

## Relations B/N Models in API Connect

Analogous concept of TABLE JOINS  
in API-Connect

**GeetaKrishna.Adhikari**

API Developer

Miracle Software Systems, Inc.

## AIM:

In reality, models are often connected or related. When you build a real-world application with multiple models, you'll typically need to define *relations* between models.

You'll find this process analogous to **Joins** in Databases.

With connected models, LoopBack exposes as a set of APIs to interact with each of the model instances and query and filter the information based on the client's needs.

## PROCEDURE:

- Create a loopback application.
- Create two Datasources and also 2 Models for each of them.
- To create Relation between 2 existing models ;
- Run the Application and note down all the available Operations.  
(for future correspondence)

Run the Command: **apic loopback:relation**

Follow the prompts and select the Model from which you want to create relation.

You'll be prompted to select the type of relation you want the Model to have.

- Belongs to
- Has One
- Has Many
- Has and Belongs to many

**Has Many** and **Has and Belongs to many** works the same, but **Has Many** needs a Mediator Model to relate 2 models.

Whereas **Has and Belongs to many** also needs a Mediator Model but generates the Mediator model based on the 2 models that are being related.

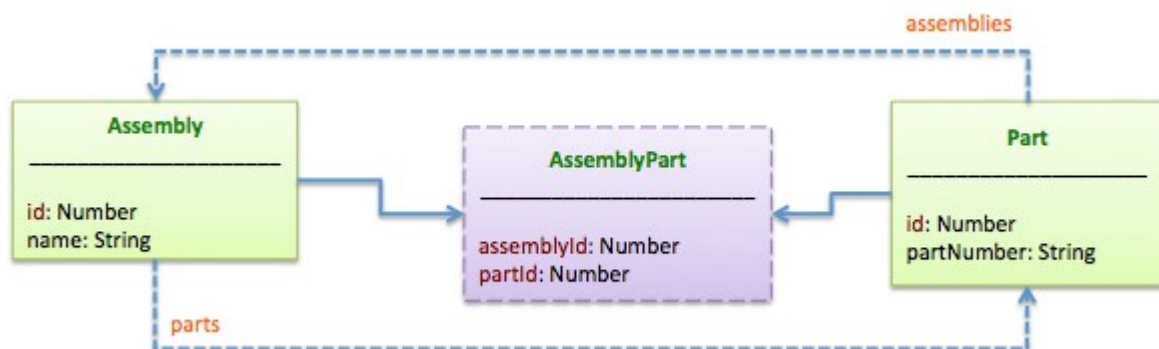
```

C:\Users\miracle\relation>apic loopback:relation
? Select the model to create the relationship from: (Use arrow keys)
? Select the model to create the relationship from: prime
? Relation type: has and belongs to many
? Choose a model to create a relationship with: sentinel
? Enter the property name for the relation: skills
? Optionally enter a custom foreign key:
Done running loopback generator

Updating swagger and product definitions
Created C:\Users\miracle\relation\definitions\relation.yaml swagger description

C:\Users\miracle\relation>_
  
```

- Follow the next prompts and provide a property through which a relation is to configured.
- You can also provide a Foreign Key .
- If you don't, the loopback will provide the Foreign Key of its own.
- The relation name appended with 'Id', for example, for relationname "customer" the default foreign key is "customerId".
- Now run the loopback application and Observe the Definitions.



- You'll find a new definition which is the appended names of related models.
- You can also see this model is combination of their Model Id's.

- You must understand that id of a particular model acts as a primary key for that model.

primesentinel

Name: primesentinel Type: object

Description:

Properties

| * Property Name                     | Description | Type   | Example |
|-------------------------------------|-------------|--------|---------|
| <input type="checkbox"/> id         |             | string |         |
| <input type="checkbox"/> primeId    |             | string |         |
| <input type="checkbox"/> sentinelId |             | string |         |

☐ Allow additional properties

Also you'll find the change in Operation capabilities such as below:

IBM API Connect relation 1.0.0

← All APIs Design <> Source Assemble

Test

Operation

Choose an operation to invoke:

- prime.prototype.\_\_findById\_\_skills
- prime.prototype.\_\_destroyById\_\_skills
- prime.prototype.\_\_updateById\_\_skills
- prime.prototype.\_\_link\_\_skills
- prime.prototype.\_\_unlink\_\_skills

invoke

- These operations are those which relate both Models.
- You can perform an operation using data of **prime** model and it affects the model **sentinel**.

