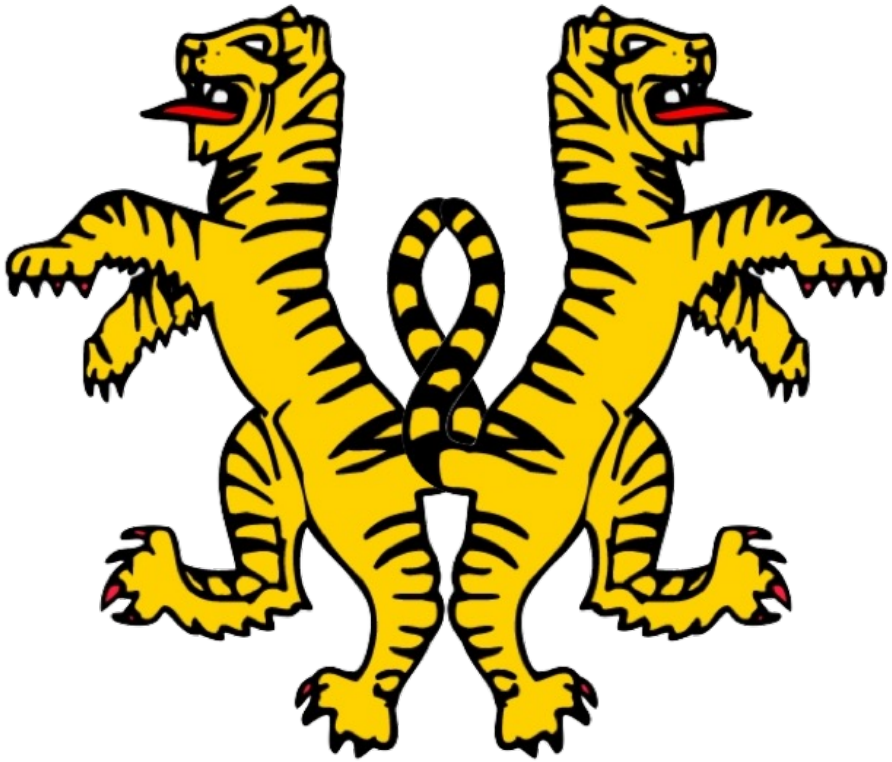
NORM_GEN:
an alternative to ORM

Hettie Dombrovskaya
DRW
In collaboration with
Boris Novikov

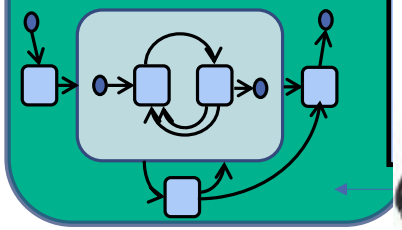
hDBn PRESENTS



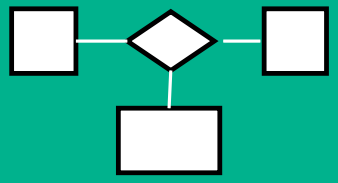
Quick Quizz

- What's ORM?
  - Object Relational Mapper
- Why developers use ORM?
  - To abstract from database specifics
- Why ORM is ~~bad~~ good?
  - Rapid development
- Why ORM **is bad**?
  - Inefficient data access patterns

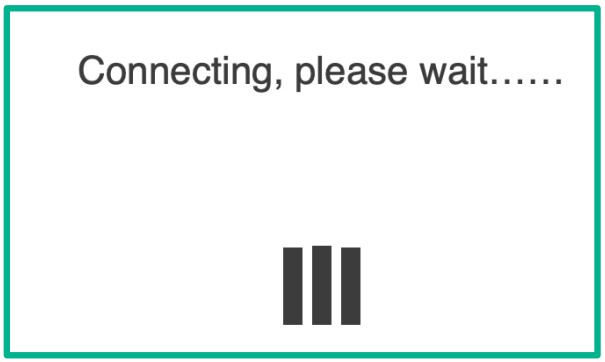
Application Model
(Object-Oriented)

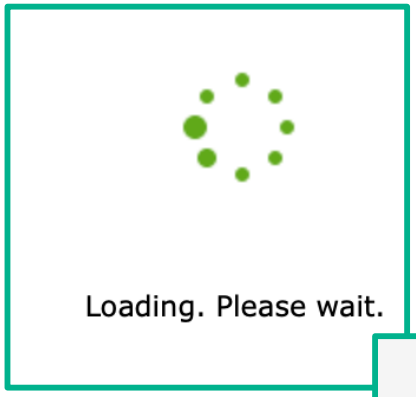


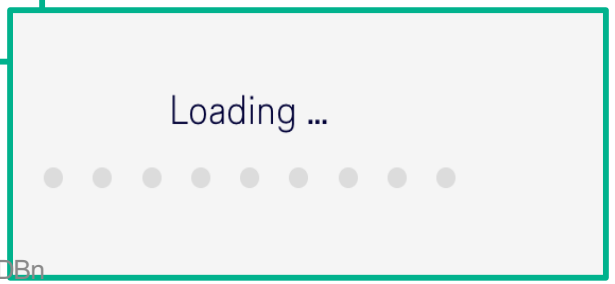
Database Model
(Object-Relational)



The Result – World-Wide Wait







Why Waiting is Bad?

