

We Know ORM Is Bad, Now What?

Hettie Dombrovskaya
Senior Cloud consultant
EDB

Swiss PG Day 2022



NORM: No ORM Framework

Now – Automated!

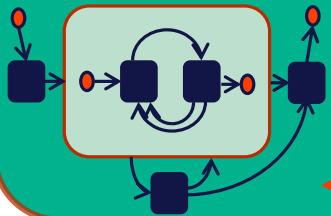


Quick Quizz

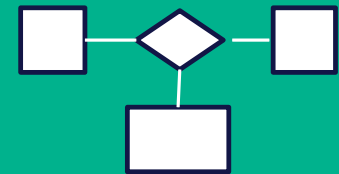
- What's ORM?
 - Object Relational Mapper
- Why developers use ORM?
 - To abstract from database specifics
- Why ORM is ~~bad~~ good?
 - Rapid development
- Why ORM **is bad**?
 - Inefficient data access patterns



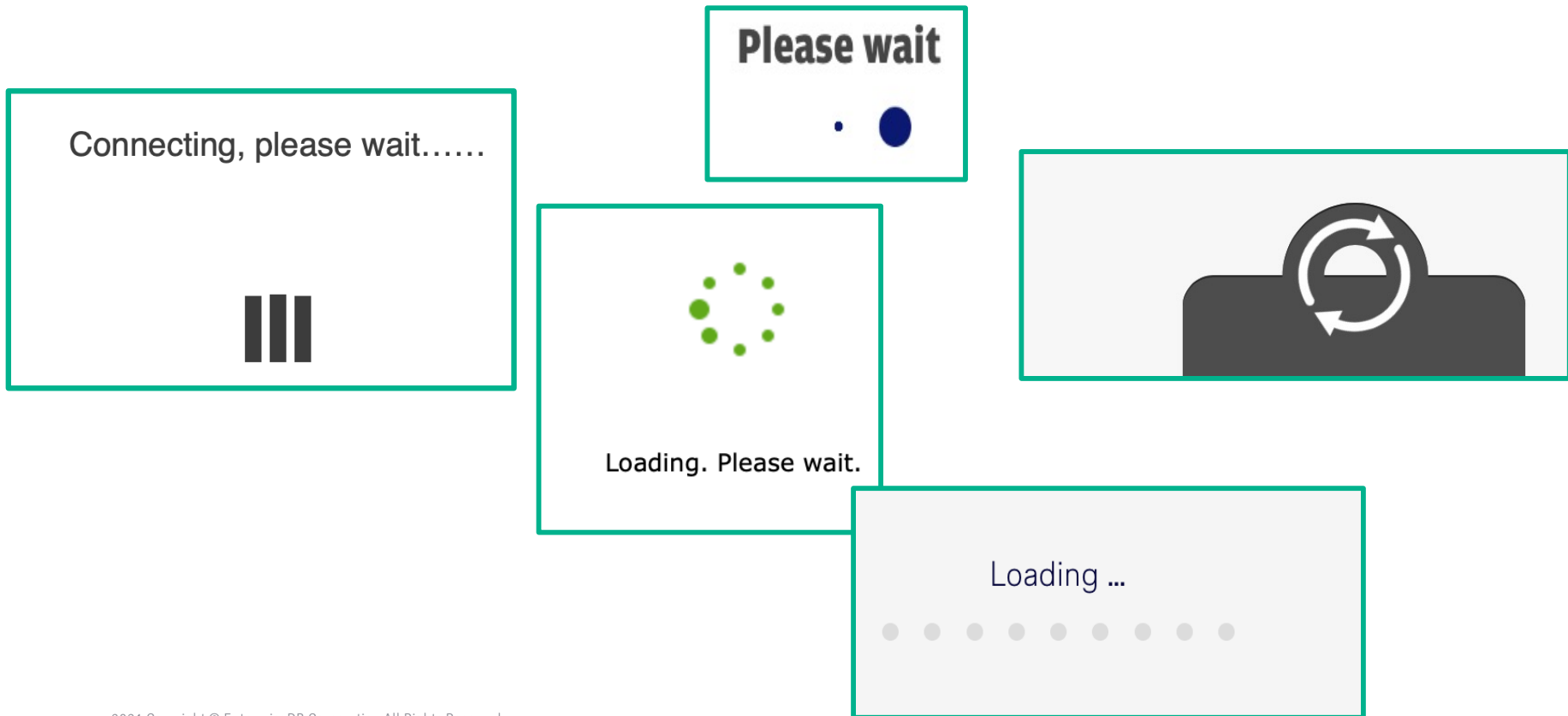
Application Model
(Object-Oriented)



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



More Hardware?...

