HITACHI
Inspire the Next

GraphQL APIを迅速に構築できる OSS「Hasura」を触ってみた

2022/03/11

株式会社 日立製作所
加藤優佑

HITACHI
Inspire the Next

Contents

HITACHI
Inspire the Next

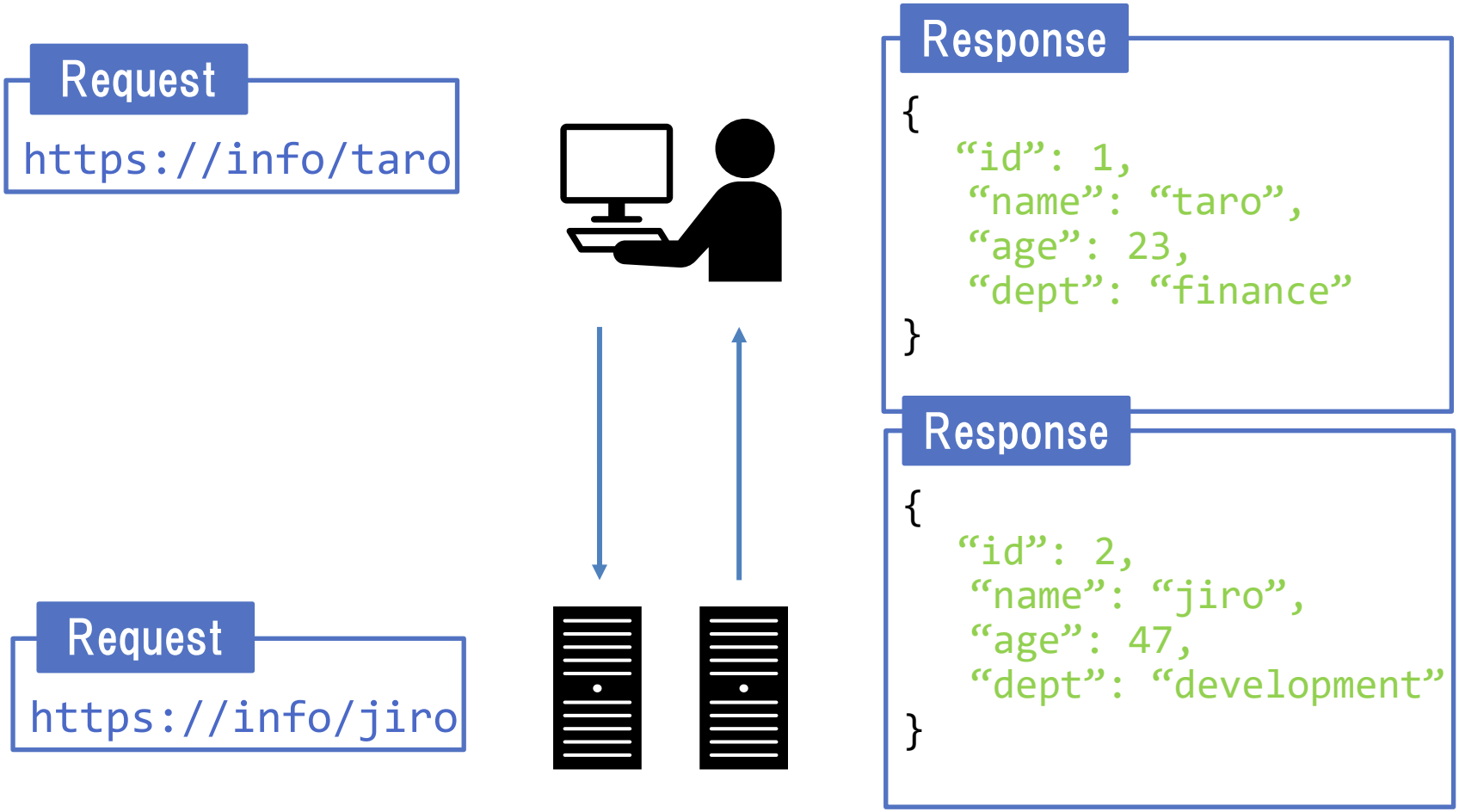
Contents

1-1. RESTful APIの問題点

HITACHI
Inspire the Next

従来のRESTful APIでは特定のURIにアクセスすれば、そこに紐づいた情報を取得できるが以下の問題点があった

- レスポンスされるデータは固定のため、不要なデータまで返ってくる
- 必要なデータが別のURIにある場合、何度もAPIを叩く必要がある



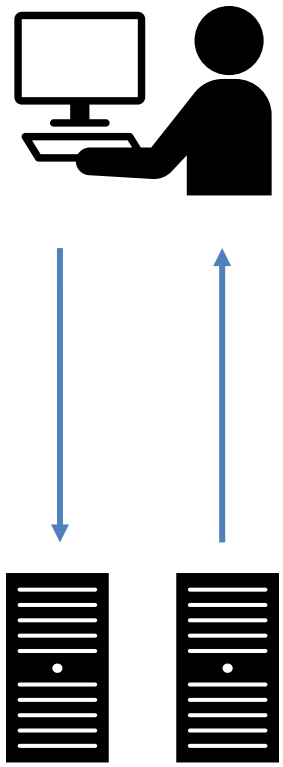
1-2. GraphQLとは

GraphQLはAPI向けのクエリ言語

- 取得したいデータをクエリに書くと、そのデータがレスポンスされる
- 1回のリクエストで必要なデータを取得可能
- データモデルの設計も行うため、RESTful APIに比べて開発コストが高い

Request

```
query {
  User {
    id
    name
    age
  }
}
```



Response

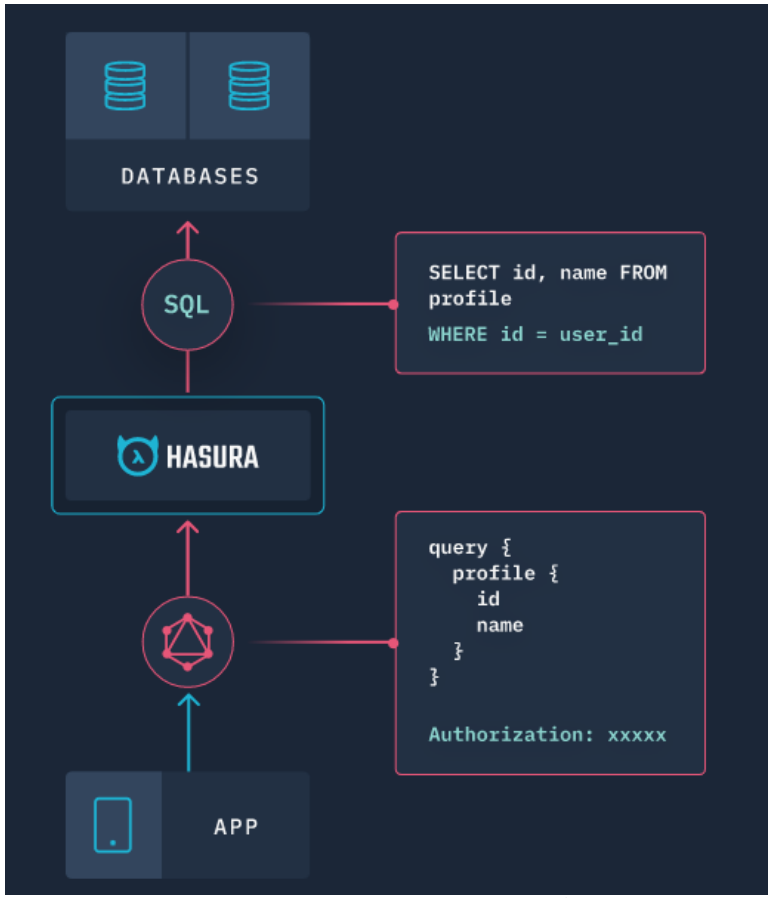
```
{
  "data": {
    "User": [
      {
        "id": 1,
        "name": "taro",
        "age": 23,
      },
      {
        "id": 2,
        "name": "jiro",
        "age": 47,
      }
    ]
  }
}
```

