (Not Found), if ID not found or invalid.
PUT	Update/ <b>Replace</b> (e.g. replace first name)	405 (Method Not Allowed), unless you want to update/replace every resource in the entire collection.	200 (OK) or 204 (No Content). 404 (Not Found), if ID not found or invalid.
PATCH	Update/ <b>Modify</b> (e.g. modify address)	405 (Method Not Allowed), unless you want to modify the collection itself.	200 (OK) or 204 (No Content). 404 (Not Found), if ID not found or invalid.
DELETE	Delete	405 (Method Not Allowed), unless you want to delete the whole collection—not often desirable.	200 (OK). 404 (Not Found), if ID not found or invalid.

Question:

- what is the difference between PUT and POST ?
- what is the difference between PUT and PATCH ?

<u>STATUS CODE RANGE</u>	MEANING
100 – 299	1XX Informational Responses. For example, 102 indicates the resource is being processed. 2XX <b>success</b> status. E.g. 200/204 etc. ( <b>Mostly okay</b> )
300 – 399	Redirects For example, 301 means <b>Moved permanently</b> (e.g. redirect)
400 – 499	Client-side errors 400 means bad request and 404 means <b>resource</b> not found ( <b>applicaiton error</b> )
500 – 599	Server-side errors For example, 500 means an internal <b>server error</b> (e.g. 503)

## REST Resource Naming Guide/Convention

**独孤九剑**：并不一定能懂原理或专业词汇，但是见多了，用到的时候自然而然就会了。

原理不懂就当多看例子，多见例子很有用

1. Use **JSON** as the Format for Sending and Receiving Data

2. Use **Nouns** Instead of **Verbs** in Endpoints

1. `POST /api/Users` //NOT `/api/createUser`
2. `GET /api/users/{id}` - `/api/getUser` WRONG
3. `DELETE /api/users/{id}` - `/api/deleteUser` WRONG
4. `UPDATE/PATCH/PUT /api/users/{id}` -> `UPDATE /api/users/101`

3. Use **Status Codes** in Error Handling

4. Use **nouns** to represent resources

1. <http://api.example.com/device-management/managed-devices>  
<http://api.example.com/device-management/managed-devices/{device-id}>  
<http://api.example.com/user-management/users>  
<http://api.example.com/user-management/users/{id}>
2. document
  1. <http://api.example.com/device-management/managed-devices/{device-id}>  
<http://api.example.com/user-management/users/{id}>  
<http://api.example.com/user-management/users/admin>
  2. What is the document in the above google drive url example?
3. collection
  1. <http://api.example.com/device-management/managed-devices>

```
http://api.example.com/user-management/users
```

```
http://api.example.com/user-management/users/{id}/accounts
```

#### 4. store

1. <http://api.example.com/song-management/users/{id}/playlists>

#### 5. controller

1. <http://api.example.com/cart-management/users/{id}/cart/checkout>  
<http://api.example.com/song-management/users/{id}/playlist/play>

### 5. Consistency is the key

1. Use forward **slash (/)** to indicate hierarchical relationships

```
1 http://api.example.com/device-management
2 http://api.example.com/device-management/managed-devices
3 http://api.example.com/device-management/managed-devices/{id}
4 http://api.example.com/device-management/managed-devices/{id}/scripts
5 http://api.example.com/device-management/managed-devices/{id}/scripts/{id}
```

2. Do NOT use **trailing forward slash (/)** in URIs

```
1 http://api.example.com/device-management/managed-devices/
  http://api.example.com/device-management/managed-devices /*This is much
  better version*/
```

3. Use **hyphens (-)** to improve the readability of URIs

```
1 http://api.example.com/device-management/managed-devices/
2 http://api.example.com/device-management/managed-devices
3 /*This is much better version then `manageddevices`*/
```

4. Do not use underscores (**\_**)

```
1 http://api.example.com/inventory-management/managed-entities/{id}/install-
  script-location //More readable
2
3 http://api.example.com/inventory-
  management/managedEntities/{id}/install_Script_Location //Less readable
```

5. Use lowercase letters in URIs



```

1 http://api.example.org/my-folder/my-doc //1
2 HTTP://API.EXAMPLE.ORG/my-folder/my-doc //2
3 http://api.example.org/My-Folder/my-doc //3

```

## 6. Do not use file extensions

```

1 http://api.example.com/device-management/managed-devices.xml /*Do not use it*/
2
3 http://api.example.com/device-management/managed-devices /*This is correct
  URI*/