How much RAM you may need?

500 GB? 1 TB? 4 TB???

What if we are in the cloud?

- Amazon instances

4	17h47m23s	107,350,233	0ms	15s457ms	0ms	<div>d£)INSERT INTO "table_xxxxxxx_v3" ("tbl_xxxxxxx_id", "tbl_xxxxxxx_yid", "xxxxxe_id", "xxxxxd_id", "xxxxxr_id", "pssssss_bbb_l_xxxxxxx_id", "pssssss_bbb_l_xxxxxxx_yid", "ppppppp_xxxxxe_id", "ppppppp_xxxxxd_id", "ppppppp_xxxxxr_id", "hhhhh_ccccccc_at", "ututututut", "atata_sssssss") 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";</div>
		<div>Details</div>				
		<div>Examples</div>	<div>User(s) involved</div>			



**We know what's the
right solution..**

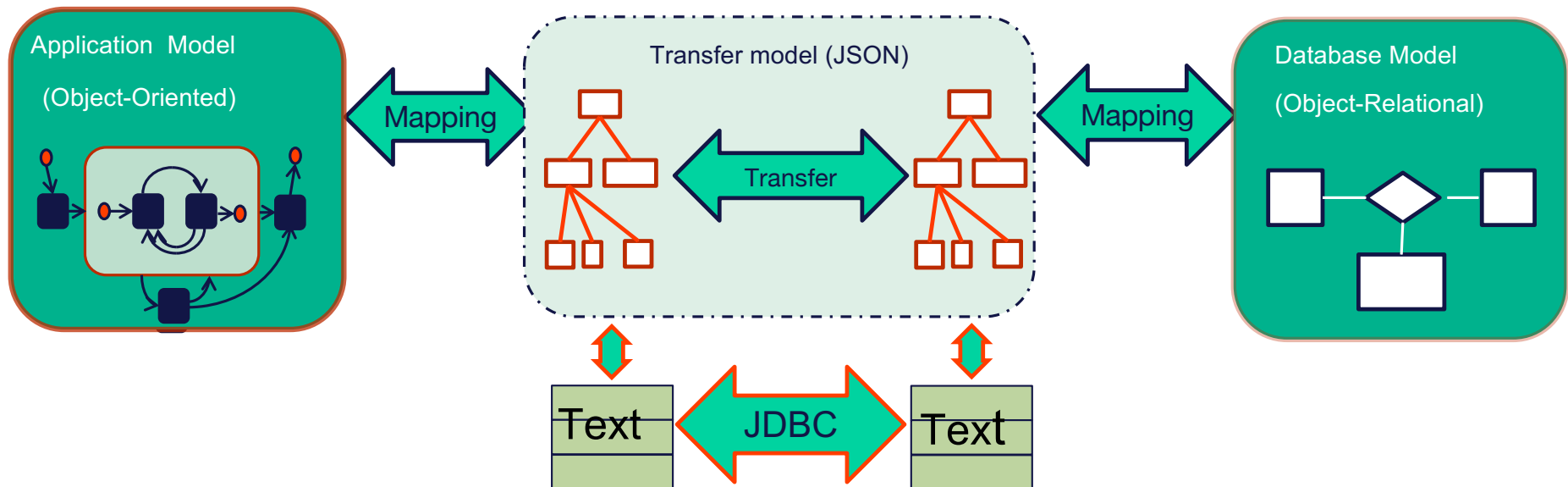


Thinking sets!

