An Adventure in Data Modeling

The Entity-Attribute-Value (EAV) Data Model

Mark Wong

PostgreSQL® Major Contributor
markwkm@postgresql.org

EDB Software Development Engineer
mark.wong@enterprisedb.com

PGDU, Sydney, Australia

2025 Oct 17

Agenda

- Introduction
- The story
  - The original data model
  - Implementing the Entity-Attribute-Value (EAV) data model
  - Problems
  - Workarounds
- Please ask questions at any time!

Introduction

- Employed by  EDB
  `https://enterprisedb.com`
- PostgreSQL® Major Contributor
- Director at United States PostgreSQL® Association
  `https://postgresql.us`
- Portland PostgreSQL® Users Group
  `https://meetup.com/pdxpug/`

Why am I here?

Tell a PostgreSQL® performance story around the experiences with the evolution of the data model around a marketing company's customer information, where there was stumbling along the way, and how we carried on.

- Implementing the EAV data model
- Living with the EAV data model
- Share what was learned
  - Spoiler alert: don't be hasty
  - It's a database anti-pattern

Once upon a time...

(Before getting into the EAV data model...)

(Sometime before 2010...)

The customer's data

Per customer customize information about their members, e.g.:

- e-mail address
- first name
- last name
- favorite database

The original data model

- Horizontally partitioned data by customer using table inheritance
- 12+ child tables created per customer - only 1 table contained person information
- Exporting person information was fast and easy - this is the primary benchmark

Example: Exporting person information

A commonly performed task is to create a CSV file:

email	first_name	last_name	favorite_dbms
markwkm at postgresql.org	Mark	Wong	PostgreSQL
mark.wong at enterprisedb.com	Mark	Wong	WarehousePG

```
COPY persons_5432
TO 'persons-5432.csv' (FORMAT CSV);
```

A table with 1 million rows can be expected to complete within seconds.

What was wrong?

Other aspects of the system was under a lot of stress:

- Over 40,000 customers in the system
- One million+ objects in the system (tables, indexes, etc.)
- Hard to mine data i.e. generate reports
- Hard to administer database system
  - ALTER TABLE to add new attribute to track
  - Database backups

Problems with simple data mining

If simple queries can't run, complex queries definitely can't run:

- How many persons are being tracked?
- Getting counts from the parent tables would need 40,000 locks, one per child table
- More complex queries would start adding tables to join, increasing locks to be taken

Problems with database backups

Is an RPO (Recovery Point Objective) greater than a day ok?

- Backups with `pg_dump` takes more than whole day
- Less than 1 terabyte of data

Time to do something **drama**tic!

Major design changes

Highlights:

- Horizontally partition the database into a fixed number of tables to reduce the number of database objects
  - 40,000 partitions reduced down to 1,000
  - Approximately 1 GB of data per partition
- Developed homegrown Python middleware layer between Web front-end and database systems
- Database schema refactor: `persons` table converted to EAV data model

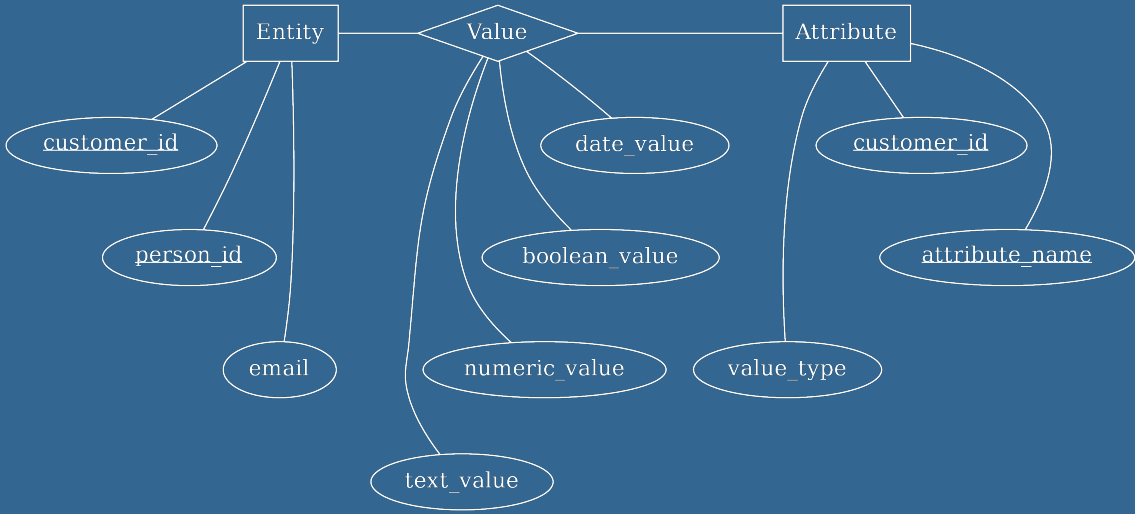
Entity-attribute-value (EAV) model is a data model to describe entities where the number of attributes (properties, parameters) that can be used to describe them is potentially vast, but the number that will actually apply to a given entity is relatively modest.

...

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

`https://en.wikipedia.org/wiki/Entity-attribute-value_model`

EAV Entity-Relationship Diagram



What was known before rolling out EAV

- Pros
  - Still a simple three table model - even though previously 1 table
  - Can add attributes with DML statements instead of DDL statements
    - Avoid `ALTER TABLE` statements - expensive heavy table locks
    - Use `INSERT`, `UPDATE` or `DELETE` - lighter finer grained row locks
- Cons
  - Data will need to be:
    - queried differently
    - transformed somewhere i.e. pivoted
  - Data type checking more complex, either implemented by:
    - multiple tables
    - multiple columns (opted for this)

The need to pivot data

Using EAV pivoted the data from the original model:

email	attribute_name	value
markwkm at postgresql.org	first_name	Mark
markwkm at postgresql.org	last_name	Wong
markwkm at postgresql.org	favorite_database	PostgreSQL

Need to pivot it back:

email	first_name	last_name	favorite_dbms
markwkm at postgresql.org	Mark	Wong	PostgreSQL

That doesn't look so bad, right?

