This talk explores why Postgres is uniquely capable of functioning as a central database in enterprises. *Title concept from Josh Berkus*

[https://momjian.us/presentations](https://momjian.us/presentations)  

Creative Commons Attribution License

Last updated: July, 2021
1. Object-relational (extensibility)
2. NoSQL
3. Data analytics
4. Foreign data wrappers (database federation)
5. Central role
Object-relational databases like Postgres support classes and inheritance, but most importantly, they define database functionality as objects that can be easily manipulated.
How Is this Accomplished?

http://www.postgresql.org/docs/current/catalogs.html
Example: ISBN Data Type

```
CREATE EXTENSION isn;
```

List of data types

<table>
<thead>
<tr>
<th>Schema</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public</td>
<td>ean13</td>
<td>International European Article Number (EAN13)</td>
</tr>
<tr>
<td>public</td>
<td>ismn</td>
<td>International Standard Music Number (ISMN)</td>
</tr>
<tr>
<td>public</td>
<td>ismn13</td>
<td>International Standard Music Number 13 (ISMN13)</td>
</tr>
<tr>
<td>public</td>
<td>issn</td>
<td>International Standard Serial Number (ISSN)</td>
</tr>
<tr>
<td>public</td>
<td>issn13</td>
<td>International Standard Serial Number 13 (ISSN13)</td>
</tr>
<tr>
<td>public</td>
<td>upc</td>
<td>Universal Product Code (UPC)</td>
</tr>
</tbody>
</table>

http://www.postgresql.org/docs/current/isn.html
ISBN Behaves Just Like Built-In Types

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>integer</td>
<td>-2 billion to 2 billion integer, 4-byte storage</td>
</tr>
<tr>
<td>isbn</td>
<td>International Standard Book Number (ISBN)</td>
</tr>
</tbody>
</table>
### The System Catalog Entry for INTEGER

```sql
SELECT * FROM pg_type WHERE typname = 'int4';
```

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>typname</td>
<td>int4</td>
</tr>
<tr>
<td>typnamespace</td>
<td>11</td>
</tr>
<tr>
<td>typowner</td>
<td>10</td>
</tr>
<tr>
<td>typlen</td>
<td>4</td>
</tr>
<tr>
<td>typbyval</td>
<td>t</td>
</tr>
<tr>
<td>typtype</td>
<td>b</td>
</tr>
<tr>
<td>typcategory</td>
<td>N</td>
</tr>
<tr>
<td>typispreferred</td>
<td>f</td>
</tr>
<tr>
<td>typisdefined</td>
<td>t</td>
</tr>
<tr>
<td>typdelim</td>
<td>,</td>
</tr>
<tr>
<td>typrelid</td>
<td>0</td>
</tr>
<tr>
<td>typelem</td>
<td>0</td>
</tr>
<tr>
