EDB Test Plan

David A. Ventimiglia

<2022-06-03 Fri>

**Table of Contents**

[1. Databases](#org5df5767)

[1.1. Postgres](#orgdac9ccf)

[1.2. Schema](#org52bd035)

[1.2.1. DONE Table Basics](#org6e10b94)

[1.2.2. DONE Table Relationships](#org307aee3)

[1.2.3. TODO Remote Relationships [DOES\_NOT\_INVOLVE\_DB]](#org5de81f7)

[1.2.4. DONE Extend with Views](#orga8805e9)

[1.2.5. DONE Extend with SQL Functions](#orgd44d416)

[1.2.6. DONE Default field values](#org9cea581)

[1.2.7. DONE Enum type fields](#org35cabfb)

[1.2.8. DONE Computed fields](#org02bce19)

[1.2.9. DONE Customize auto-generated fields [DOES\_NOT\_INVOLVE\_DB]](#orgeb0ea6d)

[1.2.10. DONE Data validations](#org6f774b2)

[1.2.11. DONE Using an existing database](#org812f473)

[1.2.12. DONE Relay Schema [DOES\_NOT\_INVOLVE\_DB]](#org41c2c30)

[1.3. Queries](#orgf3bba01)

[1.3.1. DONE Simple object queries](#org1f1d2bb)

[1.3.2. DONE Nested object queries](#org81405ea)

[1.3.3. DONE Aggregation queries](#org76c96be)

[1.3.4. DONE Filter query results / search queries](#org5821787)

[1.3.5. DONE Sort query results](#org4f5e08f)

[1.3.6. DONE Distinct query results](#org8395ef1)

[1.3.7. TODO Using multiple arguments [DOES\_NOT\_INVOLVE\_DB]](#org242ea9c)

[1.3.8. TODO Multiple queries in a request [DOES\_NOT\_INVOLVE\_DB]](#org16c454a)

[1.3.9. TODO Using variables / aliases / fragments / directives [DOES\_NOT\_INVOLVE\_DB]](#org04dd683)

[1.3.10. TODO Query performance](#org19b0763)

[1.4. Mutations](#org597c3cd)

[1.4.1. DONE Insert](#org4570d12)

[1.4.2. TODO Upsert](#orgbccf1bf)

[1.4.3. DONE Update](#org9f9f119)

[1.4.4. TODO Delete](#org60c6e04)

[1.4.5. TODO Multiple mutations in a request](#orge6eb832)

[1.5. Subscriptions](#org6422fd0)

[1.5.1. DONE Live queries](#org66f2294)

[1.6. Supported Postgres types](#orgd0df329)

[1.6.1. DONE Perform inserts on the misc table.](#orgfc60ea0)

[2. Remote Schema](#org244276d)

[2.1. Remote relationships](#org3945fad)

[2.1.1. TODO To remote database](#org12254b1)

[2.1.2. TODO To Remote Schema [DOES\_NOT\_INVOLVE\_DB]](#org30c7cbb)

[2.2. Authorization in remote schema](#orgfb8d33e)

[2.2.1. TODO Forwarding auth context to/from remote schema [DOES\_NOT\_INVOLVE\_DB]](#orgd9db446)

[2.2.2. TODO Remote schema permissions [DOES\_NOT\_INVOLVE\_DB]](#org8cc36a0)

[2.2.3. TODO Bypassing Hasura's auth for remote schema [DOES\_NOT\_INVOLVE\_DB]](#orge2c6141)

[3. Event Triggers](#org788ac13)

[3.1. Creating an Event Trigger](#org8e1f079)

[3.1.1. TODO Create an insert trigger](#org27163a1)

[3.1.2. TODO Create an update trigger](#org1ac3640)

[4. Scheduled Triggers](#orgba8c78a)

[4.1. Creating a chron trigger](#org2399d45)

[4.1.1. TODO Create a chron trigger](#org5d7d405)

[4.2. Creating a one-off scheduled event](#orgbb32098)

[4.2.1. TODO Create a one-off scheduled event](#org83d1d00)

[4.3. Cleaning up scheduled triggers data](#org8e246e0)