```

## 7. Never use CRUD function names(or Verbs) in URIs

```

1 HTTP GET http://api.example.com/device-management/managed-devices //Get all
  devices
2 HTTP POST http://api.example.com/device-management/managed-devices //Create
  new Device
3
4 HTTP GET http://api.example.com/device-management/managed-devices/{id} //Get
  device for given Id
5 HTTP PUT http://api.example.com/device-management/managed-devices/{id}
  //Update device for given Id
6 HTTP DELETE http://api.example.com/device-management/managed-devices/{id}
  //Delete device for given Id

```

## 8. Use query component to filter URI collection

```

1 http://api.example.com/device-management/managed-devices
2 http://api.example.com/device-management/managed-devices?region=USA
3 http://api.example.com/device-management/managed-devices?region=USA&brand=XYZ
4 http://api.example.com/device-management/managed-devices?
  region=USA&brand=XYZ&sort=installation-date

```

## 2. Paging

```

1 https://api.spotify.com/v1/artists/12vsllerkjdsasjc/albums?
  album_type=SINGLE&offset=20&limit=10

```

## 9. Be Clear with Versioning (mobile app -- forward/backward compatible APIs)

```

1 https://mysite.com/api/v1/posts (user without address)
2 https://mysite.com/api/v2/posts (user with address, creditCard info)

```

## Design

design 3 APIs (思考: path variable 怎么用? 有sub resources, 哪些地方该用复数)

1. find the customer's payments, like credit card 1, credit card 2, paypal, Apple Pay.
2. Find the customer's history orders from 10/10/2022 to 10/24/2022
3. find the customer's delivery addresses
4. If I also want to get customer's default payment and default delivery address, what kind of the API (URL) should be?

```
1 /api/v1/customers/amy/orders
2 /api/v1/customers/amy/payments
3 /api/v1/customers/amy/payments/{paymentTypeId}
4 /api/v1/customers/chenyu/addresses
5
6 default payment:
7 /api/v1/customers/{user name/id}/default-payment
8 /api/v1/customers/{user}/payments/default (improved version)
9
10 default delivery:
11 /api/v1/customers/{user name/id}/default-delivery
12 /api/v1/customers/{user}/addresses/default (improved version)
13
14 history orders from 10/10/2022 to 10/24/2022
15 /api/v1/orders/{userId}/daterange?start-date >= {} & end-date <= {}
16
17 /api/v1/customers/{user}/orders?start-date="10/10/2022"&end-date="10-24-2022"
    (improved version)
```

## Exercise (super useful)

Practice Request site: <https://reqres.in/>

Payload variables: <https://developers.tackle.io/reference/payload-variables>

## Reference

- <https://danielmiessler.com/study/difference-between-uri-url/>
- <https://restfulapi.net/resource-naming/>
- <https://www.freecodecamp.org/news/rest-api-best-practices-rest-endpoint-design-examples/>

## GraphQL (RoR tutorial)

## GraphQL ([NOT Tutorial](#))

面试几乎不不怎么考，但是现在越来越多的大厂开始采用, 简单介绍一下, 如果有兴趣深入了解的可以自学。

(e.g. Github, Facebook/Meta, Airbnb, Shopify)

[Understanding RPC, REST and GraphQL](#)

## RPC (Remote Procedure Call - with Http/AMQP) → Web APIs;

- **SOAP** is one of them; (using **WSDL** + **XML** vs. **JSON**(in REST) )
- **gRPC** (modern) better SOAP, uses **ProtoBuff** (schema + data) kind like **WSDL**;

## REST

- **JSON Schema** → inspired by XML-Schema;
- Can handle multiple content types **json/text**, **image/png** etc.
  - More **versatile** on this aspect;
- **Caching** is easier;

## [GraphQL \(vs.Rest\)](#)

- RPC with good part from REST/HTTP;
- **Kind like SQL**;
- defaults to providing the very smallest response from an API
- You can only speak in terms of “**Fields**”
- **Better API evolution** support; (deprecation easy)

e.g.

```
rails server --> http://localhost:3000/graphql
```

```
1 query {
2   allFood{
3     id
4     name
5   }
6 }
7
8 query{
9   nutrition{
10    id
11    calories
12    foodId
```

```
13     }
14 }
15
16
17 Create:
18 mutation {
19     foodCreate(input: {
20         name: "Apple Pie Pie",
21         placeOfOrigin: "US",
22         image: "apple-pie.png"
23     }) {
24         id
25         name
26         placeOfOrigin
27         image
28     }
29 }
30
31 Update:
32 mutation {
33     foodUpdate(input:{
34         id:4,
35         name:"super pumpkin pie !!!"
36     }){
37         id
38         name
39     }
40 }
41
42 Delete:
43 mutation {
44     foodDelete(input: {
45         id: 4
46     }) {
47         id
48     }
49 }
```

## Postman\*

---

## Public APIs

[Postman public API](#)

[Explore the World of APIs](#)