<td>typarray</td>
<td>1007</td>
</tr>
<tr>
<td>typinput</td>
<td>int4in</td>
</tr>
<tr>
<td>typoutput</td>
<td>int4out</td>
</tr>
<tr>
<td>typreceive</td>
<td>int4recv</td>
</tr>
<tr>
<td>typsend</td>
<td>int4send</td>
</tr>
<tr>
<td>typmodin</td>
<td>-</td>
</tr>
<tr>
<td>typmodout</td>
<td>-</td>
</tr>
<tr>
<td>typanalyze</td>
<td>-</td>
</tr>
</tbody>
</table>
```

[http://www.postgresql.org/docs/current/catalog-pg-type.html](http://www.postgresql.org/docs/current/catalog-pg-type.html)
The System Catalog Entry for ISBN

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>typname</td>
<td>isbn</td>
</tr>
<tr>
<td>typnamespace</td>
<td>2200</td>
</tr>
<tr>
<td>typowner</td>
<td>10</td>
</tr>
<tr>
<td>typlen</td>
<td>8</td>
</tr>
<tr>
<td>typbyval</td>
<td>t</td>
</tr>
<tr>
<td>typtype</td>
<td>b</td>
</tr>
<tr>
<td>typcategory</td>
<td>U</td>
</tr>
<tr>
<td>typispreferred</td>
<td>f</td>
</tr>
<tr>
<td>typisdefined</td>
<td>t</td>
</tr>
<tr>
<td>typdelim</td>
<td>,</td>
</tr>
<tr>
<td>typreloid</td>
<td>0</td>
</tr>
<tr>
<td>typelem</td>
<td>0</td>
</tr>
<tr>
<td>typarray</td>
<td>16405</td>
</tr>
<tr>
<td>typinput</td>
<td>isbn_in</td>
</tr>
<tr>
<td>typoutput</td>
<td>public.isn_out</td>
</tr>
<tr>
<td>typepreceive</td>
<td>-</td>
</tr>
<tr>
<td>typesend</td>
<td>-</td>
</tr>
<tr>
<td>typmodin</td>
<td>-</td>
</tr>
<tr>
<td>typmodout</td>
<td>-</td>
</tr>
<tr>
<td>typanalyze</td>
<td>-</td>
</tr>
</tbody>
</table>

...
CREATE EXTENSION plpythonu;
\dL

<table>
<thead>
<tr>
<th>Name</th>
<th>Owner</th>
<th>Trusted</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>plpgsql</td>
<td>postgres</td>
<td>t</td>
<td>PL/pgSQL procedural language</td>
</tr>
<tr>
<td>plpythonu</td>
<td>postgres</td>
<td>f</td>
<td>PL/PythonU untrusted procedural language</td>
</tr>
</tbody>
</table>

http://www.postgresql.org/docs/current/plpython.html
Available Languages

- PL/Java
- PL/Perl
- PL/pgSQL (like PL/SQL)
- PL/PHP
- PL/Python
- PL/R (like SPSS)
- PL/Ruby
- PL/Scheme
- PL/sh
- PL/Tcl
- PL/v8 (JavaScript)
- SPI (C)

http://www.postgresql.org/docs/current/external-pl.html
Specialized Indexing Methods

- BRIN
- BTree
- Hash
- GIN (generalized inverted index)
- GiST (generalized search tree)
- SP-GiST (space-partitioned GiST)

http://www.postgresql.org/docs/current/indexam.html
Index Types Are Defined in the System Catalogs Too