[4.3.1. TODO Clear Everything](#org7f10eec)

[5. Test Matrix](#org6a56093)

[5.1. NOTES](#org9d51148)

# Databases

## Postgres

## Schema

### DONE Table Basics

* [ ] Add database
* [ ] Add account and product tables
* [ ] Add account and product data
* [ ] Perform CRUD operations
  + Read

query MyQuery {

account(order\_by: {name: asc}, limit: 10) {

id

name

created\_at

updated\_at

}

}

query MyQuery {

product(order\_by: {price: asc}, limit: 10) {

id

name

price

updated\_at

created\_at

}

}

* + Insert

mutation MyMutation {

insert\_account(objects: {name: "John Doe"}) {

affected\_rows

}

}

mutation MyMutation {

insert\_product(objects: {name: "Doughnut", price: 100}) {

returning {

id

name

price

updated\_at

created\_at

}

}

}

* + Update

mutation MyMutation {

update\_account(where: {name: {\_eq: "John Doe"}}, \_set: {name: "Jane Doe"}) {

affected\_rows

}

}

* + Delete

mutation MyMutation {

delete\_product(where: {name: {\_eq: "Doughnut"}}) {

affected\_rows

}

}

### DONE Table Relationships

* [ ] Add the order and order detail tables
* [ ] Add relationships for account, order, order detail, and product
* [ ] Generate order and order detail data
* [ ] Perform queries across relationships

query MyQuery {

account(limit: 2) {

id

name

created\_at

updated\_at

orders {

id

created\_at

updated\_at

order\_details {

id

created\_at

updated\_at

units

product {

id

name

created\_at

updated\_at

price

}

}

}

}

}

### TODO Remote Relationships [DOES\_NOT\_INVOLVE\_DB]

### DONE Extend with Views

* [ ] Add accountsummary view and relationships
* [ ] Query across table and view relationships

query MyQuery {

account\_summary(limit: 10) {

id

sum

account {

name

}

}

}

### DONE Extend with SQL Functions

* [ ] Add search functions
* [ ] Query search functions

query MyQuery {

product\_search(args: {search: "apple"}) {

name

price

}

}

query MyQuery {

product\_fuzzy\_search(args: {search: "apple"}) {

name

price

}

}

### DONE Default field values

### DONE Enum type fields

* [ ] Create a native Postgres enum type for order status.
* [ ] Create a enum table for region and track it as order salesregion.

### DONE Computed fields

* [ ] Add productsku function and track it as a computed field
* [ ] Query product table with computed field

query {

product(limit: 10) {

id

name

price

sku

}

}

### DONE Customize auto-generated fields [DOES\_NOT\_INVOLVE\_DB]

* [ ] Change order.status to order.state for the GraphQL field name

### DONE Data validations

* [ ] Add nonnegativeprice check constraint
* [ ] Attempt mutations with and without negative prices

mutation MyMutation {

update\_product(where: {name: {\_eq: "Chilli Paste, Sambal Oelek"}}, \_set: {price: 10}) {

affected\_rows

}

}

mutation MyMutation {

update\_product(where: {name: {\_eq: "Pastry - Raisin Muffin - Mini"}}, \_set: {price: -10}) {

affected\_rows

}

}

### DONE Using an existing database

### DONE Relay Schema [DOES\_NOT\_INVOLVE\_DB]

* [ ] Turn on the Relay API in the Console

query MyQuery {

account\_connection(first: 10) {

edges {

node {

name

orders {

id

sales\_region {

description

}

order\_details {

units

product {

name

price

sku

}

}

}

}

cursor

}

}

}

## Queries

### DONE Simple object queries

### DONE Nested object queries

### DONE Aggregation queries

query MyQuery {

account\_aggregate {

aggregate {

count

}

}

}

query MyQuery {

account(limit: 10) {

orders {

order\_details\_aggregate {

aggregate {

sum {

units

}

}

}

}

}

}

### DONE Filter query results / search queries

### DONE Sort query results

### DONE Distinct query results

### TODO Using multiple arguments [DOES\_NOT\_INVOLVE\_DB]

### TODO Multiple queries in a request [DOES\_NOT\_INVOLVE\_DB]

### TODO Using variables / aliases / fragments / directives [DOES\_NOT\_INVOLVE\_DB]

### TODO Query performance

## Mutations

### DONE Insert

### TODO Upsert

### DONE Update

### TODO Delete

### TODO Multiple mutations in a request

## Subscriptions

### DONE Live queries

## Supported Postgres types

### DONE Perform inserts on the misc table.

mutation {

insert\_misc(objects: [

{

bigint\_field: 1

bigserial\_field: 1

boolean\_field: true

box\_field: "((0,0),(1,1))"

