

Making Postgres Central in Your Data Center

BRUCE MOMJIAN



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>



Creative Commons Attribution License

Last updated: March 2025

Outline

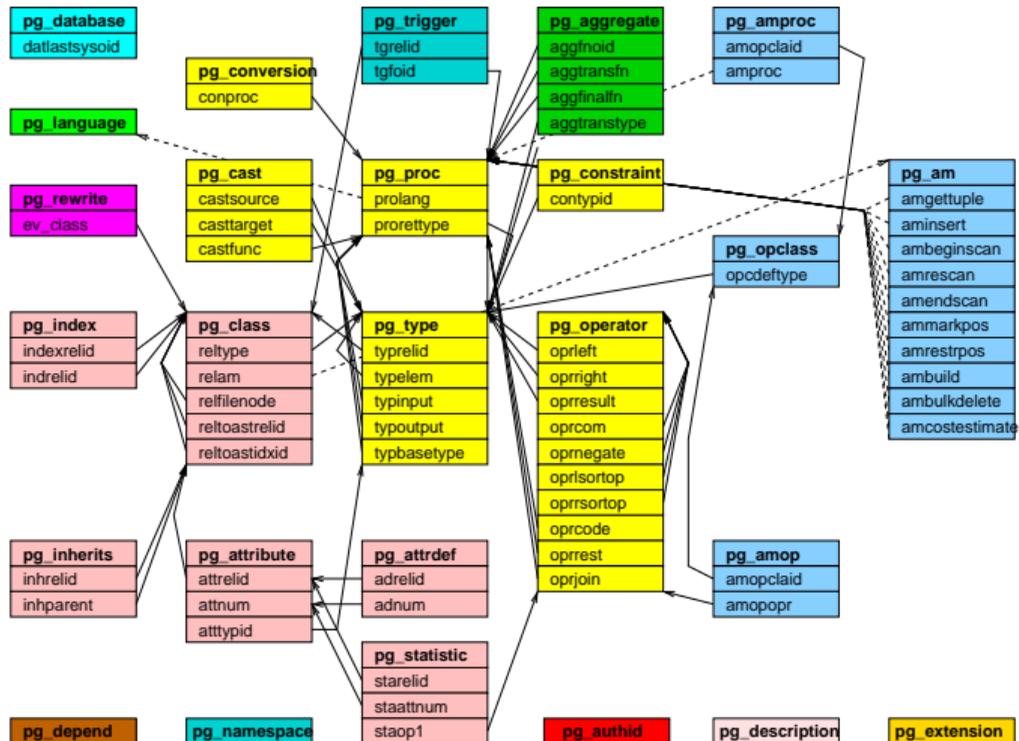
1. Object-relational (extensibility)
2. NoSQL
3. Data analytics
4. Foreign data wrappers (database federation)
5. Central role

1. Object-Relational (Extensibility)

Object-relational databases like Postgres support classes and inheritance, but most importantly, they define database functionality as objects that can be easily manipulated.

http://en.wikipedia.org/wiki/Object-relational_database

How Is this Accomplished?



<http://www.postgresql.org/docs/current/catalogs.html>

Example: ISBN Data Type

```
CREATE EXTENSION isn;
```

```
\dT
```

List of data types

Schema	Name	Description
public	ean13	International European Article Number (EAN13)
public	isbn	International Standard Book Number (ISBN)
public	isbn13	International Standard Book Number 13 (ISBN13)
public	ismn	International Standard Music Number (ISMN)
public	ismn13	International Standard Music Number 13 (ISMN13)
public	issn	International Standard Serial Number (ISSN)
public	issn13	International Standard Serial Number 13 (ISSN13)
public	upc	Universal Product Code (UPC)

<http://www.postgresql.org/docs/current/isn.html>

ISBN Behaves Just Like Built-In Types

```
\dTS
```

```
...
```

```
pg_catalog | integer | -2 billion to 2 billion integer, 4-byte storage
```

```
...
```

```
public    | isbn      | International Standard Book Number (ISBN)
```