2021 Copyright © EnterpriseDB Corporation All Rights Reserved



Our Solution - NORM



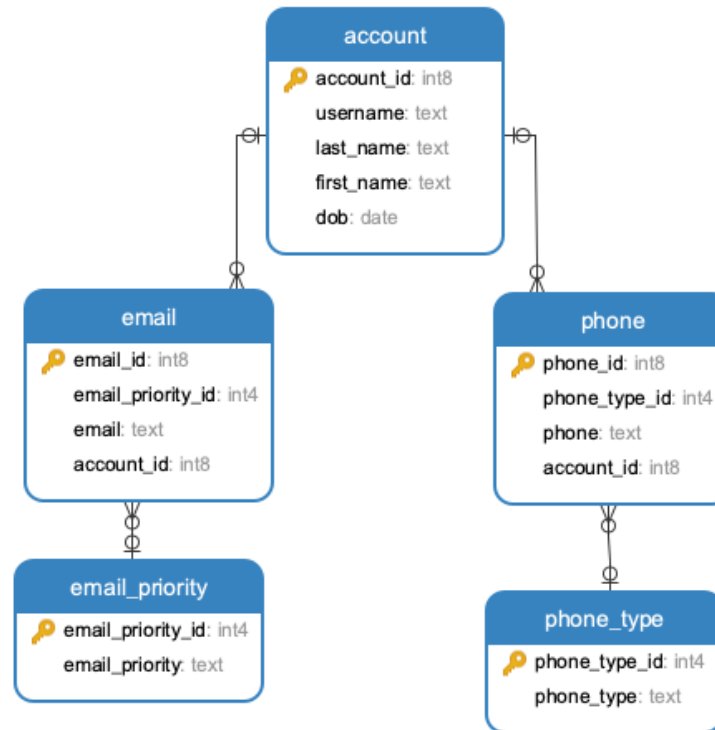
CONTRACT

- 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

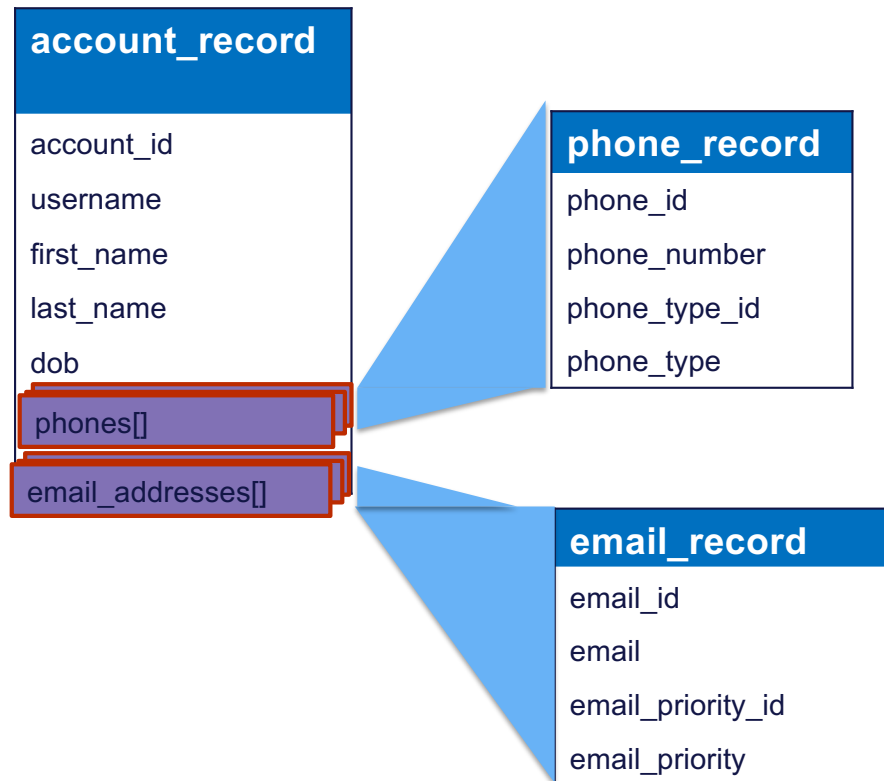
App

DB

DB Schema



Objects Mapping (Transport Object)



Transfer Object in JSON Format

```
{
  "account id": "1",
  "username": "aliceacct1",
  "last_name": "johns",
  "firs_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" }
  ]
}
```



