WenQuanYi Micro Hei [Scale=0.9]WenQuanYi Micro Hei Mono songWenQuanYi Micro Hei sfWenQuanYi Micro Hei "zh" = 0pt plus 1pt

PostgREST Documentation

åŔŚåÿČ 4.1.0

Joe Nelson

2020 åźť 03 æIJĹ 11 æŮě

Contents

PostgREST æŸŕäÿĂäÿłçŃňçńŃçŽĎ Web æĺЈ■åŁąåŹĺïijŇåŔŕäÿžæĆĺçŽĎ PostgreSQL æŢřæ■őåžŞçŻť æŐěçŤ§æĹŘ RESTful APIãĂĆ æŢřæ■őåžŞäÿ■çŽĎçżŞæđĎçžęæİ§åŠŇæİČéŹŘåĘşåőŽäžĘ API çŽĎçńŕçĆźïijĹendpointsïijĽ'åŠŇæŞ■ä¡IJãĂĆ

PostgREST æŸŕæĽ'ŃåŁĺçijŰåĘŹ CRUD çŽĎæŻ£äżčæŰźæąĹãĂĆéĂŽåÿÿçŽĎ API æĺЈ■åŁąåŹĺæŹőéĄ■éĄĞåĹřéĄĞåĹřäÿĂäÿłéŮőéćŸãĂĆéĆčåřśæŸŕçijŰåĘŹäÿŽåŁąéĂżè¿Śå¿Ăå¿ĂäijŽéĞ■åď'■å relationalïijĽ'æŸäåřĎæŸŕäÿĂçğ■å■ŸçŰŚïijĹleakyïijĽ'çŽĎæŁ¡èśąïijŇåŔŕèČ¡åŕijèĞť'äžğçŤ§ä¡ŐæŢĹçŽĎäżčçăĄ åŞšå■ęåżžçńŃäžĘäÿĂäÿłå■ŢäÿĂçŽĎåčřæŸŐæĂğçĲ§çŘĘæİěæžŘâĂŤâĂŤâĂŤâĂŤæŢřæ■őæĲňèžńãĂĆ

CHAPTER 1

åčřæŸŐæĂğçijŰçÍŃ

æŕŤèţůåŕźçİĂæ§ěèŕćçżŞæđIJéĞ■åď■åŁşåŁĺïijŇèőľæ§ěèŕćèőąåĹŠçŘĘæÿĚçżĘèŁĆä¡£ PostgreSQL äÿžæĆĺæůżåŁăæŢřæ■őæŸŕäÿĂäżűæŻť åőźæŸŞçŽĎäžŃãĂĆäÿžæŢřæ■őåžŞåŕźèśąåĹĘéĚ■æİČéŹŘæŕŤå

CHAPTER 2

Leak-proof æŁ¡èśą

æšąæIJĽ ORM åŔĆäÿŐãĂĆåĹŻåżž SQL æĂğèČ¡æŔŘçď'žçŽĎèğĘåŻ¿ãĂĆæŢřæ■őåžŞçőąçŘĘåŚŸïijĹDAï
APIïijŇæŮăéIJĂåijĂåŔŚåŔĆäÿŐãĂĆ

CHAPTER 3

æŃěæŁśåĚşçşżæĺąåđŃ

1970åźť ïijŇE. F. Codd åIJĺäżŰçŽĎæŰĞçńă âĂIJåď ğåđŃåĚśäžńæŢřæ■őåžŞæŢřæ■őåĚşçşżæĺąåđŃâĂİ äÿ■æĽ ’źèŕĎäžĘå¡ŞæŮűäÿżåŕijçŽĎæŢřæ■őåžŞåĹĘåśĆæĺąåđŃãĂĆäžŐéČ¡æŰĞçńăäijŽåŔŚçŐřåśĆæňąæŢřæ■őåž http èůŕçŤśäźŃéŮť å■ŸåIJĺæČŁäžžçŻÿäijijæĂğãĂĆèĂŇåIJĺ PostgREST äÿ■ïijŇæĹŚäżňåřİèŕŢä¡£çŤĺçĄţæť żçŽĎè£Ğæżď åŠŇåţŇåĚěïijŇèĂŇäÿ■æŸŕåţŇåěŮèůŕçŤśãĂĆ

CHAPTER 4

äÿĂäÿłéĞ■çĆź

PostgREST æIJĽäÿĂäÿłéĞ■èęĄçŽĎçĽźçĆźãĂĆåőČéĂĆçŤĺäžŐåęĆ Nginx çŽĎåůěåĚůãĂĆè£ŹåŔŕäżěåijžèąŇåřĘäżěæŢřæ■őäÿžäÿ■å£ČçŽĎ CRUD æŞ■ä¡IJäÿŐåĚűäżŰéŮőéćŸè£ŻèąŇåźšåĞĂåĹĘçężãĂĆ

CHAPTER 5

æŤźè£ŻåĚśäžń

äÿŐäżżä¡ŢåijĂæžŘéąźçŻőäÿĂæăůïijŇæĹŚäżňéČ¡åŔŕäżěäżŐåůěåĚůäÿ■çŽĎåŁ§èČ¡åŠŇä£őåď’■äÿ■èŐůçŻŁ

CHAPTER 6

çŤ§æĂĄçşżçż§

PostgREST åĚůæIJĽäÿ■æŰ■åćđéŢ£çŽĎçŤ§æĂĄçşżçż§ïijŇçď’žä¿ŃãĂĄåžŞãĂĄåőđéłŇåŠŇçŤĺæĹůãĂĆè£

6.1 åőćæĹůçńŕåžŞ

- tomberek/aor-postgrest-client - JS, admin-on-rest
- hugomrdias/postgrest-url - JS, just for generating query URLs
- john-kelly/elm-postgrest - Elm
- mithril.postgrest - JS, Mithril
- lewisjared/postgrest-request - JS, SuperAgent
- JarvusInnovations/jarvus-postgrest-apikit - JS, Sencha framework
- davidthewatson/postgrest_python_requests_client - Python
- calebmer/postgrest-client - JS
- clesiemo3/postgrestR - R
- PierreRochard/postgrest-angular - TypeScript, generate UI from API description
- thejettdurham/postgrest-sharp-client (needs maintainer) - C#, RestSharp

6.2 éćİåďŰéĂŽç§ě

åIJĺäžŐåďŰéČÍäžďäžŠäÿŁïijĹLISTEN/NOTIFYïijĽPostgreSQL
æIJĽæŃŞåśŢåĹřåďŰéČÍéŸ§åĹŮè£Żèąňè£ŻäÿĂæ■ěåďĎçŘĘçŽĎç¡ŚæąěãĂĆè£ŹåĚąèőÿå■ŸåĆÍè£ĞçÍŃåIJĺæŢ

- frafra/postgresql2websocket - Websockets
- matthewmueller/pg-bridge - Amazon SNS
- aweber/pgsql-listen-exchange - RabbitMQ
- SpiderOak/skeeter - ZeroMQ
- FGRibreau/postgresql-to-amqp - AMQP

6.3 çďžä¿ŃåžŤçŤÍ

- subzerocloud/postgrest-starter-kit - Boilerplate for new project
- NikolayS/postgrest-google-translate - Calling to external translation service
- CodeforAustralia/heritage-near-me - Elm and PostgREST with PostGIS
- timwis/handsontable-postgrest - An excel-like database table editor
- Recmo/PostgrestSkeleton - Docker Compose, PostgREST, Nginx and Auth0
- benoror/ember-postgrest-dynamic-ui - generating Ember forms to edit data
- ruslantalpa/blogdemo - blog api demo in a vagrant image
- timwis/ext-postgrest-crud - browser-based spreadsheet
- srid/chronicle - tracking a tree of personal memories
- diogob/elm-workshop - building a simple database query UI
- marmelab/ng-admin-postgrest - automatic database admin panel
- myfreeweb/moneylog - accounting web app in Polymer + PostgREST
- tyrchen/goodfilm - example film api
- begriffs/postgrest-example - sqitch versioning for API
- SMRxT/postgrest-demo - multi-tenant logging system
- PierreRochard/postgrest-boilerplate - example auth backend

6.4 Production

- Catarse
- iAdvize
- Redsmin
- Image-charts
- Drip Depot
- OpenBooking
- Convene by Thomson-Reuters
- eGull

6.5 æŃŞåśŢ

- ppKrauss/PostgREST-writeAPI - generate Nginx rewrite rules to fit an OpenAPI spec
- diogob/postgrest-ws - expose web sockets for PostgreSQL's LISTEN/NOTIFY
- pg-safeupdate - Prevent full-table updates or deletes
- srid/spas - allow file uploads and basic auth
- svmnotn/postgrest-auth - OAuth2-inspired external auth server
- nblumoe/postgrest-oauth - OAuth2 WAI middleware

6.6 åź£åŚŁ

- subZero - Automated GraphQL & REST API with built-in caching (powered in part by PostgREST)

CHAPTER 7

èţđèłĽ

"åijĂåŔŚèţůæİĕåď'ł倣ńäžĘ, æĎ§èğĽ'åřśåČŔåIJÍä¡IJåijŁ!"

—FranĂğois-G. Ribreau

"æĹŚäÿ■å¿Ůäÿ■èŕť', äÿŐ Node.js/Waterline ORM æđĎåžžçŽĎ API åŕźæŕŤ

CPU/Memory usage çőĂçŻť'æŸŕéŽ¿äżĕç¡őä£ą. å¡ŞæĹŚäżňåIJÍ 6 äÿłçď'žä¿Ń ïijĹdynosïijĽ' æŇĄçż■æśĆæČĚ 1GB æŢřæ■őæŸŕåőČçŤŽèĞşåŔłæIJĽ' 60/70 MB åď'ğåřŔ."

—Louis Brauer

"æĹŚéİđåÿÿåŰIJæňćè£ŹæăůäÿĂäÿłäžŃåőđïijŇåĄűçĎűä¡£çŤÍ SQL DDLïijĹåŠŇ V8 javascriptïijĽ'åijĂåŔŚå¿őæIJ■åŁąãĂĆ æĹŚäżňåIJÍ 6 äÿłæIJĹåĘĚåőŇåĚÍéĞ■åĘŹäžĘäÿĂäÿł Spring + MySQL éĄŮçŢŹåžŤçŤÍçÍŃåžŔãĂĆ éĂ§åžęå£ń 10 åĂ■ïijŇäžčçăĄå¿ĹçőĂæť'ĄãĂĆèĂŇäźŃåĽ'■çŽĎäžžçŤÍäžĘ 4 äÿłäžžèŁśäžĘ 3 åźť'æŮűéŮť'ãĂĆ"

—Simone Scarduzio

CHAPTER 8

èŐůå¿ŮæŤŕæŇĄ

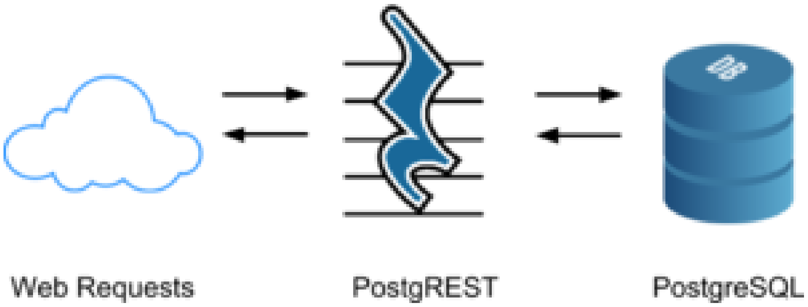
èŕěéąźçŻőæIJĽ’äÿĂäÿłåŔŃåě¡äÿŤäÿ■æŰ■æĹŘéŢ£çŽĎçď’¿åŇžãĂĆåŁăåĚěæĹŚäżňçŽĎ èĄŁåď’ĺ’åőď’ æİěèőĺèőžåŠŇæśĆåŁľ’ãĂĆåŘŇæŮűä¡ăäź§åŔŕäżěåIJÍ Github çŽĎ issues äÿŁæŘIJçť’ć bugs/featuresãĂĆ

CHAPTER 9

Tutorial 0 - èőľåőČèůŚèţůæİě

æňćè£Őä¡£çŤÍ PostgRESTïijĄèŕěåĽ■èÍĂæŸŕäÿĂäÿł Quick startïijŇåÿőåŁľä¡ăå£ńéĂ§åĹŻåżžçňňäÿĂäÿłçőĂå■ŢçŽĎ APIãĂĆ

PostgREST æŸŕäÿĂäÿłçŃňçńŃçŽĎ Web æIJ■åŁąåŹĺïijŇäÿž PostgreSQL æŢřæ■őåžŞçŤ§æĹŘ RESTful APIãĂĆ åőČæŔŘä¿Żå§žäžŐåžŢåśĆæŢřæ■őåžŞçżŞæđĎåőŽåĹűçŽĎ APIãĂĆ



æČşèęĄçŤ§æĹŘ APIïijŇæĹŚäżňåŔłéIJĂèęAåĹŻåżžäÿĂäÿłæŢřæ■őåžŞãĂĆæĽĂæIJĽçńŕçĆźåŠŇæİČéŹŘé

åIJÍæIJňæŢŹçÍŃçżŞæİ§çŽĎæŮűåĂŹïijŇæĆÍåřĘæŃěæIJĽäÿĂäÿłèČ¡çŤÍçŽĎæŢřæ■őåžŞïijŇPostgREST æIJ■åŁąåŹÍåŠŇäÿĂäÿłçőĂå■ŢçŽĎå■ŢçŤÍæĹů todo list APIãĂĆ

9.1 Step 1. æŤ¿è¡żæİ¿èĂĄéŞĄ, æĹŚäżňäijŽåÿőä¡ăçŽĎ

åIJÍä¡ăåijĂåğŃè£ŹäÿłæŢŹçÍŃæŮű, Ctrl+T äÿĂäÿŃåIJÍæŰřæăĞç■¿äÿ■æĽŞåijĂéąźçŻő èĄŁåďľľåőď . æIJĽäÿĂç¿ď å¿Ĺ nice çŽĎäžžåIJÍèĄŁåďľľåőďäÿ■æťżèůČïijŇåęĆæđIJä¡ăå■ąä¡ŔäžĘæĹŚäżňäijŽå

9.2 Step 2. åőĽ'èčĚ PostgreSQL

æĆÍåřĘéIJĂèęĄåIJÍæIJňæIJžæĹŰ Docker åőđä¿Ńäÿ■è£Řèą ŇäÿĂäÿłæŢřæ■őåžŞçŽĎçŐřäżčåĽ'ŕæIJňïijĹmod copyïijĽ'ãĂĆæĹŚäžňéIJĂèęĄ PostgreSQL 9.3 æĹŰæŽť'éńŸçĽ'ĹæIJňïijŇä¡ĘåžžèőőèĞşåřŚ 9.5 çŤĺäžŐ row-level åőĽ'åĚĺæĂğåŁ§èČ¡ïijĹåIJÍåŘŐéİćçŽĎæŢŹçĺŃäÿ■äijŽçŤĺåĹřïijĽ'ãĂĆ

åęĆæđIJæĆĺåůšçżŔçĘ§æĆĽ' PostgreSQL çŽĎä¡£çŤĺåźűåIJÍæIJňåIJřæIJĽ'åőĽ'èčĚïijŇåŔŕäżěçŻť'æŐěä¡£çŤĺ Docker äÿ■ä¡£çŤĺæŢřæ■őåžŞïijŇåŘęåĹŹæŢřæ■őåžŞéĚ■ç¡őåŕźäžŐçőĂå■ŢçŽĎæŢŹçĺŃæİěèŕť'åď'łåď'■æİĆäžĘã

åęĆæđIJä¡ăæšąæIJĽ'åőĽ'èčĚ Docker åŔŕäżěçĆź è£ŹéĞŇ. åőĽ'èčĚåě¡äžĘäźŃåŘŐïijŇèőľ'æĹŚäžňæİěæŃĽ'åŐżåźűåŘŕåŁĺæŢřæ■őåžŞçŽĎéŢIJåČŔ:

```
sudo docker run --name tutorial -p 5432:5432 \
                -e POSTGRES_PASSWORD=mysecretpassword \
                -d postgres
```

äżěäÿŁæŞ■ä¡IJäijŽäżěåőĹæŁď'è£ŻçĺŃæŰźåijŔè£Řèą Ň Docker åőđä¿Ńåźűäÿ Ťæ Žť'éIJšäÿĂäÿł 5432 çńŕæİěä¿Żä¡ăèő£éŮő PostgreSQL serverãĂĆ