The System Catalog Entry for INTEGER

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

```
-[ RECORD 1 ]--+-
```

typname	int4
typnamespace	11
typowner	10
typlen	4
typbyval	t
typtype	b
typcategory	N
typispreferred	f
typisdefined	t
typdelim	,
typrelid	0
typelem	0
typarray	1007
typinput	int4in
typoutput	int4out
typreceive	int4recv
typsend	int4send
typmodin	-
typmodout	-
typanalyze	-

```
...
```

The System Catalog Entry for ISBN

```
SELECT * FROM pg_type WHERE typname = 'isbn';
```

```
-[ RECORD 1 ]--+
```

typname	isbn
typnamespace	2200
typowner	10
typlen	8
typbyval	t
typtype	b
typcategory	U
typispreferred	f
typisdefined	t
typdelim	,
typrelid	0
typelem	0
typarray	16405
typinput	isbn_in
typoutput	public.isn_out
typreceive	-
typsend	-
typmodin	-
typmodout	-
typanalyze	-

```
...
```

Not Just Data Types, Languages

```
CREATE EXTENSION plpythonu;
```

```
\dL
```

List of languages

Name	Owner	Trusted	Description
plpgsql	postgres	t	PL/pgSQL procedural language
plpythonu	postgres	f	PL/PythonU untrusted procedural language

<http://www.postgresql.org/docs/current/plpython.html>

Server-Side Languages

- Included in the Postgres distribution
 - PL/Perl
 - PL/pgSQL
 - PL/Python
 - PL/Tcl
 - SPI
- External
 - PL/Haskell
 - PL/Java
 - PL/Lua
 - PL/R
 - PL/Rust
 - PL/sh
 - PL/v8

<http://www.postgresql.org/docs/current/external-pl.html>
https://wiki.postgresql.org/wiki/PL_Matrix

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>

<https://momjian.us/main/presentations/extended.html#indexing>

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)`

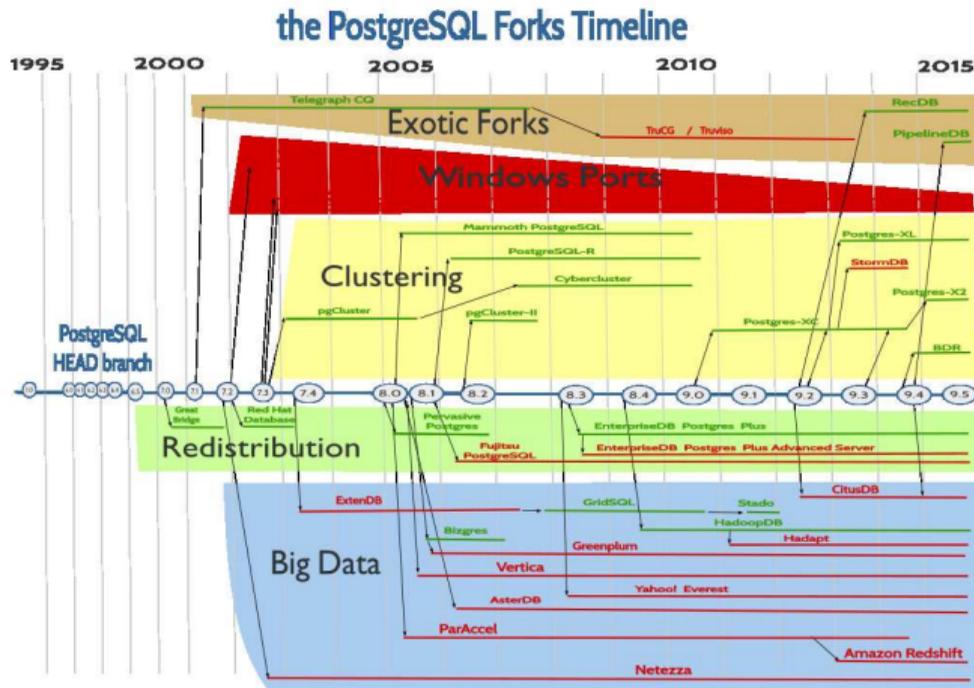
Externally Developed Plug-Ins

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

Offshoots of Postgres

- AlloyDB (GCP)
- Aurora (AWS)
- AsterDB
- CockroachDB
- EDB Advanced Server
- Greenplum
- Informix
- Netezza
- ParAccel, Redshift
- Postgres XC
- Truviso
- Vertica
- Yahoo! Everest
- YugabyteDB

