

Run Server: npm run dev1

Run Client: Thunder Client

Run Database: MySQL

## DELETE Request

→ localhost:3000/api/user/5

<https://www.prisma.io/docs/orm/overview/prisma-in-your-stack/rest>

## REST API server example

### DELETE

```
app.delete(`/post/:id`, async (req, res) => {  
  const { id } = req.params  
  const post = await prisma.post.delete({  
    where: {  
      id: Number(id),  
    },  
  })  
  res.json(post)  
})
```

## serverDeleteRequest.js

```
// delete USERT by id
// DELETE api/user/:id
app.delete('/api/user/:id', async (req, res) => {
  const paramsUserId = req.params.id;
  console.log('req.params.id: ' + paramsUserId);

  //Find USER record by ID in USER table | Validation
  try {
    const userById = await prisma.user.findUnique({
      where: { id: parseInt(req.params.id) }
    });
    if (!userById) {
      return res.status(404).json({
        err: 'could not find USER Record in table by ID: ' + req.params.id
      });
    }

    // Find PROFILE records by User-ID in PROFILE table.
    // NOTE: User and Profile is a one-to-many relationship. It can return more than one profile
    try {
      const profiles = await prisma.profile.findMany({
        where: { userId: parseInt(req.params.id) }
      });
    } catch (error) {
      console.log('Error finding profiles: ', error);
    }
  } catch (error) {
    console.log('Error finding user: ', error);
  }
});
```

## serverDeleteRequest.js

```
import { PrismaClient } from '@prisma/client';
import express from 'express';

const app = express();
const prisma = new PrismaClient();

// custom middleware
app.use((req, res, next) => {
  console.log(`${req.url} ${new Date()}`);
  next(); // call the next middleware in the stack
})

//-----
// RUN THIS SERVER:
// nodemon --watch --exec node src/serverDeleteRequest.js
```

```

//-----

/*
// SEE: prisma.schema.prisma
// USER and POST is one-to-many relationship.
// USER and PROFILES is one-to-many relationship.
// TO DELETE A POST RECORDED, WE NEED FIRST TO
// 1. DELETE THE POST WITH FOREIGN-KEY: POST-authorId=USER-id
// 2. DELETE THE PROFILE WITH FOREIGN-KEY: PROFILEID-userid=USER-id
// 3. DELETE THE USER WITH id (USER-id=POST-authorId=PROFILEID-userid)
// FOR EXAMPLE. TO DELETE A USER-id=18;
// CHECK:
// select * from post where authorId=18;
// select * from profile where userId=18;
// select * from user where Id=18;
// DELETE:
// delete from post where authorId=18;
// commit;
// delete from profile where userId=18;
// commit;
// delete from user where id =18;
// commit;
*/

// delete USERT by id
// DELETE api/user/:id
app.delete('/api/user/:id', async (req, res) => {
  const paramsUserId = req.params.id;
  console.log('req.params.id: ' + paramsUserId);

  //Find USER record by ID in USER table | Validation
  try {
    const userById = await prisma.user.findUnique({
      where: { id: parseInt(req.params.id) }
    });
    if (!userById) {
      return res.status(404).json({
        err: 'could not find USER Record in table by ID: ' + req.params.id
      });
    }
  }
}

```

```

// Find PROFILE records by User-ID in PROFILE table.
// NOTE: User and Profile is a one-to-many relationship. It can return more
than one records.
try {
  const profiles = await prisma.profile.findMany({
    where: { userId: parseInt(req.params.id) }
  });
  console.log('PROFILES records found by userId: ' + JSON.stringify(profiles));
  // delete all Profile records with Foreign key (PROFILE-User-ID = USER-ID)
  // If found process looping all found records and delete them by id.
  if (profiles && profiles.length > 0) {
    for (let i = 0; i < profiles.length; i++) {
      console.log('process delete PROFILES with ID): ' + profiles[i].id);
      let deletedProfilesById = await prisma.profile.delete({
        where: { id: parseInt(profiles[i].id) }
      });
    }
  }
}
catch (err) {
  console.log('Delete Profil records failed with err: ' + JSON.stringify(err));
  console.log('Profile record could not deleted with userId: ' + req.params.id);
}

// Find POST records by Author-ID in POST table.
// NOTE: User and POST is a one-to-many relationship. It can return more than
one records.
try {
  const posts = await prisma.post.findMany({
    where: { authorId: parseInt(req.params.id) }
  });
  console.log('POST records found by authorId: ' + JSON.stringify(posts));
  // delete all Post records with Foreign key (POST-Author-ID = USER-ID)
  // If found process looping all found records and delete them by id.
  if (posts && posts.length > 0) {
    for (let i = 0; i < posts.length; i++) {
      console.log('process delete POST with ID): ' + posts[i].id);
      let deletedPostById = await prisma.post.delete({
        where: { id: parseInt(posts[i].id) }
      });
    }
  }
}