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



**Things look good for
a while...**



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": alicejones@gmail.com,
      "email_priority_id": "1",
      "email_priority": "primary" }
  ]
}
```



“

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



Using JSON Schema to Represent the Contract

```
← → ↺ github.com/hettie-d/NORM/blob/master/NORM_GEN/ts_all_call.sql
WP Reader WP Stats WP subscription... WP Conversations Coronavirus Upda... AmazonSmile Goodreads | Rece... Google Voice - In... » Other Bookmarl

2 select norm_gen.ts_all (
3   $$ {
4     "title": "User account",
5     "description": "all account details with DB mappings",
6     "type": "array",
7     "items": {
8       "$ref": "#/definitions/account"
9     },
10    "db_mapping": {
11      "db_schema": "norm"
12    },
13    "definitions": {
14      "account": {
15        "type": "object",
16        "db_mapping": {
17          "db_table": "account",
18          "pk_col": "account_id",
19          "record_type": "account_record"
20        },
21        "properties": {
22          "account_id": {
23            "type": "number"
24          },
25          "username": {
26            "type": "string"
27          },
28          "last_name": {
29            "type": "string"
30          },
31          "first_name": {
32            "type": "string"
33          },
34          "dob": {
35            "type": "string",
36            "format": "date",
37            "db_mapping": {
```

Automating functions generation

https://github.com/hettied/NORM/NORM_GEN

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 runs them all

Example

The screenshot displays the pgAdmin 4 web interface. On the left, the 'Browser' pane shows a tree view of the database structure, with 'airlines' selected under 'local_12'. The main area is divided into three tabs: 'Query Editor', 'Query History', and 'Scratch Pad'. The 'Query Editor' tab is active, showing a SQL query that populates metadata with sample data. The query is as follows:

```
1 ---populate metadata with the sample data
2 select norm_gen.ts_all (
3     ${
4         "title": "User account",
5         "description": "all account details with DB mappings",
6         "type": "array",
7         "items": {
8             "$ref": "#/definitions/account"
9         },
10    "db_mapping": {
11        "db_schema": "norm"
12    },
13    "definitions": {
14        "account": {
15            "type": "object",
16            "db_mapping": {
17                "db_table": "account",
18                "pk_col": "account_id",
```

Below the query editor, the 'Messages' pane shows the execution results:

```
NOTICE: function norm_gen.save_schema_object(pg_catalog.int4) does not exist, skipping
NOTICE: function norm_gen.save_schema_object_key(pg_catalog.int4) does not exist, skipping
CREATE FUNCTION

Query returned successfully in 551 msec.
```



Next steps

Now, all functions can be generated, no database developer needed!

We generate

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



Example

The screenshot shows the pgAdmin 4 web interface. On the left, the 'Browser' pane displays a tree view of database objects. The 'Types (9)' folder is expanded, showing various record types like 'account_record', 'email_record', etc. The main area is divided into three panes: 'Query Editor', 'Query History', and 'Scratch Pad'. The 'Query Editor' contains a SQL script with line numbers 141 to 158. The script includes a block comment, a 'do' block with a 'declare' statement and a 'select' query, and a 'test the result' comment. The 'Data Output' pane at the bottom shows the results of the query, which is a single row with a single column labeled 'integer' containing the value '1'.

```
141 execute v_sql;
142 end;
143 $block$
144 -- generate select nested object
145
146 do $block$
147 declare v_sql text;
148 begin
149 select norm_gen.nested_root('User account') into v_sql;
150 raise notice 'SELECT: %' ,v_sql;
151 end;
152 $block$
153
154 ---generate function norm.account_search_by_ids
155 |
156 select * from norm_gen.generate_select_by_id_function('User account');
157
158 ---test the result
```

?column?
integer
1



Universal NORM-TO-DB Converter

This function is custom-built, and also can be used as a template.
Requires additional “in” types.

insert:

```
select * from norm.account_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_to_db ($${{
  "account_id":1,
  "username":"aliceacct2"}}
$$::json
)
```



Universal NORM-TO-DB Converter

This function is also custom-built, and also can be used as a template.

update embedded objects:

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

delete:

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



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?