Offshoots of Postgres



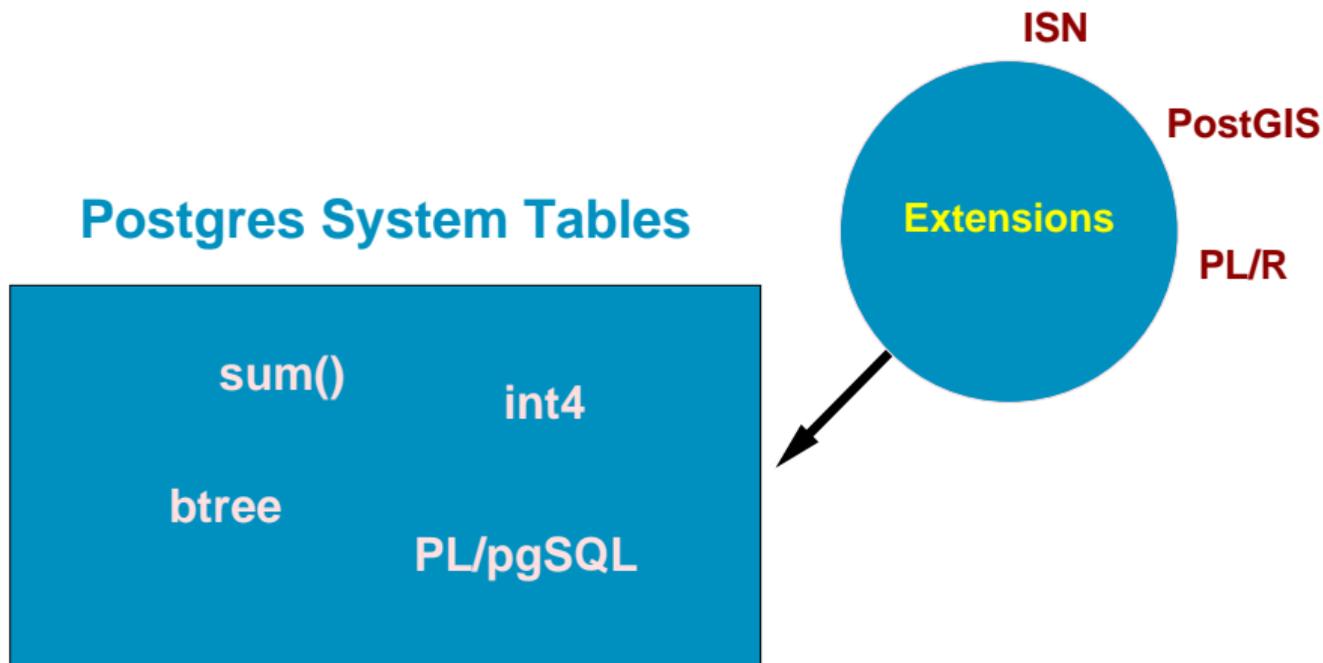
https://raw.githubusercontent.com/daamien/artwork/master/inkscape/PostgreSQL_timeline/timeline_postgresql.png