Amazon: 0.1 sec increase response time - 1% sales loss.

50 % visitors abandon the site, which is not loaded within 3 sec

79% visitors will never return again

How you scale?

More hardware?...

How much RAM you may need if you need “all data in main memory? “

500 GB? 1 TB? 4 TB???

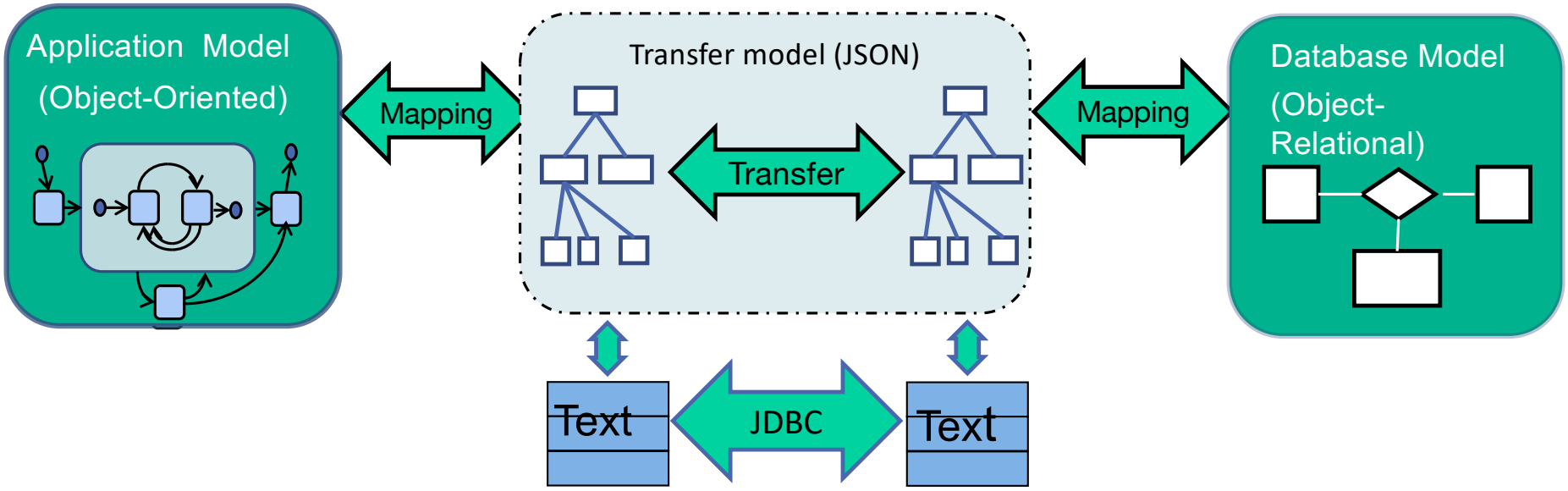
More CPU? Can’t run the sequence of calls in parallel.

What if we are in the cloud? Amazon instances and IOPS limit

4	17h47m23s	107,350,233 Details	0ms	15s457ms	0ms	d£)INSERT INTO "table_xxxxxxx_v3" ("tbl_xxxxxxx_id", "tbl_xxxxxxx_yid", "xxxxxe_id", "xxxxxd_id", "xxxxxr_id", "pssssss_bbbl_xxxxxxx_id", "pssssss_bbbl_xxxxxxx_yid", "ppppppp_xxxxxe_id", "ppppppp_xxxxxd_id", "ppppppp_xxxxxr_id", "hhhhh_ccccccc_at", "ututututut", "atata_ssssssss") VALUES (?, ? ?, ?, ?, ?, ?, ?, ?, ?, ?) ON CONFLICT ON CONSTRAINT "table_xxxxxxx_v3_pkey" DO UPDATE SET "hhhhh_ccccccc_at" = eseseses."hhhhh_ccccccc_at", "ututututut" = eseseses."ututututut "table_xxxxxxx_v3"."hhhhh_ccccccc_at" <> "eseseses"."hhhhh_ccccccc_at"; Examples User(s) involved

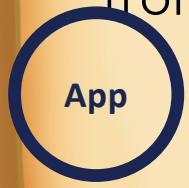
The goal of NORM is to reduce the number of round trips