bytea\_field: "\\xDEADBEEF"

character\_field: "foo"

character\_varying\_field: "bar"

cidr\_field: "192.168.100.128/25"

circle\_field: "0,0,1"

date\_field: "2022-01-01"

double\_precision\_field: 9673143120,

inet\_field: "192.168.0.1/24"

integer\_field: 1

interval\_field: "'1 month ago'"

json\_field: {}

jsonb\_field: {}

line\_field: "0,0,1,1"

lseg\_field: "0,0,1,1"

macaddr\_field: "08:00:2b:01:02:03"

macaddr8\_field: "08:00:2b:01:02:03:04:05"

money\_field: 52093.89

numeric\_field: 10

path\_field: "0,0,1,1,2,2,3,3,3,0,2,0,0,0"

pg\_lsn\_field: "FFFFFFFF/FFFFFFFF"

point\_field: "0,0"

polygon\_field: "0,0,1,0,1,1,0,1"

real\_field: 3.14159

serial\_field: 1

smallint\_field: 1

smallserial\_field: 1

text\_field: "abc"

time\_with\_time\_zone\_field: "04:05:06 PST"

time\_without\_time\_zone\_field: "04:05:06"

timestamp\_with\_time\_zone\_field: "2022-01-01 04:05:06 PST"

timestamp\_without\_time\_zone\_field: "2022-01-01 04:05:06"

txid\_snapshot\_field: "566:566:"

uuid\_field: "61a41be6-4eb4-45a5-bfb5-b68c20e9ccde"

xml\_field: "<?xml version=\"1.0\"?><book><title>Manual</title><chapter>...</chapter></book>"

}

]) {

returning {

bigint\_field

bigserial\_field

boolean\_field

box\_field

bytea\_field

character\_field

character\_varying\_field

cidr\_field

circle\_field

date\_field

double\_precision\_field

inet\_field

integer\_field

interval\_field

json\_field

jsonb\_field

line\_field

lseg\_field

macaddr\_field

macaddr8\_field

money\_field

numeric\_field

path\_field

pg\_lsn\_field

point\_field

polygon\_field

real\_field

serial\_field

smallint\_field

smallserial\_field

text\_field

time\_with\_time\_zone\_field

time\_without\_time\_zone\_field

timestamp\_with\_time\_zone\_field

timestamp\_without\_time\_zone\_field

txid\_snapshot\_field

uuid\_field

xml\_field

}

}

}

# Remote Schema

## Remote relationships

### TODO To remote database

### TODO To Remote Schema [DOES\_NOT\_INVOLVE\_DB]

## Authorization in remote schema

### TODO Forwarding auth context to/from remote schema [DOES\_NOT\_INVOLVE\_DB]

### TODO Remote schema permissions [DOES\_NOT\_INVOLVE\_DB]

### TODO Bypassing Hasura's auth for remote schema [DOES\_NOT\_INVOLVE\_DB]

# Event Triggers

## Creating an Event Trigger

### TODO Create an insert trigger

### TODO Create an update trigger

* [ ] Perform a mutation to update an order
* [ ] Update an order from the Console

mutation MyMutation {

update\_order\_by\_pk(pk\_columns: {id: "1564344e-e528-43de-b88e-dab9c3efa44e"}, \_set: {state: "fulfilled"}) {

id

state

}

}

* [ ] Check the events logs

# Scheduled Triggers

## Creating a chron trigger

### TODO Create a chron trigger

* [ ] Allow time to pass
* [ ] Check the events logs

## Creating a one-off scheduled event

### TODO Create a one-off scheduled event

## Cleaning up scheduled triggers data

### TODO Clear Everything

* [ ] Cron triggers

DELETE FROM hdb\_catalog.hdb\_cron\_events;