Sample of table sizes

Row counts are more significant than data size:

customer	entity	attribute	value
1	1,000,000	120	120,000,000
2	2,000,000	50	100,000,000
3	700,000	100	700,000

How long it takes to export data?

This page was intentionally left blank.

All exports failed for the largest customers!

Uh oh, what's going wrong?

Some customer can't export their data anymore, something is taking too long:

- PostgreSQL® statement timeouts? Disable statement timeout?
- Web server HTTP timeouts? Don't go through the Web server?
- Network switches TCP/IP idle timeouts?
- Get closer to the database server?

Application developers please take note!

Homegrown Python middleware layer was pivoting data:

customer	entity	attribute	value	execution time
1	1,000,000	120	120,000,000	DNF
2	2,000,000	50	100,000,000	DNF
3	700,000	100	700,000	4 hours

Did **N**ot **F**inish

Where to start...

As a database person suspicious that homegrown code can pivot data faster than a database management system:

- Have the database return pivoted data
- The *tablefunc* extension can pivot data with the *crosstab()* function
- `https://www.postgresql.org/docs/current/static/tablefunc.html`

Brief simpilified *crosstab()* example

```
SELECT email, first_name, last_name, favorate_dbms
FROM entity LEFT OUTER JOIN
     crosstab (<sql>) AS ct(person_id BIGINT,
                            first_name TEXT,
                            last_name TEXT,
                            favorite_dbms TEXT)
        ON entity.person_id = ct.person_id
WHERE entity.customer_id = 5432;
```

Note: `<sql>` is the `SELECT` statement to retrieve data from the EAV model.

Is *crosstab()* a positive improvement?

Results with *crosstab()*

customer	entity	attribute	value	EAV execution time	*crosstab()* execution time
1	1,000,000	120	120,000,000	DNF	22 minutes
2	2,000,000	50	100,000,000	DNF	17 minutes
3	700,000	100	700,000	4 hours	10 minutes

Yay! Much Faster!

Business as usual, for now...

There are some trade-offs:

- ORMs do not natively abstract the database pivoting data (is this still true in 2025?)
- Small exports (in the 100's) now take a little longer
- Overall still slower than the original data model

Not all obstacles have been removed

After more testing, problems with EAV not fully addressed:

- *crosstab()* will fail with any customers with between 5 to 10 million persons
- Importing person data already facing similar performance challenges

---

We still need to do better!

Biggest question at this juncture:
What if we remove this data from the PostgreSQL®?

Ok, I guess we can prototype another data model...**in PostgreSQL®**!

Exploring a non-relational data model

Try a key/value store:

- Use the PostgreSQL® *hstore* extension to proof a solution
  - This extension 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.
- `https://www.postgresql.org/docs/current/hstore.html`

Cons to *hstore* data type

Noted before diving in:

- No strict types; everything is a string
- No referential integrity constraints
- Native support varies in higher level database connectivity libraries

This is *hstore*

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

email	**hstore**
markwkm at postgresql.org	"first_name" => "Mark", "last_name" => "Wong", "favorite_database" => "PostgreSQL"

Example converting EAV to *hstore*

```
WITH u AS (
    WITH t AS (
        SELECT person_id, value.attribute_name,
        CASE WHEN value_type = 'text' THEN text_value
             WHEN value_type = 'numeric' THEN numeric_value::TEXT
             WHEN value_type = 'boolean' THEN boolean_value::TEXT
             WHEN value_type = 'date' THEN date_value::TEXT
             ELSE NULL END AS value
        FROM value, attribute
        WHERE value.attribute_name = attribute.attribute_name
    )
    SELECT person_id,
           string_agg(hstore(attribute_name, value)::TEXT, ',')::HSTORE AS hst
    FROM t GROUP BY member_id
)
UPDATE entity
SET attributes = hst
FROM u
WHERE u.person_id = member.person_id;
```

Exporting person information with *hstore*

```
COPY (
    SELECT email,
           attributes -> 'name_first' AS first_name,
           attributes -> 'name_last' AS last_name,
           attributes -> 'favorite_dbms' AS favorite_dbms
    FROM entity
    WHERE customer_id = 5432
) TO 'entity-5432.csv' (FORMAT CSV);
```

How fast is exporting person information with *hstore*?

Exporting member information is pretty fast

customer	persons	fields	values	EAV execution time	*crosstab()* execution time	*hstore* execution time
3	700,000	100	700,000	4 hours	10 minutes	15 seconds

Sorry, didn't time the other accounts... But now near the original export times!

Lessons learned

- Please try not to use the EAV data model - retrieving and inserting data is inefficient and performs poorly with under 1 million rows
- Do large data transformations in the database - Python code performance issues begins when pivoting millions+ of rows
- EAV increased on-disk usage by about 50% - varies by number of attributes
- If the EAV data model is already in use there are options to help get by:
  - *tablefunc* extension with the *crosstab()* data pivoting functions will help into millions of rows - in minutes
  - *hstore* data transformation can be done on-the-fly in seconds
- JSON was not yet a native data type in PostgreSQL® at the time of this adventure - similar experience expected as with *hstore*

Thank you!

Mark Wong
PostgreSQL® Major Contributor
markwkm@postgresql.org


EDB Software Development Engineer
mark.wong@enterprisedb.com