EDB Test Plan

David A. Ventimiglia

<2022-06-03 Fri>

Contents

1 Databases

1.1 Postgres

1.2 Schema

1.2.1 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
    }
  }
```

1.2.2 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
}
      }
    }
  }
}
```

1.2.3 TODO Remote RelationshipsDOES_NOT_INVOLVE_DB

1.2.4 DONE Extend with Views

☒ Add $\text{account}_{\text{summary}}$ view and relationships

☒ Query across table and view relationships

```
query MyQuery {
  account_summary(limit: 10) {
    id
    sum
    account {
      name
    }
  }
}
```

1.2.5 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
  }
}
```

1.2.6 DONE Default field values

1.2.7 DONE Enum type fields

☒ Create a native Postgres enum type for order status.

☒ Create a enum table for region and track it as order $\text{sales}_{\text{region}}$.

1.2.8 DONE Computed fields

☒ Add $\text{product}_{\text{sku}}$ function and track it as a computed field

☒ Query product table with computed field

```
query {
  product(limit: 10) {
    id
    name
    price
    sku
  }
}
```

1.2.9 DONE Customize auto-generated fields DOES_NOT_INVOLVE_DB

☒ Change order.status to order.state for the GraphQL field name

1.2.10 DONE Data validations

☒ Add $\text{non}_{\text{negativeprice}}$ check constraint

☒ Attempt mutations with and without negative prices

```
mutation MyMutation {
  update_product(where: {name: {_eq: "Chilli Paste, Sambal Oelek"}}, _set: {price
    affected_rows
  }
}

mutation MyMutation {
  update_product(where: {name: {_eq: "Pastry - Raisin Muffin - Mini"}}, _set: {pri
    affected_rows
  }
}
```

1.2.11 DONE Using an existing database

1.2.12 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
  region
  order_details {
    units
    product {
      name
      price
      sku
```

```
    }
  }
}
      }
      cursor
    }
  }
}
```

1.3 Queries

1.3.1 DONE Simple object queries

1.3.2 DONE Nested object queries

1.3.3 DONE Aggregation queries

```
query MyQuery {
  account_aggregate {
    aggregate {
      count
    }
  }
}

query MyQuery {
  account(limit: 10) {
    orders {
      order_details_aggregate {
aggregate {
  sum {
    units
  }
}
      }
    }
  }
}
```

1.3.4 DONE Filter query results / search queries

1.3.5 DONE Sort query results

1.3.6 DONE Distinct query results

1.3.7 TODO Using multiple argumentsDOES_NOT_INVOLVE_DB

1.3.8 TODO Multiple queries in a requestDOES_NOT_INVOLVE_DB

1.3.9 TODO Using variables / aliases / fragments / directives DOES_NOT_INVOLVE_DB

1.3.10 TODO Query performance

1.4 Mutations

1.4.1 DONE Insert

1.4.2 TODO Upsert

1.4.3 DONE Update

1.4.4 TODO Delete

1.4.5 TODO Multiple mutations in a request

1.5 Subscriptions

1.5.1 TODO Live queries

1.6 Supported Postgres types

1.6.1 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></
      }
    ]) {
    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
    }
  }
}
```

2 Remote Schema

2.1 Remote relationships

2.1.1 TODO To remote database

2.1.2 TODO To Remote SchemaDOES_NOT_INVOLVE_DB

2.2 Authorization in remote schema

2.2.1 TODO Forwarding auth context to/from remote schema DOES_NOT_INVOLVE_DB

2.2.2 TODO Remote schema permissionsDOES_NOT_INVOLVE_DB

2.2.3 TODO Bypassing Hasura's auth for remote schemaDOES_NOT_INVOLVE_DB

3 Event Triggers

3.1 Creating an Event Trigger

3.1.1 DONE Create an insert trigger

3.1.2 DONE 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: {
    id
    state
  }
}
```

- ☐ Check the events logs

4 Scheduled Triggers

4.1 Creating a chron trigger

4.1.1 TODO Create a chron trigger

- ☐ Allow time to pass
- ☐ Check the events logs

4.2 Creating a one-off scheduled event

4.2.1 TODO Create a one-off scheduled event

4.3 Cleaning up scheduled triggers data

4.3.1 TODO Clear Everything

- ☐ Cron triggers

  ```
  DELETE FROM hdb_catalog.hdb_cron_events;
  ```

- ☐ Scheduled events

  ```
  DELETE FROM hdb_catalog.hdb_scheduled_events;
  ```

5 Test Matrix

Function	Test	Outcome	Comments
Remote Relationships	NO		Remote Schema con
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 inv
Data validation with a database constraint	YES	SUCCESS	
Test using the Relay API	YES	SUCCESS	Doesn't actually inv
Simple object queries	YES	SUCCESS	
Nested object queries (involves JOINs)	YES	SUCCESS	
Aggregation queries (count)	YES	SUCCESS	We didn't test min,
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 inv
Multiple queries in a request	NO		Doesn't actually inv
Multiple variables / aliases / fragments / directives	NO		Doesn't actually inv
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 thi
ON CONFLICT (an "upsert")	TBD		We forgot to test thi
Multiple mutations in a request	TBD		We forgot to test thi
Subscriptions (Live Queries)	TBD	SUCCESS	
Test all Postgres/Hasura types (mutation, query)	TBD	SUCCESS	
Remote Database	TBD		We forgot to test thi
Creating event triggers	YES	FAIL	Needs a Hasura fix.
Creating a chron trigger	TBD	FAIL	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	

5.1 NOTES

Test do we test this (YES, NO, TBD)

YES we should test this (and have)

NO we may not need to test this

TBD we should test this (but have not yet, possibly because we cannot)

SUCCESS we tested it and it passed

FAIL we tested it and it did not pass

"We forgot to test this!" either we forgot, or we suspect we may not need to (e.g. "Remote Database")

"Doesn't actually involve the DB" a Hasura function which we believe shouldn't work differently on BDR, since the function doesn't actually interact with the database.

"Needs a Hasura fix" either we have a FAIL test or a TBD test, because of an identified gap in Hasura that needs to be fixed.

5.2 Additional Notes

1. Event Trigger creation fails with the following error:


```
{
    "internal": {
"statement": "\n  INSERT INTO hdb_catalog.hdb_source_catalog_version(version, upgraded
"prepared": false,
"error": {
    "exec_status": "FatalError",
    "hint": "To enable updating the table, set REPLICA IDENTITY using ALTER TABLE.",
    "message": "cannot run INSERT ON CONFLICT DO UPDATE on table \"hdb_source_catalog_
    "status_code": "55000",
    "description": null
},
"arguments": [
    "(Oid 25,Just (\"2\",Binary))"
]
    },
    "path": "$.args[0].args",
```

```
    "error": "database query error",
    "code": "unexpected"
}
```


1. Cannot alter table with volatile function. This error is the result of trying to add a UUID column to an existing table:


```
{
    "internal": {
"statement": "CREATE EXTENSION IF NOT EXISTS pgcrypto;\nalter table \"public\".\"misc\
"prepared": false,
"error": {
    "exec_status": "FatalError",
    "hint": "null.",
    "message": "ALTER TABLE ... ADD COLUMN ... (mutable) DEFAULT may not affect replic
    "status_code": "0A000",
    "description": null
},
"arguments": []
    },
    "path": "$",
    "error": "query execution failed",
    "code": "postgres-error"
}
```


Workaround is detailed in `https://www.enterprisedb.com/docs/pgd/latest/bdr/ddl/#adding-a-column`