9.3 Step 3. åőĽ'èčĚ PostgREST

PostgREST æŸŕä¡IJäÿžäÿĂäÿłå■ŢçŃňçŽĎäžŇè£ŻåĹűæŰĞäżűåŔŚåÿČçŽĎïijŇæŤŕæŇĄ Linux/BSD/Windows çŽĎäÿżèęĄåŔŚèąŇçĽ'ĹãĂĆèő£éŮő æIJĂæŰřçĽ'ĹæIJň äžěèŐůåŔŰäÿŃè¡¡åĹŮèąĺãĂĆåęĆæđIJæĆĺçŽĎåźşåŘřäÿ■æŸŕéćĎåĚĹæđĎåżžçŽĎåźşåŘřïijŇèŕůåŔĆéŸĚ *æžŘäżčçăĄçijŰèŕŚ* äžěèŐůåŔŰæIJĽ'åĚşåęĆä¡ŢèĞłèąŇæđĎåżžçŽĎèŕť'æŸŐãĂĆäź§åŔŕäżěéĂŽç§ěæĹŚäžňåIJÍäÿ

çŤĺäžŐäÿŃè¡¡çŽĎäžŇè£ŻåĹűæŰĞäżűæŸŕ .tar.xz åŐŃçijľ'æŰĞäżűïijĹéŹď'äžĘ Windows æŸŕ zip æŰĞäżűïijĽ'ãĂĆ èęĄæŔŘåŔŰäžŇè£ŻåĹűæŰĞäżűïijŇè£ŻåĚěçżĹçńŕåźűè£Řèą Ň

```
# download from https://github.com/begriffs/postgrest/releases/latest

tar xfJ postgrest-<version>-<platform>.tar.xz
```

ä¡ăäijŽå¿ŮåĹřäÿĂäÿłåŘ■äÿž postgrest (Windows äÿŁæŸŕ postgrest.exe) çŽĎæŰĞäżű. åĹřäžĘè£Źäÿ Ăæ■ĕïijŇåŔŕäżěåřİèŕŢè£Řèą Ň

```
./postgrest
```

åęĆæđIJäÿĂåĹĞæ■čåÿÿïijŇåőČåřĘæL'Şå■řåĞžæIJĽ'åĚşéĚ■ç¡őçŽĎçĽ'ĹæIJňåŠŇä£ąæĄŕãĂĆæĆĺåŔŕäżěçż Linuxäÿŀ çŽĎ /usr/local/bin ïijŇäżěä¿£æĆĺåŔŕäżěäżŐäżżä¡ŢçŻőå¡Ţè£Řèą ŇåőČ ãĂĆ

æşĺèğč: PostgREST ä¿İèţŰ libpq ïijĹPostgreSQL çŽĎ C èŕ■èĺĂåžŞïijĽ'çŽĎåőĽ'èčĚãĂĆæšąæIJĽ'è£Źäÿłåž ŞçŽĎèŕİä¡ăäijŽèŐůå¿ŮäÿĂäÿłå¡ćåęĆ "error while loading shared libraries: libpq.so.5." çŽĎæŁĕéŤŹïijŇäżĕäÿŃæŸŕèğčåĘşæŰźæąĹ:

9.4 Step 4. äÿž API åĹŻåżžæŢřæ■őåžŞ

äÿžäžĘè£đäÿŁåőźåŹĺåĘĚçŽĎ SQL æŐğåĹűåŔř (psql)ïijŇä¡ăéIJĂèęĄè£Řèą Ňåę Ćäÿ Ńå Ś¡äżď:

```
sudo docker exec -it tutorial psql -U postgres
```

ä¡ăåžŤèŕěçIJŃåĹřäžĘ psql çŽĎåŚ¡äżďèąŇæŔŘçďž:

```
psql (9.6.3)
Type "help" for help.

postgres=#
```

æĹŚäżňèęĄåĄŽçŽĎçňňäÿĂäżűäžŃæŸŕäÿžèęĄæŽť éIJšåIJĺ API äÿ■çŽĎæŢřæ■őåžŞåŕźèśąåĹŻåżžäÿĂäÿł åŚ¡åŘ■çŽĎ schemaãĂĆæĹŚäżňåŔŕäżĕä¡£çŤĺäżżä¡ŢæĹŚäżňåŰIJæňćçŽ
"api" æĂŐäźĹæăůãĂĆåIJĺä¡ăåĹŽåĹŽåŘŕåŁĺçŽĎåŚ¡äżďèąŇåůĕåĚůåĘĚæĽğèąŇèŕĕæŞ■ä¡IJïijŽ

```
create schema api;
```

æĹŚäżňçŽĎ API åĞĘåďĞéĂŽè£Ğèąĺæİĕèŏ¿ç¡őäÿĂäÿłçńŕçĆź /todosãĂĆ

```
create table api.todos (
  id serial primary key,
  done boolean not null default false,
  task text not null,
  due timestamptz
);

insert into api.todos (task) values
  ('finish tutorial 0'), ('pat self on back');
```

æŐĕäÿŃæİĕïijŇåĹŻåżžäÿĂäÿłèğŠèĽšæİěçŤĺäžŐè£ŻèąŇåŇ£åŘ■çŽĎ web èŕůæśĆãĂĆå¡ŞäÿĂäÿłèŕůæśĆè£ŻæİěïijŇPostgREST äijŽåIJĺæŢřæ■őåžŞäÿ■åĹĞæ■ćåĹřèŕěèğŠèĽšè£ŻèąŇæ§ěèŕ

```
create role web_anon nologin;
grant web_anon to postgres;

grant usage on schema api to web_anon;
grant select on api.todos to web_anon;
```

web_anon èğŠèĽšæŃěæIJĽèőŕéŮő api schema çŽĎæİČéŹŘïijŇåŔŕäżĕèŕżåŔŰ todos èąĺäÿ■çŽĎæŢřæ■őïijĹrowsïijĽãĂĆ çŐřåIJÍåŔŕäžěéĂĂåĞž psqlïijŇ æŸŕæŮűåĂŹåijĂåğŃä¡£çŤÍ API äžĘïijĄ

```
\q
```

9.5 Step 5. è£ŘèąŇ PostgREST

PostgREST ä¡£çŤÍäÿĂäÿłéĚ■ç¡őæŰĞäżűæİèçąőåőŽåęĆä¡Ţè£đæŐěæŢřæ■őåžŞãĂĆåĽŻåżžäÿĂäÿłæŰĞäżű tutorial.conf åźűåŁăäÿŁåęĆäÿŃåĘĚåőź:

```
db-uri = "postgres://postgres:mysecretpassword@localhost/postgres"
db-schema = "api"
db-anon-role = "web_anon"
```

èŕęçżĘéĚ■ç¡őåĘĚåőźåŔĆèğĄ *options*ãĂĆçŐřåIJÍåŔŕäžěè£ŘèąŇæIJ■åŁąåŹÍ:

```
./postgrest tutorial.conf
```

ä¡ăåžŤèŕěçIJŃåĹř

```
Listening on port 3000
Attempting to connect to the database...
Connection successful
```

çŐřåIJÍåŔŕäžěè£ŻèąŇ web èŕůæśĆäžĘãĂĆåÿĆéİćäÿŁæIJĽå¿Ĺåď ŽåŔŕäžěçŤÍçŽĎåŻ¿å¡ćåŇŰ API èŕůæśĆåůěåĚůïijŇäÿ■è£ĞåIJÍæIJňæŢŹçÍŃåĘĚæĹŚäżňä¡£çŤÍ curlãĂĆæĽŞåijĂäÿĂäÿłæŰřçŽĎ terminal (ä£İæŇĄ PostgREST ä¿İæŮğè£ŘèąŇ)ãĂĆåřİèŕŢåŕź todos åĄŽäÿĂäÿł HTTP èŕůæśĆãĂĆ

```
curl http://localhost:3000/todos
```

API è£ŤåŻđ:

```
[
  {
    "id": 1,
    "done": false,
    "task": "finish tutorial 0",
    "due": null
  },
  {
    "id": 2,
    "done": false,
    "task": "pat self on back",
    "due": null
```

(äÿŃéąţçżğçż■)

(çż■äÿŁéąţ)

```
  }
]
```

éĂŽè£Ğå¡ŞåĽ■çŽĎèğŠèĽšæİČéŹŘïijŇåŇ£åŘ■èŕůæśĆæIJĽ todos èąĺçŽĎåŔłèŕżæİČéŹŘãĂĆåęĆæđIJæĹŚäżňèŕŢåŻ¿æůżåŁăäÿĂäÿłæŰřçŽĎ todo äijŽèćńæŃŠçżİãĂĆ

```
curl http://localhost:3000/todos -X POST \
     -H "Content-Type: application/json" \
     -d '{"task": "do bad thing"}'
```

åŞ■åžŤæŸŕ 401 Unauthorized:

```
{
  "hint": null,
  "details": null,
  "code": "42501",
  "message": "permission denied for relation todos"
}
```

There we have it, a basic API on top of the database! åIJĺäÿŃäÿĂçŕĞæŢŹçĺŃäÿ■ïijŇæĹŚäżňåřĘäijŽçIJŃåĹřåęĆä¡ŢæŃŞåśŢè£Źäÿłä¿Ńå■ŘïijŇä¡£çŤĺæŻť åď■æİĆçŽÌ

CHAPTER 10

Tutorial 1 - éĞŚéŠěåŇŹ

åIJÍ *Tutorial 0 - èőľ’åőČèůŚèţůæİě* äÿ■æĹŚäżňåĹŻåżžäžĘäÿĂäÿłèŐůåŔŰ todos æŢřæ■őåŔłèŕżçŽĎ APIãĂĆä¡£çŤĺå■Ţäÿłçńŕç ĆźåĹŮåĞž todosãĂĆ æĹŚäżňæIJL’å¿Ĺåď ŽåŁđæşŢåŔŕäżĕä¡£è£Źäÿł API æŻť æIJL’èűčïijŇä¡ĘäÿĂäÿłåĕ¡çŽĎåijĂåğŃæŸŕåĔĄèőÿäÿĂä

10.1 Step 1. æůżåŁăäÿĂäÿłåŔŮä£ąçŽĎçŤĺæĹů

äÿŁäÿĂèŁĆäÿ■ïijŇè£Żèąŕå Ň£åŘ■ Web èŕůæśĆçŽĎäžŃåŘŐåIJĺæŢřæ■őåžŞäÿ■åĹŻåżžäžĘäÿĂäÿł `web_anon` èğŠèĽ’šãĂĆèőľ’æĹŚäżňåIJĺåĹŻåżžäÿĂäÿłèğŠèĽ’šåŔńåĄŽ `todo_user` çŤĺäžŐä¡£çŤĺ API è£Żèąŕèžńäż¡éłŇèŕĄçŽĎçŤĺæĹůïijŇè£Źäÿłèğ ŠèĽ’šåřĘæIJL’æİČåŕź todo list åĄŽäżżä¡ŢäžŃæČĔãĂĆ

```
-- run this in psql using the database created
-- in the previous tutorial

create role todo_user nologin;
grant todo_user to postgres;

grant usage on schema api to todo_user;
grant all on api.todos to todo_user;
grant usage, select on sequence api.todos_id_seq to todo_user;
```

10.2 Step 2. çŤ§æĹŘäÿĂäÿłåŕĘçăĄ

åőćæĹůçńŕéĂŽè£Ğ API ä¡£çŤÍ JSON Web Token è£ŻèąŇèžńäż¡éłŇèŕĄãĂĆJTW æŸŕä¡£çŤÍäžĚæIJĽæĹŚäżňåŠŇæIJ■åŁąåŹÍç§ěéĄŞçŽĎåŕĘçăĄè£ŻèąŇåŁăåŕĘç■¿åŘ■çŽĎ JSON åŕźèśąãĂĆ çŤśäžŐåőćæĹůçńŕäÿ■ç§ěéĄŞåŕĘçăĄïijŇæĽĂäżěäÿ■èČ¡çŕąæŤź token çŽĎåĘĚåőźãĂĆ PostgREST äijŽæčĂæţŃäijłéĂăçŽĎ token åźűæŃŠçżİåőČäżňãĂĆ

æĹŚäżňæİěåĹŻåżžäÿĂäÿłåŕĘçăĄåźűæŔŘä¿ŻçżŹ PostgRESTãĂĆæIJĂåě¡æČşæČşäÿĂäÿłåď■æİĆçŽĎéŢ£ä

æşĺèğč: OpenSSL toolkit æŔŘäÿłäÿĂäÿłçőĂå■ŢçŽĎæŰźåijŔæİěçŤ§æĹŘåőĽåĚĺçŽĎåŕĘçăĄãĂĆåęĆæđIJä¡ăæ

```
openssl rand -base64 32
```

æĽŞåijĂ `tutorial.conf` (åIJĺäÿŁäÿĂèŁĆäÿ■åĹŻåżžçŽĎ) åźűåřĘåŕĘçăĄæůżåŁäåIJĺæŰřçŽĎäÿĂèąŇ:

```
# add this line to tutorial.conf

jwt-secret = "<the password you created>"
```

åęĆæđIJ PostgREST server äż■æŮğåIJĺè£ŘèąŇäÿ■ïijŇéĆčäźĹéIJĂèęĄéĞ■åŘŕåőČäżěä¿£åŁăè¡¡æIJĂæŰřçŽ

10.3 Step 3. çŤ§æĹŘ token

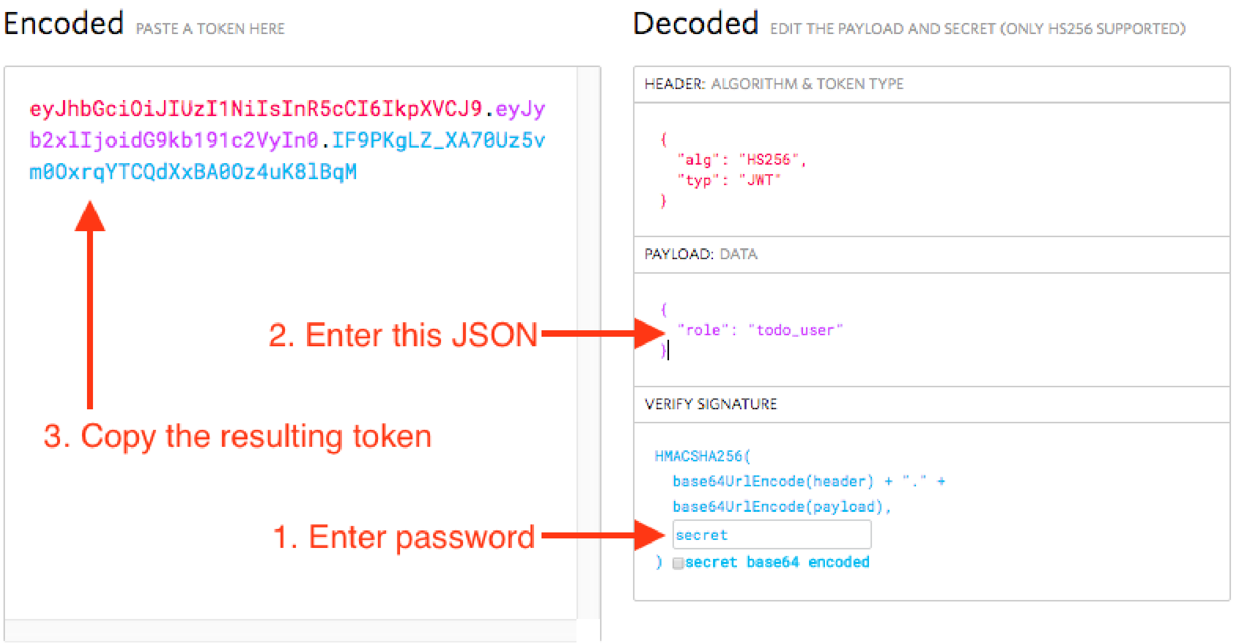
éĂŽåÿÿä¡ăèĞłåůśçŽĎäżčçăĄåIJĺæŢřæ■őåžŞæĹŰåĚűäżŰæIJ■åŁąåŹÍäÿ■åřĘåĹŻåżžåźűç■¿ç¡šèžńäż¡éłŇèŕĄ tokenïijŇä¡ĘæŸŕåIJĺæIJňæŢŹçĺŃäÿ■ïijŇæĹŚäżňåřĘâĂIJèĞłåůśåŁĺæĽŃâĂİãĂĆèůşè¡ňåĹř jwt.ioïijŇåźűåąńåĘŹåęĆäÿŃå■ŮæőţïijŽ