```

```

    }
  }
  catch (err) {
    console.log('Delete POST records failed with err: ' + JSON.stringify(err));
    console.log('Post record could not be deleted with authorId: ' + req.params.id);
  }

  // delete User
  try {
    const deletedUser = await prisma.user.delete({
      where: {
        id: parseInt(req.params.id)
      }
    })
    console.log('delete User successfully. deleted-User: ' +
JSON.stringify(deletedUser));
    return res.status(200).json(deletedUser);
  }
  catch (err) {
    console.log('Delete User record failed with Id: ' + req.params.id);
    console.log('err: ' + JSON.stringify(err));
    return res.status(404).json(err);
  }
}
catch (err) {
  console.log('User record not found with id: ' + params.userid);
  return res.status(404).json(err);
}
});

app.listen(3000, () => {
  console.log('Server is running at port 3000');
})

```

On package.json: add dev1.

"dev1": "nodemon --watch --exec node src/serverDeleteRequest.js",

```
{
  "name": "hello-prisma",
  "type": "module",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "dev": "nodemon --watch --exec node src/server.js",
    "dev1": "nodemon --watch --exec node src/serverDeleteRequest.js",
    "test": "echo \\\"Error: no test specified\\\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "devDependencies": {
    "@types/node": "^22.5.5",
    "nodemon": "^3.1.4",
    "prisma": "^5.19.1",
    "ts-node": "^10.9.2",
    "typescript": "^5.6.2"
  },
  "dependencies": {
    "@prisma/client": "^5.19.1",
    "@types/express": "^4.17.21",
    "express": "^4.21.0"
  }
}
```

```
{
  "name": "hello-prisma",
  "type": "module",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "dev": "nodemon --watch --exec node src/server.js",
    "dev1": "nodemon --watch --exec node src/serverDeleteRequest.js",
    "test": "echo \\\"Error: no test specified\\\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "devDependencies": {
    "@types/node": "^22.5.5",
    "nodemon": "^3.1.4",
    "prisma": "^5.19.1",
    "ts-node": "^10.9.2",
    "typescript": "^5.6.2"
  },
  "dependencies": {
    "@prisma/client": "^5.19.1",
    "@types/express": "^4.17.21",
    "express": "^4.21.0"
  }
}
```

### Package.json

```
{
  "name": "hello-prisma",
  "type": "module",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "dev": "nodemon --watch --exec node src/server.js",
    "dev1": "nodemon --watch --exec node src/serverDeleteRequest.js",
    "test": "echo \"Error: no test specified\" && exit 1"
  },
}
```

### → RE-RUN SERVER...

(As long as change something in the `serverDeleteRequest.js`. We need to re-run the server.)

→ **Ctrl + C** → Type → **J** → Stop the Server

```
tle":"Hello World","content":null,"published":true,"authorId":1}
Batchvorgang abbrechen (J/N)? j
PS C:\Users\Family\git\create-crud-api-with-express-for-prisma-mysql-app> |
```



