An Adventure in Data Modeling

The Entity-Attribute-Value Data Model

PGCon
May 23, 2014

Mark Wong
mark.wong@myemma.com
Emma Email Marketing

@emmaemailtech

Who is Emma?

At Emma, we're out to create a world-class brand that's known and loved by marketers, designers and business owners everywhere. And we're well on our way, supporting the email marketing efforts of roughly 40,000 businesses, nonprofits and agencies doing all sorts of interesting things in all sorts of interesting places, assuming Belgium makes your list of interesting places, and why wouldn't it?

http://myemma.com

Stylish Email Marketing

Why am I here?

Tell a story about some of our Postgres performance experiences with the evolution of the data model around our member information, where we stumbled along the way, and how we are carrying on.

What is member information?

Member information is an account's email list and any additional attributes that the customer desires such as:

- first name
- last name
- favorite database

- Horizontally partitioned data by account using table inheritance
- 14 child tables created per account
- Exporting member information was fast and easy because all member information were contained in a single table

Once upon a time…

- If my members were the PostgreSQL Core Team:

```
COPY userdata_88888_members
TO ‘members-88888.csv’ (FORMAT CSV)
```

email	first_name	last_name	favorite_dbms
josh at agliodbs.com	Josh	Berkus	PostgreSQL
peter_e at gmx.net	Peter	Eisentraut	SQLite
magnus at hagander.net	Magnus	Hagander	PostgreSQL
tgl at sss.pgh.pa.us	Tom	Lane	PostgreSQL
bruce at momjian.us	Bruce	Momjian	PostgreSQL
dpage at pgadmin.org	Dave	Page	PostgreSQL

Example of member information

- Over 40,000 accounts in the system
  - Hard to mine data
  - Well over one million objects in the system (tables, indexes, sequences, etc.)
  - Hard to administer database system
- Induced **ALTER TABLE** statements whenever an attribute is added on a heavily accessed table

What was wrong?

- How many marketing campaigns were sent yesterday?
- Getting counts from the parent tables would need 40,000 locks, one per child table
- More complex queries would start adding tables to join

Example of simple data mining exercise

Issues with a large system catalog

- Backups with *pg_dump* takes more than whole day for less than 1 terabyte of data
- Would only run backups over the weekend

Time to do something dramatic!

Highlighting a few of the changes that occurred:

- Reduced the number of database objects by horizontally partitioning into a fixed number of tables (1024 partitions)
- Approximately 1 GB of data per partition
- Developed home grown Python middleware layer between Web front end and database systems
- Major database schema refactor: applied entity-attribute-value data model to member information

A few years ago…

Entity-attribute-value model (EAV) is a data model to describe entities where the number of attributes (properties, parameters) that can be used to describe them is potentially vast, but the number that will actually apply to a given entity is relatively modest.

…

EAV is also known as object-attribute-value model, vertical database model and open schema.

http://en.wikipedia.org/wiki/Entity-attribute-value_model

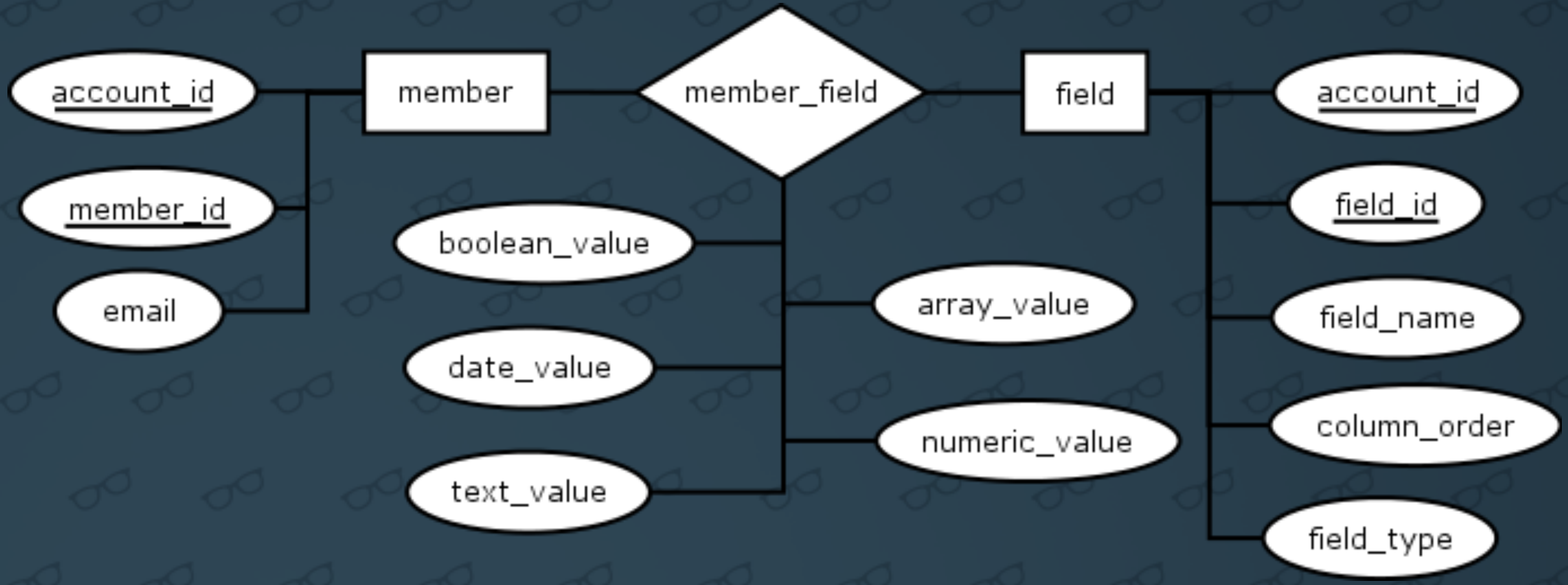
- Pros
  - Avoid expensive **ALTER TABLE** statements when adding or removing member attributes
- Cons
  - Data will need to be queried differently
  - Data type checking either done using multiple tables or multiple columns (opted for latter)