https://wiki.postgresql.org/wiki/PostgreSQL_derived_databases

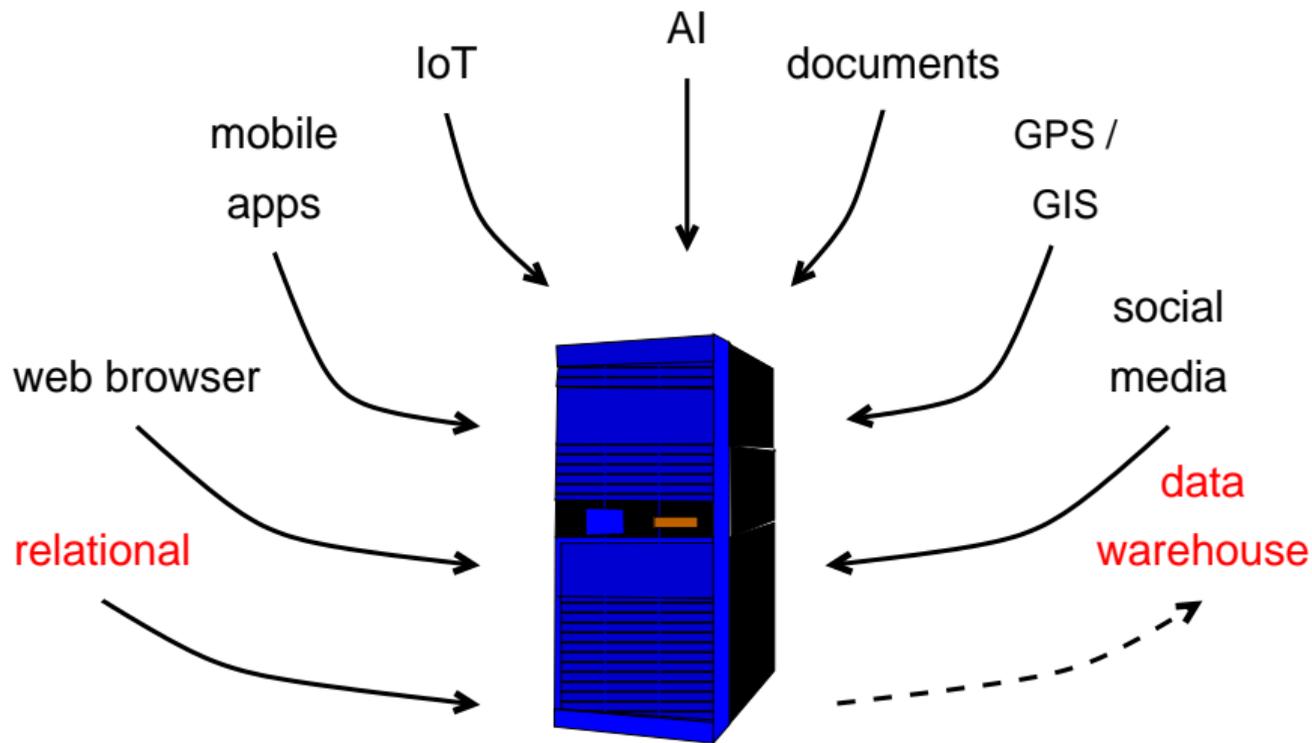
Plug-In Is Not a Bad Word

Many databases treat extensions as special cases, with serious limitations. Postgres built-ins use the same API as extensions, so extensions operate just like built-in functionality.

Extensions and Built-In Facilities Behave the Same



Why Extensibility Matters Today



Red indicates traditional relational workloads.

2. NoSQL



NoSQL Types

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

The Best of Both Worlds: Postgres

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

Schema-Less Data: JSONB

```
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';
```

```
?column?
```

```
-----
```

```
"Bill"
```