RUN SERVER... | npm run **dev1**

C:\Users\Family\git\create-crud-api-with-express-for-prisma-mysql-app> **npm run dev1**

```
{
  "name": "hello-prisma",
  "type": "module",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "dev": "nodemon --watch --exec node src/server.js",
    "dev1": "nodemon --watch --exec node src/serverDeleteRequest.js",
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": ""
}
```

```
deletedUser: {"id":21,"email":"alice1726693020@prisma.io","name":"Alice"}
Batchvorgang abbrechen (Y/N)? j
PS C:\Users\Family\git\create-crud-api-with-express-for-prisma-mysql-app> npm run dev1

> hello-prisma@1.0.0 dev1
> nodemon --watch --exec node src/serverDeleteRequest.js

[nodemon] 3.1.4
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): --exec
[nodemon] watching extensions: js,mjs,cjs,json
[nodemon] starting `node node src/serverDeleteRequest.js`
Server is running at port 3000
/api/user/20 Thu Sep 19 2024 13:57:45 GMT+0200 (Mittleuropäische Sommerzeit)
req.params.id: 20
PROFILES records found by userId: [{"id":16,"bio":"I like turtles","userId":20}]
```



```
> hello-prisma@1.0.0 dev1  
> nodemon --watch --exec node src/serverDeleteRequest.js
```

```
[nodemon] 3.1.4  
[nodemon] to restart at any time, enter `rs`  
[nodemon] watching path(s): --exec  
[nodemon] watching extensions: js,mjs,cjs,json  
[nodemon] starting `node node src/serverDeleteRequest.js`  
Server is running at port 3000  
█
```

➔ Thnder Client | http Client



<https://www.prisma.io/docs/orm/overview/prisma-in-your-stack/rest>

## REST API server example

### DELETE

```
app.delete(`/post/:id`, async (req, res) => {
  const { id } = req.params
  const post = await prisma.post.delete({
    where: {
      id: Number(id),
    },
  })
  res.json(post)
})
```

MySQL database USER table without ID=91. Use for ERROR validation.

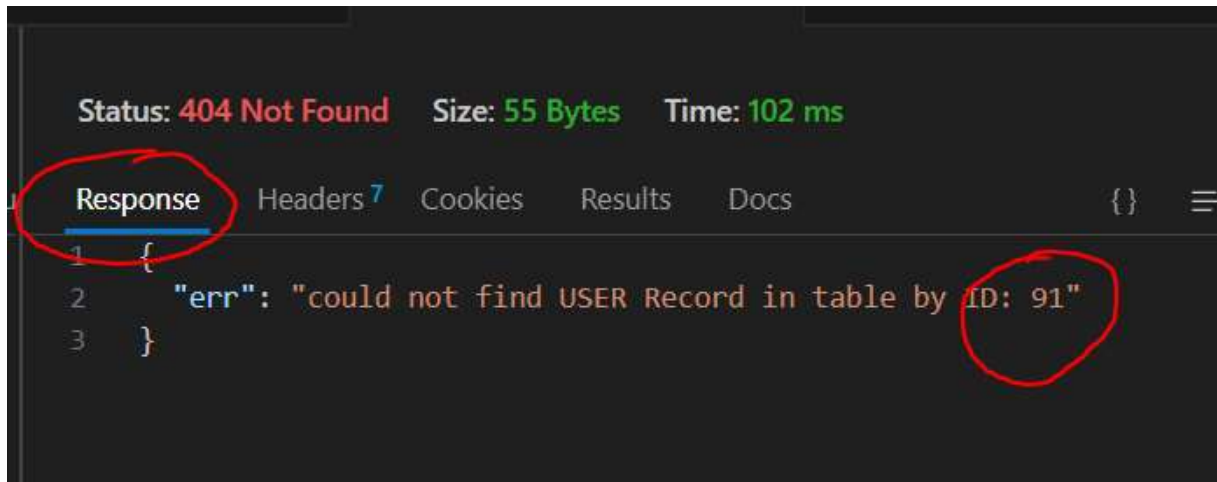
The screenshot shows the VS Code interface with a MySQL database connection named 'PrisamDBDemo' established. The 'Tables' list on the left includes '\_prisma\_migrations', 'post', 'profile', 'user', and 'Views'. The 'user' table is selected, and its data is displayed in the main editor. A red circle highlights the 'id' column, and a red arrow points from the 'user' table in the left sidebar to this circle. The table contains 9 records, with IDs ranging from 19 to 30. The 'email' column contains email addresses for each user.

id	email
19	alice1726690090@prisma.io
23	alice1726693739@prisma.io
24	alice1726694027@prisma.io
25	alice1726694052@prisma.io
26	alice1726694254@prisma.io
27	alice1726694292@prisma.io
28	alice1726694353@prisma.io
29	alice1726694427@prisma.io
30	alice1726694480@prisma.io