What we knew before applying EAV

EAV table descriptions

Three tables make up the model:

- Entity: **member** table contains attributes that all members must have, e.g. email address
- Attribute: **field** table contains the custom attributes that users defines, e.g. favorite database management system
- Value: **member_field** table contains the values for custom attributes defined in the **field** table



EAV ER Digram

- Uses SQLAlchemy ORM to pull data and performs a data pivot

- Restricts API calls to return up to 500 members per call

The middleware layer

- Before pivot:

email	field_name	value
josh at agliodbs.com	first_name	Josh
josh at agliodbs.com	last_name	Berkus
josh at agliodbs.com	favorite_dbms	PostgreSQL

- After pivot:

email	first_name	last_name	favorite_dbms
josh at agliodbs.com	Josh	Berkus	PostgreSQL

Pivoting data

That doesn’t look so bad, right?
How much data might our customers have?

Ranked in order of potentially most values:

rank	account	members	fields	values	max values
1	41383	994,684	119	32,079,663	118,367,396
2	21322	1,902,163	59	5,354,715	112,227,617
3	2451	4,661,264	22	844,881	102,547,808
4	1703180	3,884,321	26	9,933,392	100,992,346
5	41997	737,432	87	4,115,583	64,156,584
6	18528	1,120,968	52	6,310,398	58,290,336
7	4393	656,672	85	5,175,631	55,817,120
8	1366214	470,107	109	7,272,797	51,241,663

Sample of account sizes

How long it takes to export member information?

All exports failed for our largest accounts!

Something is taking too long:

- PostgreSQL statement timeouts; disable statement timeout?
- Apache HTTP timeouts; don't go through the Web server?
- Network switches TCP/IP idle timeouts; get closer to the database server?

Where are we failing?

After bypassing as many things as possible and extracting Python code to run directly against the database:

rank	account	members	values	runtime
1	41383	994,684	32,079,663	DNF
2	21322	1,902,163	5,354,715	DNF
7	4393	656,672	5,175,631	4 hours

Exporting directly from the database system

Maybe the middleware shouldn't be trying to do that much work.

Maybe the database management system can help…

PostgreSQL provides the extension *tablefunc* containing the *crosstab()* data pivoting functions.

http://www.postgresql.org/docs/current/static/tablefunc.html

The database can pivot data

Spoiler alert!

Postgres pivots data faster than how we did it in Python

If you use the correct *crosstab* function…

I have Emma’s favorite DBMS, but not her last name. These *crosstab* functions puts only non-NULL data into the next column pivoted and pads any remaining columns with NULLs.

email	first_name	last_name	favorite_dbms
josh at agliodbs.com	Josh	Berkus	PostgreSQL
peter_e at gmx.net	Peter	Eisentraut	SQLite
magnus at hagander.net	Magnus	Hagander	PostgreSQL
tgl at sss.pgh.pa.us	Tom	Lane	PostgreSQL
emma at myemma.com	Emma	**MongoDB**	

crosstab(text sql) and *crosstabN(text sql)*

This *crosstab* function aligns the data with the column it is pivoted to.

email	first_name	last_name	favorite_dbms
josh at agliodbs.com	Josh	Berkus	PostgreSQL
peter_e at gmx.net	Peter	Eisentraut	SQLite
magnus at hagander.net	Magnus	Hagander	PostgreSQL
tgl at sss.pgh.pa.us	Tom	Lane	PostgreSQL
emma at myemma.com	Emma		**MongoDB**

crosstab(text source_sql, text category_sql)

How much of a positive improvement was *crosstab*?