èŕůèőŕä¡ŔæĆĺåąńåĘŹçŽĎåŕĘçăĄïijŇèĂŇäÿ■æŸŕåŻ¿çĽĞéĞŇçŽĎ `secret`ãĂĆåąńåĘŹåŕĘçăĄåŠŇ payload äźŃåŘŐïijŇåůęä¿ğçŽĎçijŰçăĄæŢřæ■őäijŽåĹůæŰřïijŇèŕěæŢřæ■őå■ş token åď■åĹűåőČãĂĆ

æşĺèğč: èŹ¡çĎűäżď'çĽŇåŔŕèČ¡çIJŃèţůæİěå¿ĹæĺąçşŁïijŇä¡Ęå¿Ĺåőźæ ŸŞéĂĘåŘŚåĞžçŽĎ payloadãĂĆtoken äżĚäżĚæŸŕèćńç■¿åŘ■ïijŇæšąæIJĽåŁăåŕĘïijŇæĽĂäżěåęĆæđIJä¡ăæIJĽäÿ■æČşèőľåőćæĹůçńŕçIJ

10.4 Step 4. è£ŻèąŇèŕůæśĆ

åŽđåĹř terminalïijŇæĹŚäżňæİěçŤÍ `curl` æůżåŁăäÿĂäÿł todoãĂĆèŕěèŕůæśĆåřĘåŇĚæŃňäÿĂäÿłåŇĚåŘńèžńäż¡éłŇèŕĄ token çŽĎ HTTP åďť ãĂĆ



Encoded PASTE A TOKEN HERE

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJy
b2xlIjoidG9kb191c2VyIn0.IF9PKgLZ_XA70Uz5v
m0OxrqYTCQdXxBA0Oz4uK8lBqM
```

Decoded EDIT THE PAYLOAD AND SECRET (ONLY HS256 SUPPORTED)

HEADER: ALGORITHM & TOKEN TYPE

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

PAYLOAD: DATA

```
{
  "role": "todo_user"
}
```

VERIFY SIGNATURE

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  secret
) □secret base64 encoded
```

åŽ¿ 1: åęĆä¡ŢåIJÍ https://jwt.io åĹŻåžž Token

```
export TOKEN="<paste token here>"

curl http://localhost:3000/todos -X POST \
     -H "Authorization: Bearer $TOKEN"   \
     -H "Content-Type: application/json" \
     -d '{"task": "learn how to auth"}'
```

çŐřåIJÍæĹŚäżňåůšçżŔåőŇæĹŘäžĘæĹŚäżňçŽĎ todo list äÿ■çŽĎæĽĂæIJĽäÿĽäÿłéąźçŻőïijŇæĽĂäżěæĹŚä PATCH èŕůæśĆåřĘäżŰäżňåĚĺèő¿ç¡őäÿž doneãĂĆ

```
curl http://localhost:3000/todos -X PATCH \
     -H "Authorization: Bearer $TOKEN"    \
     -H "Content-Type: application/json"  \
     -d '{"done": true}'
```

èŕůæśĆäÿĂäÿŃ todo çIJŃçIJŃè£ŹäÿĽéąźïijŇåĚĺéČĺéČ¡åůšåőŇæĹŘäžĘ.

```
curl http://localhost:3000/todos
```

```
[
  {
    "id": 1,
    "done": true,
    "task": "finish tutorial 0",
```

(äÿŃéąţçżğçż■)

(çż■äÿŁéąţ)

```
    "due": null
  },
  {
    "id": 2,
    "done": true,
    "task": "pat self on back",
    "due": null
  },
  {
    "id": 3,
    "done": true,
    "task": "learn how to auth",
    "due": null
  }
]
```

10.5 Step 4. æůżåŁăè£ĞæIJ§æŮűéŮť

çŻőåĽ■ïijŇæĹŚäżňçŽĎèőď èŕĄ token åŕźäžŐæĽĂæIJĽèŕůæśĆéČ¡æŸŕäÿĂèĞť æIJĽæŢĹçŽĎãĂĆæIJ■åŁą
JWT åŕĘçăĄïijŇåřśäijŽéĂŽè£ĞéłŇèŕĄãĂĆ

æŻť åě¡çŽĎç■ŰçŢĕæŸŕèőľ token ä¡£çŤÍ exp åčřæŸŐäÿĂäÿłè£ĞæIJ§æŮűéŮť æĹşãĂĆè£ŹæŸŕ PostgREST çĽźåĹńåŕźå¿ĚçŽĎäÿď äÿł JWT åčřæŸŐäźŃäÿĂãĂĆ

Claim	Interpretation
`role`	The database role under which to execute SQL for API request
`exp`	Expiration timestamp for token, expressed in "Unix epoch time"

æşĺèğč: Unix æŮűéŮť æĹş (Unix epoch time) èćńåőŽäźĽäÿžèĞł 1970 åźť 1 æIJĹ 1 æŮě 00:00:00 å■ŔèřČäÿŰçŢŇæŮűïijĹUTCïijĽäżěæİĕåĹřçŐřåIJÍçŽĎæĂżçğŠæŢřïijŇäÿ■èĂČèŹŚéŮřçğŠãĂĆ

äÿžäžĘåIJÍèąŇåŁĺäÿ■èğĆåŕ§è£ĞæIJ§ïijŇæĹŚäżňåřĘæůżåŁăäÿĂäÿłåIJÍ 5min äźŃåŘŐè£ĞæIJ§çŽĎ `exp` åčřæŸŐãĂĆéęŰåĚĹæĽ¿åĹřäżŐå¡ŞåĽ■æŮűéŮť çőŮèţůåĹř 5min äźŃåŘŐçŽĎæŮűéŮť æĹşãĂĆ åIJÍ psql äÿ■è£ŘèąŇïijŽ

```sql
select extract(epoch from now() + '5 minutes'::interval) :: integer;
```

åŻđåĹř jwt.io åźűä£őæŤź payload

```
{
  "role": "todo_user",
  "exp": "<computed epoch value>"
}
```

æŃůèťİæŰřçŽĎ tokenïijŇçĎűåŘŐåřĘåĚűä£İå■ŸäÿžäÿĂäÿłæŰřçŽĎçŐŕåćČåŔŸéĞŔãĂĆ

```
export NEW_TOKEN="<paste new token>"
```

åřİèŕŢåIJÍè£ĞæIJ§æŮűéŮťçŽĎåĽ■åŘŐä¡£çŤÍ curl è£Żèąŗèŕěèŕůæś Ć:

```
curl http://localhost:3000/todos \
     -H "Authorization: Bearer $NEW_TOKEN"
```

è£ĞæIJ§äżěåŘŐ, èŕě API äijŽè£ŤåŻđäÿĂäÿł HTTP 401 Unauthorized:

```
{"message":"JWT expired"}
```

10.6 éŹĎåŁăéćŸ: çńŃå■şæŠď éŤĂ

Even with token expiration there are times when you may want to immediately revoke access for a specific token. For instance, suppose you learn that a disgruntled employee is up to no good and his token is still valid.

To revoke a specific token we need a way to tell it apart from others. Let's add a custom `email` claim that matches the email of the client issued the token.

Go ahead and make a new token with the payload

```
{
  "role": "todo_user",
  "email": "disgruntled@mycompany.com"
}
```

Save it to an environment variable:

```
export WAYWARD_TOKEN="<paste new token>"
```

PostgREST allows us to specify a stored procedure to run during attempted authentication. The function can do whatever it likes, including raising an exception to terminate the request.

First make a new schema and add the function:

```
create schema auth;
grant usage on schema auth to web_anon, todo_user;
```

(äÿŃéąţçżğçż■)

(çż■äÿŁéąţ)

```
create or replace function auth.check_token() returns void
  language plpgsql
  as $$
begin
  if current_setting('request.jwt.claim.email', true) =
     'disgruntled@mycompany.com' then
    raise insufficient_privilege
      using hint = 'Nope, we are on to you';
  end if;
end
$$;
```

Next update `tutorial.conf` and specify the new function:

```
# add this line to tutorial.conf

pre-request = "auth.check_token"
```

Restart PostgREST for the change to take effect. Next try making a request with our original token and then with the revoked one.

```
# this request still works

curl http://localhost:3000/todos \
     -H "Authorization: Bearer $TOKEN"

# this one is rejected

curl http://localhost:3000/todos \
     -H "Authorization: Bearer $WAYWARD_TOKEN"
```

The server responds with 403 Forbidden:

```
{
  "hint": "Nope, we are on to you",
  "details": null,
  "code": "42501",
  "message": "insufficient_privilege"
}
```

CHAPTER 11

åŔŕæĽğèąŇæŰĞäżű

[äÿŃè¡¡éąţ]

äÿŃè¡¡éąţéİćåĔůæIJĽ Mac OS XãĂĄWindows åŠŇåĞăäÿł Linux åŔŚèąŇçĽĹçŽĎéćĎçijŰèŕŚæŰĞäżűãĂĆèğčåŐŃäźŃåŘŐåŔŕäżěè£ŘèąŇåŔŕæĽğèąŇæŰĞäżűåŁă `--help` æăĞå£ŮæİĕæşĕçIJŃä¡£çŤÍèŕťæŸŐ:

```
# èğčåŐŃ tar åŇĚ (available at https://github.com/begriffs/postgrest/
↪releases/latest)

$ tar Jxf postgrest-[version]-[platform].tar.xz

# åřİèŕŢè£ŘèąŇ
$ ./postgrest --help

# You should see a usage help message
```

CHAPTER 12

Homebrew

åIJÍ Mac äÿŁä¡ăåŔŕäżěä¡£çŤÍ Homebrew æİěåőĽèčĚ PostgREST

```
# çąőä£İ brew æŸŕæIJĂæŰřçŽĎ
brew update

# æčĂæ§ě brew çŽĎ setup æIJL'æšąéŮőéćŸ
brew doctor

# åőL'èčĚ postgrest
brew install postgrest
```

èŕěåŚ¡äżď äijŽèĞłåŁĺåřĘ PostgreSQL å¡ŞåĄŽä¿İèţŰåőĽèčĚ. èŕěèĞçĺŃå¿Ăå¿ĂéIJĂèęĄéŢ£è¿¿15åĹĘéŠ§æĽ■èČ¡åőĽèčĚè¡ŕäżűåŇĚåŔŁåĚűä¿İèţŰãĂĆ

åőĽèčĚåőŇæĹŘåŘŐïijŇèŕěåůěåĚůäijŽèćńæůżåŁäåĹř $PATH äÿ■ïijŇä¡ăåŔŕäżěåIJĺäżżæĎŔä¡■ç¡őä¡£çŤĺïijŽ

```
postgrest --help
```

CHAPTER 13

PostgreSQL ä¿İèţŰ

èęĄä¡£çŤĺ PostgREST æĆÍåřĘéIJĂèęĄåőĽèčĚæŢřæ■őåžŞïijĹPostgreSQL 9.3 æĹŰæŻť éńŸçĽĹæIJňïijĽãĂĆ æĆÍåŔŕäżěä¡£çŤÍåČŔ Amazon RDS è£ŹæăůçŽĎäÿIJèě£ïijŇä¡ĘæŸŕåIJÍæIJňåIJřåőĽèčĚæIJňèžńæŕŤè¿Čä¿£åőIJïijŇæŻť ä¿£äžŐåijĂåŔŚãĂĆ

- OS X èŕť æŸŐ
- Ubuntu 14.04 èŕť æŸŐ
- Windows åőĽèčĚåŇĚ

CHAPTER 14

æžŘäżčçăĄçijŰèŕŚ

æşĺèğč: æĹŚäżňäÿ■åżžèőőåIJĺ **Alpine Linux** äÿŁæđĎåżžåŠŇä¡£çŤĺ PostgRESTïijŇåŻăäÿžåIJĺèŕěåźşåŔřæIJĽ’è£Ğ GHC åĘĚå■ŸæşĎæijŔçŽĎæŁĕåŚŁãĂĆ

å¡ŞæĆĺçŽĎçşżçż§æšąæIJĽ’éćĎæđĎåżžçŽĎåŔŕæĽ’ğèąŇæŰĞäżűæŮűïijŇåŔŕäżĕäżŐæžŘäżčçăĄæđĎåżžéąź
Stack ãĂĆåőČåřĘåIJĺæĆĺçŽĎçşżçż§äÿŁåőĽ’èčĚäżżä¡Ţå£ĚèęĄçŽĎ Haskell ä¿İèţŰãĂĆ

- åőĽ’èčĚ Stack
- åőĽ’èčĚä¿İèţŰåžŞ

Operating System	Dependencies
Ubuntu/Debian	libpq-dev, libgmp-dev
CentOS/Fedora/Red Hat	postgresql-devel, zlib-devel, gmp-devel
BSD	postgresql95-server
OS X	postgresql, gmp

- æđĎåżžåźűåőĽ’èčĚ

```
git clone https://github.com/begriffs/postgrest.git
cd postgrest

# adjust local-bin-path to taste
stack build --install-ghc --copy-bins --local-bin-path /usr/local/
↪bin
```

æşĺèğč: åęĆæđIJä¡ăæđĎåżžåď śèť ĕïijŇèĂŇäÿŤä¡ăçŽĎçşżçż§åŔłæIJĽäÿ■åĹř 1GB åĘĚå■ŸïijŇåřİèŕŢæůżåŁăäÿĂäÿł swap æŰĞäżűãĂĆ

- æčĂæ§ěåőĽèčĚæŸŕåŘęæĹŘåŁ§: `postgrest --help`.

14.1 PostgREST æţŃèŕŢåěŮäżű

14.1.1 åĹŻåżžæţŃèŕŢåžŞ

äÿžäžĘæ■čçąőèŁŘèąŇpostgrestèŁŻèąŇæţŃèŕŢïijŇéęŰåĚĹéIJĂèęĄåĹŻåżžäÿĂäÿłæŢřæ■őåžŞãĂĆäÿžæ■ď'ï

èĎŽæIJňéIJĂèęĄäżěäÿŃåŔĆæŢřïijŽ

```
test/create_test_db connection_uri database_name [test_db_user] [test_
↪db_user_password]
```

ä¡£çŤÍ'connection URI <https://www.postgresql.org/docs/current/static/libpq-connect.html#AEN45347>'_ åŐżæŇĞåőŽæŢřæ■őåžŞçŤĺæĹůãĂĄåŕĘçăĄãĂĄäÿżæIJžäżěåŔŁçńŕåŔčãĂĆäÿ■èęĄåIJĺæŢřæ■őå

èĎŽæIJňäÿ■çŽĎ:code:'database_name'åŔĆæŢřïijŇæŸŕåřĘèęĄè£đæŐěåĹřçŽĎæŢřæ■őåžŞåŘ■çğřãĂĆåęĆ

åęĆæđIJä¡£çŤĺæŇĞåőŽçŽĎæŢřæ■őåžŞçŤĺæĹůè£ŻèąŇåăĘæăĹæţŃèŕŢãĂĆæŕŔæňąæţŃèŕŢè£ŘèąŇåŘŐïijŇ

åęĆæđIJæIJłæŇĞåőŽçŤĺæĹůïijŇèĎŽæIJňåřĘäijŽçŤ§æĹŘèğŠèĽšåŘ■:code:*postgrest_test*_ïijŇåźűäżěæĽĂ

åęĆæđIJä¡£çŤĺäÿĂäÿłåůšçżŔå■ŸåIJĺçŽĎçŤĺæĹůæİěè£ŻèąŇæţŃèŕŢè£đæŐěïijŇéĆčäźĹè£ŸéIJĂèęĄæŇĞåč

èŕěèĎŽæIJňåřĘè£ŤåŻđæţŃèŕŢè£ĞçĺŃäÿ■ä¡£çŤĺçŽĎæŢřæ■őuri - èŕĕuriäÿŐåřĘåIJĺçŤ§äžğäÿ■ä¡£çŤĺçŽĎéĚ■ç¡őæŰĞäżűåŔĆæŢř:code:'db-uri'çŻÿåŕźåžŤãĂĆ

