1. Java Script. 1.1. Extend JS Date object with a method daysTo() which returns number of complete days between any pair of JS date objects: d1.daysTo(d2) should return quantity of complete days from d1 to d2.

```javascript
Date.prototype.daysTo = function(targetDate) {

    if (isNaN(this.getTime()) || isNaN(targetDate.getTime())) {
        throw new Error("date is invalid."); // validate the d1, d2 input date
    }

    const timeDifference = targetDate.getTime() - this.getTime(); // convert date to milisecond timestamp
    const countDays = Math.floor(timeDifference / (1000 * 60 * 60 * 24));
    return countDays;
};

const d1 = new Date('2024-12-12');
const d2 = new Date('2024-12-15');
console.log(d1.daysTo(d2));
```

```python
from datetime import datetime

def days_to(d1, d2):
    return (d2 - d1).days  # gives the complete days difference

d1 = datetime(2024, 12, 10)
d2 = datetime(2024, 12, 15)

print(days_to(d1, d2))
```

✓ 0.0s

```
5
```

1.2. Please order by Total Develop a program which produces ordered array of sales. Input: array of objects with the following structure {amount: 10000, quantity: 10}. Output: new array of ordered sales. Array element structure should be: {amount: 10000, quantity: 10, Total: 100000}, where Total = amount * quantity. Please order by Total and note that input array shall remain intact.

```javascript
function orderSalesByTotal(sales) {
    const salesWithTotal = sales.map(sale => ({
        ...sale, // copy sales properties
        Total: sale.amount * sale.quantity // calculate Total
    }));

    return salesWithTotal.sort((a, b) => b.Total - a.Total);
}

const sales = [
    { amount: 10000, quantity: 10},
    { amount: 5000, quantity: 15 },
    { amount: 2000, quantity: 20 }
];

console.log('Original Sales:', sales);
const orderedSales = orderSalesByTotal(sales);
console.log('Ordered Sales:', orderedSales);
```

```python
def ordered_sales(input_array):
    sales_with_total = [
        {**sale, "Total": sale["amount"] * sale["quantity"]} for sale in sales # create new dictionary with the total
    ]
    ordered_sales = sorted(sales_with_total, key=lambda x: x["Total"],reverse=True)   # sort in ascending order
    return ordered_sales

sales = [
    {"amount": 10000, "quantity": 10},
    {"amount": 5000, "quantity": 15},
    {"amount": 2000, "quantity": 20}
]

output = ordered_sales(sales) # get ordered sales
print("Ordered Sales:", output)
```

19] ✓ 0.0s Python

```
': 10000, 'quantity': 10, 'Total': 100000}, {'amount': 5000, 'quantity': 15, 'Total': 75000}, {'amount': 2000, 'quantity': 20, 'Total': 40000}]
```

1.3. Develop a program "Object Projection". Input: any JSON object; prototype object. Output: projected object. Projected object structure shall be intersection of source object and prototype object structures. Values of properties in projected object shall be the same as values of respective properties in source object.

```javascript
function Object_projection(src, proto) {
    const result = {};
    function traverse(srcObj, protoObj, resObj) {
        for (let key in protoObj) {
            if (protoObj.hasOwnProperty(key) && srcObj.hasOwnProperty(key)) { // key exists in both source and prototype
                if (protoObj[key] && typeof protoObj[key] === 'object' && protoObj[key] !== null) {
                    resObj[key] = {};    // Initialize the sub-object in result
                    traverse(srcObj[key], protoObj[key], resObj[key]); // Recurse into the nested objects
                } else {
                    resObj[key] = srcObj[key]; //prototype has no nested objects, assign the value from the source
                }
            }
        }
    }

    traverse(src, proto, result);
    return result;
}
```