## ERROR Validation

DELETE → localhost:3000/api/user/91

:id = 91



GET Response deleted error info with Status 404 Not Found OK 😊

```
{
  "err": "could not find USER Record in table by ID: 91"
}
```

LOG →

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  DEVDB  SQL CONSOLE

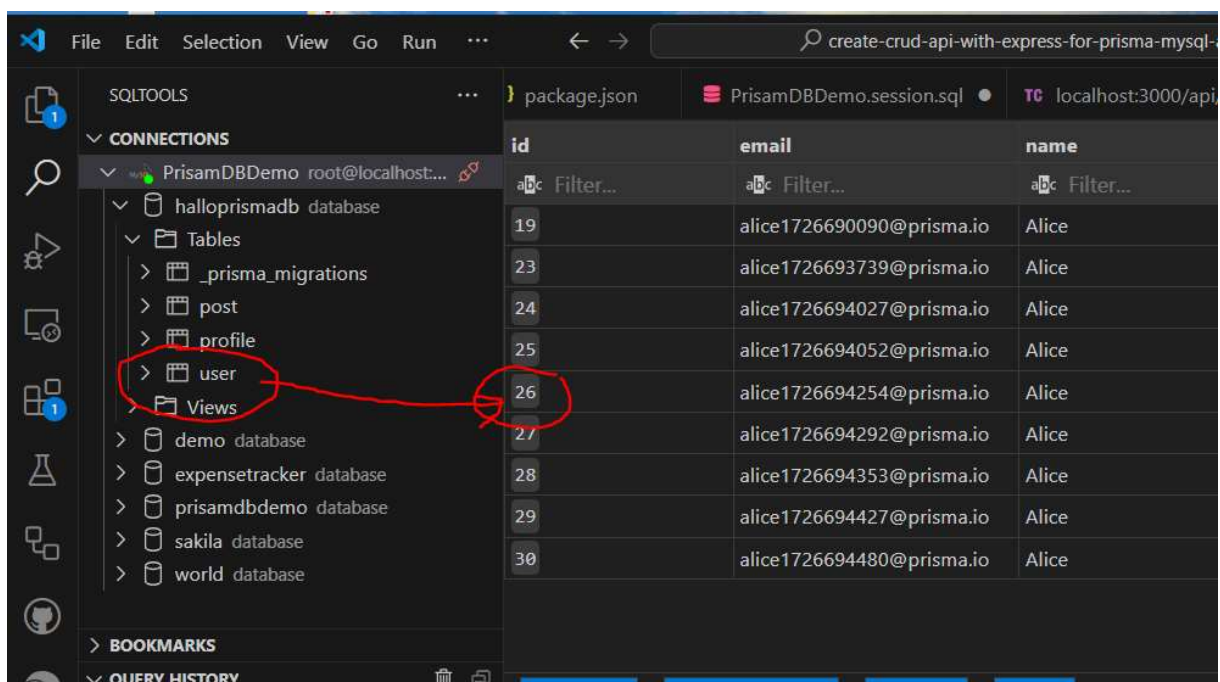
[nodemon] 3.1.4
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): --exec
[nodemon] watching extensions: js,mjs,cjs,json
[nodemon] starting `node node src/serverDeleteRequest.js`
Server is running at port 3000
/api/user/91 Fri Sep 20 2024 17:48:11 GMT+0200 (Mittleuropäische Sommerzeit)
req.params.id: 91
█
```

## DELETE USER with ID=26

DELETE → localhost:3000/api/user/26

:id = 26