çŤ§æĹŘçŤĺæĹůåŠŇåŕĘçăĄåĚĄèőÿåĹŻåżžæŢřæ■őåžŞåźűåŕźäżżä¡ŢpostgresæIJ■åŁąåŹĺè£ŘèąŇæţŃèŕŢïijŇ

14.1.2 è£ŘèąŇæţŃèŕŢ

äÿžäžĘè£ŘèąŇæţŃèŕŢïijŇå£ĚéążåIJĺçŐŕåćČåŔŸéĞŔäÿ■æŔŘä¿ŻæŢřæ■őåžŞçŽĎuriä£ąæĄŕïijŇåŕźåžŤçŽĎ

éĂŽåÿÿæČĚåĘţäÿŃïijŇåĹŻåżžæŢřæ■őåžŞäÿŐè£ŘèąŇæţŃèŕŢåŚ¡äżď'äijŽåIJĺåŘŇäÿĂåŚ¡äżď'èąŇäÿ■æĽğè

```
POSTGREST_TEST_CONNECTION=$(test/create_test_db "postgres://
↪postgres:pwd@database-host" test_db) stack test
```

åIJĺåŘŇäÿĂæŢřæ■őåžŞäÿ■éĞ■åď'■è£ŘèąŇæŮűïijŇåžŤèŕěåŕijåĞžæŢřæ■őåžŞè£đæŐěåŔŸéĞŔä£ąæĄŕïijŽ

```
export POSTGREST_TEST_CONNECTION=$(test/create_test_db "postgres://
↪postgres:pwd@database-host" test_db)
stack test
stack test
...
```

åęĆæđIJçŐŕåćČåŔŸéĞŔäÿžçľ žæĹŰæIJłæŇĞåőŽïijŇéĆčäźĹætŃèŕŢçŽĎè£ŘèąŇçĺŇåžŔåřĘäijŽéżŸèőď è£

```
postgres://postgrest_test@localhost/postgrest_test
```

äÿŁè£řè£đæŐěåĄĞåőŽætŃèŕŢçŽĎæIJ■åŁąåŹĺåIJĺæIJňåIJř:code:*localhost:code:*ïijŇåźűäÿŤæŢřæ■őåžŞçŤĺ

14.1.3 éŤĂæŕĄæŢřæ■őåžŞ

ætŃèŕŢåőŇæĹŘäźŃåŘŐïijŇætŃèŕŢæŢřæ■őåžŞåřĘäijŽèćńä£İçŢŹïijŇåŘŇæŮűè£ŸäijŽåIJĺpostgresæIJ■åŁ

```
test/destroy_test_db connection_uri database_name
```

14.1.4 ä¡£çŤĺ Docker ætŃèŕŢ

äÿžäžĘçőĂåŇŰè£đæŐěéİđæIJňåIJřçŐŕåćČPostgreSQLçŽĎætŃèŕŢçŐŕåćČèő¿ç¡őïijŇåŔŕäżěä¡£çŤĺäÿĂçğ■é

ä¿ŃåęĆïijŇåęĆæđIJæŸŕåIJĺmacäÿŁåĄŽæIJňåIJřäijĂåŔŚïijĹäÿŤåůšçżŔåőĽèčĚäžĘDockeræIJ■åŁąïijĽïijŇ

```
$ docker run --name db-scripting-test -e POSTGRES_PASSWORD=pwd -p␣
↪5434:5432 -d postgres
$ POSTGREST_TEST_CONNECTION=$(test/create_test_db "postgres://
↪postgres:pwd@localhost:5434" test_db) stack test
```

æ■d'åd'ŰïijŇåęĆæđIJéĂŽè£ĞåĹŻåżždockeråőźåŹĺè£ŘèąŇæİěè£ŘèąŇåäĘæăĹætŃèŕŢïijĹåŕźäžŐGHCä¡Őä Sierraæ Ÿŕå£Ěèę ĄçŽĎïijŇåIJĺ:code:‘stack test‘äijŽæŔŘçď žåijĆåÿÿïijĽïijŇä¡ăåŔŕäżěåIJĺå■ŢçŃňçŽĎåőźåŹĺäÿ■è

ä¡£çŤĺäżěäÿŃèĎŽæIJňæđĎåżžætŃèŕŢä¡£çŤĺçŽĎåőźåŹĺ:code:*test/Dockerfile.test*ïijŽ

```
$ docker build -t pgst-test - < test/Dockerfile.test
$ mkdir .stack-work-docker ~/.stack-linux
```

åIJĺætŃèŕŢåőźåŹĺéęŰæňąè£ŘèąŇæŮűïijŇåřĘäijŽèŁśèť źè¿ČéŢ£çŽĎæŮűéŮť ïijŇåŻăäÿžéIJĂèęAçijŞå■ŸçŽ *linux‘æŰĞäżűåď źä¡IJäÿžåőźåŹĺçŽĎæŇĆè¡¡å■ůïijŇäżěçąőä£İæĹŚäżňåIJĺäÿĂæňąæĂğæĺąåijŔäÿŃè£ŘèąŇåőźåŹ* work-docker‘åŘŇæăůéIJĂèęĄæŸăåřĎèĞşåőźåŹĺäÿ■ïijŇåIJĺä¡£çŤĺLinuxäÿ■çŽĎstackæŮűå£ĚéążæŇĞåőŽïijŇä *work*ãĂĆïijĹåIJĺSierraäÿŁ:code:‘stack build‘åŔŕäżěæ■čåÿÿä¡£çŤĺïijŇèĂŇ:code:‘stack test‘åIJĺGHC 8.0.1äÿ■äÿ■äijŽèţůä¡IJçŤĺïijĽ

æŰĞäżűåď źæŸăåřĎèĞşdockeråőźåŹĺäÿ■ïijŽ

```
$ docker run --name pg -e POSTGRES_PASSWORD=pwd  -d postgres
$ docker run --rm -it -v `pwd`:`pwd` -v ~/.stack-linux:/root/.stack --
↪link pg:pg -w="`pwd`" -v `pwd`/.stack-work-docker:`pwd`/.stack-work␣
↪pgst-test bash -c "POSTGREST_TEST_CONNECTION=$(test/create_test_db
↪"postgres://postgres:pwd@pg" test_db) stack test"
```

åIJĺmacäÿŁïijŇDockerçŽĎååĘæăĹæţŃèŕŢæŰźåijŔåęĆäÿŃïijŽ

```
$ host_ip=$(ifconfig en0 | grep 'inet ' | cut -f 2 -d' ')
$ export POSTGREST_TEST_CONNECTION=$(test/create_test_db "postgres://
↪postgres@$HOST" test_db)
$ docker run --rm -it -v `pwd`:`pwd` -v ~/.stack-linux:/root/.stack -v␣
↪`pwd`/.stack-work-docker:`pwd`/.stack-work -e "HOST=$host_ip" -e
↪"POSTGREST_TEST_CONNECTION=$POSTGREST_TEST_CONNECTION" -w="`pwd`"␣
↪pgst-test bash -c "stack test"
$ test/destroy_test_db "postgres://postgres@localhost" test_db
```

CHAPTER 15

éĚ■ç¡ő

The PostgREST server reads a configuration file to determine information about the database and how to serve client requests. There is no predefined location for this file, you must specify the file path as the one and only argument to the server:

```
postgrest /path/to/postgrest.conf
```

The file must contain a set of key value pairs. At minimum you must include these keys:

```
# postgrest.conf

# The standard connection URI format, documented at
# https://www.postgresql.org/docs/current/static/libpq-connect.html
↪#AEN45347
db-uri       = "postgres://user:pass@host:5432/dbname"

# The name of which database schema to expose to REST clients
db-schema    = "api"

# The database role to use when no client authentication is provided.
# Can (and probably should) differ from user in db-uri
db-anon-role = "anon"
```

The user specified in the db-uri is also known as the authenticator role. For more information about the anonymous vs authenticator roles see the *èğŠèL'šçşżçż§æęĆè£ř*.

Here is the full list of configuration parameters.

Name	Type	Default	Required
db-uri	String		Y
db-schema	String		Y
db-anon-role	String		Y
db-pool	Int	10	
server-host	String	*4	
server-port	Int	3000	
server-proxy-uri	String		
jwt-secret	String		
secret-is-base64	Bool	False	
max-rows	Int	∞	
pre-request	String		

db-uri The standard connection PostgreSQL URI format. Symbols and unusual characters in the password or other fields should be percent encoded to avoid a parse error. On older systems like Centos 6, with older versions of libpq, a different db-uri syntax has to be used. In this case the URI is a string of space separated key-value pairs (key=value), so the example above would be `"host=host user=user port=5432 dbname=dbname password=pass"`. Also allows connections over Unix sockets for higher performance.

db-schema The database schema to expose to REST clients. Tables, views and stored procedures in this schema will get API endpoints.

db-anon-role The database role to use when executing commands on behalf of unauthenticated clients.

db-pool Number of connections to keep open in PostgREST's database pool. Having enough here for the maximum expected simultaneous client connections can improve performance. Note it's pointless to set this higher than the `max_connections` GUC in your database.

server-host Where to bind the PostgREST web server.

server-port The port to bind the web server.

server-proxy-uri Overrides the base URL used within the OpenAPI self-documentation hosted at the API root path. Use a complete URI syntax `scheme:[//[user:password@]host[:port]][/]path[?query][#fragment]`. Ex. `https://postgrest.com`

```
{
  "swagger": "2.0",
  "info": {
    "version": "0.4.0.0",
    "title": "PostgREST API",
    "description": "This is a dynamic API generated by PostgREST"
  },
```

(äÿŃéąţçżğçż■)

(çż■äÿŁéąţ)

```
  "host": "postgrest.com:443",
  "basePath": "/",
  "schemes": [
    "https"
  ]
}
```

jwt-secret The secret used to decode JWT tokens clients provide for authentication. If this parameter is not specified then PostgREST refuses authentication requests. Choosing a value for this parameter beginning with the at sign such as `@filename` loads the secret out of an external file. This is useful for automating deployments. Note that any binary secrets must be base64 encoded.

secret-is-base64 When this is set to `true`, the value derived from `jwt-secret` will be treated as a base64 encoded secret.

max-rows A hard limit to the number of rows PostgREST will fetch from a view, table, or stored procedure. Limits payload size for accidental or malicious requests.

pre-request A schema-qualified stored procedure name to call right after switching roles for a client request. This provides an opportunity to modify SQL variables or raise an exception to prevent the request from completing.

15.1 åŘŕåŁÍ Server

PostgREST outputs basic request logging to stdout. When running it in an SSH session you must detach it from stdout or it will be terminated when the session closes. The easiest technique is redirecting the output to a logfile or to the syslog:

```
ssh foo@example.com \
  'postgrest foo.conf </dev/null >/var/log/postgrest.log 2>&1 &'

# another option is to pipe the output into "logger -t postgrest"
```

(Avoid `nohup postgrest` because the HUP signal is used for manual *Schema éĞ■è¡¡*.)

CHAPTER 16

çąňåŇŰ PostgREST

PostgREST is a fast way to construct a RESTful API. Its default behavior is great for scaffolding in development. When it's time to go to production it works great too, as long as you take precautions. PostgREST is a small sharp tool that focuses on performing the API-to-database mapping. We rely on a reverse proxy like Nginx for additional safeguards.

The first step is to create an Nginx configuration file that proxies requests to an underlying PostgREST server.

```
http {
  ...
  # upstream configuration
  upstream postgrest {
    server localhost:3000;
    keepalive 64;
  }
  ...
  server {
    ...
    # expose to the outside world
    location /api/ {
      default_type  application/json;
      proxy_hide_header Content-Location;
      add_header Content-Location  /api/$upstream_http_content_
 ↪location;
      proxy_set_header  Connection "";
      proxy_http_version 1.1;
```

(äÿŃéąţçżğçż■)

(çż■äÿŁéąţ)

```
      proxy_pass http://postgrest/;
    }
    ...
  }
}
```

16.1 éŸżæ■ćåĚÍèąÍæŞ■ä¡lJ

Each table in the admin-selected schema gets exposed as a top level route. Client requests are executed by certain database roles depending on their authentication. All HTTP verbs are supported that correspond to actions permitted to the role. For instance if the active role can drop rows of the table then the DELETE verb is allowed for clients. Here's an API request to delete old rows from a hypothetical logs table:

```
DELETE /logs?time=lt.1991-08-06 HTTP/1.1
```

However it's very easy to delete the **entire table** by omitting the query parameter!

```
DELETE /logs HTTP/1.1
```

This can happen accidentally such as by switching a request from a GET to a DELETE. To protect against accidental operations use the pg-safeupdate PostgreSQL extension. It raises an error if UPDATE or DELETE are executed without specifying conditions. To install it you can use the PGXN network:

```
sudo -E pgxn install safeupdate

# then add this to postgresql.conf:
# shared_preload_libraries='safeupdate';
```

This does not protect against malicious actions, since someone can add a url parameter that does not affect the result set. To prevent this you must turn to database permissions, forbidding the wrong people from deleting rows, and using row-level security if finer access control is required.

16.2 Count-Header DoS

For convenience to client-side pagination controls PostgREST supports counting and reporting total table size in its response. As described in *Limit åŠŇåĹĘéąţ*, responses ordinarily include a range but leave the total unspecified like

```
HTTP/1.1 200 OK
Range-Unit: items
Content-Range: 0-14/*
```

However including the request header `Prefer: count=exact` calculates and includes the full count:

```
HTTP/1.1 206 Partial Content
Range-Unit: items
Content-Range: 0-14/3573458
```

This is fine in small tables, but count performance degrades in big tables due to the MVCC architecture of PostgreSQL. For very large tables it can take a very long time to retrieve the results which allows a denial of service attack. The solution is to strip this header from all requests:

```
Nginx stuff. Remove any prefer header which contains the word count
```

æşÍèğč: In future versions we will support `Prefer: count=estimated` to leverage the PostgreSQL statistics tables for a fast (and fairly accurate) result.

16.3 HTTPS

See the ssl section of the authentication guide.

16.4 éŹŘéĂ§

Nginx supports "leaky bucket" rate limiting (see official docs). Using standard Nginx configuration, routes can be grouped into *request zones* for rate limiting. For instance we can define a zone for login attempts:

```
limit_req_zone $binary_remote_addr zone=login:10m rate=1r/s;
```

This creates a shared memory zone called "login" to store a log of IP addresses that access the rate limited urls. The space reserved, 10 MB (`10m`) will give us enough space to store a history of 160k requests. We have chosen to allow only allow one request per second (`1r/s`).

Next we apply the zone to certain routes, like a hypothetical stored procedure called `login`.

```
location /rpc/login/ {
  # apply rate limiting
```

(äÿŃéąţçżğçż■)

(çż■äÿŁéąţ)

```
  limit_req zone=login burst=5;
}
```

The burst argument tells Nginx to start dropping requests if more than five queue up from a specific IP.

Nginx rate limiting is general and indescriminate. To rate limit each authenticated request individually you will need to add logic in a *Custom Validation* function.

CHAPTER 17

èřČèŕŢ

The PostgREST server logs basic request information to stdout, including the requesting IP address and user agent, the URL requested, and HTTP response status. However this provides limited information for debugging server errors. It's helpful to get full information about both client requests and the corresponding SQL commands executed against the underlying database.

A great way to inspect incoming HTTP requests including headers and query params is to sniff the network traffic on the port where PostgREST is running. For instance on a development server bound to port 3000 on localhost, run this:

```
# sudo access is necessary for watching the network
sudo ngrep -d lo0 port 3000
```

The options to ngrep vary depending on the address and host on which you've bound the server. The binding is described in the *Configuration* section. The ngrep output isn't particularly pretty, but it's legible. Note the `Server` response header as well which identifies the version of server. This is important when submitting bug reports.

Once you've verified that requests are as you expect, you can get more information about the server operations by watching the database logs. By default PostgreSQL does not keep these logs, so you'll need to make the configuration changes below. Find `postgresql.conf` inside your PostgreSQL data directory (to find that, issue the command `show data_directory;`). Either find the settings scattered throughout the file and change them to the following values, or append this block of code to the end of the configuration file.

```
# send logs where the collector can access them
log_destination = "stderr"
```

(äÿŃéąţçżğçż■)

(çż■äÿŁéąţ)

```
# collect stderr output to log files
logging_collector = on

# save logs in pg_log/ under the pg data directory
log_directory = "pg_log"

# (optional) new log file per day
log_filename = "postgresql-%Y-%m-%d.log"

# log every kind of SQL statement
log_statement = "all"
```

Restart the database and watch the log file in real-time to understand how HTTP requests are being translated into SQL commands.