NORM – No ORM

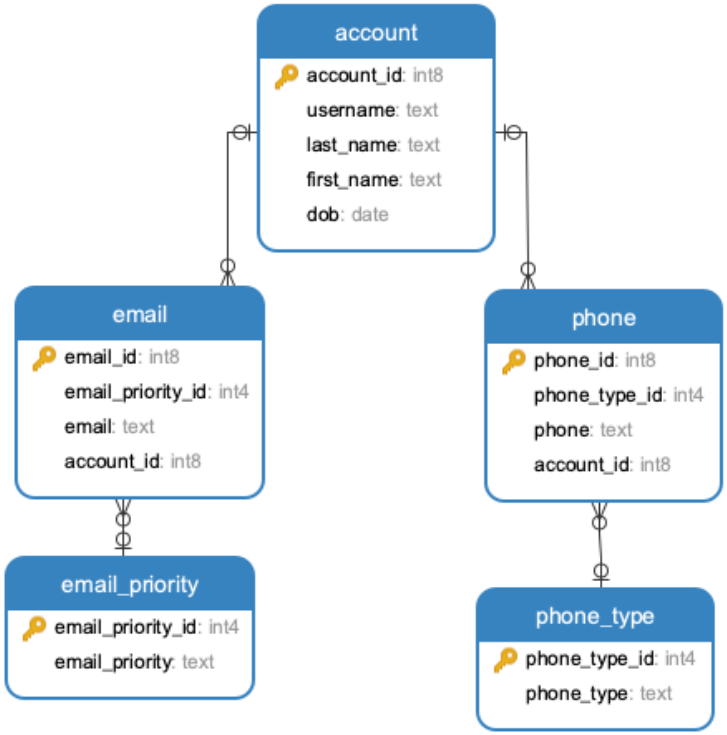

Application Model
(Object-Oriented)
Mapping
Transfer model (JSON)
Transfer
Mapping
Database Model
(Object-
Relational)
Text
JDBC
Text

CONTRACT

- Both sides (an application and a database) convert internal representations into complex hierarchical object
- Contract establishes object structure implemented on both sides
- Now, for any application endpoint it takes one database call to transfer data to and from database



DB Schema


account
account_id: int8
username: text
last_name: text
first_name: text
dob: date
email
email_id: int8
email_priority_id: int4
email: text
account_id: int8
phone
phone_id: int8
phone_type_id: int4
phone: text
account_id: int8
email_priority
email_priority_id: int4
email_priority: text
phone_type
phone_type_id: int4
phone_type: text

Transfer Object in JSON Format

```
{”account_id”: ”1”,
”username”: ”aliceacct1”,
”last_name”: ”johns”,
“first_name”:”alice”,
”dob” : ”1996-04-01” ,
”phones” : [
        { ”phone_id”: 1,
          ”phone_number” : ”2021234567” ,
          ”phone_type_id”:”2”, ”phone_type”: ”cell” }
        ],
”email_addresses”:[
     {”email_address_id”:1 ,
      ”email_address”:alicejons@gmail.com,
      “email_priority_id”: “1”,
      “email_priority”:”primary” }
      ]}
```

Functions Return UDT

```
SELECT ROW(account_id ,
            username,
            first_name,
            last_name,
            dob ,
          (SELECT array_agg(row(phone_id,
                        phone,
                        p.phone_type_id,
                        phone_type )::phone_record)
                        FROM phone p
                             JOIN phone_type pt USING (phone_type_id) WHERE p.account_id=a.account_id),
           (SELECT array_agg(row(email_id,
                        email,
                        e.email_priority_id,
                        email_priority )::email_record)
                        FROM email e
                             JOIN email_priority ep using(email_priority_id) WHERE e.account_id=a.account_id)
                         )::account_record AS single_item
           FROM account a WHERE a.account_id=p_account_id
```

Refresher

Why not storing JSON?!

- Single hierarchy
- The search is slower

Why not returning JSON?!

- By using JSON as a return type for functions we are losing the strong type dependencies

  That's why we organize functions in "packages":
  we want to ensure that all functions from one "package" return the same object

master NORM / sql / account_pkg.sql Go to file ...

 **hettie-d** account_update function and more usage examples are added Latest commit a6983ff on Apr 30, 2020 History

1 contributor

341 lines (316 sloc) | 10.5 KB Raw Blame

```
1  set search_path to norm;
2
3  drop type if exists phone_record cascade ;
4  create type phone_record as (
5                    phone_id  bigint,
6                    phone_number  text,
7                    phone_type_id int,
8                    phone_type   text);
9  drop type if exists email_record cascade;
10 create type email_record as(
11                   email_id bigint,
12                       email  text,
13                   email_priority_id int,
14                   email_priority text);
15  drop type if exists account_record cascade;
16  create type account_record as (
17                 account_id  bigint,
18                 username text,
19                 first_name text,
20                 last_name text,
21                 dob date,
```

Any Problem With That?

Yes!

“We need a database developer to do this!”

With an ORM, you do not need to do anything special!

Let’s Go Back to JSON:

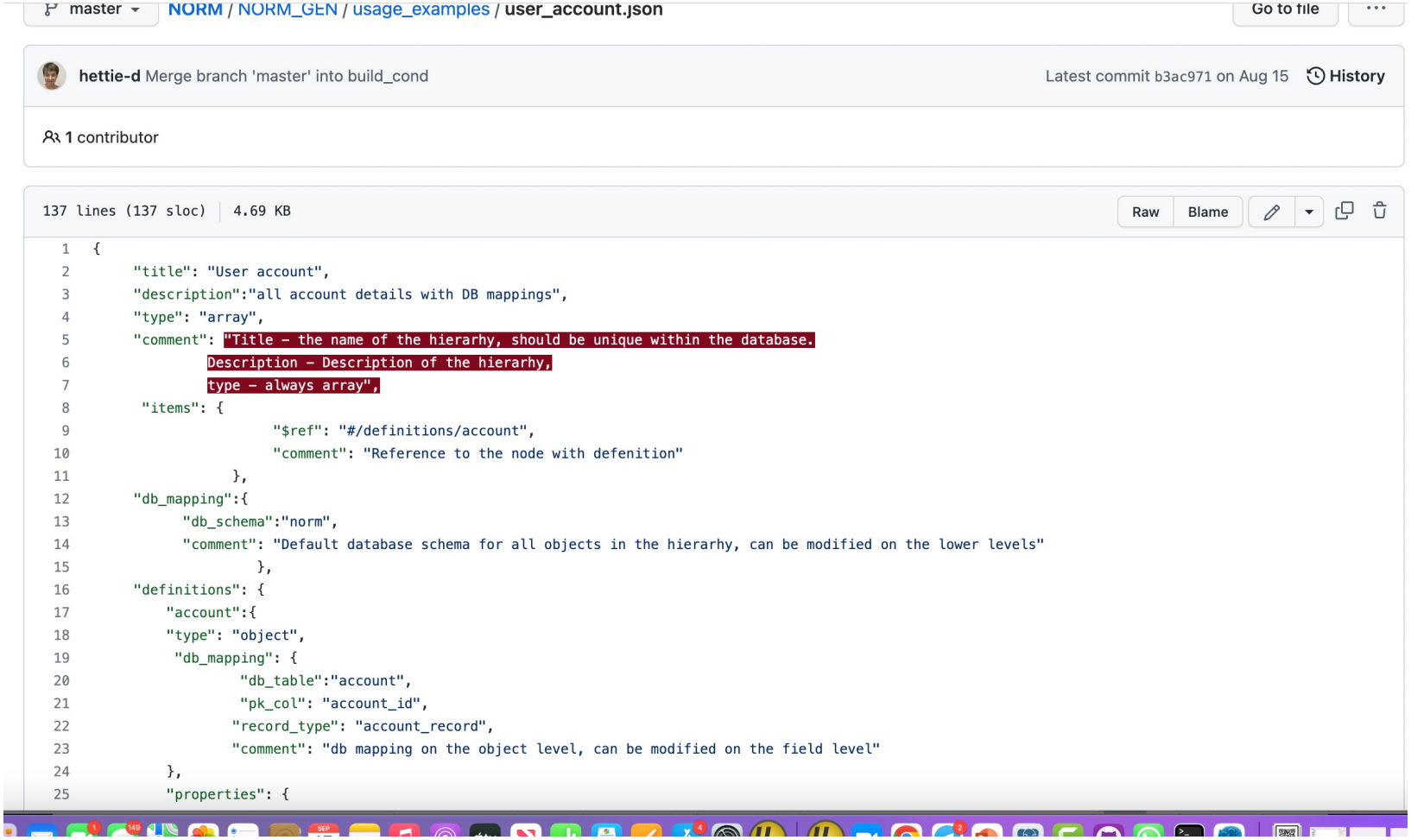
```
{
”account_id”: ”1”,
”username”: ”aliceacct1”,
”last_name”: ”johns”,
“first_name”:”alice”,
”dob” : ”1996-04-01” ,
”phones” : [
          { ”phone_id”: 1,
            ”phone_number” : ”2021234567” ,
            ”phone_type_id”:”2”, ”phone_type”: ”cell” }
          ],
”email_addresses”:[
     {”email_address_id”:1 ,
      ”email_address”:alicejons@gmail.com,
      “email_priority_id”: “1”,
      “email_priority”:”primary” }
      ]}
```

That’s great, but that’s not a contract!

We focused on:

- formalizing contracts (mapping)
- generating search functions (reads)
- generating DML functions (writes)

https://github.com/hettie-d/NORM/NORM_GEN


master
NORM / NORM_GEN / usage_examples / user_account.json
Go to file
hettie-d Merge branch 'master' into build_cond
Latest commit b3ac971 on Aug 15
History
1 contributor
137 lines (137 sloc)
4.69 KB
Raw
Blame
{
"title": "User account",
"description":"all account details with DB mappings",
"type": "array",
"comment": "Title – the name of the hierarhy, should be unique within the database.
Description – Description of the hierarhy,
type – always array",
"items": {
"$ref": "#/definitions/account",
"comment": "Reference to the node with defenition"
},
"db_mapping":{
"db_schema":"norm",
"comment": "Default database schema for all objects in the hierarhy, can be modified on the lower levels"
},
"definitions": {
"account":{
"type": "object",
"db_mapping": {
"db_table":"account",
"pk_col": "account_id",
"record_type": "account_record",
"comment": "db mapping on the object level, can be modified on the field level"
},
"properties": {

```
{
     "title": "ACCOUNT_hierarchy",
     "description":"all account details with DB mappings, variation of User account",
     "type": "array",
      "items": {
                 "$ref": "#/definitions/account"
                 },
     "db_mapping":{
          "db_schema":"norm"
                    },
     "definitions": {
         "account":{
         "type": "object",
          "db_mapping": {
                  "db_table":"account",
                  "pk_col": "account_id",
                 "record_type": "account_record"
         },
         "properties": {
```

```
"properties": {
            "dob": {
                "type": "string",
                "format": "date",
                "db_mapping":{
                  "db_col": "dob",
                  "db_type": "date"
                  }
            },
            "emails": {
                "type": "array",
                "items": {
                    "$ref": "#/definitions/email"
                }
            },
            "phones": {
                "type": "array",
                "items": {
                    "$ref": "#/definitions/phone"
                }
            },
```

```
"email": {
                "type": "object",
                "db_mapping": {
                    "parent_fk_col": "account_id",
                    "record_type": "email_record",
                    "db_table": "email",
                    "pk_col":"email_id",
                   "embedded":[
                       {"alias":"ep",
                        "db_schema":"dictionaries",
                        "db_table":"email_priority",
                        "pk_col":"email_priority_id",
                        "fk_col":"email_priority_id"
                        }
                        ]
                },
            "properties": {
                    "email_address": {
                        "type": "string"
                    },
```