## Working On Operations:

- First open the Database and see the available tables and data in it.
- For example my database Optimus has only 2 models(tables) with id's 1 & 4.
- Now take the operation **Link** and give the foreignkey (here **sentinelId**), corresponding data and **ModelId(of Model1 i.e., prime)**
- Or follow the image below

The screenshot shows the API management interface with the following components:

- Navigation Bar:** Products, APIs, Models, Data Sources.
- Left Panel (API List):**
  - relation 1.0.0
    - GET /primes/{id}/skills/{fk}
    - DELETE /primes/{id}/skills/{fk}
    - PUT /primes/{id}/skills/{fk}
    - PUT /primes/{id}/skills/rel/{fk}** (highlighted)
    - DELETE /primes/{id}/skills/rel/{fk}
    - HEAD /primes/{id}/skills/rel/{fk}
    - GET /primes/{id}/skills
    - POST /primes/{id}/skills
    - DELETE /primes/{id}/skills
    - GET /primes/{id}/skills/count
    - POST /primes
    - PUT /primes
    - GET /primes
- Right Panel (API Details):**
  - Foreign key for skills:** fk, sentinelId
  - data:**

```
{
  "id": "3",
  "sentinelId": "4",
  "primeId": "1"
}
```
  - Show schema | Generate**
  - PersistedModel id:** id, 1

## OUTPUT

The screenshot shows the API management interface with the following components:

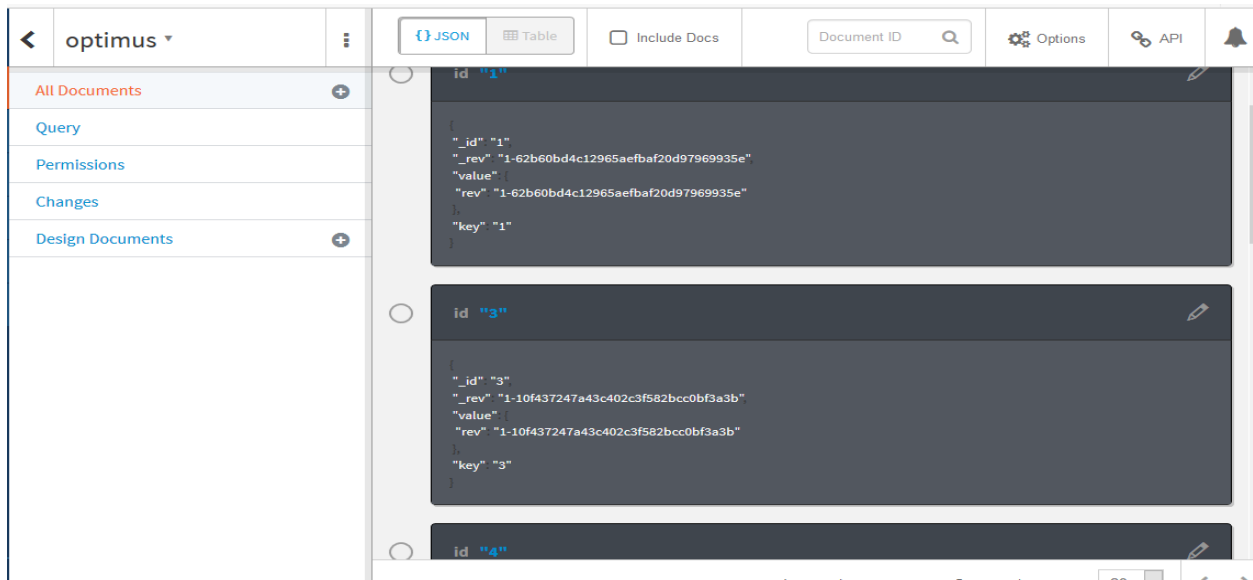
- Navigation Bar:** Products, APIs, Models, Data Sources.
- Left Panel (API List):**
  - lation 1.0.0
    - GET /primes/{id}/skills/{fk}
    - DELETE /primes/{id}/skills/{fk}
    - PUT /primes/{id}/skills/{fk}
    - PUT /primes/{id}/skills/rel/{fk}** (highlighted)
    - DELETE /primes/{id}/skills/rel/{fk}
    - HEAD /primes/{id}/skills/rel/{fk}
    - GET /primes/{id}/skills
    - POST /primes/{id}/skills
    - DELETE /primes/{id}/skills
    - GET /primes/{id}/skills/count
    - POST /primes
    - PUT /primes
    - GET /primes
    - GET /primes/{id}/exists
- Right Panel (API Details):**
  - Request:**

```
PUT https://localhost:4002/api/primes/1/skills/rel/sentinelId
APIM-Debug: true
Content-Type: application/json
Accept: application/json
X-IBM-Client-Id: default
X-IBM-Client-Secret: SECRET
```
  - Response:**