17.1 Schema éĞ■è¡¡

Users are often confused by PostgREST's database schema cache. It is present because detecting foreign key relationships between tables (including how those relationships pass through views) is necessary, but costly. API requests consult the schema cache as part of *èţĎæžŘåţŇåěŮ*. However if the schema changes while the server is running it results in a stale cache and leads to errors claiming that no relations are detected between tables.

To refresh the cache without restarting the PostgREST server, send the server process a SIGHUP signal:

```
killall -HUP postgrest
```

In the future we're investigating ways to keep the cache updated without manual intervention.

CHAPTER 18

åďĞçŤÍ URL çżŞæđĎ

As discussed in *å■ŢæŢřæĹŰåď’■æŢř*, there are no special URL forms for singular resources in PostgREST, only operators for filtering. Thus there are no URLs like `/people/1`. It would be specified instead as

```
GET /people?id=eq.1
Accept: application/vnd.pgrst.object+json
```

This allows compound primary keys and makes the intent for singular response independent of a URL convention.

Nginx rewrite rules allow you to simulate the familiar URL convention. The following example adds a rewrite rule for all table endpoints, but you’ll want to restrict it to those tables that have a numeric simple primary key named "id."

```
# support /endpoint/:id url style
location ~ ^/([a-z_]+)/([0-9]+) {

  # make the response singular
  proxy_set_header Accept 'application/vnd.pgrst.object+json';

  # assuming an upstream named "postgrest"
  proxy_pass http://postgrest/$1?id=eq.$2;

}
```

CHAPTER 19

èąĺ & èğĘåŻ¿

All views and tables in the active schema and accessible by the active database role for a request are available for querying. They are exposed in one-level deep routes. For instance the full contents of a table *people* is returned at

```
GET /people HTTP/1.1
```

There are no deeply/nested/routes. Each route provides OPTIONS, GET, POST, PATCH, and DELETE verbs depending entirely on database permissions.

æşĺèğč: Why not provide nested routes? Many APIs allow nesting to retrieve related information, such as `/films/1/director`. We offer a more flexible mechanism (inspired by GraphQL) to embed related information. It can handle one-to-many and many-to-many relationships. This is covered in the section about *èţĎœžŘåţŇåĕŮ*.

19.1 æřťåźşè£Ǧæżď (Rows)

You can filter result rows by adding conditions on columns, each condition a query string parameter. For instance, to return people aged under 13 years old:

```
GET /people?age=lt.13 HTTP/1.1
```

Adding multiple parameters conjoins the conditions:

```
GET /people?age=gte.18&student=is.true HTTP/1.1
```

These operators are available:

abbreviation	meaning
eq	equals
gte	greater than or equal
gt	greater than
lte	less than or equal
lt	less than
neq	not equal
like	LIKE operator (use * in place of %)
ilike	ILIKE operator (use * in place of %)
in	one of a list of values e.g. `?a=in.1,2,3` – also supports commas in quoted strings like `?a=in."hi,there","yes,you"`
is	checking for exact equality (null,true,false)
@@	full-text search using to_tsquery
@>	contains e.g. `?tags=@>.{example, new}`
<@	contained in e.g. `?values=<@{1,2,3}`
not	negates another operator, see below

To negate any operator, prefix it with `not` like `?a=not.eq.2`.

For more complicated filters (such as those involving disjunctions) you will have to create a new view in the database, or use a stored procedure. For instance, here's a view to show "today's stories" including possibly older pinned stories:

```
CREATE VIEW fresh_stories AS
SELECT *
  FROM stories
 WHERE pinned = true
    OR published > now() - interval '1 day'
ORDER BY pinned DESC, published DESC;
```

The view will provide a new endpoint:

```
GET /fresh_stories HTTP/1.1
```

æşĺèğč: We're working to extend the PostgREST query grammar to allow more complicated boolean logic, while continuing to prevent performance problems from arbitrary client queries.

19.2 åđĆçŻť è£Ğæżď (Columns)

When certain columns are wide (such as those holding binary data), it is more efficient for the server to withold them in a response. The client can specify which columns are required using the `select` parameter.

```
GET /people?select=fname,age HTTP/1.1
```

The default is `*`, meaning all columns. This value will become more important below in *èţĎæžŘåţŇåěŮ*.

19.2.1 èőąçőŮåĹŮ

Filters may be applied to computed columns as well as actual table/view columns, even though the computed columns will not appear in the output. For example, to search first and last names at once we can create a computed column that will not appear in the output but can be used in a filter:

```
CREATE TABLE people (
  fname text,
  lname text
);

CREATE FUNCTION full_name(people) RETURNS text AS $$
  SELECT $1.fname || ' ' || $1.lname;
$$ LANGUAGE SQL;

-- (optional) add an index to speed up anticipated query
CREATE INDEX people_full_name_idx ON people
  USING GIN (to_tsvector('english', full_name(people)));
```

A full-text search on the computed column:

```
GET /people?full_name=@@.Beckett HTTP/1.1
```

As mentioned, computed columns do not appear in the output by default. However you can include them by listing them in the vertical filtering `select` param:

```
GET /people?select=*,full_name HTTP/1.1
```

19.3 æŐŠåžŔ

The reserved word `order` reorders the response rows. It uses a comma-separated list of columns and directions:

```
GET /people?order=age.desc,height.asc HTTP/1.1
```

If no direction is specified it defaults to ascending order:

```
GET /people?order=age HTTP/1.1
```

If you care where nulls are sorted, add nullsfirst or nullslast:

```
GET /people?order=age.nullsfirst HTTP/1.1
```

```
GET /people?order=age.desc.nullslast HTTP/1.1
```

You can also use *èőąçőŮåĹŮ* to order the results, even though the computed columns will not appear in the output.

19.4 Limit åŠŇåĹĘéąţ

PostgREST uses HTTP range headers to describe the size of results. Every response contains the current range and, if requested, the total number of results:

```
HTTP/1.1 200 OK
Range-Unit: items
Content-Range: 0-14/*
```

Here items zero through fourteen are returned. This information is available in every response and can help you render pagination controls on the client. This is an RFC7233-compliant solution that keeps the response JSON cleaner.

There are two ways to apply a limit and offset rows: through request headers or query params. When using headers you specify the range of rows desired. This request gets the first twenty people.

```
GET /people HTTP/1.1
Range-Unit: items
Range: 0-19
```

Note that the server may respond with fewer if unable to meet your request:

```
HTTP/1.1 200 OK
Range-Unit: items
Content-Range: 0-17/*
```

You may also request open-ended ranges for an offset with no limit, e.g. `Range:  10-`.

The other way to request a limit or offset is with query parameters. For example

```
GET /people?limit=15&offset=30 HTTP/1.1
```

This method is also useful for embedded resources, which we will cover in another section. The server always responds with range headers even if you use query parameters to limit the query.

In order to obtain the total size of the table or view (such as when rendering the last page link in a pagination control), specify your preference in a request header:

```
GET /bigtable HTTP/1.1
Range-Unit: items
Range: 0-24
Prefer: count=exact
```

Note that the larger the table the slower this query runs in the database. The server will respond with the selected range and total

```
HTTP/1.1 206 Partial Content
Range-Unit: items
Content-Range: 0-24/3573458
```

19.5 çŻÿåžŤæăijåijŔ

PostgREST uses proper HTTP content negotiation (RFC7231) to deliver the desired representation of a resource. That is to say the same API endpoint can respond in different formats like JSON or CSV depending on the client request.

Use the Accept request header to specify the acceptable format (or formats) for the response:

```
GET /people HTTP/1.1
Accept: application/json
```

The current possibilities are

- */*
- text/csv
- application/json
- application/openapi+json
- application/octet-stream

The server will default to JSON for API endpoints and OpenAPI on the root.

19.6 å■ŢæŢřæĹŰåď■æŢř

By default PostgREST returns all JSON results in an array, even when there is only one item. For example, requesting `/items?id=eq.1` returns

```
[
  { "id": 1 }
]
```

This can be inconvenient for client code. To return the first result as an object unenclosed by an array, specify `vnd.pgrst.object` as part of the `Accept` header

```
GET /items?id=eq.1 HTTP/1.1
Accept: application/vnd.pgrst.object+json
```

This returns

```
{ "id": 1 }
```

When a singular response is requested but no entries are found, the server responds with an empty body and 404 status code rather than the usual empty array and 200 status.

æşĺèğč: Many APIs distinguish plural and singular resources using a special nested URL convention e.g. */stories* vs */stories/1*. Why do we use */stories?id=eq.1*? The answer is because a singlular resource is (for us) a row determined by a primary key, and primary keys can be compound (meaning defined across more than one column). The more familiar nested urls consider only a degenerate case of simple and overwhelmingly numeric primary keys. These so-called artificial keys are often introduced automatically by Object Relational Mapping libraries.

Admittedly PostgREST could detect when there is an equality condition holding on all columns constituting the primary key and automatically convert to singular. However this could lead to a surprising change of format that breaks unwary client code just by filtering on an extra column. Instead we allow manually specifying singular vs plural to decouple that choice from the URL format.

19.7 äžŇè£ŻåĹűè¿ŞåĞž

If you want to return raw binary data from a `bytea` column, you must specify `application/octet-stream` as part of the `Accept` header and select a single column `?select=bin_data`.

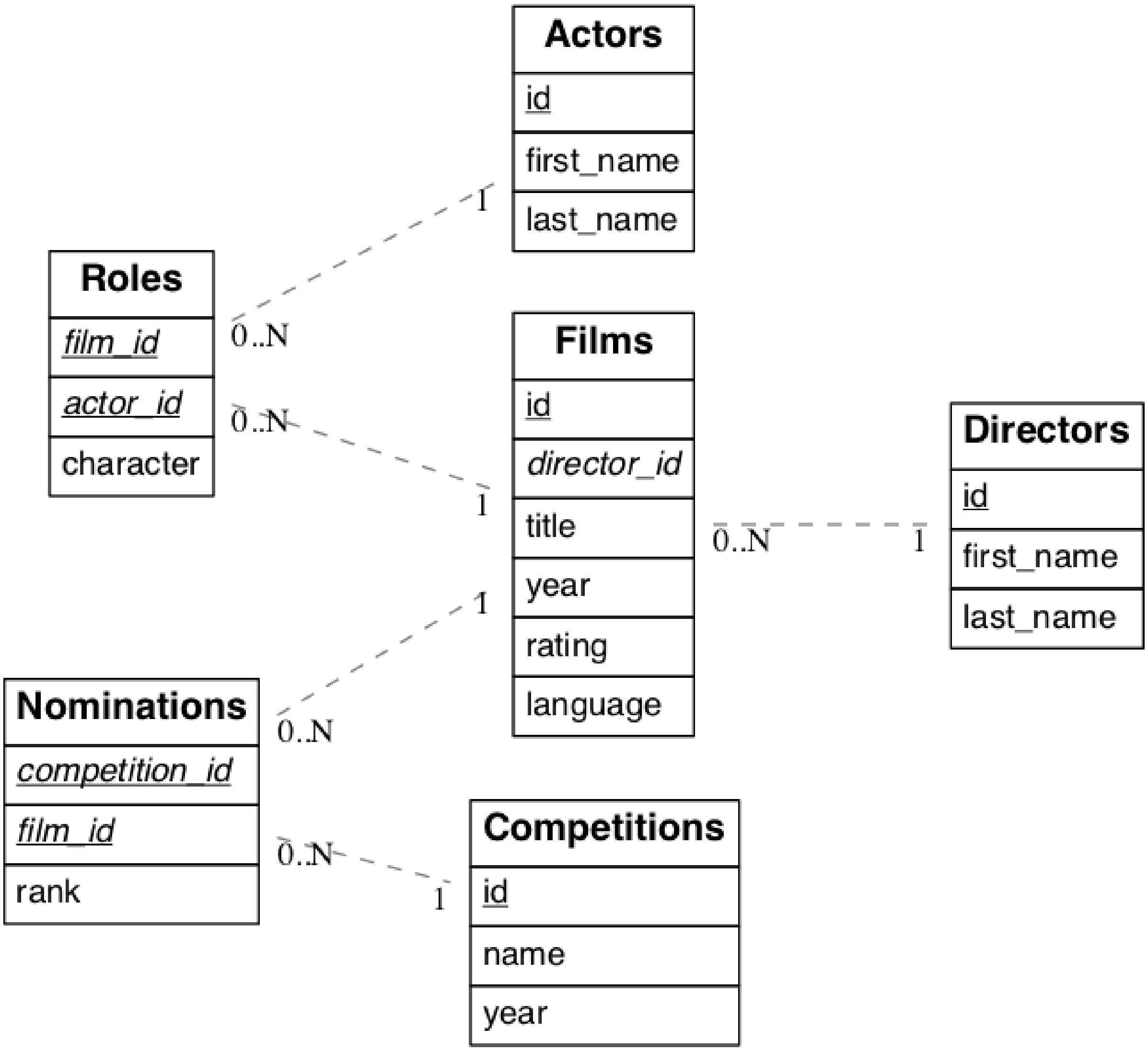
```
GET /items?select=bin_data&id=eq.1 HTTP/1.1
Accept: application/octet-stream
```

æşĺèğč: If more than one row would be returned the binary results will be concatenated with no delimiter.

CHAPTER 20

èţĎæžŘåţŇåěŮ

In addition to providing RESTful routes for each table and view, PostgREST allows related resources to be included together in a single API call. This reduces the need for multiple API requests. The server uses foreign keys to determine which tables and views can be returned together. For example, consider a database of films and their awards:



As seen above in *åđĆçŻť'è£Ğæżď' (Columns)* we can request the titles of all films like this:

```
GET /films?select=title HTTP/1.1
```

This might return something like

```
[
  { "title": "Workers Leaving The LumiÃÍre Factory In Lyon" },
  { "title": "The Dickson Experimental Sound Film" },
  { "title": "The Haunted Castle" }
]
```

However because a foreign key constraint exists between Films and Directors, we can request this information be included:

```
GET /films?select=title,directors(last_name) HTTP/1.1
```

Which would return

```
[
  { "title": "Workers Leaving The LumiÃÍre Factory In Lyon",
    "directors": {
      "last_name": "LumiÃÍre"
    }
  },
  { "title": "The Dickson Experimental Sound Film",
    "directors": {
      "last_name": "Dickson"
    }
  },
  { "title": "The Haunted Castle",
    "directors": {
      "last_name": "MÃl'liÃÍs"
    }
  }
]
```

æşÍèğč: As of PostgREST v4.1, parens `()` are used rather than brackets `{}` for the list of embedded columns. Brackets are still supported, but are deprecated and will be removed in v5.

PostgREST can also detect relations going through join tables. Thus you can request the Actors for Films (which in this case finds the information through Roles). You can also reverse the direction of inclusion, asking for all Directors with each including the list of their Films:

```
GET /directors?select=films(title,year) HTTP/1.1
```

æşÍèğč: Whenever foreign key relations change in the database schema you must refresh PostgREST's schema cache to allow resource embedding to work properly. See the section *Schema éĞ■è¡¡*.

20.1 åţŇåĚěè£Ğæżďå ŠŇæŐŠåžŔ

Embedded tables can be filtered and ordered similarly to their top-level counterparts. To do so, prefix the query parameters with the name of the embedded table. For instance, to order the actors in each film:

```
GET /films?select=*,actors(*)&actors.order=last_name,first_name HTTP/1.
↪1
```

This sorts the list of actors in each film but does *not* change the order of the films themselves. To filter the roles returned with each film:

```
GET /films?select=*,roles(*)&roles.character=in.Chico,Harpo,Groucho␣
↪HTTP/1.1
```

Once again, this restricts the roles included to certain characters but does not filter the films in any way. Films without any of those characters would be included along with empty character lists.