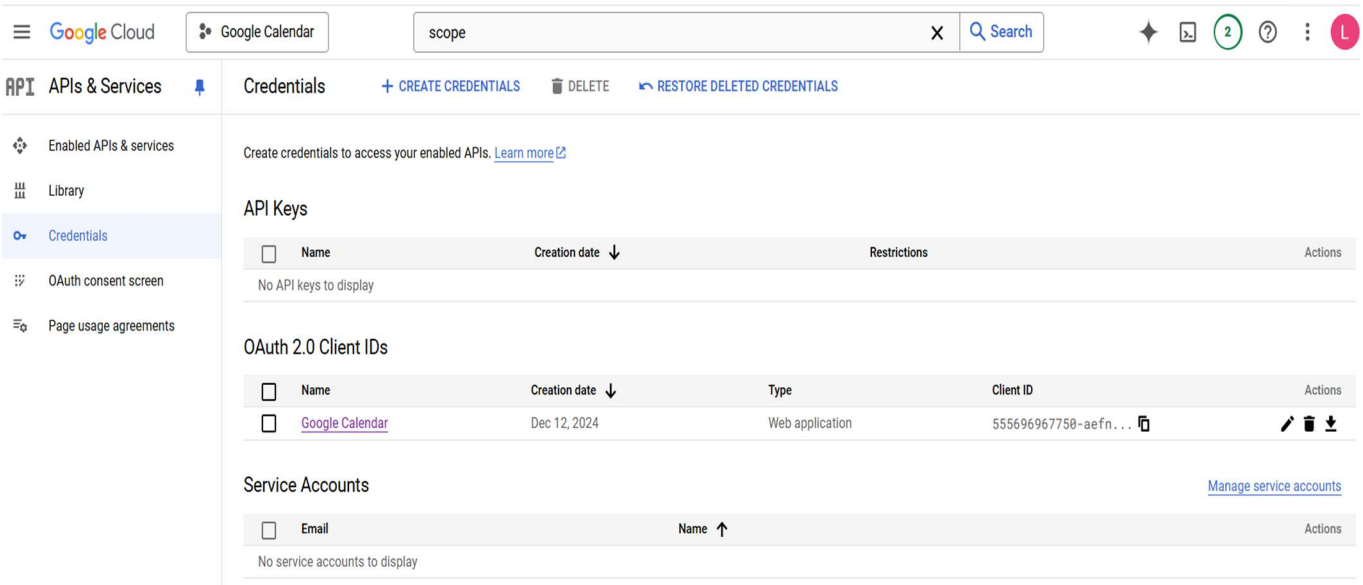
```javascript
const src = {
    prop11: {
        prop21: 21,
        prop22: {
            prop31: 31,
            prop32: 32
        }
    },
    prop12: 12
};

const proto = {
    prop11: {
        prop22: null
    }
};

const projectedObject = Object_projection(src, proto);
console.log(JSON.stringify(projectedObject, null, 2));
```

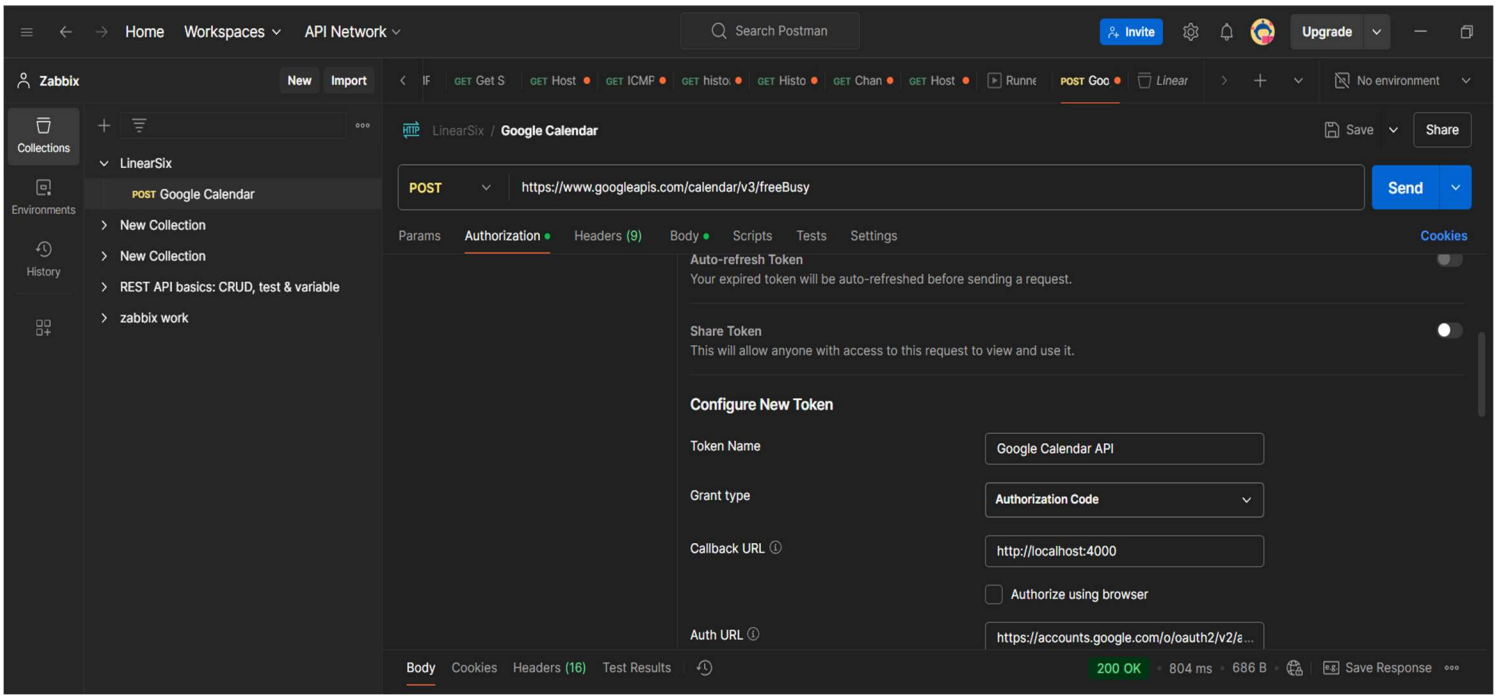
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS JUPYTER SERIAL MONITOR

```
PS G:\Semi-8\LinearSix> node "g:\Semi-8\LinearSix\1.3_src_proto.js"
{
  "prop11": {
    "prop22": {
      "prop31": 31,
      "prop32": 32
    }
```