Metadata Tables

create_norm_gen_tables.sql

transfer_schema

transfer_schema_object

transfer_schema_key

All of them are automatically populated by the “contract analyzer”

Parsing JSON Schema

process_schema_pkg.sql

save_transfer_schema (json, Boolean) – initial parsing, saving json, assigning the schema_id

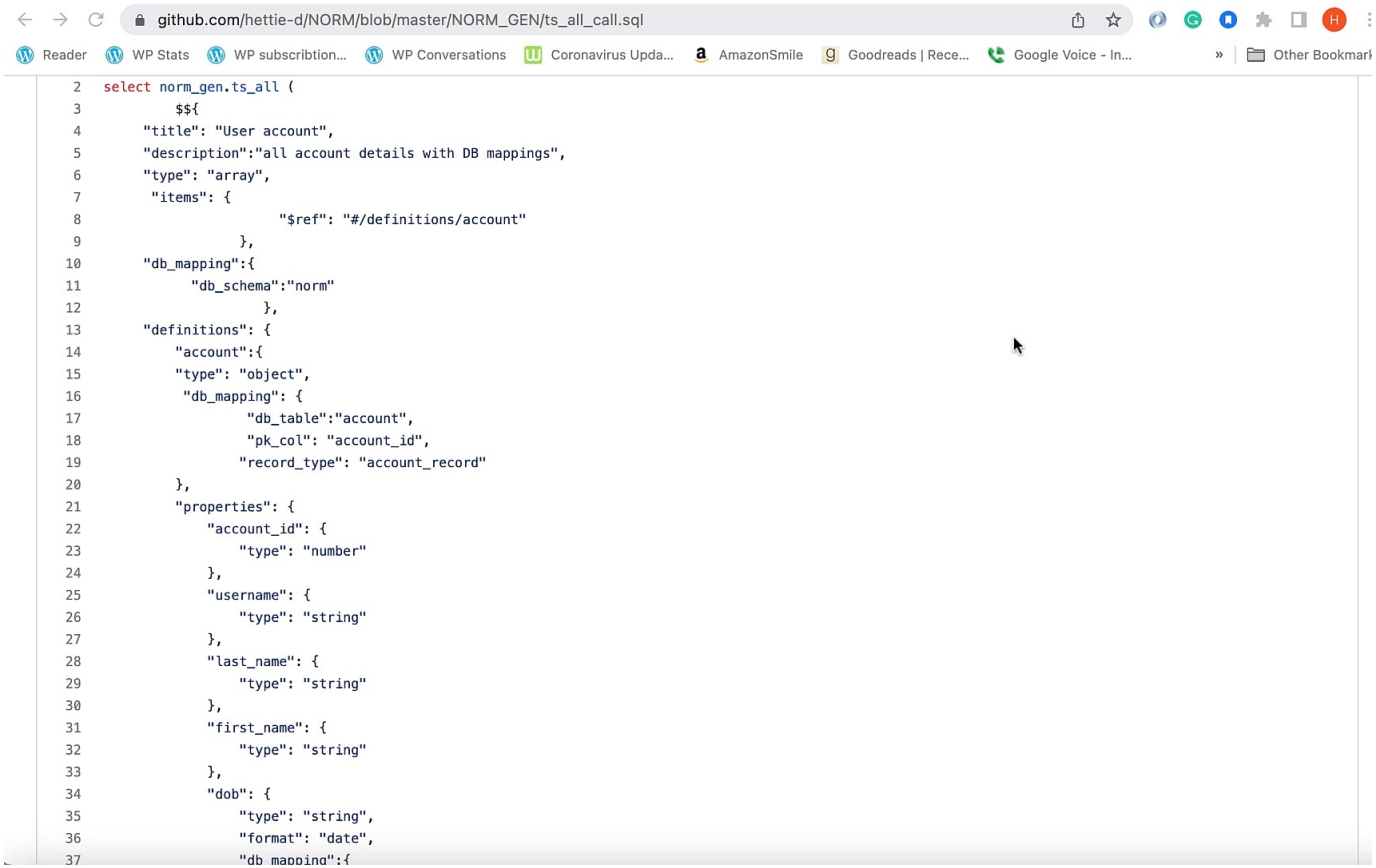
save_schema_object (int) - parse and save details of transfer objects for a given schema

save_schema_key (int) – parse and save details of individual keys, updates parent references for objects