CHAPTER 21

èĞłåőŽäźĽæ§ěèŕć

The PostgREST URL grammar limits the kinds of queries clients can perform. It prevents arbitrary, potentially poorly constructed and slow client queries. It’s good for quality of service, but means database administrators must create custom views and stored procedures to provide richer endpoints. The most common causes for custom endpoints are

- Table unions and OR-conditions in the where clause
- More complicated joins than those provided by *èţĎœžŘåţŇåěŮ*
- Geospatial queries that require an argument, like "points near (lat,lon)"
- More sophisticated full-text search than a simple use of the `@@` filter

CHAPTER 22

å■ŸåĆÍè£ĞçÍŃ

Every stored procedure in the API-exposed database schema is accessible under the /rpc prefix. The API endpoint supports only POST which executes the function.

```
POST /rpc/function_name HTTP/1.1
```

Such functions can perform any operations allowed by PostgreSQL (read data, modify data, and even DDL operations). However procedures in PostgreSQL marked with stable or immutable volatility can only read, not modify, the database and PostgREST executes them in a read-only transaction compatible for read-replicas.

Procedures must be used with named arguments. To supply arguments in an API call, include a JSON object in the request payload and each key/value of the object will become an argument.

For instance, assume we have created this function in the database.

```
CREATE FUNCTION add_them(a integer, b integer)
RETURNS integer AS $$
 SELECT $1 + $2;
$$ LANGUAGE SQL IMMUTABLE STRICT;
```

The client can call it by posting an object like

```
POST /rpc/add_them HTTP/1.1

{ "a": 1, "b": 2 }
```

The keys of the object match the parameter names. Note that PostgreSQL converts parameter names to lowercase unless you quote them like CREATE FUNCTION foo("mixedCase" text) .... You can also call a function that takes a single parameter of type json by sending the header `Prefer: params=single-object` with your request. That way the JSON request body will be used as the single argument.

æşĺèğč: We recommend using function arguments of type json to accept arrays from the client. To pass a PostgreSQL native array you'll need to quote it as a string:

```
POST /rpc/native_array_func HTTP/1.1

{ "arg": "{1,2,3}" }
```

```
POST /rpc/json_array_func HTTP/1.1

{ "arg": [1,2,3] }
```

PostgreSQL has four procedural languages that are part of the core distribution: PL/pgSQL, PL/Tcl, PL/Perl, and PL/Python. There are many other procedural languages distributed as additional extensions. Also, plain SQL can be used to write functions (as shown in the example above).

By default, a function is executed with the privileges of the user who calls it. This means that the user has to have all permissions to do the operations the procedure performs. Another option is to define the function with with the `SECURITY DEFINER` option. Then only one permission check will take place, the permission to call the function, and the operations in the function will have the authority of the user who owns the function itself. See PostgreSQL documentation for more details.

æşĺèğč: Why the */rpc* prefix? One reason is to avoid name collisions between views and procedures. It also helps emphasize to API consumers that these functions are not normal restful things. The functions can have arbitrary and surprising behavior, not the standard "post creates a resource" thing that users expect from the other routes.

We are considering allowing GET requests for functions that are marked non-volatile. Allowing GET is important for HTTP caching. However we still must decide how to pass function parameters since request bodies are not allowed. Also some query string arguments are already reserved for shaping/filtering the output.

22.1 èŐůåŔŰèŕůæśĆçŽĎ Headers/Cookies

Stored procedures can access request headers and cookies by reading GUC variables set by PostgREST per request. They are named `request.header.XYZ` and `request.cookie.`

XYZ. For example, to read the value of the Origin request header:

```
SELECT current_setting('request.header.origin', true);
```

22.2 åď■æİĆåÿČåřŤéĂżè¿Ś

For complex boolean logic you can use stored procedures, an example:

```
CREATE FUNCTION key_customers(country TEXT, company TEXT, salary␣
↪FLOAT) RETURNS SETOF customers AS $$
  SELECT * FROM customers WHERE (country = $1 AND company = $2) OR␣
↪salary = $3;
$$ LANGUAGE SQL;
```

Then you can query by doing:

```
POST /rpc/key_customers HTTP/1.1

{ "country": "Germany", "company": "Volkswagen", salary": 120000.00 }
```

22.3 æŁŻåĞžéŤŹèŕŕ

Stored procedures can return non-200 HTTP status codes by raising SQL exceptions. For instance, here's a saucy function that always errors:

```
CREATE OR REPLACE FUNCTION just_fail() RETURNS void
  LANGUAGE plpgsql
  AS $$
BEGIN
  RAISE EXCEPTION 'I refuse!'
    USING DETAIL = 'Pretty simple',
          HINT = 'There is nothing you can do.';
END
$$;
```

Calling the function returns HTTP 400 with the body

```
{
  "message":"I refuse!",
  "details":"Pretty simple",
  "hint":"There is nothing you can do.",
  "code":"P0001"
}
```

You can customize the HTTP status code by raising particular exceptions according to the PostgREST *error to status code mapping*. For example, `RAISE insufficient_privilege` will respond with HTTP 401/403 as appropriate.

CHAPTER 23

æŔŠåĚě/ä£őæŤź

All tables and auto-updatable views can be modified through the API, subject to permissions of the requester's database role.

To create a row in a database table post a JSON object whose keys are the names of the columns you would like to create. Missing properties will be set to default values when applicable.

```
POST /table_name HTTP/1.1

{ "col1": "value1", "col2": "value2" }
```

The response will include a `Location` header describing where to find the new object. If the table is write-only then constructing the Location header will cause a permissions error. To successfully insert an item to a write-only table you will need to suppress the Location response header by including the request header `Prefer:  return=minimal`.

On the other end of the spectrum you can get the full created object back in the response to your request by including the header `Prefer:  return=representation`. That way you won't have to make another HTTP call to discover properties that may have been filled in on the server side. You can also apply the standard *åđĆçŻt'è£Ğæżd' (Columns)* to these results.

æşĺèğč: When inserting a row you must post a JSON object, not quoted JSON.

```
Yes
{ "a": 1, "b": 2 }

```

(äÿŃéąţçżğçż■)

(çż■äÿŁéąţ)

```
No
"{ \"a\": 1, \"b\": 2 }"
```

Some javascript libraries will post the data incorrectly if you're not careful. For best results try one of the *åőćœĹůçńŕåžŞ* built for PostgREST.

To update a row or rows in a table, use the PATCH verb. Use *æřť'åźşè£Ğæżď' (Rows)* to specify which record(s) to update. Here is an exmaple query setting the `category` column to child for all people below a certain age.

```
PATCH /people?age=lt.13 HTTP/1.1

{ "category": "child" }
```

Updates also support `Prefer: return=representation` plus *åđĆçŻť'è£Ğæżď' (Columns)*.

æşÍèğč: Beware of accidentally updating every row in a table. To learn to prevent that see *éŸżæ■ćåĔÍèąĺæŞ■ä¡IJ*.

23.1 æĽźéĞŔæŔŠåĔě

Bulk insert works exactly like single row insert except that you provide either a JSON array of objects having uniform keys, or lines in CSV format. This not only minimizes the HTTP requests required but uses a single INSERT statement on the backend for efficiency. Note that using CSV requires less parsing on the server and is much faster.

To bulk insert CSV simply post to a table route with `Content-Type: text/csv` and include the names of the columns as the first row. For instance

```
POST /people HTTP/1.1
Content-Type: text/csv

name,age,height
J Doe,62,70
Jonas,10,55
```

An empty field (`,,`) is coerced to an empty string and the reserved word `NULL` is mapped to the SQL null value. Note that there should be no spaces between the column names and commas.

To bulk insert JSON post an array of objects having all-matching keys

```
POST /people HTTP/1.1
Content-Type: application/json

[
  { "name": "J Doe", "age": 62, "height": 70 },
  { "name": "Janus", "age": 10, "height": 55 }
]
```

CHAPTER 24

åĹăéŹď

To delete rows in a table, use the DELETE verb plus *æřť’åźşè£Ğæżď’ (Rows)*. For instance deleting inactive users:

```
DELETE /user?active=is.false HTTP/1.1
```

æşĺèğč: Beware of accidentally deleting all rows in a table. To learn to prevent that see *éŸżæ■ćåĚĺèąĺæŞ■ä¡IJ*.

CHAPTER 25

OpenAPI æŤŕæŇĄ

Every API hosted by PostgREST automatically serves a full OpenAPI description on the root path. This provides a list of all endpoints, along with supported HTTP verbs and example payloads.

You can use a tool like Swagger UI to create beautiful documentation from the description and to host an interactive web-based dashboard. The dashboard allows developers to make requests against a live PostgREST server, provides guidance with request headers and example request bodies.

æşĺèğč: The OpenAPI information can go out of date as the schema changes under a running server. To learn how to refresh the cache see *Schema éĞ■è¡¡*.

CHAPTER 26

HTTP çŁűæĂĄçăĄ

PostgREST translates PostgreSQL error codes into HTTP status as follows:

PostgreSQL error code(s)	HTTP status	Error description
08*	503	pg connection err
09*	500	triggered action exception
0L*	403	invalid grantor
0P*	403	invalid role specification
23503	409	foreign key violation
23505	409	uniqueness violation
25*	500	invalid transaction state
28*	403	invalid auth specification
2D*	500	invalid transaction termination
38*	500	external routine exception
39*	500	external routine invocation
3B*	500	savepoint exception
40*	500	transaction rollback
53*	503	insufficient resources
54*	413	too complex
55*	500	obj not in prereq state
57*	500	operator intervention
58*	500	system error
F0*	500	conf file error
HV*	500	foreign data wrapper error
P0001	400	default code for "raise"
P0*	500	PL/pgSQL error
XX*	500	internal error
42883	404	undefined function
42P01	404	undefined table
42501	if authed 403, else 401	insufficient privileges
other	500	

CHAPTER 27

èğŠèĽšçşżçż§æęĆè£ř

PostgRESTæŮĺåIJĺåřĘæŢřæ■őåžŞä£İæŇĄåIJĺAPIåőĽåĔĺæĂğçŽĎäÿ■å£ČãĂĆ æĽĂæIJĽæŐĹæİČéČ¡éĂŽè£ĞæŢřæ■őåžŞèğŠèĽšåŠŇæİČéŹŘè£ŻèąŇãĂĆ PostgRESTçŽĎåůěä¡IJæŸŕ**éłŇèŕĄ**èŕůæśĆ - å■şéłŇèŕĄåőćæĹůçńŕæŸŕåŘęæŸŕäżŰäżňæĽĂèŕť çŽĎ - çĎűåŘŐèőľæŢřæ■őåžŞ**æŐĹæİČ**åőćæĹůçńŕæŞ■ä¡IJãĂĆ

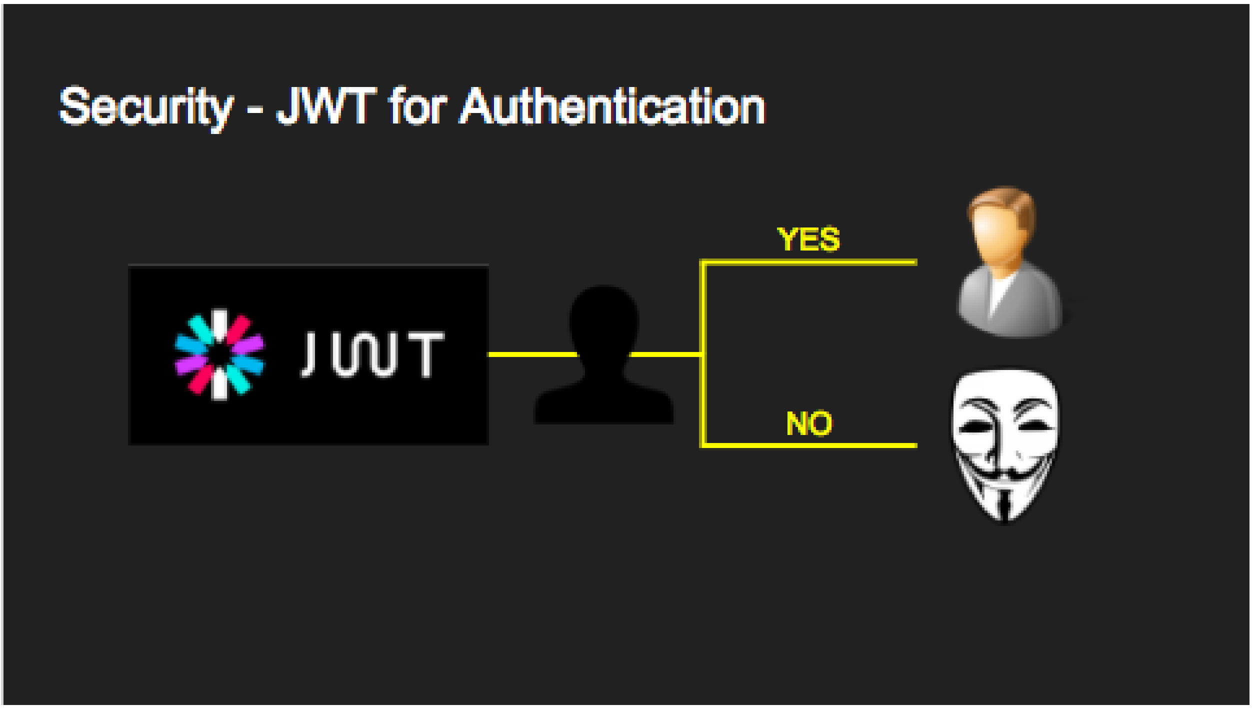
27.1 éłŇèŕĄåžŔåĹŮ

the **authenticator**, **anonymous** and **user** roles. PostgRESTä¡£çŤĺäÿĽçğ■çśżåđŃçŽĎèğŠèĽšïijŇ**èžńäż¡éłŇèŕĄåŹĺ**ïijŇ**åŇ£åŘ■**åŠŇ**çŤĺæĹů**èğŠèĽšãĂĆ æŢřæ■őåžŞçőąçŘĘåŚŸåĹŻåżžè£ŹäžŻèğŠèĽšåźűéĚ■ç¡őPostgRESTäżěä¡£çŤĺåőČäżňãĂĆ

Security - Roles for Authorization

Anonymous
Authenticator
User(s)

The authenticator åžŤèŕěåĹŻåżžèžńäż¡éłŇèŕĄåŹĺïijŽäżčçăĄïijŽ'NOINHERIT'åźűåIJĺæŢřæ■őåžŞäÿ■éĚ■ç¡
åőČæŸŕäÿĂäÿłåŔŸèĽšé¿ŹïijŇåĚűåůěä¡IJæŸŕâĂIJæĹŘäÿžâĂİåĚűäżŰçŤĺæĹůæİěäÿžçżŔè£Ğèžńäż¡éłŇèŕĄçŽĎ
äÿŃåŻ¿æŸ¿çď'žäžĘæIJ■åŁąåŹĺåęĆä¡Ţåď'ĎçŔĘèžńäż¡éłŇèŕĄãĂĆ åęĆæđIJau-
thæĹŘåŁ§ïijŇåőČåřĘåĹĞæ■ćåĹřèŕůæśĆæŇĞåőŽçŽĎçŤĺæĹůèğŠèĽšïijŇåŘęåĹŹåřĘåĹĞæ■ćåĹřåŇ£åŘ■èğŠèĽ



Security - JWT for Authentication

Here are the technical details. We use JSON Web Tokens to authenticate API requests.æĆĺåŔŕèČ¡è£Ÿèőřå¿ŮJWTåŇĚåŘńåŁăåŕĘç■¿åŘ■åčřæŸŐçŽĎåĹŮèąĺãĂĆ æĽĂæIJĽçťćèţŤéČ¡æŸŕåĚĄèőÿçŽĎïijŇä¡ĘPostgRESTçĽźåĹńåĚşæşĺäÿĂäÿłåŘ■äÿžèğŠèĽšroleçŽĎåčřæŸŐãĂ

```
{
  "role": "user123"
}
```

å¡ŞèŕůæśĆåŇĚåŘńåĚůæIJĽèğŠèĽšåčřæŸŐçŽĎæIJĽæŢĹJWTæŮűïijŇPostgRESTåřĘåIJĺHTTPèŕůæś

```
SET LOCAL ROLE user123;
```

èŕůæşĺæĎŔïijŇéĂŽè£ĞåĚĹåĽ■çŽĎæŞ■ä¡IJïijŇæŢřæ■őåžŞçőąçŘĘåŚŸå£ĚéążåčřæŸŐèŕÿèžńäż¡élŇèŕĄèĂĚè

```
GRANT user123 TO authenticator;
```