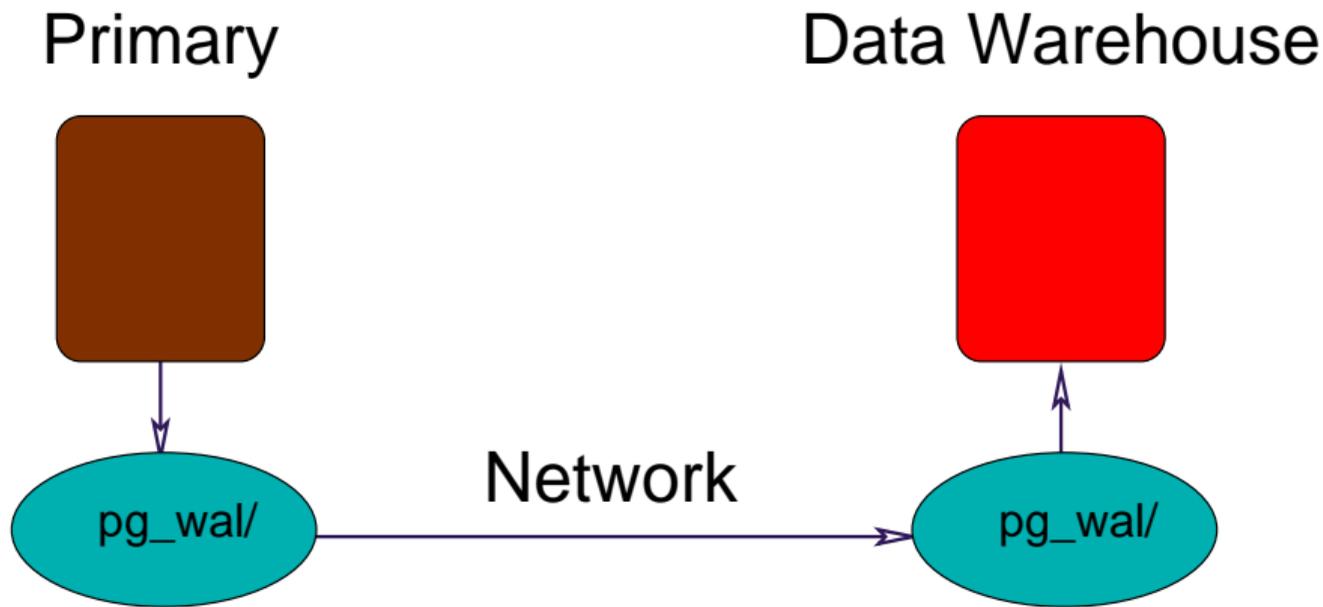
Easy Relational Schema Changes

```
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

- SQL
 - aggregates, GROUPING SETS, ROLLUP, CUBE
 - window functions
 - common table expressions (CTE)
 - server-side languages, e.g., PL/R
- Performance
 - optimizer, <https://danolivo.substack.com/p/does-postgresql-respond-to-the-challenge>
 - bitmap heap scans
 - BRIN and bloom indexes
 - materialized views
 - just-in-time compilation (JIT)
- Large data sets
 - data partitioning
 - tablespaces
 - 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.

4. Foreign Data Wrappers (Database Federation)

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

Many of these, e.g., the Postgres foreign data wrapper, can push down WHERE clauses, joins, sorts, and aggregates.

<http://www.postgresql.org/docs/current/ddl-foreign-data.html>
http://wiki.postgresql.org/wiki/Foreign_data_wrappers

Foreign Data Wrappers to Interfaces

- JDBC
- ODBC
- LDAP

Foreign Data Wrappers to Non-Traditional Data Sources

- Files
- HTTP
- AWS S3
- Twitter

Foreign Data Wrapper Example

```
CREATE EXTENSION postgres_fdw;
```

```
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
```

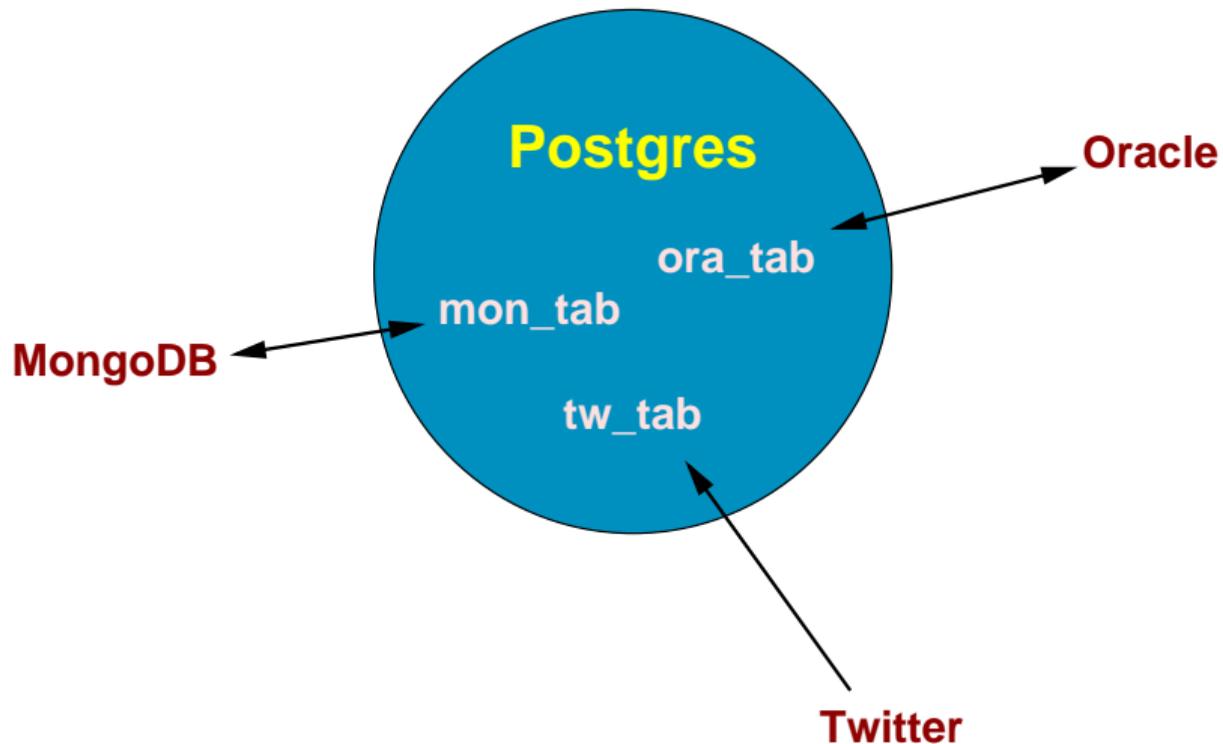
```
List of foreign tables
```