1-3. Hasuraとは

HasuraはGraphQLのバックエンドサービス

- クライアントとDBの間に入り、GraphQLとSQLを相互変換する
- DBへアクセスするGraphQL APIの開発時間を短縮可能

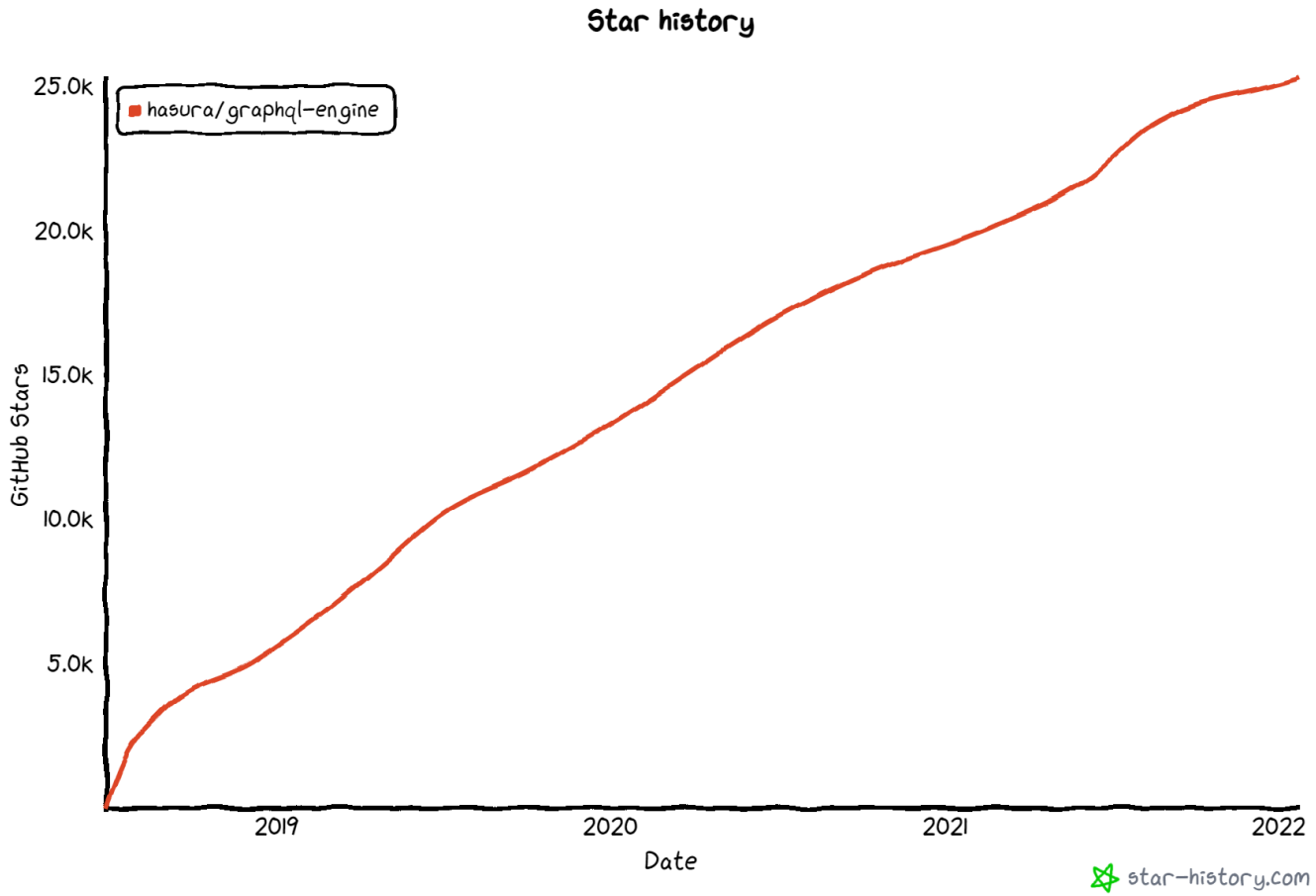
HASURA



(https://hasura.io)

1-3. Hasuraとは

- ライセンスはApache License 2.0であり、バージョン2.2.2までリリース
- Githubのスター数推移からみても注目度の高いOSSである



(https://star-history.com/#hasura/graphql-engine)

HITACHI
Inspire the Next

Contents

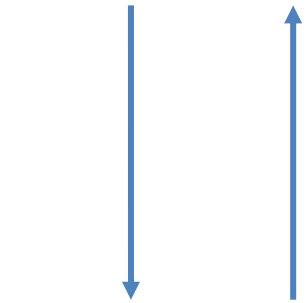
2-1. Database

HITACHI
Inspire the Next

Hasura上でDBに対してテーブルを作成するとGraphQLスキーマに
自動変換され、GraphQL APIで使用できるようになる

Request

```
query {
  User {
    user_id
    name
    age
  }
}
```



HASURA

Response

```
{
  "data": {
    "User": [
      {
        "id": 1,
        "name": "taro",
        "age": 23,
      },
      {
        "id": 2,
        "name": "jiro",
        "age": 47,
      }
    ]
  }
}
```

user_id	name	age	dept
1	taro	23	finance
2	jiro	47	development

2-1. Database

HITACHI
Inspire the Next

Request

```
Mutation insert_article {
  insert_User (
    objects: [
    {
      user_id: 3
      name: “saburo”
      age: 35
      dept: “legal”
    }
    ]
  ) {
      returning {
        user_id
        name
        age
      }
    }
  }
}
```

HASURA

Response

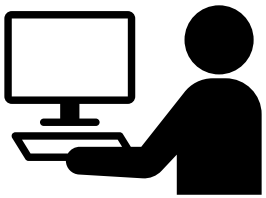
```
{
  “data”: {
    “insert_User”: {
      “returning”: [
        {
        “id”: 3,
        “name”: “saburo”,
        “age”: 35,
        }
      ]
    }
  }
}
```

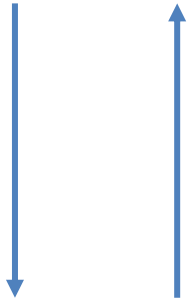
user_id	name	age	dept
1	taro	23	finance
2	jiro	47	development
3	saburo	35	legal

2-1. Database

HITACHI
Inspire the Next

Hasura上でテーブル間のRelationを設定するとGraphQLスキーマで
テーブル同士を結合することが可能





device_id	name	rental_date	id
1	ipad	2022/11/23	3
2	iphone	2022/02/03	1

user_id	name	age	dept
1	taro	23	finance
2	jiro	47	development
3	saburo	35	legal

2-1. Database

HITACHI
Inspire the Next

Hasura上でテーブル間のRelationを設定すると
テーブル同士を結合することが可能

Request

```
query {
  Device {
    device_id
    name
    User {
      name
      department
    }
  }
}
```

HASURA

device_id	name	rental_date	id
1	ipad	2022/11/23	3
2	iphone	2022/02/03	1

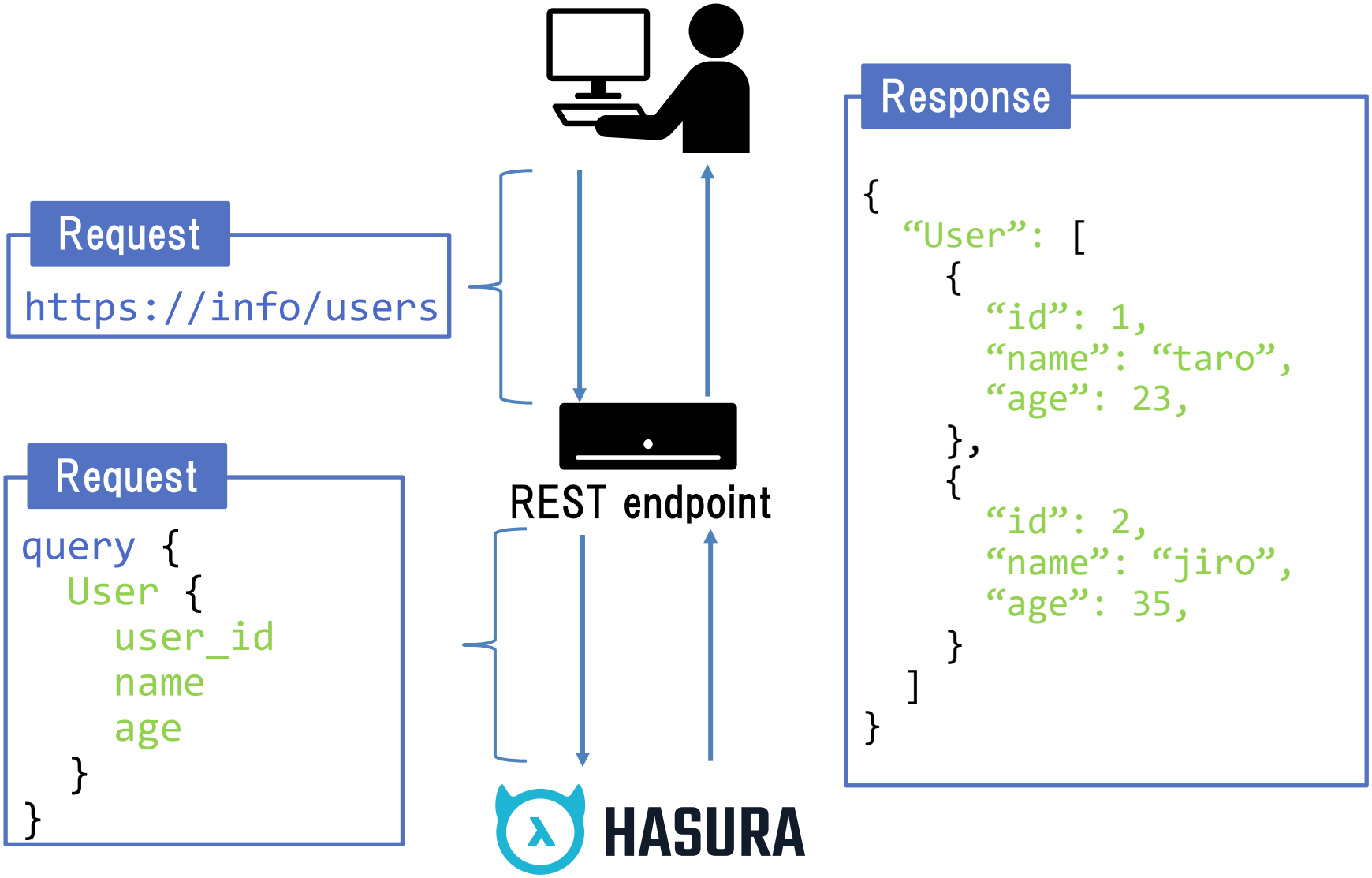
Response

```
{
  "data": {
    "Device": [
      {
        "device_id": 3,
        "name": "ipad",
        "User": {
          "name": "saburo",
          "dept": "legal",
        }
      },
      {
        "device_id": 1,
        "name": "iphone",
        "User": {
          "name": "taro",
          "dept": "finance",
        }
      }
    ]
  }
}
```