åęĆæđIJåőćæĹůçńŕäÿ■åŇĚåŘńJWTïijĹæĹŰæšąæIJĽèğŠèĽšåčřæŸŐçŽĎJWTïijĽïijŇåĹŹPostgRESTåřĘå
æŢřæ■őåžŞçőąçŘĘåŚŸå£Ěéążæ■čçąőèő¿ç¡őåŇ£åŘ■èğŠèĽšæİČéŹŘïijŇäżĕéŸšæ■ćåŇ£åŘ■çŤĺæĹůæ§ěçIJŃæĹ

27.2 çŤĺæĹůåŠŇçżĎ

PostgreSQLä¡£çŤĺèğŠèĽšçŽĎæęĆå£ţçőąçŘĘæŢřæ■őåžŞèő£éŮőæİČéŹŘãĂĆ
åŔŕäżĕåřĘèğŠèĽšroleèğĘäÿžæŢřæ■őåžŞçŤĺæĹůæĹŰäÿĂçżĎæŢřæ■őåžŞçŤĺæĹůïijŇåĚůä¡ŞåŔŰåĘşäžŐèğŠèĽš

27.2.1 æŕŔäÿł Web çŤĺæĹůçŽĎèğŠèĽš

PostgRESTåŔŕäżěéĂĆåžŤäżżä¡ŢäÿĂäÿłèğĆçĆźãĂĆ åęĆæđIJæĆĺåřĘèğŠèĽšèğĘäÿžå■ŢäÿłçŤĺæĹůïijŇéĆč
å¡ŞçżŔè£Ğèžńäż¡élŇèŕĄçŽĎçŤĺæĹůåŔŚåĞžèŕůæś ĆæŮűïijŇPostgRESTåřĘåĹĞæ■ćåĹřèŕěçŤĺæĹůçŽĎèğŠèĽšï

æĆĺåŔŕäżĕä¡£çŤĺèąŇçžğåőĽåĚĺæĂğçĄţæť żåIJřéŹŘåĹűå¡ŞåĽ■çŤĺæĹůçŽĎåŔŕèğĄæĂğåŠŇèő£éŮőæİČéŹ
äżĕäÿŃæŸŕæİĕèĞłTomas Vondraç ŽĎçďžä¿Ń<http://blog.2ndquadrant.com/application-users-vs-row-level-security/>‘_ïijŇè£ŹæŸŕäÿĂäÿłå■ŸåĆĺçŤĺæĹůäźŃéŮť åŔŚéĂĄçŽĎæűĹæĄŕçŽĎ
çŤĺæĹůåŔŕäżĕåIJĺåĚűäÿ■æŔŠåĚěèąŇäżĕåŘŚåĚűäżŰçŤĺæĹůåŔŚéĂĄæűĹæĄŕïijŇåźűæ§ěèŕćåőČäżĕæ§ěçIJŃåĚ

```
CREATE TABLE chat (
  message_uuid    UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
  message_time    TIMESTAMP NOT NULL DEFAULT now(),
  message_from    NAME      NOT NULL DEFAULT current_user,
  message_to      NAME      NOT NULL,
  message_subject VARCHAR(64) NOT NULL,
  message_body    TEXT
);
```

æĹŚäżňåÿŇæIJŻåőđæŰ¡äÿĂéąźæŤ£ç■ŰïijŇçąőä£İçŤĺæĹůåŔłèČ¡çIJŃåĹřäżŰåŔŚéĂĄæĹŰæĽŞçőŮåŔŚçż
æ■ď åď ŰïijŇæĹŚäżňè£ŸåÿŇæIJŻéŸżæ■ćçŤĺæĹůä¡£çŤĺåĚűäżŰäžžçŽĎåğŞåŘ■äijłéĂămessage_fromåĹŮãĂĆ

PostgreSQLïijĹ9.5åŔŁæŽť éńŸçĽĹæIJňïijĽåĚĄèőÿæĹŚäżňä¡£çŤĺèąŇçžğåőĽåĚĺæĂğèő¿ç¡őæ■ď ç■ŰçŢĕïi

```
CREATE POLICY chat_policy ON chat
  USING ((message_to = current_user) OR (message_from = current_user))
  WITH CHECK (message_from = current_user)
```

èő£éŮőçŤ§æĹŘçŽĎèĄŁåď'l'èąÍAPIçńŕçĆźçŽĎäżżä¡Ţäžžé Č¡åřĘçIJŃåĹřäżŰäžňåžŤèŕěåĞĘçąőçŽĎèąŇïijŇ

27.2.2 Web çŤÍæĹůåĚśäžńèğŠèĽ'š

æĹŰèĂĔïijŇæŢřæ■őåžŞèğŠèĽ'šåŔŕäżĕäżčèąÍçżĎèĂŇäÿ■æŸŕäÿłåĹńçŤÍæĹůïijĹæĹŰäÿłåĹńéŹď'åď'ŰïijĽ'ã
æĆÍåŔŕäżĕéĂĽ'æŃl'WebåžŤçŤÍçÍŃåžŘçŽĎæĽ'ĂæIJĽ'åůšçŹżå¡ŢçŤÍæĹůåĚśäžńåŘŇäÿĂäÿłwebuserèğŠèĽ'šãĂĆ
æĆÍåŔŕäżĕéĂŽè£ĞåIJÍJWTäÿ■åŇĚåŘńéćİåď'ŰåčřæŸŐæİěçŤĎåĹń/æŐŠéŹď'åĚůä¡Şæ§Řäÿłç ŤÍæĹůïijŇä¿Nåę

```
{
  "role": "webuser",
  "email": "john@doe.com"
}
```

SQLäżčçăĄåŔŕäżĕéĂŽè£ĞPostgRESTæŇĽ'èŕůæśĆèő¿ç¡őçŽĎGUCåŔŸéĞŔèőáéŮőåčřæŸŐãĂĆ
ä¿Nåę ĆïijŇèęĄèŐůåŔŰçŤţå■ŘéĆőäżűåčřæŸŐïijŇèŕůèřČçŤÍæ■ď'åĞ¡æŢřïijŽ

```
current_setting('request.jwt.claim.email', true)
```

This allows JWT generation services to include extra information and your database code to react to it. For instance the RLS example could be modified to use this current_setting rather than current_user. The second 'true' argument tells current_setting to return NULL if the setting is missing from the current configuration.

27.2.3 æůůåŘĹçŤÍæĹůçżĎèğŠèĽ'š

æŃěæIJĽ'èőÿåď'ŽæŢřæ■őåžŞèğŠèĽ'šæšąæIJĽ'æĂğèČ¡æ■§åď'śïijŇåř¡çőąèğŠèĽ'šæŸŕæŇĽ'ç¿ď'éŻĘåŚ¡åŘ■è
åęĆæđIJéIJĂèęĄïijŇæĆÍåŔŕäżĕèĞłçŤśäÿžWebåžŤçŤÍçÍŃåžŘäÿ■çŽĎæŕŔäÿłçŤÍæĹůåĹĘéĚ■æŰřèğŠèĽ'šãĂĆ
æĆÍåŔŕäżĕæůůåŘĹçżĎåŠŇå■Ţäÿłèğ ŠèĽ'šç■ŰçŢěãĂĆ ä¿NåęĆïijŇæĹŚäžňäż■çĎűåŔŕäżĕæŃěæIJĽ'äÿĂäÿłwebus

```
CREATE ROLE webuser NOLOGIN;
-- grant this role access to certain tables etc

CREATE ROLE user000 NOLOGIN;
GRANT webuser TO user000;
-- now user000 can do whatever webuser can

GRANT user000 TO authenticator;
-- allow authenticator to switch into user000 role
-- (the role itself has nologin)
```

27.3 èĞłåőŽäźĽ’éłŇèŕĄ

PostgRESTéĂŽè£ĞäżčçăĄïijŽ‘exp‘åčřæŸŐäżď’çĽ’ŇåĹřæIJ§ïijŇæŃŠçżİè£ĞæIJ§çŽĎäżď’çĽ’ŇãĂĆ ä¡ĘæŸŕïijŇåőČäÿ■äijŽåijžåĹűæĽ’ğèąŇäżżä¡ŢéćİåďŰçŽĎçžęæİ§ãĂĆ éćİåďŰçžęæİ§çŽĎäÿĂäÿłçď’žä¿ŃæŸŕçńŃå■şæŠď’éŤĂåŕźçĽ’źåőŽçŤĺæĹůçŽĎèő£éŮőãĂĆ éĚ■ç¡őæŰĞäżűåŔĆæŢřïijŽcodeïijŽ‘pre-request‘æŇĞåőŽåIJĺéłŇèŕĄèĂĚåĹĞæ■ćåĹřæŰřèğŠèĽ’šäźŃåŘŐåŠŇäÿ

è£ŹæŸŕäÿĂäÿłä¿Ńå■ŘãĂĆ åIJĺéĚ■ç¡őæŰĞäżűäÿ■æŇĞåőŽå■ŸåĆĺè£ĞçĺŃïijŽ

```
pre-request = "public.check_user"
```

åIJĺèŕěåĞ¡æŢřäÿ■ïijŇæĆĺåŔŕäżěè£ŘèąŇäżżæĎŔäżčçăĄæİěæčĂæ§ěèŕůæśĆïijŇåźűæăźæ■őéIJĂèęĄåijŢåŔŚ

```
CREATE OR REPLACE FUNCTION check_user() RETURNS void
  LANGUAGE plpgsql
  AS $$
BEGIN
  IF current_user = 'evil_user' THEN
    RAISE EXCEPTION 'No, you are evil'
      USING HINT = 'Stop being so evil and maybe you can log in';
  END IF;
END
$$;
```

CHAPTER 28

åőćæĹůçńŕ Auth

èęĄè£ŻèąŇçżŔè£Ğèžńäż¡éłŇèŕĄçŽĎèŕůæśĆïijŇåőćæĹůçńŕå£Ěéążå ŇĚåŘńïijŽcodeïijŽ*Authorization* HTTPæăĞåďť ïijŇåĚűåĂijäÿžïijŽcodeïijŽ*Bearer <jwt>*ãĂĆ ä¿ŃåęĆïijŽ

```
GET /foo HTTP/1.1
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
↪eyJyb2xlIjoiamRvZSIsImV4cCI6MTQ3NTUxNjI1MH0.
↪GYDZV3yM0gqvuEtJmfpplLBXSGYnke_Pvnl0tbKAjB4
```

28.1 JWT Generation

æĆĺåŔŕäżĕäżŐæŢřæ■őåžŞåĘĚéČĺæĹŰéĂŽè£Ğåď ŰéČĺæĲ■åŁąåĹŻåżžæĲĽæŢĹçŽĎJWTãĂĆ æŕŔäÿłäżď çĽŇéČ¡ä¡£çŤĺçğŸåŕĘåŕĘçăĄåŁäåŕĘç■¿åŘ■ - ç■¿åŘ■èĂĚåŠŇéłŇèŕĄèĂĚåĚśäžńçğŸåŕĘãĂĆ åŽäæ■dďïijŇäÿŐPostgRESTæĲ■åŁąåŹĺåĚśäžńåŕĘçăĄçŽĎäżżä¡ŢæĲ■åŁąéČ¡åŔŕäżĕåĹŻåżžæĲĽæŢĹçŽĎJWTã ïijĹPostgRESTçŻőåĽ■äżĚæŤŕæŇĄHMAC-SHA256ç■¿åŘ■çőŮæşŢãĂĆïijĽ

28.1.1 JWT from SQL

æĆĺåŔŕäżĕä¡£çŤĺ‘pgjwt extension <https://github.com/michelp/pgjwt>‘_åIJĺSQLäÿ■åĹŻåżžJWTäżď çĽŇãĂĆ åőČä¿ĹçőÄå■ŢïijŇåŔłéIJĂèęĄpgcryptoãĂĆ åęĆæđIJæĆĺä¡£çŤĺçŽĎæŸŕäÿ■æŤŕæŇĄåőĽèčĚæŰřæĽľåśŢçŽĎAmazon RDSç■Ľ çŐŕåćČïijŇæĆĺäż■çĎűåŔŕäżĕåIJĺpgjwtäÿ■æĽŃåŁĺè£ŘèąŇSQLïijŇäżŐèĂŇåĹŻåżžæĆĺéIJĂèęĄçŽĎåŁ§èĆ æŐěäÿŃæİěçijŰåĘŹäÿĂäÿłè£ŤåŽđäżďçĽŇçŽĎå■ŸåĆĺè£ĞçĺŃãĂĆ äÿŃéİćçŽĎäÿĂäÿłè£ŤåŽđäÿĂäÿłåÿęæIJĽçąňçijŰçăĄèğŠèĽšçŽĎäżďçĽŇïijŇèŕěèğŠèĽšåIJĺåŔŚåÿČåŘŐäžŤåĹ èŕůæşĺæĎŔïijŇæ■ďåĞ¡æŢřäź§æIJĽäÿĂäÿłçąňçijŰçăĄçŽĎåŕĘçăĄãĂĆ

```
CREATE TYPE jwt_token AS (
  token text
);

CREATE FUNCTION jwt_test() RETURNS public.jwt_token
    LANGUAGE sql
    AS $$
  SELECT sign(
    row_to_json(r), 'mysecret'
  ) AS token
  FROM (
    SELECT
      'my_role'::text as role,
      extract(epoch from now())::integer + 300 AS exp
  ) r;
$$;
```

PostgRESTéĂŽè£Ğåŕź'/rpc/jwt_test'è£ŻèąŇPOSTèŕůæśĆïijŇæİěåŘŚåőćæĹůçńŕæŽťéIJšæ■ďåĞ¡æŢřïijĹåĆ

æşĺèğč: èęĄéĄ£åĚ■åŕźå■ŸåĆĺè£ĞçĺŃäÿ■çŽĎåŕĘéŠěè£ŻèąŇçąňçijŰçăĄïijŇèŕůåřĘåĚűåŘęå■ŸäÿžæŢřæ■őåžŞ

```
-- run this once
ALTER DATABASE mydb SET "app.jwt_secret" TO '!!secret!!';

-- then all functions can refer to app.jwt_secret
SELECT sign(
  row_to_json(r), current_setting('app.jwt_secret')
) AS token
FROM ...
```

28.1.2 JWT from Auth0

åČŔAuth0 <https://auth0.com/>'_è£ŹæăůçŽĎåďŰéČĺæIJ■åŁąåŔŕäżěåřĘOAuthäżŐGithubïijŇTwitterïijŇG Auth0è£ŸåŔŕäżěåďĎçŘĘçŤţå■ŘéĆőäżűæşĺåĘŇåŠŇåŕĘçăĄéĞ■ç¡őæţĄçĺŃãĂĆ

èęĄä¡£çŤĺAuth0ïijŇèŕůåřĘåĚűåőćæĹůçńŕåŕĘéŠěåď■åĹűåĹřPostgRESTéĚ■ç¡őæŰĞäżűäÿ■ïijŇåęĆäÿŃæĽ *secret*ãĂĆ ïijĹæŮğåijŔçŽĎAuth0çğŸåŕĘæŸŕBase64çijŰçăĄçŽĎãĂĆåŕźäžŐè£ŹäžŻçğŸåŕĘèő¿ç¡őïijŽäżčçăĄïij *is-base64* toïijŽcodeïijŽ*true*ïijŇæĹŰèĂĚåŔłåĹůæŰřAuth0çğŸåŕĘãĂĆïijĽä¡ăåŔŕäżěåIJĺåőćæĹůçńŕèő¿ç¡őäÿ■æĽ Auth0çőąçŘĘæŐğåĹűåŔřãĂĆ

æĹŚäżňçŽĎäżčçăĄéIJĂèęĄJWTäÿ■çŽĎæŢřæ■őåžŞèğŠèĽšãĂĆèęĄæůżåŁäåőČïijŇæĆléIJĂèęĄåřĘæŢřæ■
metadata <https://auth0.com/docs/rules/metadata-in-rules>'_äÿ■ãĂĆçĎűåŘŐïijŇæĆlåřĘéIJĂèęĄçijŰåĘŹäÿĂäÿ
param <https://auth0.com/docs/libraries/lock/v10/sending-authentication-parameters#scope-string->äÿ■åŇĚåŘńïijŽcodeïijŽ'role'åčřæŸŐãĂĆ

```
// Example Auth0 rule
function (user, context, callback) {
  user.app_metadata = user.app_metadata || {};
  user.role = user.app_metadata.role;
  callback(null, user, context);
}
```

```
// Example using Auth0Lock with role claim in scope
new Auth0Lock ( AUTH0_CLIENTID, AUTH0_DOMAIN, {
  container: 'lock-container',
  auth: {
    params: { scope: 'openid role' },
    redirectUrl: FQDN + '/login', // Replace with your redirect url
    responseType: 'token'
  }
})
```