update_db_type (int) – sets up all key types which are not explicitly defined

ts_all.sql

Using JSON Schema to Represent the Contract



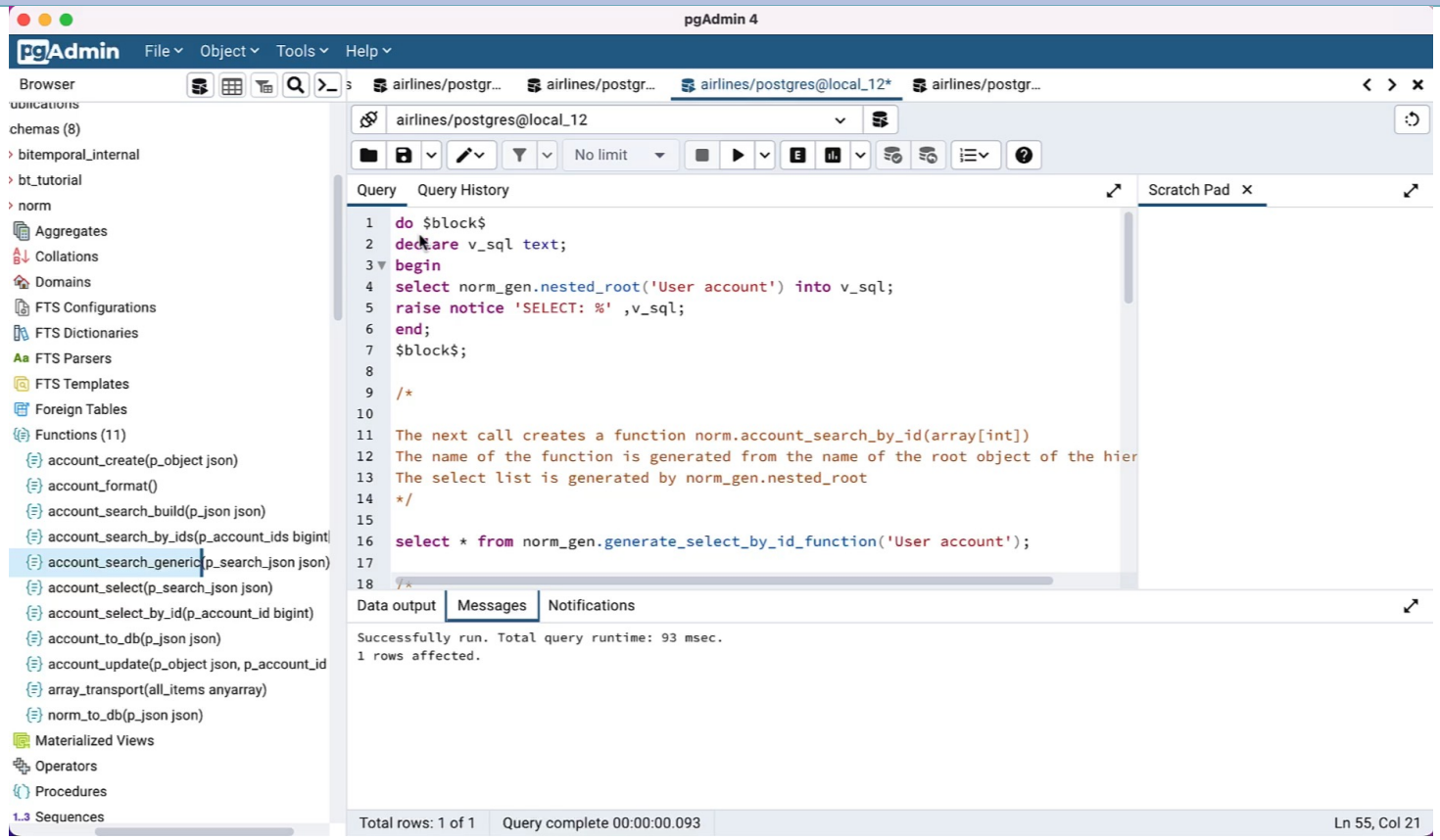

```
select norm_gen.ts_all (
        $${
    "title": "User account",
    "description":"all account details with DB mappings",
    "type": "array",
     "items": {
                    "$ref": "#/definitions/account"
                },
    "db_mapping":{
         "db_schema":"norm"
                },
    "definitions": {
        "account":{
        "type": "object",
         "db_mapping": {
                "db_table":"account",
                "pk_col": "account_id",
               "record_type": "account_record"
        },
        "properties": {
            "account_id": {
                "type": "number"
            },
            "username": {
                "type": "string"
            },
            "last_name": {
                "type": "string"
            },
            "first_name": {
                "type": "string"
            },
            "dob": {
                "type": "string",
                "format": "date",
                "db_mapping":{
```

Next step – generate all functions

(no database developer needed)

- SELECT of the nested object for a given hierarchy
- SEARCH by ID
- GENERIC SEARCH
- NORM_TO_DB converter