2-2. RESTified GraphQL Endpoints

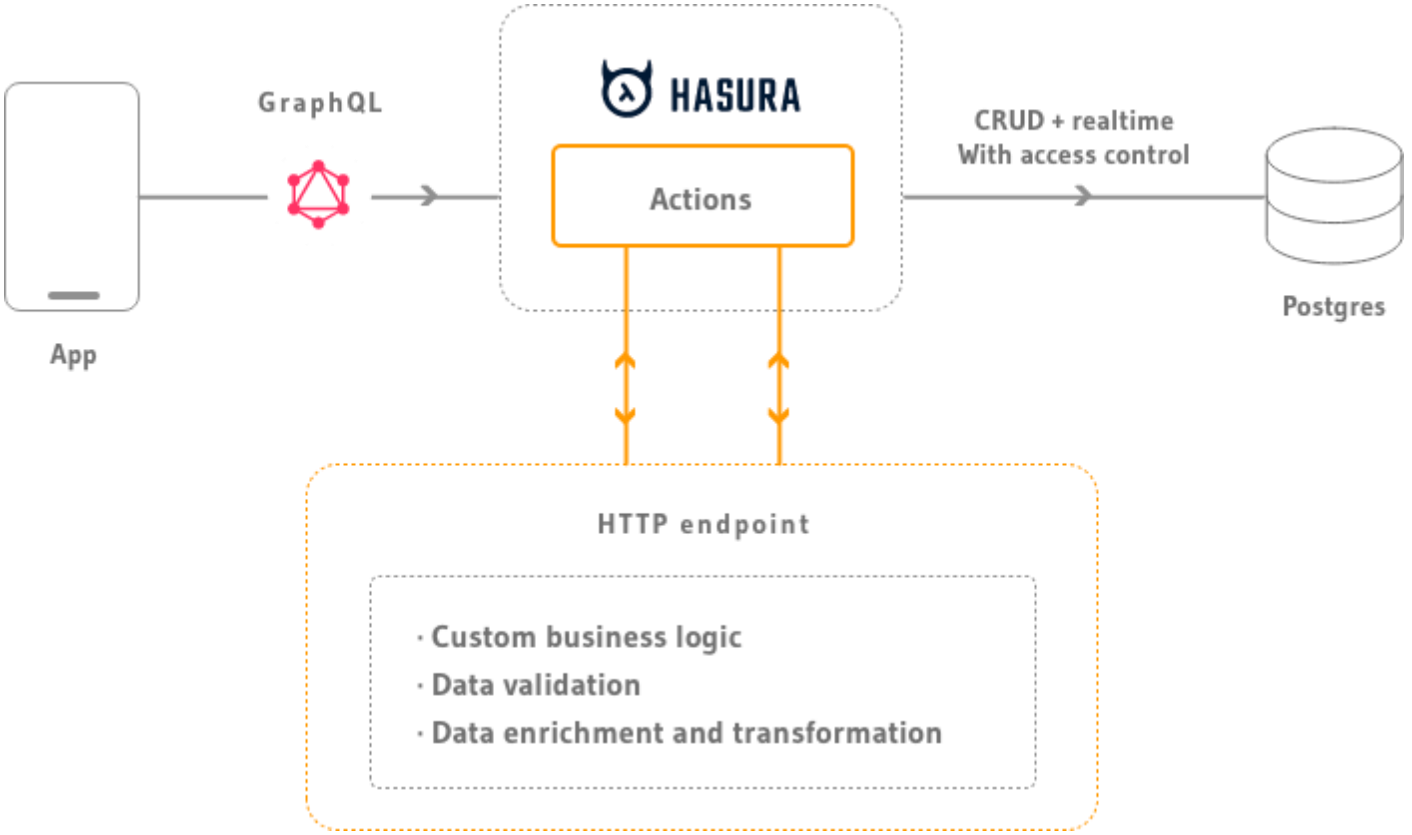
HITACHI
Inspire the Next

HasuraではGraphQLリクエストをRESTエンドポイントとして公開可能



2-3. Actions

- Hasuraでは外部APIと連携することでDB操作以外のユーザー独自のビジネスロジックを組み込むことが可能
- Actionは特にRESTエンドポイントを呼び出す際に使用



(https://hasura.io/docs/latest/graphql/core/actions/index.html)

2-3. Actions

HITACHI
Inspire the Next

- Hasuraでは外部APIと連携することでDB操作以外のユーザー独自のビジネスロジックを組み込むことが可能
- Actionは特にRESTエンドポイントを呼び出す際に使用

Action

```
type Query {
  Login (
    arg1: AccessInput!
  ): AccessOutput
}

type AccessInput {
  user: String!
  pass: String!
}

type AccessOutput {
  accesstoken: String!
}
```

Handler

```
const handler = async (req, res) => {
  const {user, pass} = req.bod,input;
  return res.json ({
    “accesstoken”: “4f3GFm”
  })
};
```

2-3. Actions

- Hasuraでは外部APIと連携することでDB操作以外のユーザー独自のビジネスロジックを組み込むことが可能
- Actionは特にRESTエンドポイントを呼び出す際に使用

Request

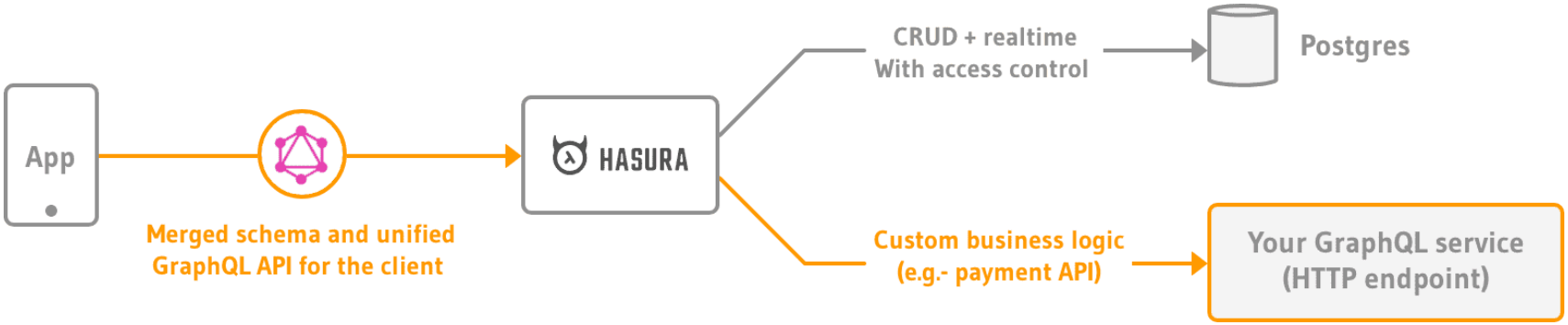
```
query {
  Login (
    user: “test”,
    pass: “pass”
  ) {
    accesstoken
  }
}
```