* [ ] Scheduled events

DELETE FROM hdb\_catalog.hdb\_scheduled\_events;

# Test Matrix

| Function | Test | Outcome | Comments |
| --- | --- | --- | --- |
| Remote Relationships | NO |  | Remote Schema connect to GraphQL servers |
| Add Database | YES | SUCCESS |  |
| Add tables and relationships in Console | YES | SUCCESS |  |
| Perform CRUD operations in API | YES | SUCCESS |  |
| Set up and use table relationships | YES | SUCCESS |  |
| Track views | YES | SUCCESS |  |
| Manually add relationships to views | YES | SUCCESS |  |
| Query across view/table relationships | YES | SUCCESS |  |
| Track a function as a table and use | YES | SUCCESS |  |
| Track a function as a computed field and use | YES | SUCCESS |  |
| Use defaults for field values | YES | SUCCESS |  |
| Use a native enum as a Hasura enum | YES | SUCCESS |  |
| Use a table as a Hasura enum | YES | SUCCESS |  |
| Customize field-names in API | YES | SUCCESS | Doesn't actually involve the DB |
| Data validation with a database constraint | YES | SUCCESS |  |
| Test using the Relay API | YES | SUCCESS | Doesn't actually involve the DB |
| Simple object queries | YES | SUCCESS |  |
| Nested object queries (involves JOINs) | YES | SUCCESS |  |
| Aggregation queries (count) | YES | SUCCESS | We didn't test min, max, avg, etc. |
| Filter queries (involves WHERE) | YES | SUCCESS |  |
| Sort queries (involves ORDER BY) | YES | SUCCESS |  |
| Distinct queries (involves DISTINCT) | YES | SUCCESS |  |
| Limit queries (involves LIMIT) | YES | SUCCESS |  |
| Using multiple arguments | NO |  | Doesn't actually involve the DB |
| Multiple queries in a request | NO |  | Doesn't actually involve the DB |
| Multiple variables / aliases / fragments / directives | NO |  | Doesn't actually involve the DB |
| INSERT (see "Perform CRUD operations in API" above | YES | SUCCESS |  |
| UPDATE (see "Perform CRUD operations in API" above | YES | SUCCESS |  |
| DELETE (see "Perform CRUD operations in API" above | TBD |  | We forgot to test this! |
| ON CONFLICT (an "upsert") | TBD |  | We forgot to test this! |
| Multiple mutations in a request | TBD |  | We forgot to test this! |
| Subscriptions (Live Queries) | TBD | SUCCESS |  |
| Test all Postgres/Hasura types (mutation, query) | TBD | SUCCESS |  |
| Remote Database | TBD |  | We forgot to test this! |
| Creating event triggers | YES | FAIL | Needs a Hasura fix |
| Creating a chron trigger | TBD |  | Needs a Hasura fix |
| Creating a one-off scheduled event | TBD |  | Needs a Hasura fix |
| Cleaning up scheduled trigger data | TBD |  | Needs a Hasura fix |
| CI/CD: hasura metadata (apply, clear, reload, status) | YES | SUCCESS |  |
| CI/CD: hasura migrate (apply, apply –down all, delete) | YES | SUCCESS |  |

## NOTES

1. Test
   1. do we test this (YES, NO, TBD)
2. YES
   1. we should test this (and have)
3. NO
   1. we may not need to test this
4. TBD
   1. we should test this (but have not yet, possibly because we cannot)
5. SUCCESS
   1. we tested it and it passed
6. FAIL
   1. we tested it and it did not pass
7. "We forgot to test this!"
   1. either we forgot, or we suspect we may not need to (e.g. "Remote Database")
8. "Doesn't actually involve the DB"
   1. a Hasura function which we believe shouldn't work differently on BDR, since the function doesn't actually interact with the database.
9. "Needs a Hasura fix"
   1. either we have a FAIL test or a TBD test, because of an identified gap in Hasura that needs to be fixed.