2.1. Develop a program in JS which returns array of free/busy intervals in a given time period for any shared Google calendar. Input: shared Google calendar ID; time period (starting and ending moments). Output: array of busy intervals.

Alternatively (if 2.1 is too difficult to develop) provide sequence of REST API calls that can be executed in REST API client (Postman) in order to achieve the same result.





Zabbix

New Import

Collections

Environments

History

LinearSix

POST Google Calendar

New Collection

New Collection

REST API basics: CRUD, test & variable

zabbix work

LinearSix / Google Calendar

Save Share

POST https://www.googleapis.com/calendar/v3/freeBusy Send

Params Authorization Headers (9) Body Scripts Tests Settings Cookies

none form-data x-www-form-urlencoded raw binary GraphQL JSON Beautify

```
{
  "timeMin": "2024-12-12T00:00:00Z",
  "timeMax": "2024-12-12T23:59:59Z",
  "items": [
    {
      "id": "s.s.luxshan@gmail.com"
    }
  ]
}
```

Body Cookies Headers (16) Test Results 200 OK 804 ms 686 B Save Response

Pretty Raw Preview Visualize JSON

```
{
    "kind": "calendar#freeBusy",
    "timeMin": "2024-12-12T00:00:00.000Z",
    "timeMax": "2024-12-12T23:59:59.000Z",
    "calendars": {
        "s.s.luxshan@gmail.com": {
            "busy": [
                {
                    "start": "2024-12-12T13:30:00Z",
                    "end": "2024-12-12T14:30:00Z"
                }
            ]
        }
```

Online Find and replace Console Postbot Runner Start Proxy Cookies Vault Trash

Zabbix

New Import

Collections

Environments

History

LinearSix

POST Google Calendar

New Collection

New Collection

REST API basics: CRUD, test & variable

zabbix work

LinearSix / Google Calendar

Save Share

POST https://www.googleapis.com/calendar/v3/freeBusy Send

Params Auth Headers (9) Body Scripts Tests Settings Cookies

raw JSON Beautify

```
{
  "timeMin": "2024-12-12T00:00:00Z",
  "timeMax": "2024-12-12T23:59:59Z",
  "items": [
    {
      "id": "s.s.luxshan@gmail.com"
    }
  ]
}
```

Body 200 OK 804 ms 686 B

Pretty Raw Preview Visualize JSON

```
{
    "kind": "calendar#freeBusy",
    "timeMin": "2024-12-12T00:00:00.000Z",
    "timeMax": "2024-12-12T23:59:59.000Z",
    "calendars": {
        "s.s.luxshan@gmail.com": {
            "busy": [
                {
                    "start": "2024-12-12T13:30:00Z",
                    "end": "2024-12-12T14:30:00Z"
                }
            ]
        }
```