Response

```
{
  “data”: {
    “Login”: {
      “accesstoken”: “4f3GFm”
    }
  }
}
```

2-4. Remote Schemas

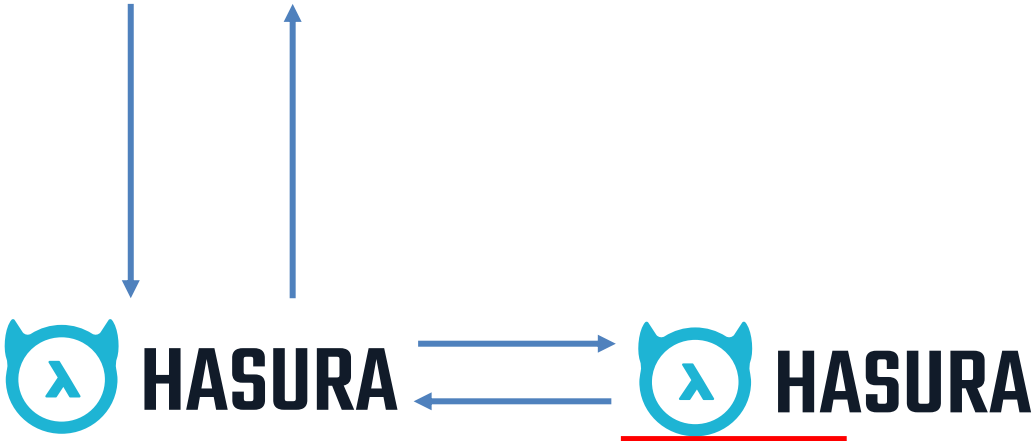
- Hasuraは他のGraphQLスキーマに接続し、GraphQL APIを統合することが可能
- Remote Schemasを利用することで統合元のスキーマと区別せずに外部のGraphQLスキーマにアクセスすることが可能



(https://hasura.io/docs/latest/graphql/core/remote-schemas/index.html)

2-4. Remote Schemas

- Hasuraは他のGraphQLスキーマに接続し、GraphQL APIを統合することが可能
- Remote Schemasを利用することで統合元のスキーマと区別せずに外部のGraphQLスキーマにアクセスすることが可能



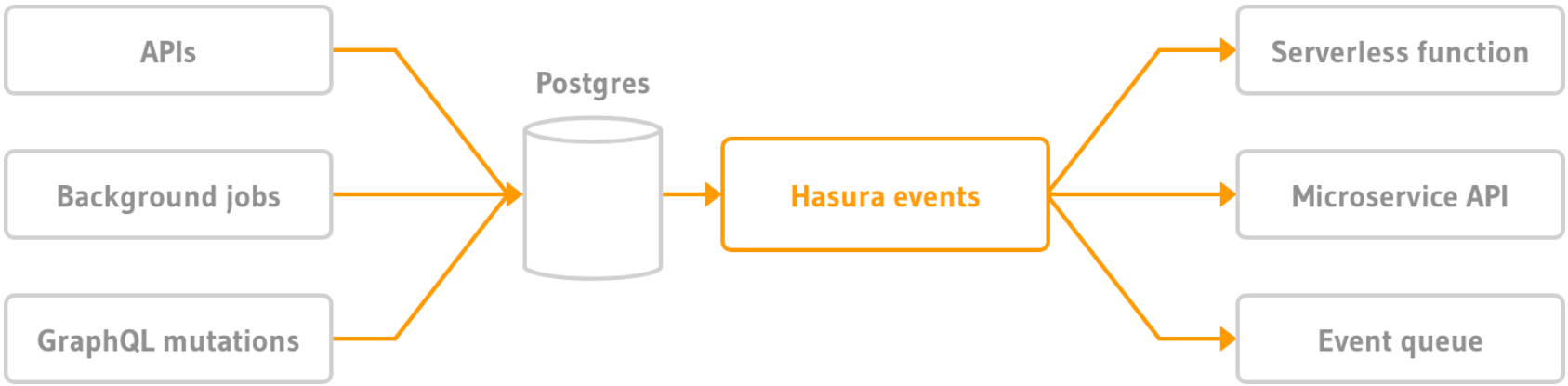
device_id	name	rental_date	id
1	ipad	2022/11/23	3
2	iphone	2022/02/03	1

user_id	name	age	dept
1	taro	23	finance
2	jiro	47	development
3	saburo	35	legal

2-5. Event Triggers

HITACHI
Inspire the Next

- DBで特定のイベント(INSERT、UPDATE、DELETE操作等)が発生した際にあらかじめ定義された外部APIを呼び出す機能
- 外部APIへの通知はat least once方式で実施

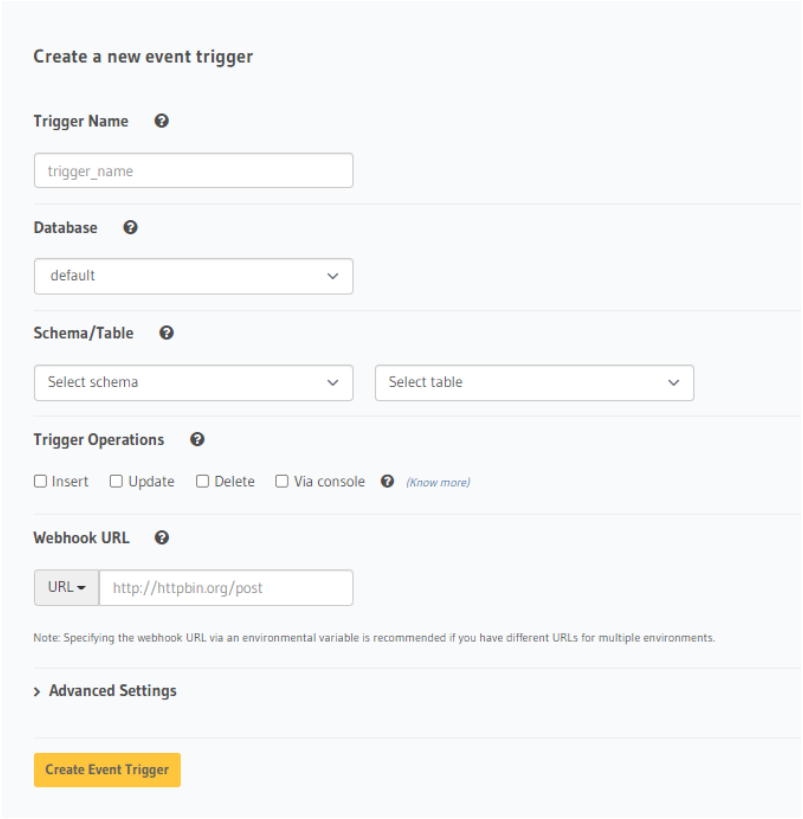


(https://hasura.io/docs/latest/graphql/core/event-triggers/index.html)

2-5. Event Triggers

HITACHI
Inspire the Next

- 用途としては会員データ登録後のユーザー通知といったポーリングするほど高頻度でないDB更新で処理が開始されるケースがある
- またEvent TriggersのJSON形式のPayloadには発生原因となったイベントや変更前後の値が含まれているので、ボイラープレートとしても利用可能

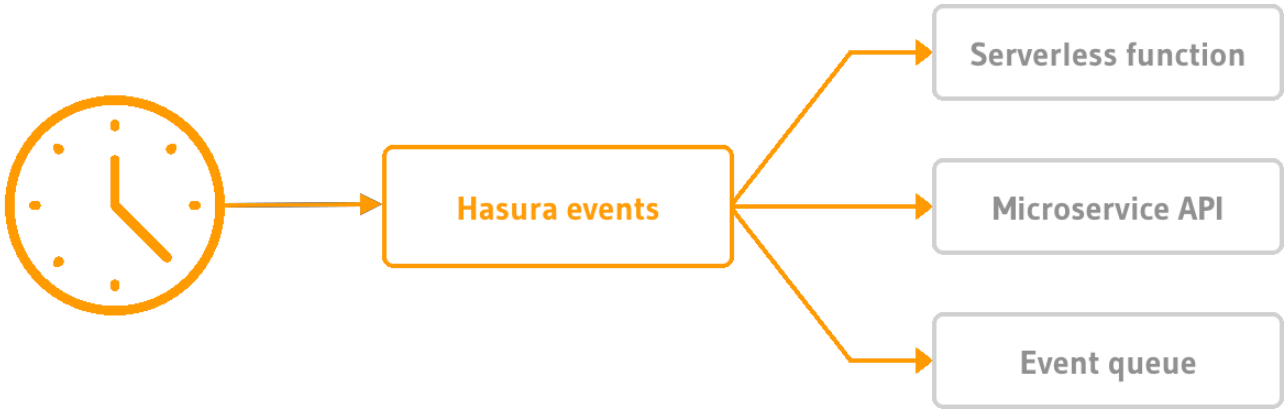



```
{
  "event": {
      "session_variables": <session-variables>,
      "op": "<op-name>",
      "data": {
          "old": <column-values>,
          "new": <column-values>
      }
  },
  "created_at": "<timestamp>",
  "id": "<uuid>",
  "trigger": {
      "name": "<name-of-trigger>"
  },
  "table":  {
      "schema": "<schema-name>",
      "name": "<table-name>"
  }
}
```