```
Schema | Table | Server
```

```
-----+-----+-----  
public | other_world | postgres_fdw_test
```

Foreign Postgres server name in red; foreign table name in blue

Read and Read/Write Data Sources



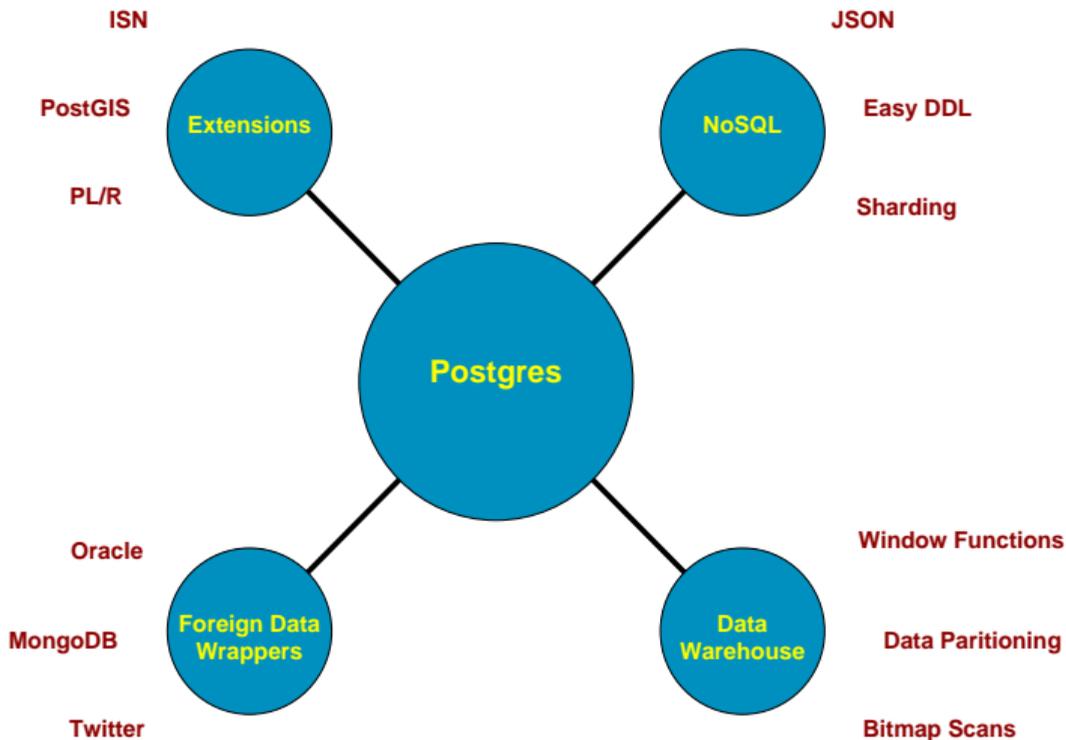
5. Postgres Centrality

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



Conclusion



<https://momjian.us/presentations>

https://www.flickr.com/photos/kenny_barker/