```
SELECT amname FROM pg_am ORDER BY 1;
  amname
  --------
    brin
    btree
    hash
    gin
    gist
    spgist
```

http://www.postgresql.org/docs/current/catalog-pg-am.html
Operators Have Similar Flexibility

Operators are function calls with left and right arguments of specified types:

```
doS
  Schema   | Name   | Left arg type | Right arg type | Result type | Description
  ...     pg_catalog | +     | integer       | integer      | integer     | add
```

```
dfS
  Schema   | Name   | Result data type | Argument data types | Type
  ...     pg_catalog | int4pl | integer       | integer, integer | normal
```
Other Extensibility

- Aggregates are defined in `pg_aggregate`, `sum(int4)`
- Casts are defined in `pg_cast`, `int4(float8)`
Externally Developed Plug-Ins

- PostGIS (Geographical Information System)
- PL/v8 (server-side JavaScript)
- experimentation, e.g., full text search was originally externally developed
Offshoots of Postgres

- Aurora (Amazon)
- AsterDB
- Greenplum
- Informix
- Netezza
- ParAccel
- Postgres XC
- Redshift (Amazon)
- Truviso
- Vertica
- Yahoo! Everest

https://wiki.postgresql.org/wiki/PostgreSQL_derived_databases
http://de.slideshare.net/pgconf/elephant-roads-a-tour-of-postgres-forks
Offshoots of Postgres

https://raw.githubusercontent.com/daamien/artwork/master/inkscape/PostgreSQL_timeline/timeline_postgresql.png
Many databases treat extensions as special cases, with serious limitations. Postgres built-ins use the same API as extensions, so all extensions operate just like built-in functionality.
Extensions and Built-In Facilities Behave the Same

Postgres System Tables

- `sum()`
- `int4`
- `btree`
- `PL/pgSQL`

Extensions

- `ISN`
- `PostGIS`
- `PL/R`
2. NoSQL
There is no single NoSQL technology. They all take different approaches and have different features and drawbacks:

- Key-value stores, e.g., Redis
- Document databases, e.g., MongoDB (JSON)
- Columnar stores: Cassandra
- Graph databases: Neo4j
Why NoSQL Exists

Generally, NoSQL is optimized for:

- Auto-sharding
- Fast simple queries
- Flexible schemas
NoSQL Sacrifices

- A powerful query language
- A sophisticated query optimizer
- Data normalization
- Joins
- Referential integrity
- Durability
Are These Drawbacks Worth the Cost?

- **Difficult Reporting** Data must be brought to the client for analysis, e.g., no aggregates or data analysis functions. Schema-less data requires complex client-side knowledge for processing.

- **Complex Application Design** Without powerful query language and query optimizer, the client software is responsible for efficiently accessing data and for data consistency.

- **Durability** Administrators are responsible for data retention.
When Should NoSQL Be Used?

- Massive write scaling is required, more than a single server can provide
- Only simple data access pattern is required
- Additional resource allocation for development is acceptable
- Strong data retention or transactional guarantees are not required
- Unstructured duplicate data that greatly benefits from column compression
When Should Relational Storage Be Used?

- Easy administration
- Variable workloads and reporting
- Simplified application development
- Strong data retention
Postgres has many NoSQL features without the drawbacks:

- Schema-less data types, with sophisticated indexing support
- Transactional schema changes with rapid additional and removal of columns
- Durability by default, but controllable per-table or per-transaction
CREATE TABLE customer (id SERIAL, data JSONB);

INSERT INTO customer VALUES (DEFAULT, '{"name" : "Bill", "age" : 21}');

SELECT data->'name' FROM customer WHERE data->>'age' = '21';

----------
"Bill"
BEGIN WORK;
ALTER TABLE customer ADD COLUMN debt_limit NUMERIC(10,2);
ALTER TABLE customer ADD COLUMN creation_date TIMESTAMP WITH TIME ZONE;
ALTER TABLE customer RENAME TO cust;
COMMIT;
3. Data Analytics

- Aggregates
- Optimizer
- Server-side languages, e.g., PL/R
- Window functions
- Bitmap heap scans
- Tables
- Data partitioning
- Materialized views
- Common table expressions (CTE)
- BRIN indexes
- GROUPING SETS, ROLLUP, CUBE
- Just-in-time compilation (JIT)
- Parallelism
- Sharding (in progress)
Read-Only Replicas for Analytics

Tables from multiple clusters can be collected and synchronized on one cluster using logical replication, and a single table can be broadcast to multiple clusters too.
Foreign data wrappers (SQL MED) allow queries to read and write data to foreign data sources. Foreign database support includes:

- CouchDB
- Informix
- MongoDB
- MySQL
- Neo4j
- Oracle
- Postgres
- Redis

The transfer of joins, aggregates, and sorts to foreign servers is not yet implemented.
Foreign Data Wrappers to Interfaces

- JDBC
- ODBC
- LDAP
Foreign Data Wrappers to Non-Traditional Data Sources

- Files
- HTTP
- AWS S3
- Twitter
CREATE SERVER postgres_fdw_test
FOREIGN DATA WRAPPER postgres_fdw
OPTIONS (host 'localhost', dbname 'fdw_test');

CREATE USER MAPPING FOR PUBLIC
SERVER postgres_fdw_test
OPTIONS (password '');

CREATE FOREIGN TABLE other_world (greeting TEXT)
SERVER postgres_fdw_test
OPTIONS (table_name 'world');

\det
List of foreign tables
<table>
<thead>
<tr>
<th>Schema</th>
<th>Table</th>
<th>Server</th>
</tr>
</thead>
<tbody>
<tr>
<td>public</td>
<td>other_world</td>
<td>postgres_fdw_test</td>
</tr>
</tbody>
</table>

Foreign Postgres server name in red; foreign table name in blue
Read and Read/Write Data Sources

- Postgres
  - ora_tab
  - mon_tab
  - tw_tab

- MongoDB

- Twitter

- Oracle
Postgres can rightly take a central place in the data center with its:

- Object-relation flexibility and extensibility
- NoSQL-like workloads
- Powerful data analytics capabilities
- Access to foreign data sources

No other database has all of these key components.
Postgres’s Central Role