Timed python script running directly against database system:

rank	account	members	values	previously	runtime
1	41383	994,684	32,079,663	DNF	22 min
2	21322	1,902,163	5,354,715	DNF	17 min
7	4393	656,672	5,175,631	4 hours	10 min

Results from using *crosstab*

Much faster!

- Cannot use ORM to model pivoted data
- Small exports (in the 100’s) appear to take a little longer

There are some tradeoffs

Exports will fail again if we take on accounts somewhere between 5 to 10 million members

Not all problems solved

- Retrieving data from EAV model seems inefficient

- Performance issues begin when pivoting only millions of rows

What we knew after having EAV

We still need to do better

What can we do?

Time to explore other options

“What if we remove the **member_field** table from the database altogether?”

–Most popular question asked within Emma.

Let’s prototype a different data model in Postgres

First look at *hstore* as a key/value data store…

This module implements the hstore data type for storing sets of key/value pairs within a single PostgreSQL value. This can be useful in various scenarios, such as rows with many attributes that are rarely examined, or semi-structured data. Keys and values are simply text strings.

http://www.postgresql.org/docs/current/static/hstore.html

Maybe the *hstore* extension can help proof a solution

Things to note before going in:

- No strict types; everything is a string
- No referential integrity constraints; cannot create a foreign key between an *hstore* key and a table column
- psycopg2 and SQLAlchemy support for *hstore* not released at the time, but are now

Cons to *hstore* data type

What does *hstore* look like

Put the member attribute values into the **member** table as the *hstore* column field. The key in field's key/value pair is the field name.

email	field
josh at agliodbs.com	“first_name”=>”Josh”, “last_name”=>”Berkus”, “favorite_dbms”=>“PostgreSQL”
peter_e at gmx.net	“first_name”=>”Peter”, “last_name”=>”Eisentraut”, “favorite_dbms”=>“PostgreSQL”
magnus at hagander.net	“first_name”=>”Magnus”, “last_name”=>”Hagander”, “favorite_dbms”=>“PostgreSQL”

Is it hard to convert EAV to key/value model?

Approximately 2 minutes to transform a single partition:

```
WITH u AS (
    WITH t AS (
        SELECT member_id, shortcut_name,
               CASE WHEN f.field_type = 'text' THEN mf.text_value
                    WHEN f.field_type = 'text[]' THEN mf.array_value::TEXT
                    WHEN f.field_type = 'numeric' THEN mf.numeric_value::TEXT
                    WHEN f.field_type = 'boolean' THEN mf.boolean_value::TEXT
                    WHEN f.field_type = 'date' THEN date_value::TEXT
                    ELSE NULL END AS value
        FROM field f, member_field mf
        WHERE f.field_id = mf.field_id
    )
    SELECT member_id,
           string_agg(hstore(shortcut_name, value)::TEXT, ',')::HSTORE AS hst
    FROM t GROUP BY member_id
)
UPDATE member
SET field = hst
FROM u
WHERE u.member_id = member.member_id;
```

Converting to *hstore* is fairly fast

```
COPY (
    SELECT email,
           field -> 'name_first' AS first_name,
           field -> 'name_last' AS last_name,
           field -> 'favorite_dbms' AS favorite_dbms
    FROM member m
    WHERE m.account_id = 88888)
TO ‘audience-88888.csv’ (FORMAT CSV)
```

Exporting member information with *hstore*

How fast is exporting member information with *hstore?*

rank	account	members	values	SQLAlchemy	crosstab	hstore
7	4393	656,672	5,175,631	4 hours	10 min	15 sec

Exporting member information is pretty fast

Are we done yet?

Have only looked at *hstore* thus far…

- Use *crosstab* to pivot data in parts
- Use HSTORE on the fly since converting data seems relatively quick
- JSON to get some strict type checking (except dates)
- BSON?
- External data store
- Yet another data model
- XML might be used to get strict type checking with DTD

Other things to try, maybe

Thank you!