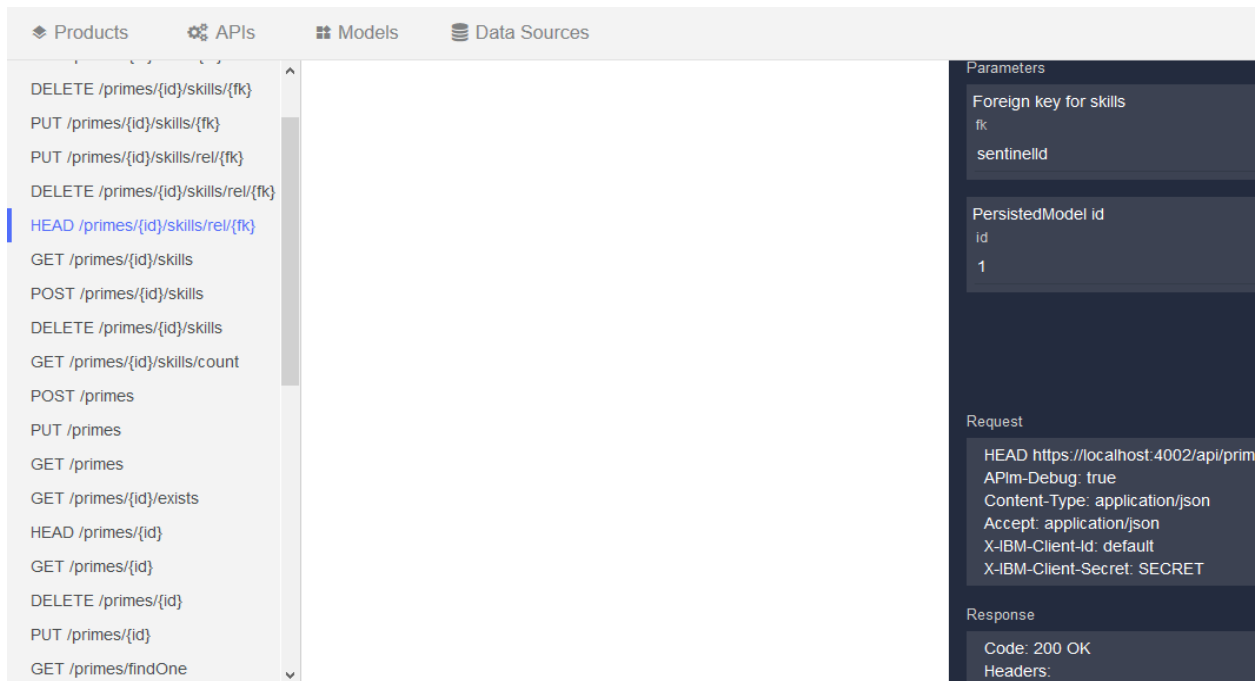
```
Code: 200 OK
Headers:
content-type: application/json; charset=utf-8
x-ratelimit-limit: 100
x-ratelimit-remaining: 99
x-ratelimit-reset: 308369

{
  "id": "2",
  "primeId": "1",
  "sentinelId": "sentinelId"
}
```

- To check if there exists a relation you can check your database for the additional table.



- Or you can use the Operation **EXISTS** (or follow as in image below)



## UNLINK Operation:

The screenshot shows the Miracle Software Systems API interface. The top navigation bar includes 'Products', 'APIs', 'Models', and 'Data Sources'. The left sidebar lists various API endpoints under 'relation 1.0.0'. The 'DELETE /primes/{id}/skills/rel/{fk}' endpoint is selected. The right panel displays the 'Content-Type' as 'application/json', the 'Accept' as 'application/json', and the 'Parameters' section showing 'Foreign key for skills' with 'fk' and 'sentinelId'. The 'PersistedModel id' is also shown as 'id' with the value '1'.

- If you don't want the relation to exist you can **DELETE** or **UNLINK**.
- You can check your Database and find the the related table with missing here **id:3**.
- **Now** you can understand that Link operation creates a new model(table) with properties of both Parent Models.
- It's like a joined table after Join a operation.
- Whereas UNLINK is Inverse of Link operation.
- Explore the other HTTP methods.