Code snippet

NodeJs - Request

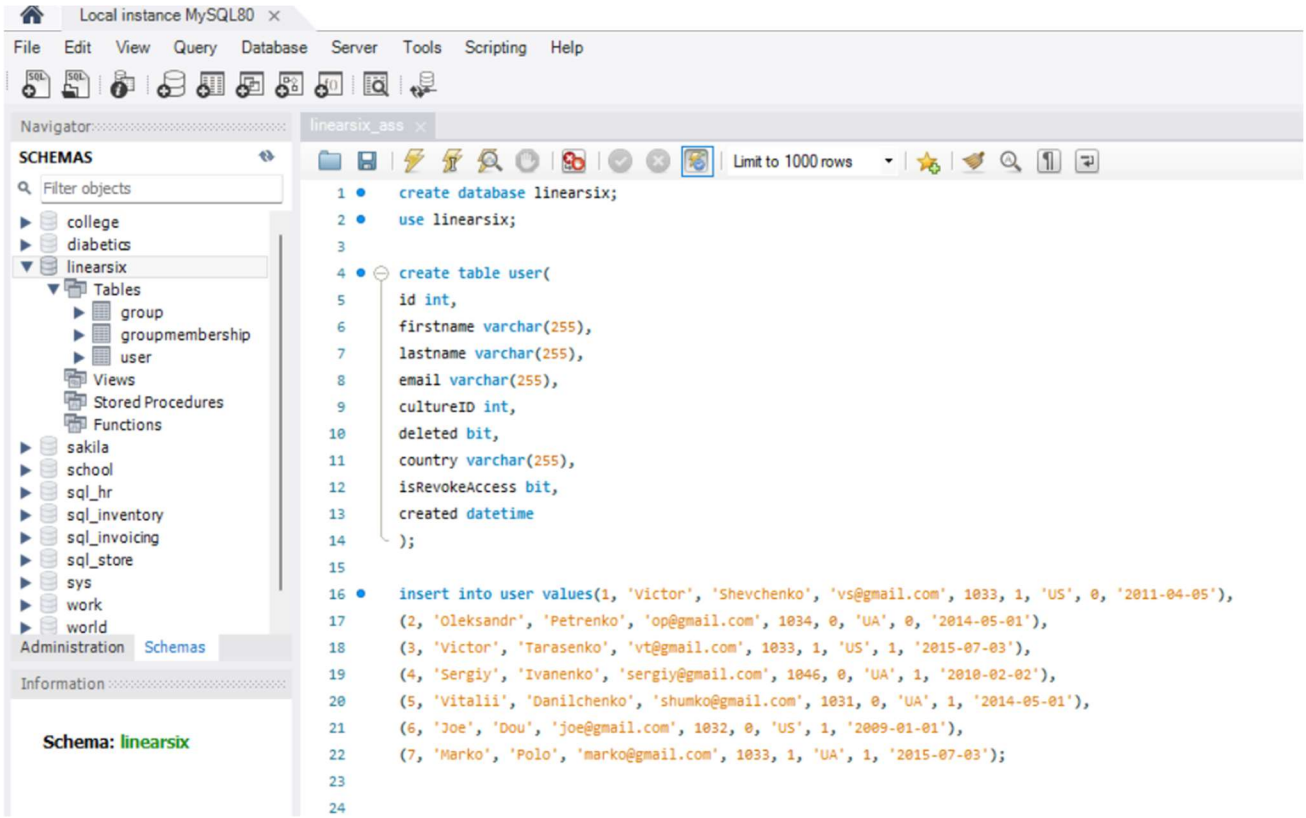
```
var request = require('request');
var options = {
  'method': 'POST',
  'url': 'https://www.googleapis.com/calendar/v3/freeBusy',
  'headers': {
    'Content-Type': 'application/json',
    'Authorization': '••••••'
  },
  body: JSON.stringify({
    "timeMin": "2024-12-12T00:00:00Z",
    "timeMax": "2024-12-12T23:59:59Z",
    "items": [
      {
        "id": "s.s.luxshan@gmail.com"
      }
    ]
  })

};
request(options, function (error, response) {
  if (error) throw new Error(error);
  console.log(response.body);
});
```