Example


pgAdmin 4
pgAdmin
File
Object
Tools
Help
Browser
airlines/postgr...
airlines/postgr...
airlines/postgres@local_12*
airlines/postgr...
airlines/postgres@local_12
No limit
Query
Query History
Scratch Pad
chemas (8)
bitemporal_internal
bt_tutorial
norm
Aggregates
Collations
Domains
FTS Configurations
FTS Dictionaries
FTS Parsers
FTS Templates
Foreign Tables
Functions (11)
account_create(p_object json)
account_format()
account_search_build(p_json json)
account_search_by_ids(p_account_ids bigint
account_search_generic(p_search_json json)
account_select(p_search_json json)
account_select_by_id(p_account_id bigint)
account_to_db(p_json json)
account_update(p_object json, p_account_id
array_transport(all_items anyarray)
norm_to_db(p_json json)
Materialized Views
Operators
Procedures
Sequences
1 do $block$
2 declare v_sql text;
3 begin
4 select norm_gen.nested_root('User account') into v_sql;
5 raise notice 'SELECT: %' ,v_sql;
6 end;
7 $block$;
8
9 /*
10
11 The next call creates a function norm.account_search_by_id(array[int])
12 The name of the function is generated from the name of the root object of the hier
13 The select list is generated by norm_gen.nested_root
14 */
15
16 select * from norm_gen.generate_select_by_id_function('User account');
17
18 /*
Data output
Messages
Notifications
Successfully run. Total query runtime: 93 msec.
1 rows affected.
Total rows: 1 of 1
Query complete 00:00:00.093
Ln 55, Col 21

Building search

```
select norm.account_search_generic($${
"phone_type":"cell",
"email_priority":"primary",
"account":{"last_name":"johns",

"emails":{"email_address":{"$like":"%gmail%"}},
     "dob":{"$gt":"1901-01-01"},
     "phones":{"phone_number":{"$like":"312%"}}
     }
}$$::json);
```

```
from  norm.account  top   where  account_id IN (
   select account_id from norm.email where
    email_priority_id IN (
   select email_priority_id from
 norm.email_priority where
    email_priority  =  ('primary'::text) )
AND  email  LIKE  ('%gmail%'::text) )
AND  account_id IN (
   select account_id from norm.phone where
    phone_type_id IN (
   select phone_type_id from norm.phone_type
where
    phone_type  =  ('cell'::text) )
AND  phone  LIKE  ('312%'::text) )
AND  dob  >  ('1901-01-01'::date)  AND
  last_name  =  ('johns'::text)
```

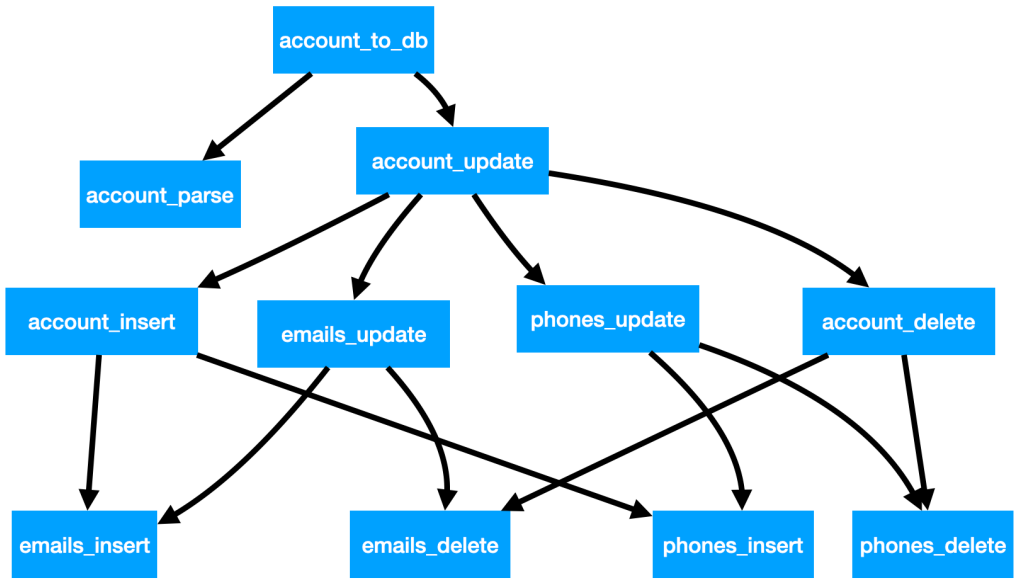
Generating DML

```
select * from  norm_gen.generate_to_db ('User account’)
```

Builds NORM_TO_DB converter for a specified hierarchy

By default, the name is set to “root object” name plus the schema_id

How it works


account_to_db
account_parse
account_update
account_insert
emails_update
phones_update
account_delete
emails_insert
emails_delete
phones_insert
phones_delete

NORM-TO-DB Converter for User account

insert:

```
select * from norm.account_1_to_db ($$[{
  "username":"johnsmithaccount",
  "first_name":"john",
  "last_name":"smith",
  "dob":"1991-04-01",
   "phones":[{"phone_number":"3123334556", "phone_type_id":"1"}]
 }]
$$::json)
```

update:

```
select * from norm.account_1_to_db ($$[{
   "account_id":1,
   "username":"aliceacct2"}]
 $$::json
)
```