## PUT using Foriegn Key:

The screenshot shows the Miracle Software Systems API interface. The top navigation bar includes 'Products', 'APIs', 'Models', and 'Data Sources'. The left sidebar lists various API endpoints under 'relation 1.0.0'. The 'PUT /primes/{id}/skills/{fk}' endpoint is selected. The right panel displays the 'Content-Type' as 'application/json', the 'Parameters' section showing 'Foreign key for skills' with 'fk' and the value '4'. The 'data' section shows a JSON object with 'height': 80324208, 'name': 'maiores exercitationem sed consequuntur', 'character': 'ratione ad in', 'skill': 'occaecat dolore', and 'id': 4. The 'PersistedModel id' is also shown as 'id' with the value '1'.

## OUTPUT:

relation 1.0.0

- GET /primes/{id}/skills/{fk}
- DELETE /primes/{id}/skills/{fk}
- PUT /primes/{id}/skills/{fk}
- PUT /primes/{id}/skills/rel/{fk}
- DELETE /primes/{id}/skills/rel/{fk}
- HEAD /primes/{id}/skills/rel/{fk}
- GET /primes/{id}/skills
- POST /primes/{id}/skills
- DELETE /primes/{id}/skills
- GET /primes/{id}/skills/count
- POST /primes
- PUT /primes
- GET /primes
- GET /primes/{id}/exists
- HEAD /primes/{id}
- GET /primes/{id}
- DELETE /primes/{id}

Request

```

PUT https://localhost:4002/api/primes/1/skills/4
APIm-Debug: true
Content-Type: application/json
Accept: application/json
X-IBM-Client-Id: default
X-IBM-Client-Secret: SECRET

```

Response

```

Code: 200 OK
Headers:
content-type: application/json; charset=utf-8
x-ratelimit-limit: 100
x-ratelimit-remaining: 97
x-ratelimit-reset: 2387371

{
  "character": "ratione ad in",
  "height": 80324208,
  "name": "maiores exercitationem sed consequuntur",
  "skill": "occaecat dolorum",
  "id": "4"
}

```

## GET using Foreign Key:

Products APIs Models Data Sources

relation 1.0.0

- GET /primes/{id}/skills/{fk}
- DELETE /primes/{id}/skills/{fk}
- PUT /primes/{id}/skills/{fk}
- PUT /primes/{id}/skills/rel/{fk}
- DELETE /primes/{id}/skills/rel/{fk}
- HEAD /primes/{id}/skills/rel/{fk}
- GET /primes/{id}/skills
- POST /primes/{id}/skills
- DELETE /primes/{id}/skills
- GET /primes/{id}/skills/count
- POST /primes
- PUT /primes
- GET /primes
- GET /primes/{id}/exists
- HEAD /primes/{id}
- GET /primes/{id}

Identification

Client ID

default

Client secret

SECRET

Content-Type

application/json

Accept

application/json

Parameters

Foreign key for skills

fk

4

PersistedModel id

id

1



## OUTPUT:

The screenshot shows the API Explorer interface with the following components:

- Navigation Bar:** Products, APIs, Models, Data Sources.
- Left Panel (API List):** A list of API endpoints under 'relation 1.0.0'. The endpoint `GET /primes/{id}/skills/{fk}` is selected.
- Request Panel:**

```
GET https://localhost:4002/api/primes/1/skills/4
APIM-Debug: true
Content-Type: application/json
Accept: application/json
X-IBM-Client-Id: default
X-IBM-Client-Secret: SECRET
```
- Response Panel:**

```
Code: 200 OK
Headers:
content-type: application/json; charset=utf-8
x-ratelimit-limit: 100
x-ratelimit-remaining: 98
x-ratelimit-reset: 2331921

{
  "character": "ratione ad in",
  "height": 80324208,
  "name": "maiores exercitationem sed consequuntur",
  "skill": "occaecati dolore",
  "id": "4"
}
```

## The Other GET operation:

The screenshot shows the API Explorer interface with the following components:

- Navigation Bar:** Products, APIs, Models, Data Sources.
- Left Panel (API List):** A list of API endpoints under 'relation 1.0.0'. The endpoint `GET /primes/{id}/skills` is selected.
- Request Panel:**

```
GET https://localhost:4002/api/primes/1/skills
APIM-Debug: true
Content-Type: application/json
Accept: application/json
X-IBM-Client-Id: default
X-IBM-Client-Secret: SECRET
```
- Response Panel:**

```
Code: 200 OK
Headers:
content-type: application/json; charset=utf-8
x-ratelimit-limit: 100
x-ratelimit-remaining: 98
x-ratelimit-reset: 2303259

[
  {
    "character": "ratione ad in",
    "height": 80324208,
    "name": "maiores exercitationem sed consequuntur",
    "skill": "occaecati dolore",
    "id": "4"
  }
]
```