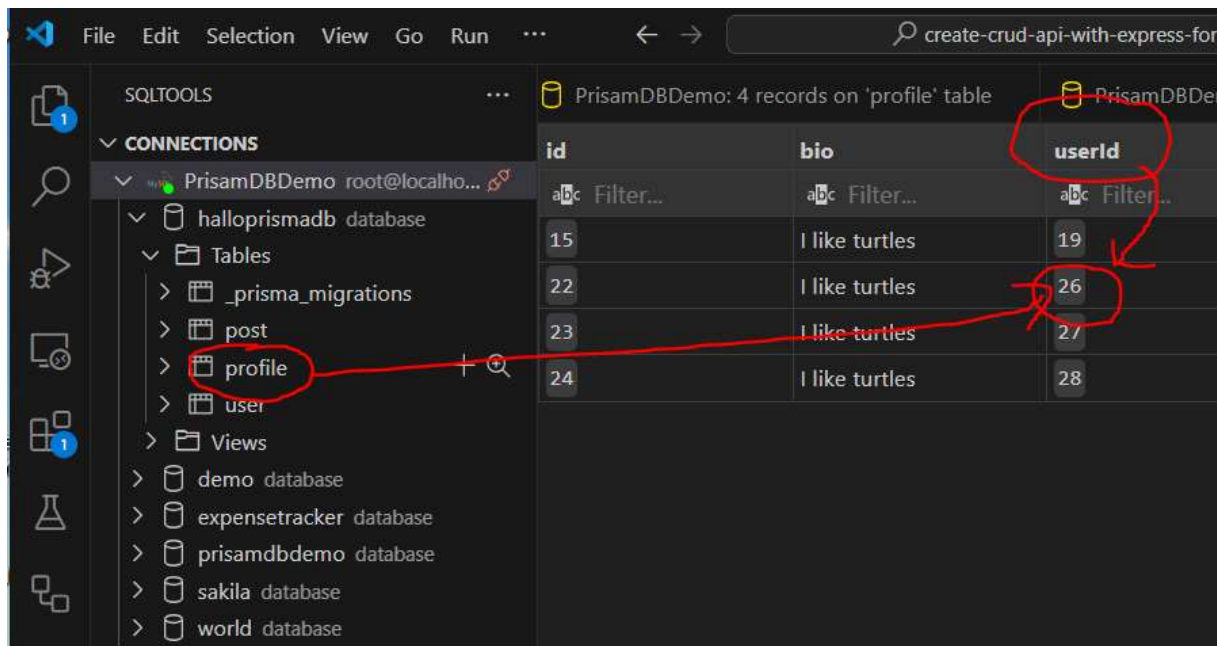
User contains id=26



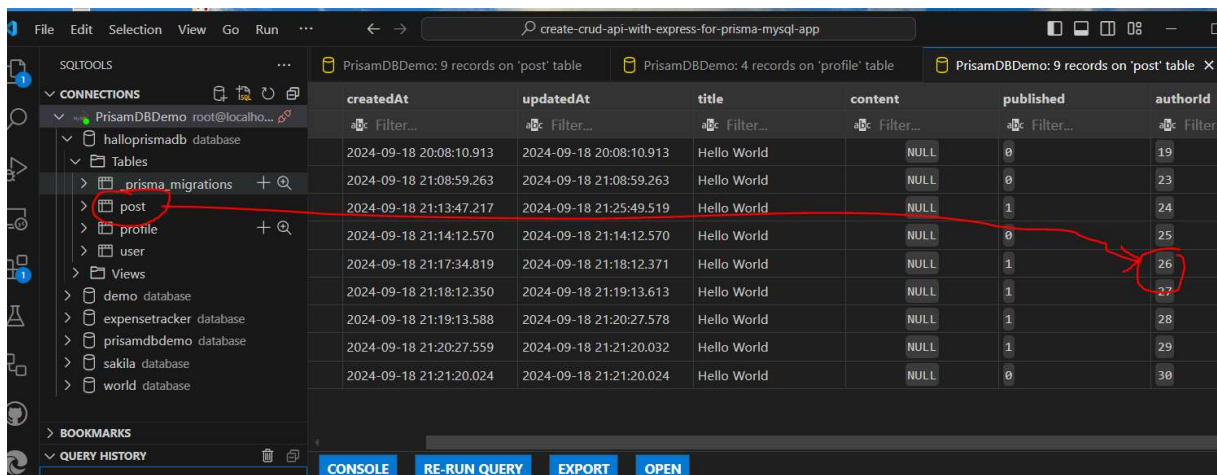
The screenshot shows the VS Code interface with the 'PrisamDBDemo' database connected. The 'user' table is selected in the 'CONNECTIONS' sidebar. The main table view displays a list of users with columns 'id', 'email', and 'name'. The user with 'id' 26 is circled in red.

id	email	name
19	alice1726690090@prisma.io	Alice
23	alice1726693739@prisma.io	Alice
24	alice1726694027@prisma.io	Alice
25	alice1726694052@prisma.io	Alice
26	alice1726694254@prisma.io	Alice
27	alice1726694292@prisma.io	Alice
28	alice1726694353@prisma.io	Alice
29	alice1726694427@prisma.io	Alice
30	alice1726694480@prisma.io	Alice