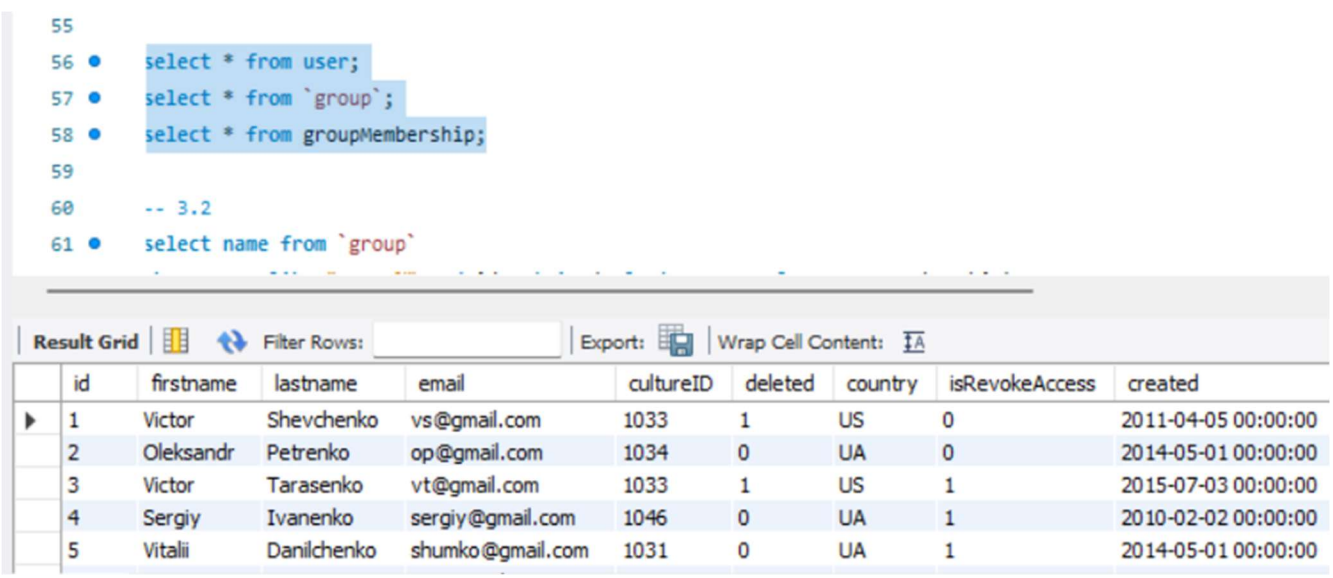
NodeJs - Request

```javascript
var request = require('request');
var options = {
  'method': 'POST',
  'url': 'https://www.googleapis.com/calendar/v3/freeBusy',
  'headers': {
    'Content-Type': 'application/json',
    'Authorization': '••••••'
  },
  body: JSON.stringify({
    "timeMin": "2024-12-12T00:00:00Z",
    "timeMax": "2024-12-12T23:59:59Z",
    "items": [
      {
        "id": "s.s.luxshan@gmail.com"
      }
    ]
  })

};
request(options, function (error, response) {
  if (error) throw new Error(error);
  console.log(response.body);
});

```

Please prepare scripts executable on this Try-SQL Editor

3.1. Create tables and insert data





id	firstname	lastname	email	cultureID	deleted	country	isRevokeAccess	created
1	Victor	Shevchenko	vs@gmail.com	1033	1	US	0	2011-04-05 00:00:00
2	Oleksandr	Petrenko	op@gmail.com	1034	0	UA	0	2014-05-01 00:00:00
3	Victor	Tarasenko	vt@gmail.com	1033	1	US	1	2015-07-03 00:00:00
4	Sergiy	Ivanenko	sergiy@gmail.com	1046	0	UA	1	2010-02-02 00:00:00
5	Vitalii	Danilchenko	shumko@gmail.com	1031	0	UA	1	2014-05-01 00:00:00

Result Grid | Filter Rows:

id	name	created
10	Support	2010-02-02 00:00:00
12	Dev team	2010-02-03 00:00:00
13	Apps team	2011-05-06 00:00:00
14	TEST-dev-team	2013-05-06 00:00:00
15	Guest	2014-02-02 00:00:00

user 1 | group 2 × | groupMembership 3

Result Grid | Filter Rows: | Exp

id	userID	groupID	created
110	2	10	2010-02-02 00:00:00
112	3	15	2010-02-03 00:00:00
114	1	10	2014-02-02 00:00:00
115	1	17	2011-05-02 00:00:00
117	4	12	2014-07-13 00:00:00

user 1 | group 2 | groupMembership 3 ×

3.2. Select names of all empty test groups (group name starts with “TEST-”).

```
-- 3.2
select name from `group`
where name like "TEST-%" and id not in (select groupID from groupMembership);

-- 3.3
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

name
TEST-dev-team
TEST-QA-team

3.3. Select user first names and last names for the users that have Victor as a first name and are not members of any test groups (they may be members of other groups or have no membership in any groups at all).

```
-- 3.3
select firstname, lastname from user
where firstname = 'Victor' and id not in (select userID from groupMembership
where groupID in (select id from `group` where name like "TEST-%"));
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

firstname	lastname
Victor	Tarasenko

3.4. Select users and groups for which user was created before the group for which he(she) is member of.

```
-- 3.4
select user.firstname, user.lastname, `group`.name
from user
join groupMembership
on user.id = groupMembership.userID
join `group`
on groupMembership.groupID = `group`.id
where user.created < `group`.created;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

firstname	lastname	name
Sergiy	Ivanenko	Dev team