(https://hasura.io/docs/latest/graphql/core/event-triggers/payload.html)

2-6. Scheduled Triggers

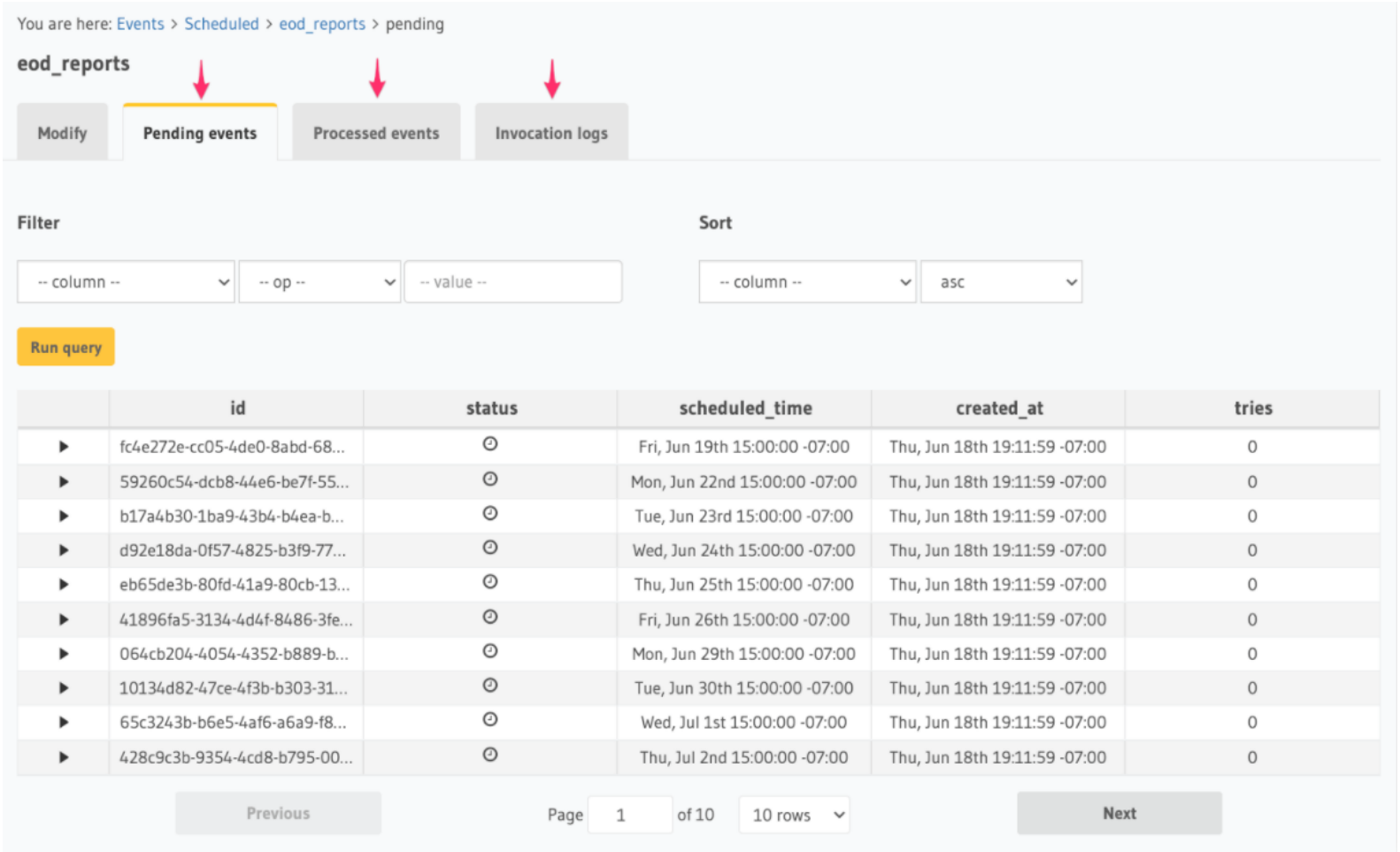
- ユーザーが定義したロジックを指定した時間に実行する機能
- トリガの設定方式には定期実行するcron triggersと設定した時刻に1度だけ実行するone-off scheduled eventsがある



(https://hasura.io/docs/latest/graphql/core/scheduled-triggers/index.html)

2-6. Scheduled Triggers

- どちらの形式でもHasura上で各イベントのステータス、ログが見れる
- リトライ回数やインターバル、リトライ期限も設定できるため、ジョブ管理サーバーとして用いることができる



(https://hasura.io/docs/latest/graphql/core/scheduled-triggers/create-cron-trigger.html)

2-7. Migrations＆Metadata

HITACHI
Inspire the Next

- HasuraではDBのスキーマやHasuraのメタデータをファイルとして出力でき、これを使用して類似環境の構築が可能
- 出力したファイルは編集することで環境のカスタマイズが可能
- 各時点でのスキーマやメタデータをバージョンで管理するので、環境をロールバックすることも可能

Hasura CLI

```
hasura migrate apply
--version 1234567890
--type down
--database-name db1

git checkout meta1
Hasura metadata apply
```

HASURA

2-8. Deploying

HITACHI
Inspire the Next

- Hasuraではデプロイする際のガイドや本番環境でセキュアなGraphQLエンジンを実現するためのガイドが提供されている
- 具体例としては以下が挙げられる

Set an admin secret
- コンソールやエンドポイントへの不正アクセスを防ぐために設定
- Hasuraでユーザー認証を行う際には必ず設定

Enable HTTPS
- Hasura自身はAPIをSSL/TLSで保護できないため、APIをセキュアにするにはリバースプロキシを設定する必要がある

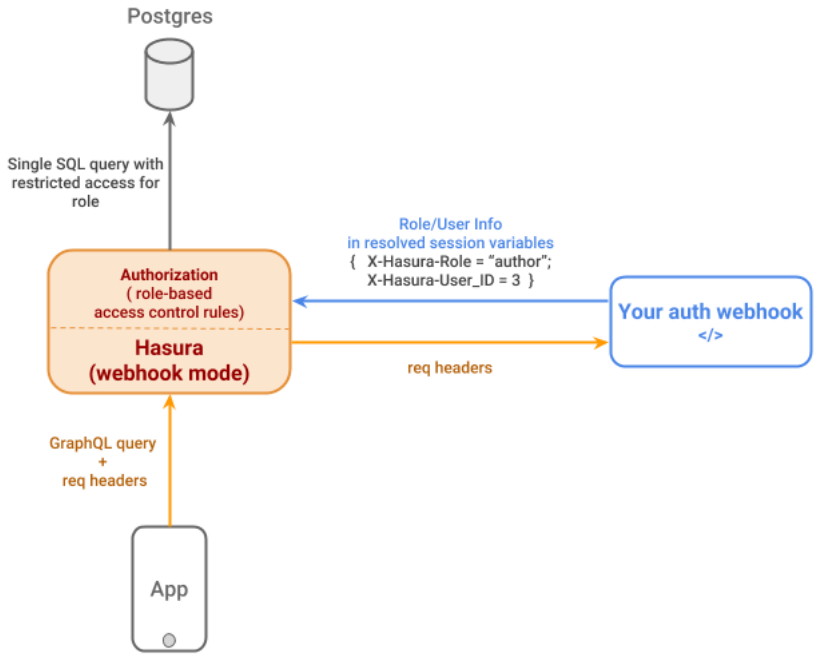
Configure logging
- Hasuraではquery-log(Hasuraのquery全般に関するlog)やhttp-log(HasuraにHTTPリクエストした際のlog)等のlogが12種類提供されており、その中で出力するlogを設定できる