API 练习网站: <https://regres.in/>

## API Collection

Home Workspaces ▾ API Network ▾ Explore

My Workspace New Import

Overview GET Book

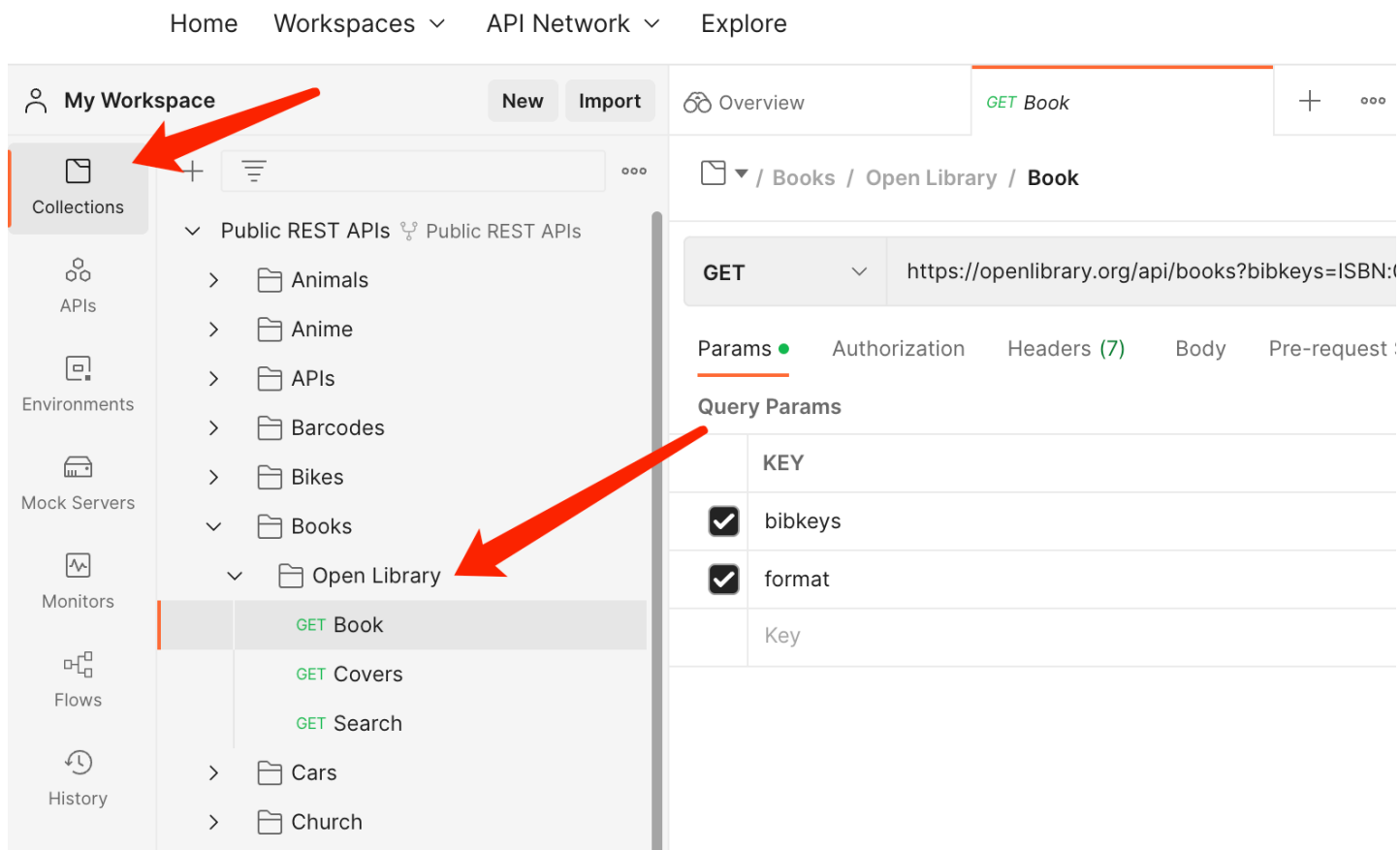
▼ / Books / Open Library / Book

GET ▼ https://openlibrary.org/api/books?bibkeys=ISBN:1

Params ● Authorization Headers (7) Body Pre-request :

Query Params

	KEY
<input checked="" type="checkbox"/>	bibkeys
<input checked="" type="checkbox"/>	format
	Key



## Environments

Home Workspaces ▾ API Network ▾ Explore Search Postman

My Workspace New Import

Overview GET Book PROD

PROD

	VARIABLE	TYPE ①	INITIAL VALUE ②
<input checked="" type="checkbox"/>	URL	default	▼ https://openlibrary.org
	Add a new variable		