Profile contains foreign key **userId=26**



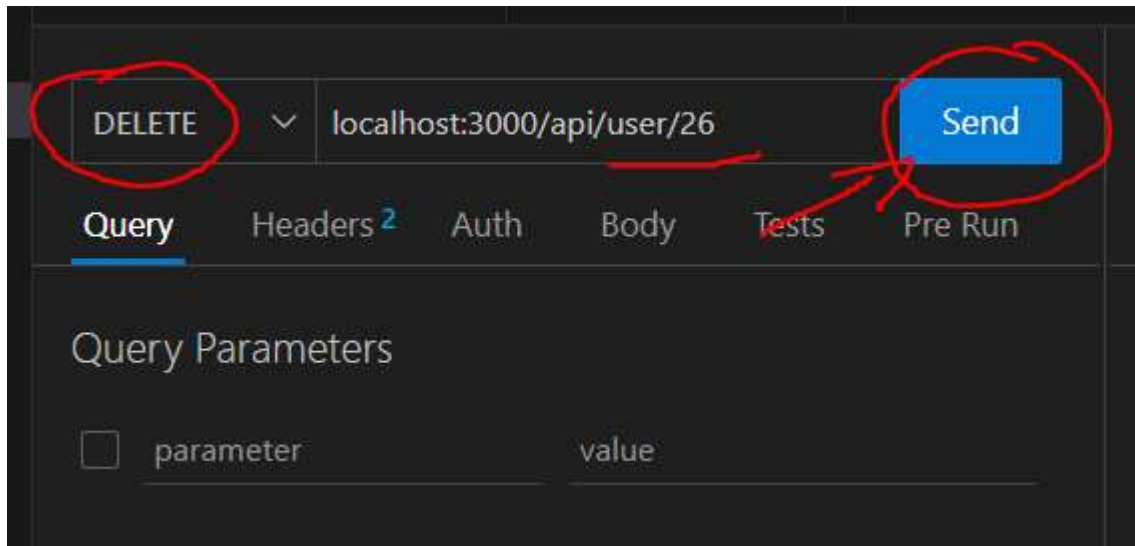
POST contains foreign key authorId **26**



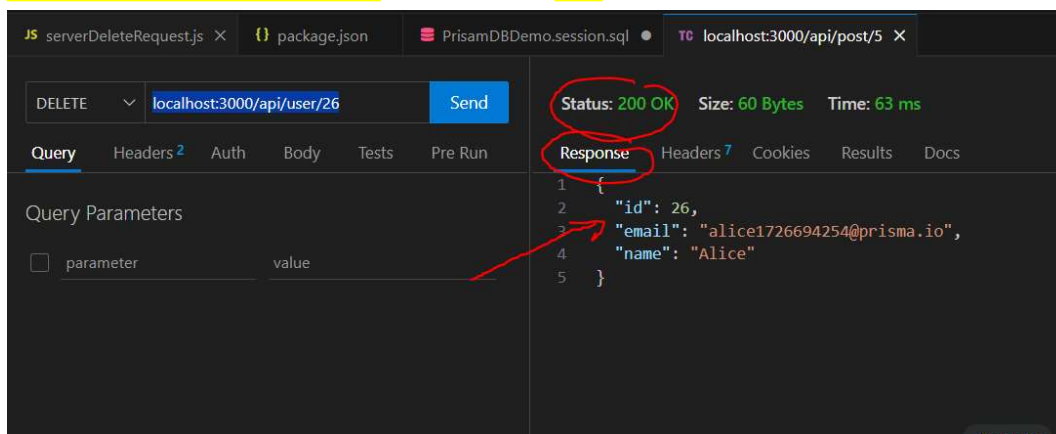
Click → Thunder Client



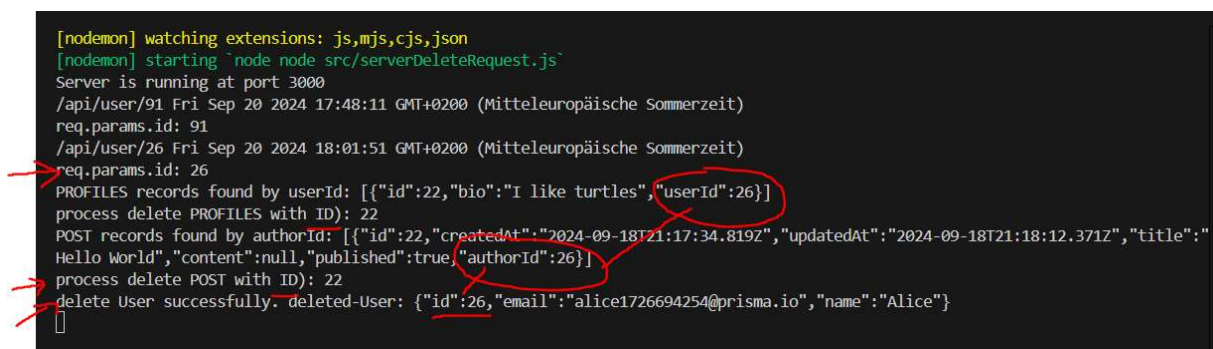
DELETE → localhost:3000/api/user/26



USER, POST, PROFILE were deleted with Status 200 OK and Response message with id=26



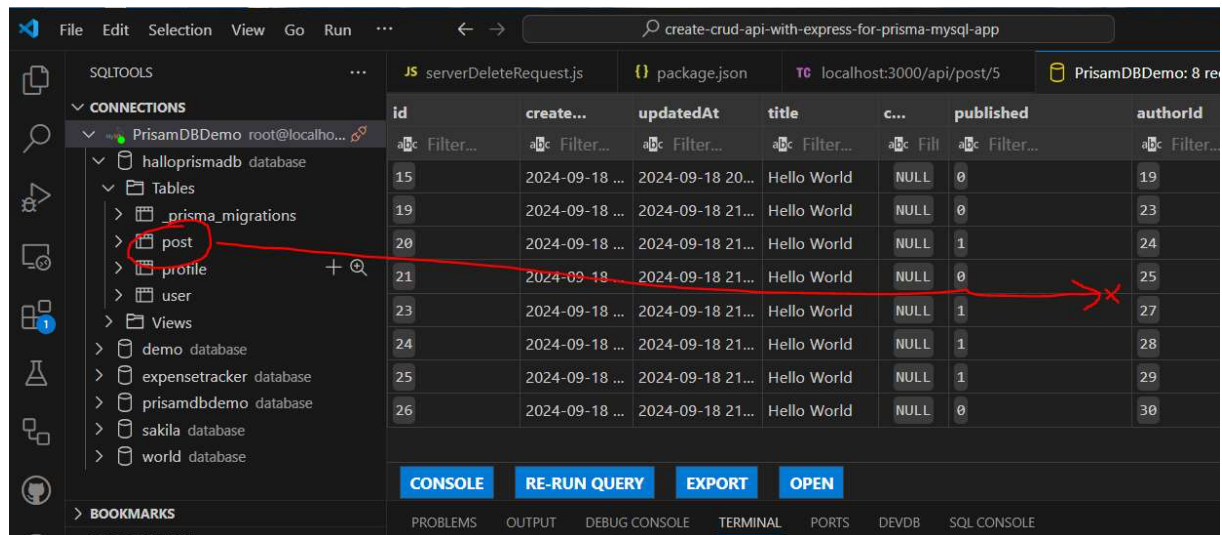
LOG





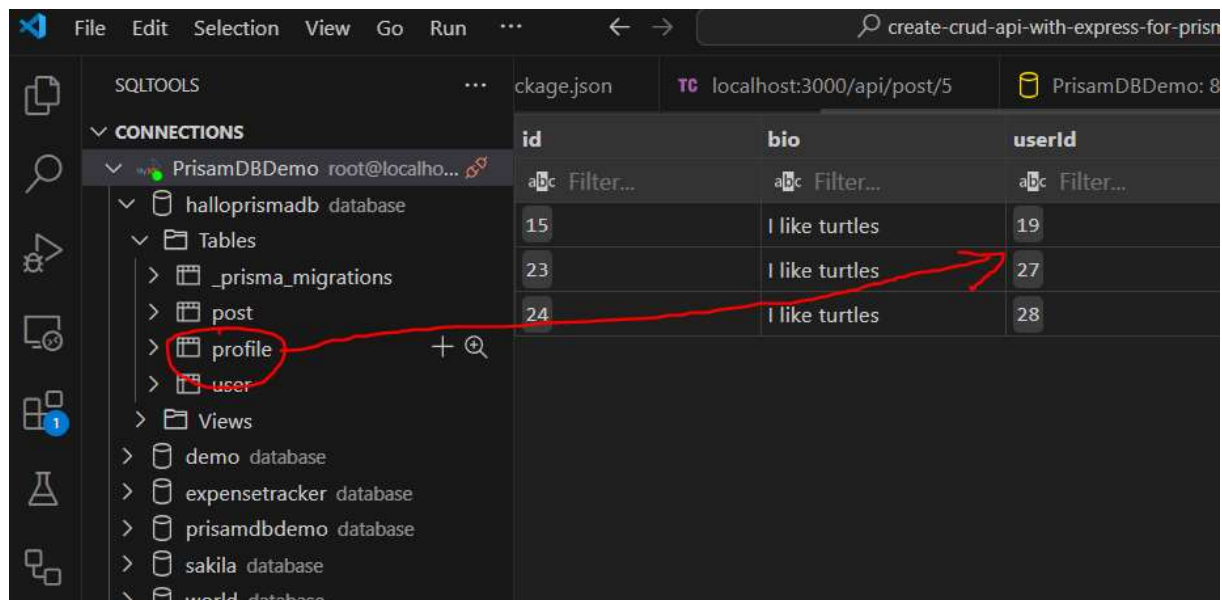
## My SQL Database

POST authorId=26 was deleted.



id	createAt	updatedAt	title	createdAt	published	authorId
15	2024-09-18 ...	2024-09-18 20...	Hello World	NULL	0	19
19	2024-09-18 ...	2024-09-18 21...	Hello World	NULL	0	23
20	2024-09-18 ...	2024-09-18 21...	Hello World	NULL	1	24