HITACHI
Inspire the Next

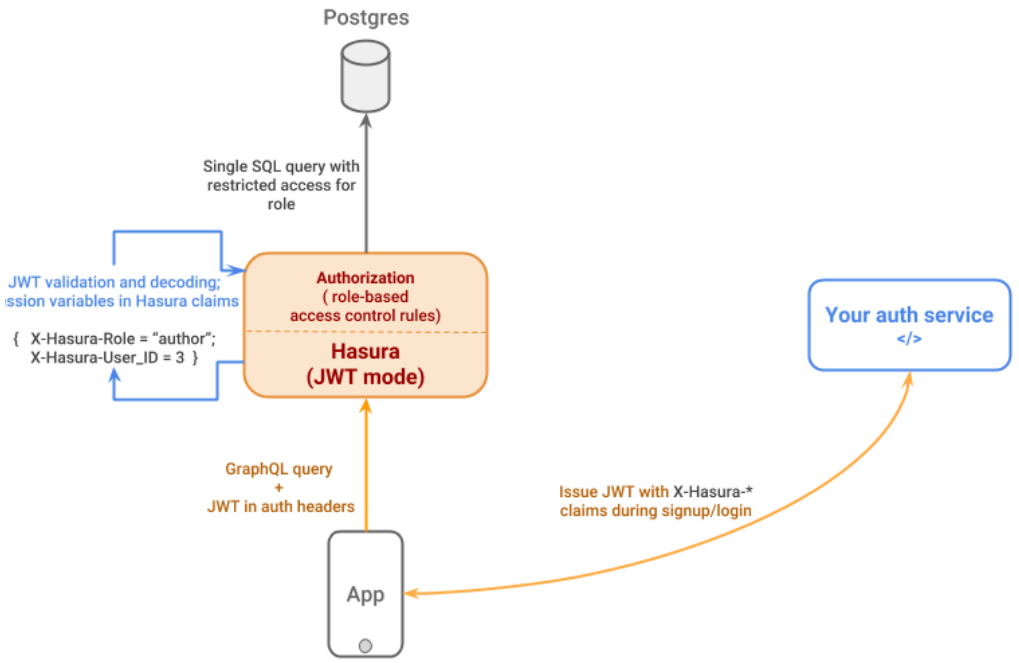
Contents

3-1. Authentication

- Hasuraではユーザー認証はHasura外で行う必要があり、認証方式にはwebhook認証とJWT(JSON Web Token)認証の2種類が存在
- いずれの方式もSession Variablesという変数(X-Hasura-Role等)をHasuraに渡すことでユーザーの識別を実施



webhook認証



JWT認証

(https://hasura.io/docs/latest/graphql/core/auth/authentication/index.html)

3-1. JWT認証

- JWT認証ではまずクライアントが認証サーバーからJWTトークンを取得し、そのJWTをヘッダに含めたHTTPリクエストをHasuraに送信
- JWT内のclaimに記述されたX-Hasura-Allowed-Rolesの中からユーザーに割り当てるRoleを選択し、後述のAuthorizationを実施

認証サーバー

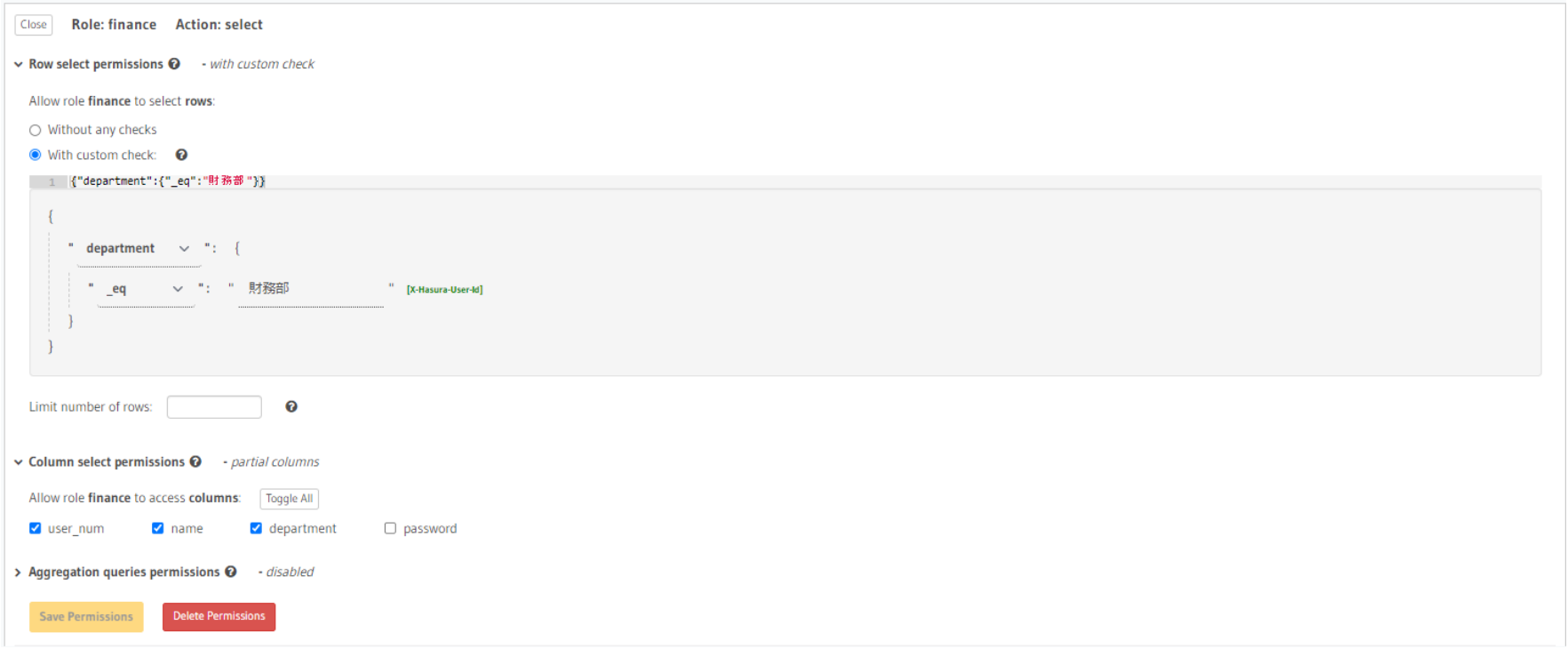
JWT

```
{
  “exp”: 1637347424
  “iat”: 1637347424
  “sub”: “123456789”
  “typ”: “Bearer”
  https://hasura.io/jwt/claims: {
    “x-hasura-allowed-roles”: [
      “finance”
    ]
    “x-hasura-user-id”: “123456789”
  }
}
```

3-2. Authorization

- Hasuraにおいてアクセス制御はRoleを用いて実施
- Role、Table、CRUD毎にPermissionと呼ばれるアクセスルールを設定し、DBの行・列単位で制御

Role	insert	select	update	delete	
admin	✔	✔	✔	✔	
finance	✖	▼	✖	✖	☐
manager	✖	✔	✖	✖	☐
user	✖	▼	✖	✖	☐
Enter new role	✖	✖	✖	✖	