28.1.3 JWT åőĽåĚĺ

åŕźäžŐä¡£çŤĺJWTïijŇèĞşåřŚæIJĽäÿĽçğ■åÿÿèğĄçŽĎæĽźèŕĎïijŽ1ïijĽéŠĹåŕźæăĞåĞĘæIJňèžńïijŇ2ïijĽåŔ

åĚşäžŐâĂIJJWTæăĞåĞĘ<https://tools.ietf.org/html/rfc7519>'_çŽĎæĽźèŕĎåIJĺç¡ŠäÿŁåĚűäżŰåIJřæŰźèŕę
//paragonie.com/blog/2017/03/jwt- JSON-WebçŽĎæăĞèőř - æŸŕ
- åİŔæăĞåĞĘæŸŕïijŇæŕŔäÿłäžžïijŇåžŤèŕěèğĎéĄ£>'_ãĂĆ Post-
gRESTæIJĂçŻÿåĚşçŽĎéČĺåĹĘæŸŕæĽĂèřŞçŽĎïijŽäżčçăĄïijŽ*alg* =
none'éŮőéćŸãĂĆäÿĂäžŻåőđçŐřJWTçŽĎæIJ■åŁąåŹĺåĚĄèőÿåőćæĹůçńŕéĂĽæŃľçŤĺäžŐç■¿ç¡šJWTçŽĎçőŮæşŢ
= *none*ãĂĆ

åŕźJWTåžŞçŽĎæĽźèŕĎäżĚéĂŽè£ĞåőČä¡£çŤĺçŽĎåžŞäÿŐPostgRESTçŻÿåĚşãĂĆåęĆäÿŁæĽĂè£řïijŇäÿ■å
//auth0.com/blog/critical-vulnerabilities-in-json-web-token-libraries/>'_äÿ■æĽ¿åĹřæŻť åď Žä£ąæĄŕãĂĆæIJĽåĚ
<https://jwt.io/>'_ãĂĆ

æIJĂåŘŐäÿĂçğ■æĽźèŕĎçŽĎéĞ■çĆźæŸŕæżěçŤĺJWTæİěçżť æŁď ç¡ŠçżIJäijŽèŕİãĂĆå§žæIJňåžžèőőæŸŕâĂ
//cryto.net/~joepie91/blog/2016/06/13/stop-using-jwt-for-sessions/>'_åŽäåÿžåď ğåď ŽæŢřïijĹåęĆæđIJäÿ■æŸŕåĚ
ïijŇå¡Şä¡ăåĄŽçŽĎæŮűåĂŹåĞžçŐřçŽĎéŮőéćŸçŽĎèğčåĘşæŰźæąĹïijŇ''äÿ■åůĕä¡IJ<http://cryto.net/~joepie91/blog/2016/06/19/stop-using-jwt-for-sessions-part-2-why - æĆĺçŽĎæžűæűš-
çŁŕèğĎåůĕä¡IJ/>'_ãĂĆéŞ¿æŐěçŽĎæŰĞçńăæůśåĚĕèőĺèőžäžĘè£ŹäžŻéŮőéćŸïijŇä¡ĘéŮőéćŸçŽĎåőđèťĺæŸŕJW

PostgRESTäÿżèęĄä¡£çŤĺJWTè£ŻèąŇèžńäż¡éłŇèŕĄåŠŇæŐĹæİČïijŇåźűéijŞåŁśçŤĺæĹůäź§è£Źæăůåžã
.. _ssl:

28.2 SSL

PostgRESTæŮĺåIJĺåĄŽåě¡äÿĂäżűäžŃïijŽäÿžPostgreSQLæŢřæ■őåžŞæůżåŁăHTTPæŐěåŔčãĂĆ
äÿžäžĘä£İæŇĄäżčçăĄåřŔèĂŇéŻĘäÿ■ïijŇæĹŚäżňäÿ■åőđçŐřSSLãĂĆ
ä¡£çŤĺåČŔNGINXè£ŹæăůçŽĎåŔ■åŘŚäżčçŘĘæİĕæůżåŁăåőČïijŇâĂIJè£ŹéĞŇæŸŕåęĆä¡Ţ<https://nginx.org/en/docs/http/configuring_https_servers.html>'_ãĂĆ èŕůæşĺæĎŔïijŇåČŔHerokuè£ŹæăůçŽĎæ§ŘäžŻ

CHAPTER 29

æđűæđĎéŽŤçęż

PostgRESTåőđä¿ŃéĚ■ç¡őäÿžåĔňåijĂæIJ■åŁąåŹĺéĚ■ç¡őæŰĞäżűäÿ■æŇĞåőŽçŽĎå■ŢäÿłæĺąåijŔçŽĎæĽĂa
è£ŹæĎŔåŚşçİĂçğĄæIJĽæŢřæ■őæĹŰåőđçŐřçżĘèŁĆåŔŕäżĕè£ŻåĔěçğĄæIJĽæĺąåijŔïijŇåźűäÿŤåŕźHTTPåőćæĹ
çĎűåŘŐïijŇæĆĺåŔŕäżĕåĔňåijĂèğĘåŻ¿åŠŇå■ŸåĆĺè£ĞçĺŃïijŇäżŐèĂŇåřĘåĘĔéČĺçżĘèŁĆäÿŐåď’ŰéČĺäÿŰçŢŇé
åőČä¡£äżčçăĄæŻt’åőźæŸŞéĞ■æđĎïijŇåźűæŔŘä¿ŻäžĘäÿĂçğ■èĞłçĎűçŽĎæŰźåijŔæİĕè£ŻèąŇAPIçĽĹæIJňæŐ
æIJĽåĔşä¡£çŤĺåĔňåĔśèğĘåŻ¿åŇĔèčĔçğĄæIJĽèąĺçŽĎçď’žä¿ŃïijŇèŕůåŔĆéŸĔäÿŃéİćçŽĎïijŽrefïijŽ‘public_ui‘é

CHAPTER 30

SQL çŤĺæĹůçőąçŘĘ

30.1 å■ŸåĆĺçŤĺæĹůåŠŇåŕĘçăĄ

åęĆäÿŁæĽĂè£řïijŇåď ŰéČĺæIJ■åŁąåŔŕäżĕæŔŘä¿ŻçŤĺæĹůçőąçŘĘåźűä¡£çŤĺJWTäÿŐPostgRESTæIJ■åŁąå
äź§åŔŕäżĕåőŇåĚĺéĂŽè£ĞSQLæŤŕæŇĄçŹżå¡ŢãĂĆ è£ŹæŸŕäÿĂéąźçŻÿå¡Şåď ŽçŽĎåůĕä¡IJïijŇæĽĂäżĕåĞĘåď Ğ

äÿŃèąĺïijŇåĞ¡æŢřåŠŇèğęåŔŚåŹĺåřĘå■ŸåIJĺäžŐïijŽcodeïijŽ‘basic_auth‘æĺąåijŔäÿ■ïijŇæĆĺäÿ■åžŤåIJĺAPIå
åĔňåĔśèğĘåŻ¿åŠŇåĞ¡æŢřåřĘå■ŸåIJĺäžŐäÿ■åŘŇçŽĎæĺąåijŔäÿ■ïijŇèŕĕæĺąåijŔåIJĺåĘĔéČĺåijŢçŤĺæ■ď åĘĔéČĺå

éęŰåĚĹïijŇæĹŚäżňéIJĂèęĄäÿĂäÿłèąĺæİĕèů§èÿłæĹŚäżňçŽĎçŤĺæĹůïijŽ

```
-- æĹŚäżňåřĘåĘĔåőźç¡őäžŐbasic_authæĺąåijŔäÿ■ïijŇ
-- äżĕåřĘåĔűéŽŘèŮŔåIJĺåĔňåĔśèğĘåŻ¿äÿ■ãĂĆ
-- æ§ŘäžŻåĔňåĔśè£ĞçĺŃ/èğĘåŻ¿åřĘåijŢçŤĺåĘĔéČĺçŽĎåÿőåŁl’çĺŃåžŔåŠŇèąĺãĂĆ
create schema if not exists basic_auth;

create table if not exists
basic_auth.users (
  email    text primary key check ( email ~* '^.+@.+\..+$' ),
  pass     text not null check (length(pass) < 512),
  role     name not null check (length(role) < 512)
);
```

æĹŚäżňåÿŇæIJŻèŕĕèğŠèĽšroleæŸŕåőđéŹĚæŢřæ■őåžŞèğŠèĽšçŽĎåď ŰéŤőïijŇä¡ĘæŸŕPostgreSQLäÿ■æŤŕ
æĹŚäżňåřĘä¡£çŤĺèğęåŔŚåŹĺæĽŃåŁĺåijžåĹűæĽğèąŇåőČãĂĆ

```
create or replace function
basic_auth.check_role_exists() returns trigger
  language plpgsql
  as $$
begin
  if not exists (select 1 from pg_roles as r where r.rolname = new.
↪role) then
    raise foreign_key_violation using message =
      'unknown database role: ' || new.role;
    return null;
  end if;
  return new;
end
$$;

drop trigger if exists ensure_user_role_exists on basic_auth.users;
create constraint trigger ensure_user_role_exists
  after insert or update on basic_auth.users
  for each row
  execute procedure basic_auth.check_role_exists();
```

æŐěäÿŃæİěïijŇæĹŚäżňåřĘä¡£çŤĺpgcryptoæĽľåśŢåŠŇèğęåŔŚåŹĺæİěä£İåŕĘåŕĘçăĄïijŽcodeïijŽ‘users‘èąĺã

```
create extension if not exists pgcrypto;

create or replace function
basic_auth.encrypt_pass() returns trigger
  language plpgsql
  as $$
begin
  if tg_op = 'INSERT' or new.pass <> old.pass then
    new.pass = crypt(new.pass, gen_salt('bf'));
  end if;
  return new;
end
$$;

drop trigger if exists encrypt_pass on basic_auth.users;
create trigger encrypt_pass
  before insert or update on basic_auth.users
  for each row
  execute procedure basic_auth.encrypt_pass();
```

ä¡£çŤĺèŕěèąĺïijŇæĹŚäżňåŔŕäżěåÿőåŁľæčĂæ§ěåŁäåŕĘåĹŮçŽĎåŕĘçăĄãĂĆ åęĆæđIJçŤţå■ŘéĆőäżűåŠŇåŕĘçăĄæ■čçąőïijŇåőČåřĘè£ŤåŻđçŤĺæĹůçŽĎæŢřæ■őåžŞèğŠèĽšãĂĆ

```
create or replace function
basic_auth.user_role(email text, pass text) returns name
  language plpgsql
  as $$
begin
  return (
  select role from basic_auth.users
   where users.email = user_role.email
     and users.pass = crypt(user_role.pass, users.pass)
  );
end;
$$;
```

30.2 Public çŤÍæĹůçŢŇéİć

åIJÍäÿŁäÿĂèŁĆäÿ■ïijŇæĹŚäżňåĹŻåżžäžĘäÿĂäÿłçŤÍäžŐå■ŸåĆÍçŤÍæĹůä£ąæĄŕçŽĎåĘĚéČÍèąĺãĂĆ
åIJÍè£ŹéĞŇïijŇæĹŚäżňåĹŻåżžäÿĂäÿłçŹżå¡ŢåĞ¡æŢřïijŇåőČæŐěåŔŮäÿĂäÿłçŤţå■ŘéĆőäżűåIJřåİĂåŠŇåŕĘçăĄï

30.2.1 çŹżå¡Ţ

åęĆ‘‘JWT from SQL‘_äÿ■æĽĂè£řïijŇæĹŚäżňåřĘåIJÍçŹżå¡ŢåĞ¡æŢřäÿ■åĹŻåżžäÿĂäÿłJWTãĂĆ
èŕůæşĺæĎŔïijŇæĆĺéIJĂèęĄåřĘæ■ďçďžä¿Ńäÿ■çąňçijŰçăĄçŽĎåŕĘéŠěèřČæŢť äÿžæĆĺéĂĽæŃľçŽĎåőĽåĚÍåŕĘé

```
create or replace function
login(email text, pass text) returns basic_auth.jwt_token
  language plpgsql
  as $$
declare
  _role name;
  result basic_auth.jwt_token;
begin
  -- check email and password
  select basic_auth.user_role(email, pass) into _role;
  if _role is null then
    raise invalid_password using message = 'invalid user or password';
  end if;

  select sign(
      row_to_json(r), 'mysecret'
    ) as token
    from (
      select _role as role, login.email as email,
```

(äÿŃéąţçżğçż■)

(çż■äÿŁéąţ)

```
        extract(epoch from now())::integer + 60*60 as exp
    ) r
    into result;
  return result;
end;
$$;
```

èřČçŤÍæ■ď åĞ¡æŢřçŽĎAPIèŕůæśĆåęĆäÿŃæĽĂçď žïijŽ

```
POST /rpc/login HTTP/1.1

{ "email": "foo@bar.com", "pass": "foobar" }
```

åŞ■åžŤçIJŃèţůæİěåČŔäÿŃéİćçŽĎäżčçăĄæőţãĂĆ åřİèŕŢåIJÍ‘jwt.io <https://jwt.io/>‘_èğčçăĄäżď çĽŇãĂĆ ïijĹåőČçŽĎçijŰçăĄåÿęæIJĽäżěäÿŃçğŸåŕĘïijŽäżčçăĄïijŽ*mysecret*ïijŇåęĆäÿŁéİćçŽ

```
{
  "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
↪eyJlbWFpbCI6ImZvb0BiYXIuY29tIiwicm9sZSI6ImF1dGhvciJ9.fpf3_
↪ERi5qbWOE5NPzvauJgvulm0zkIG9xSm2w5zmdw"
}
```

30.2.2 æİČéŹŘ

æĆĺçŽĎæŢřæ■őåžŞèğŠèĽšéIJĂèęĄèő£éŮőæĺąåijŔïijŇèąĺïijŇèğĘåŻ¿åŠŇåĞ¡æŢřæĽ■èČ¡äÿžHTTPèŕůæśĆ åŻđæČşäÿĂäÿŃâĂIJèğŠèĽšçşżçż§æęĆè£řâĂİ_ïijŇPostgRESTä¡£çŤĺçĽźæőŁèğŠèĽšæİěåď ĎçŘĘèŕůæśĆïijŇå■ äżěäÿŃæŸŕåĚĄèőÿåŇ£åŘ■çŤĺæĹůåĹŻåżžåÿŘæĹůåźűåřİèŕŢçŹżå¡ŢçŽĎæİČéŹŘçď žä¿ŃãĂĆ

```
-- åŘ■çğřâĂIJanonâĂİåŠŇâĂIJauthenticatorâĂİæŸŕåŔŕéĚ■ç¡őçŽĎ
--␣
↪èĂŇäÿ■æŸŕåĔşéŤőèŕ■ïijŇæĹŚäżňåŔłæŸŕäÿžäžĘæÿĚæŹřèţůèğĄèĂŇéĂL’æŃl’åőČäżň
create role anon;
create role authenticator noinherit;
grant anon to authenticator;

grant usage on schema public, basic_auth to anon;
grant select on table pg_authid, basic_auth.users to anon;
grant execute on function login(text,text) to anon;
```

æĆĺåŔŕèČ¡äijŽæŃĚå£ČïijŇåŇ£åŘ■çŤĺæĹůåŔŕäżěäżŐïijŽcodeïijŽ‘basic_auth.users‘èąĺäÿ■èŕżåŔŰæĽĂæIJ ä¡ĘæŸŕïijŇæ■ď èąĺäÿ■éĂĆçŤĺäžŐçŻť æŐěæ§ěèŕćïijŇåŻăäÿžåőČä¡■äžŐå■ŢçŃňçŽĎæđűæđĎäÿ■ãĂĆ åŇ£åŘ■èğŠèĽšéIJĂèęĄèő£éŮőïijŇåŻăäÿžpublicïijŽcodeïijŽ‘users‘èğĘåŻ¿ä¡£çŤĺèřČçŤĺçŤĺæĹůçŽĎæİČéŹŘèŕż ä¡ĘæĹŚäżňåůšçąőä£İèğĘåŻ¿æ■čçąőéŹŘåĹűåŕźæŢŔæĎ§ä£ąæĄŕçŽĎèő£éŮőãĂĆ