21	2024-09-18 ...	2024-09-18 21...	Hello World	NULL	0	25
23	2024-09-18 ...	2024-09-18 21...	Hello World	NULL	1	27
24	2024-09-18 ...	2024-09-18 21...	Hello World	NULL	1	28
25	2024-09-18 ...	2024-09-18 21...	Hello World	NULL	1	29
26	2024-09-18 ...	2024-09-18 21...	Hello World	NULL	0	30

Profile contains userId=26 was deleted



id	bio	userId
15	I like turtles	19
23	I like turtles	27
24	I like turtles	28

User contains id=26 was deleted

The screenshot shows the SQLTools application interface. On the left, the 'CONNECTIONS' panel lists several databases, including 'PrisamDBDemo'. Under 'PrisamDBDemo', the 'Tables' section is expanded, showing 'user', 'post', and 'profile'. The 'user' table is selected, and its schema is displayed: 'id INT', 'email VARCHAR(191)', and 'name VARCHAR(191)'. A red circle highlights the 'user' table, and a red arrow points to a '+' icon next to it. On the right, a table of records is shown with columns 'id', 'email', and 'name'. The records are filtered to show only those with 'id' values between 19 and 30. The table contains 12 rows of data, all with the name 'Alice'.

id	email	name
19	alice1726690090@prisma.io	Alice
23	alice1726693739@prisma.io	Alice
24	alice1726694027@prisma.io	Alice
25	alice1726694052@prisma.io	Alice
27	alice1726694292@prisma.io	Alice
28	alice1726694353@prisma.io	Alice
29	alice1726694427@prisma.io	Alice
30	alice1726694480@prisma.io	Alice

USER, POST, PROFILE were deleted successfully 😊