HITACHI
Inspire the Next

Contents

4-1. 検証環境の構築

HITACHI
Inspire the Next

Hasuraに接続された以下のテーブルに
3つのRole”manager”、”user”、”finance”でアクセスすることを想定

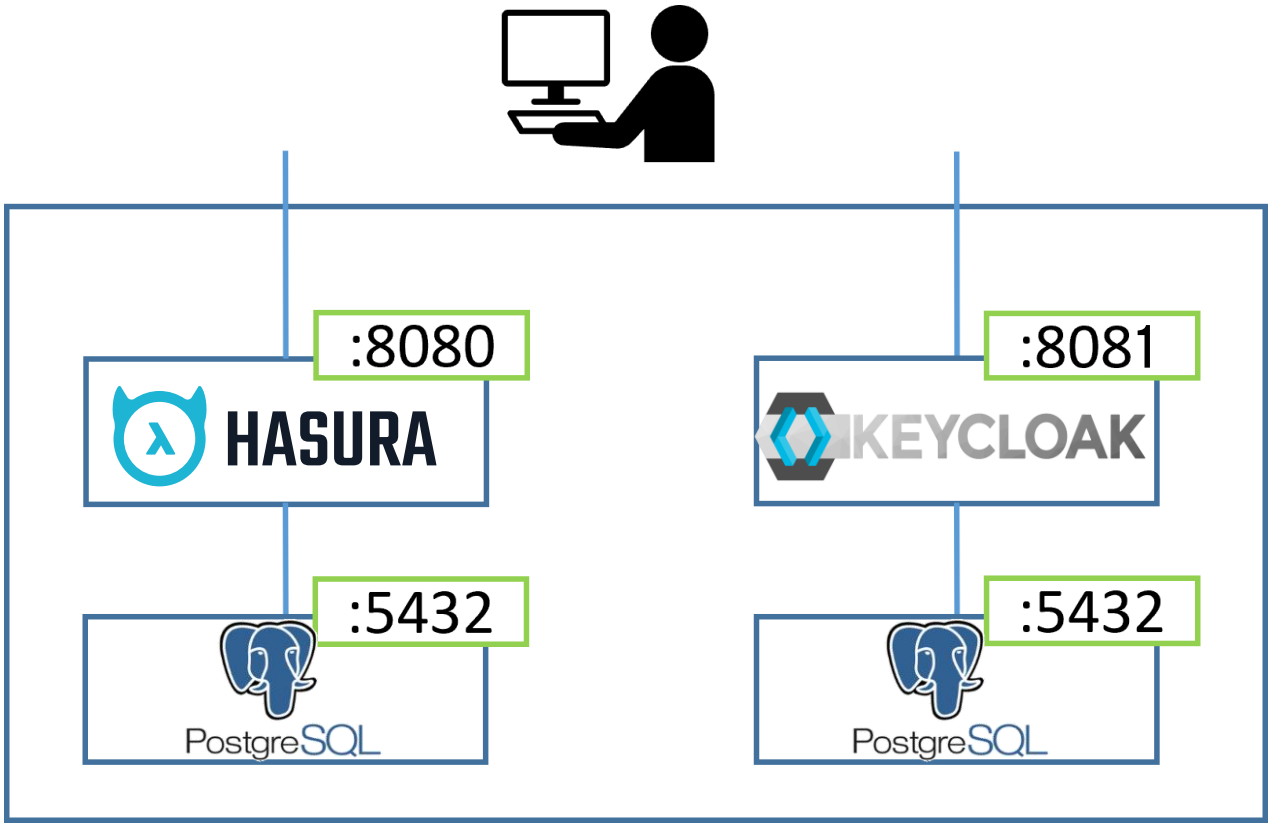
user_num	name	department	password
1	太郎	財務部	aa1
2	次郎	法務部	bki4
3	三郎	総務部	csu27
4	四郎	法務部	dte256
5	五郎	財務部	eno3125
6	六郎	財務部	fha46656

Role	説明
manager	全てのユーザー情報にアクセスできるRole
user	自らのユーザー番号と一致するデータにのみアクセスできるRole
finance	財務部に所属するユーザーのpassword以外の属性情報を取得できるRole

4-1. 検証環境の構築

HITACHI
Inspire the Next

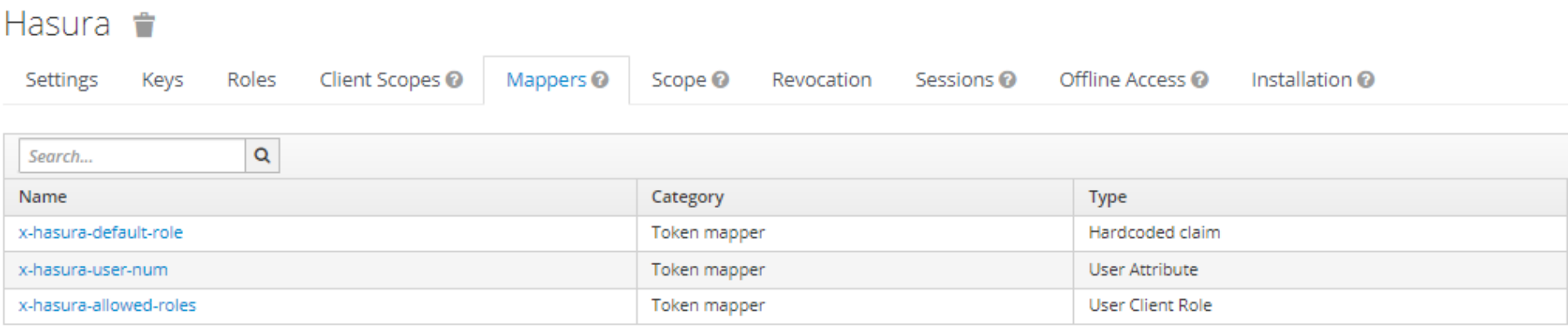
認証・認可サーバにはKeycloakを使用し、Keycloakから取得した各RoleのHasuraにおける挙動を確認する



4-2. Keycloakの設定

HITACHI
Inspire the Next

“Clients”として”hasura”を作成し、”hasura”内の“Mapper”でJWTのclaimに組み込むfieldを指定する



Mapper名	説明
x-hasura-allowed-roles(必須)	認証したユーザーに割り当てるRole一覧をセットするfield
x-hasura-default-role(必須)	Hasuraへのリクエストヘッダでroleを指定しない場合に割り当てるRoleを設定するfield
x-hasura-user-num	Roleとは関係ないですが、HasuraでSession Variablesとして使用するfield

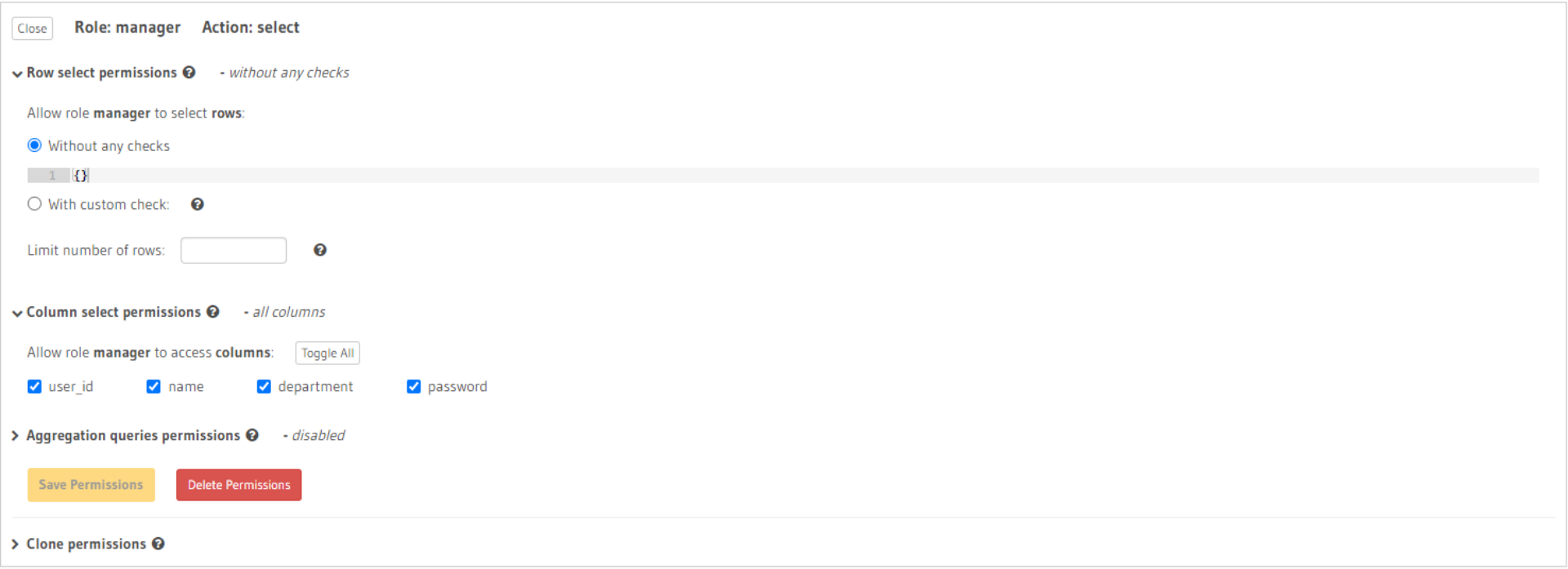
4-2. Keycloakの設定

次に”hasura”内の”Roles”でHasuraに渡すRoleを作成
(今回は”manager”、”user”、”finance”を作成)