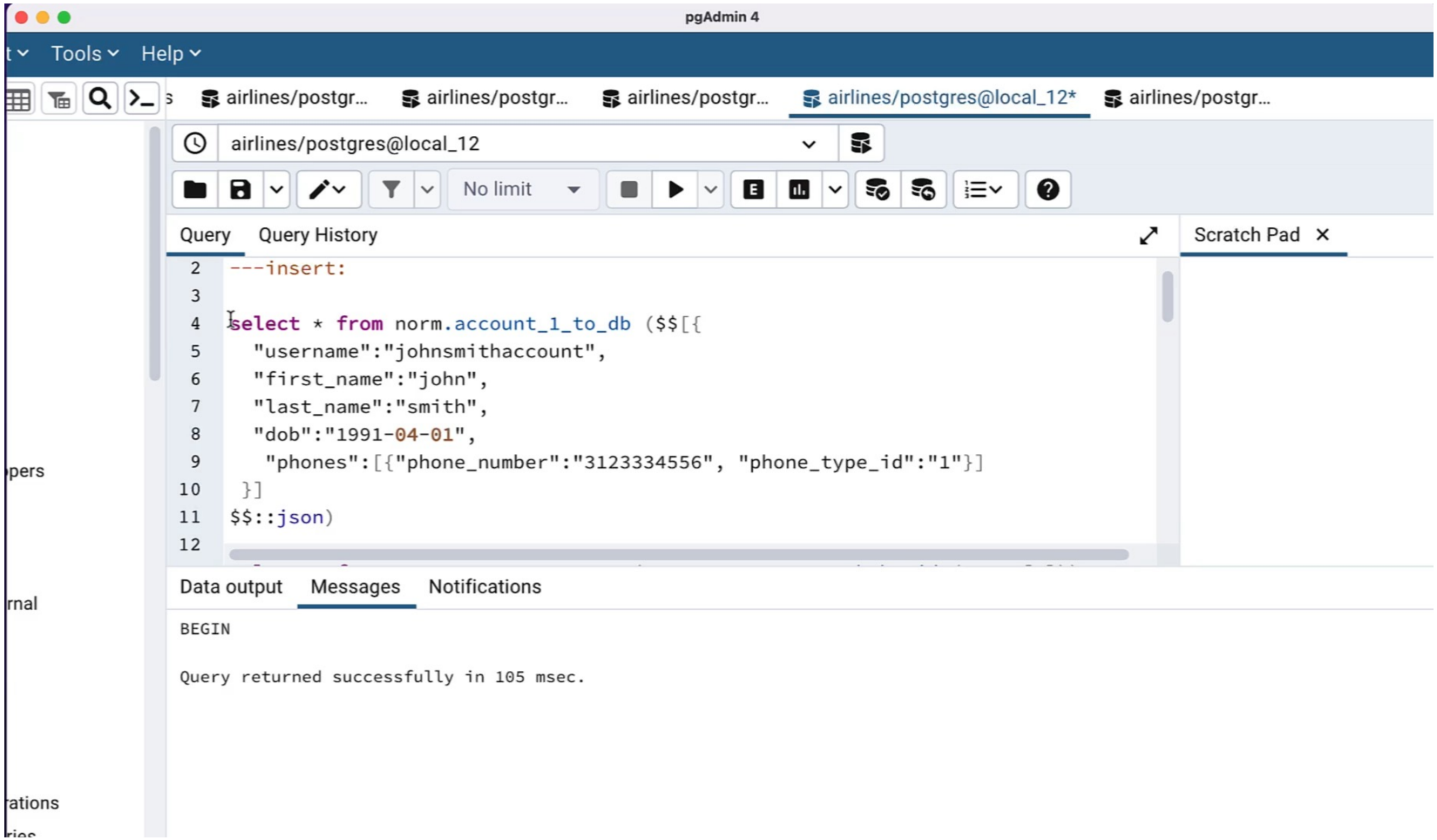
NORM-TO-DB Converter for User account

update embedded objects:

```
select * from norm.account_1_to_db($$[{
   "account_id":"3",
   "emails":[{"email_address":"new.email@hotmail.com",
                                      "email_priority_id":"1"}]
      }]
     $$::json)
```

delete:

```
select * from norm. account_1_to_db($$[{
   "account_id":"1",
   "phones":[{"phone_id":"2", "command":"delete"}]
   }]
 $$::json)
```


pgAdmin 4
Tools
Help
airlines/postgr...
airlines/postgr...
airlines/postgr...
airlines/postgres@local_12*
airlines/postgr...
airlines/postgres@local_12
No limit
Query
Query History
Scratch Pad
2 ---insert:
3
4 select * from norm.account_1_to_db ($$[{
5 "username":"johnsmithaccount",
6 "first_name":"john",
7 "last_name":"smith",
8 "dob":"1991-04-01",
9 "phones":[{"phone_number":"3123334556", "phone_type_id":"1"}]
10 }]
11 $$::json)
12
Data output
Messages
Notifications
BEGIN
Query returned successfully in 105 msec.

Additional Considerations and Summary

How this is different from other ORMs?

"Thinking sets"
Mapping complex objects
Doing DB work inside DB

Other programming languages

Details may vary (no need for converting to text, etc.)
Active Record requires additional work, because it reads the DB catalog directly

Storing JSON vs. building JSON

Search
Multiple hierarchies

Questions?