4-2. Hasuraの設定

HITACHI
Inspire the Next

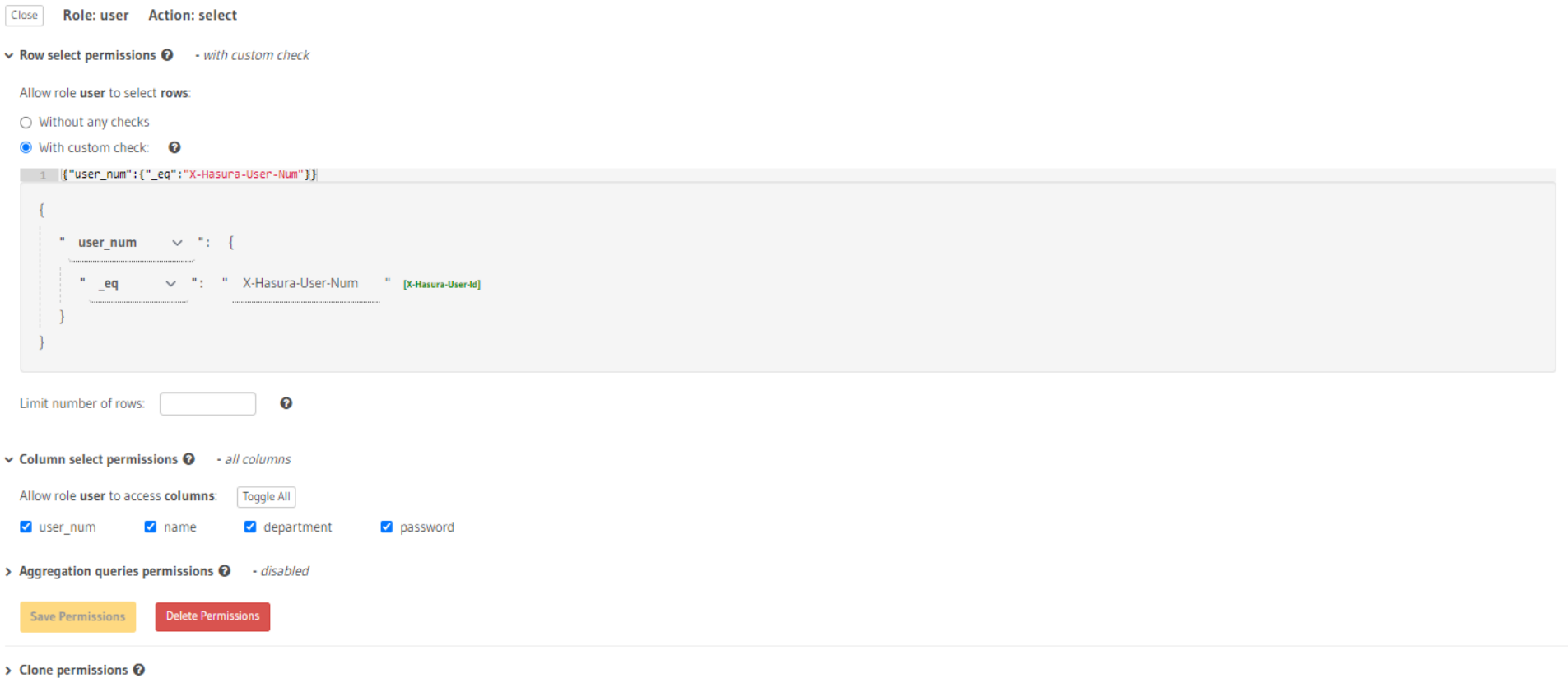
Role”manager”はDBの全てのユーザー情報にアクセスできるように Permissionには何も設定をしない



4-2. Hasuraの設定

HITACHI
Inspire the Next

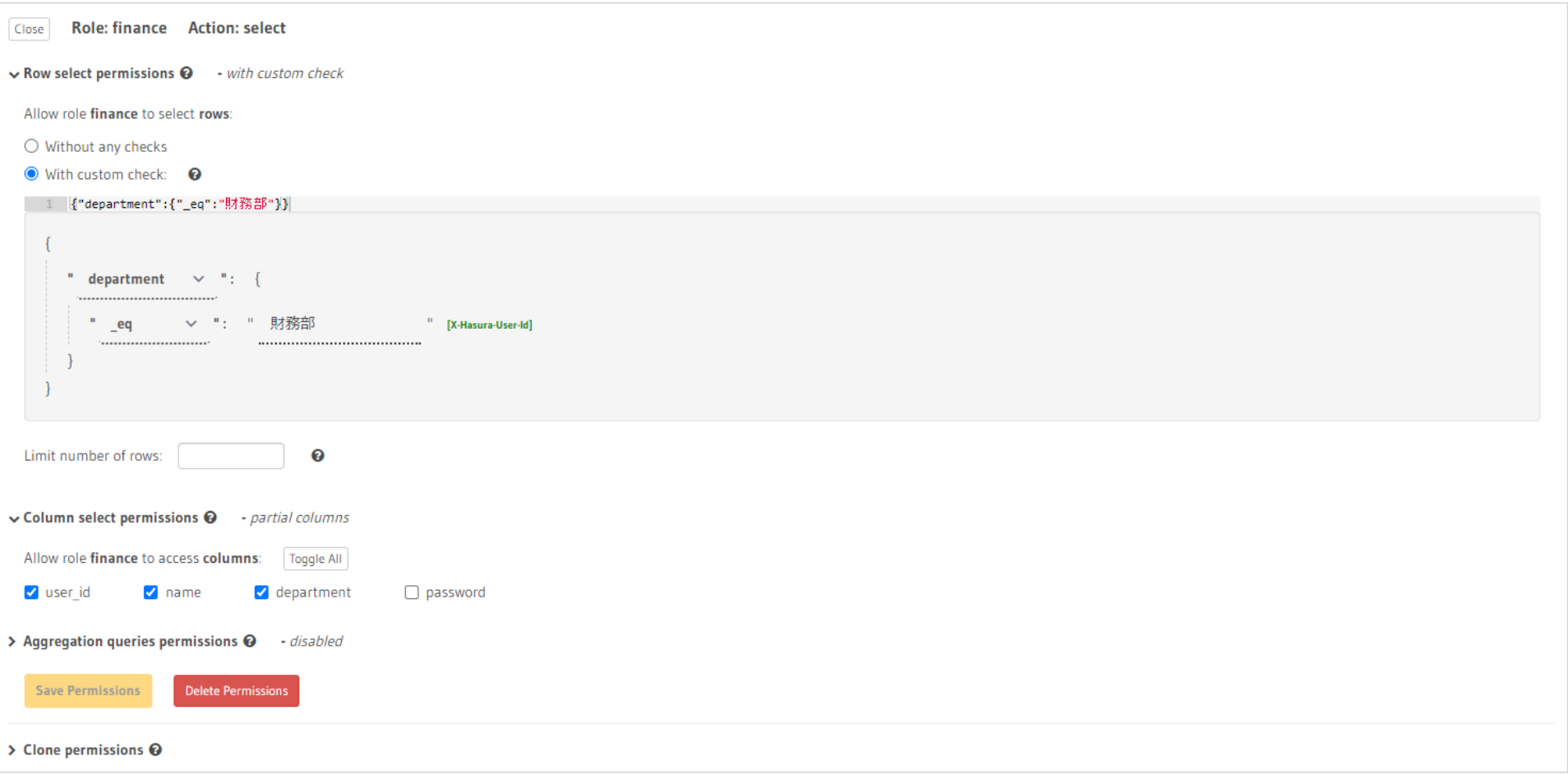
Role”user”は自らのユーザー番号と一致するデータにのみアクセスできるようしたいので、JWT中のX-Hasura-User-Numとテーブルのuser_numが一致する行のみ取得できるよう”Row select permissions”を設定



4-2. Hasuraの設定

HITACHI
Inspire the Next

Role”finance”は財務部に所属する人のpassword以外の情報を取得するよう
”Row select permissions”はdepartmentが財務部に一致する行み、
”Column select permissions”ではpassword以外の列を設定



HITACHI
Inspire the Next

Contents

5. まとめ

HITACHI
Inspire the Next

- Hasuraを使用することで対象をDBに限定されるもののGraphQL APIを容易に開発できる
  - Hasura上でテーブルを作成するとGraphQLスキーマを自動生成

- Role, Table, CRUD毎にPermissionと呼ばれるアクセスルールを設定することで、DBの行単位、列単位での制御を実施
  - JWT認証ではクライアントがアクセストークンを取得するのでクライアント側のAppによってRoleを切り替えることも可能

HITACHI
Inspire the Next

Trademarks

- HASURAは、HASURA, INC.の登録商標です。
- PostgreSQLはPostgreSQL Community Association of Canadaの登録商標です。
- GraphQLはGraphQL Foundationが管理する商標です。商標およびロゴの使用は、LF Projectsの商標ポリシーに従います
- その他記載の会社名、製品名などは、それぞれの会社の商標もしくは登録商標です。

HITACHI

Inspire the Next