DESIGN OF POSTGRES

Michael Stonebraker and Lawrence A. Rowe
June 1986

By Bhagyasri Velpula

Agenda

- Abstract
- Motivation
- Background
- Design Goals
- Design Concepts
- Summary
- Future directions for this research
- My learning
- Q & A

Abstract

- Preliminary design of a new database management system.
- Design goals
  - provide better support for complex objects,
  - provide user extendibility
  - provide facilities for active databases
  - simplify the DBMS code for crash recovery,
  - make use of new technologies whenever possible.
  - make as few changes as possible to the existing relational model.
- Design concepts of POSTGRES and the basic system

Motivation

Motivation

- Need of additional data types to be integrated by database vendors making extension of the system impractical for end-users.
- To solve this problem required two additions to database architecture.
  - the ability to store arbitrary complex objects.
  - allow users to define their own types and the ways that those types are accessed
- requirement of modern databases is to support stored triggers and alerts for changes in data

Background

Background

- POSTGRES is the successor to the INGRES relational database system
- INGRES implemented during 1975-1977 at the University of California
- ‘‘hacked up enough’’ to make the inclusion of substantial new function extremely difficult
- proposed ideas would be difficult to integrate into that system because of earlier design decisions.
- Born the idea of building a new database system, called POSTGRES

Design Goals

1. Provide better support for complex objects

- Engineering data can be very complex and dynamic, unlike most of the business data
- Data can be represented in the existing data types in relational system but the performance of applications is unacceptable.
- Need a extensibility feature to store and access the data objects easily

2. Provide user extendibility for data types, operators and access methods

- allow new data types, new operators and new access methods to be included in the existing data database management system
- Should be easily accessible for the non-expert users
- which means easy-to-use interfaces should be preserved for any code that will be written by a user

3. Provide facilities for active databases (i.e., alerters and triggers) and inferencing including forward and backward-chaining

- Main idea is to support active databases and rules
- Developing applications become easier when we use alerters and triggers
- An alert is like a notification
- A trigger is part of the database and connected to a table, where it executes actions on insert, update or delete on that table.
- Example
  - Alerters – Bug reporting system
  - Triggers – Database consistency.
- Data in most of the application systems are operate on data that is described in rules rather than as a data values
- Its natural to store them as rules and infer them rather than storing them as data values and enforcing complex integrity constraints on them.

4. Simplify the DBMS code for crash recovery

- Problems with the existing crash recovery code
- Need of simple and easily extensible code to support crash recovery

- IDEA PROPOSED :

Treat the log as normal data managed by the DBMS which will simplify the recovery code and simultaneously provide support for access to the historical data.

5. produce a design that can take advantage of optical disks, workstations composed of multiple tightly-coupled processors, and custom designed VLSI chips,

- make use of new technologies whenever possible.
- Intended to support optical disks in the storage hierarchy
- workstation-sized processors with several CPU’s was the expected upcoming technology and this design of POSTGES should take the advantage of this
- Support the custom design VLSI chips

6. Make as few changes as possible (preferably none) to the relational model.

- Relational Model will be popular in the market and many business data processing users will need it , hence this framework should be preserved
- Many semantic models were existing but no small model was good enough to solve the current problems
- a new system should be built on a small, simple model that is extendible rather than large, complex data model,

DESIGN CONCEPTS

- design of POSTGRES
- the basic system architecture proposed

POSTQUEL

- Query language inherited from INGRES which is called QUEL
- many extensions and changes are made and the new language is called POSTQUEL
- The following built-in data types are provided;
  - integers,
  - floating point,
  - fixed length character strings,
  - unbounded varying length arrays of fixed types with an arbitrary number of dimensions, 5) POSTQUEL, and
  - procedure

Complex Objects

- Shared complex objects can be represented by a field of type POSTQUEL
- Basically stores only one copy of the procedure's source code in the database and it stores only one copy of the commands to fetch the data that represents the object.

Example:

POLYGON (id, other fields)

CIRCLE (id, other fields)

LINE (id, other fields)

Object relation can be defined like this

create OBJECT (name = char[10], obj = postquel)

create OBJPROC(name=char[12], proc=cproc)
append to OBJPROC(name=‘‘display-list’’, proc=‘‘...source code...’’)

execute (OBJPROC.proc)
with (‘‘apple’’)
where OBJPROC.name=‘‘displa

execute (OBJECT.obj)
where OBJECT.name=*argumen*

Name	OBJ
apple	retrieve (POLYGON.all) where POLYGON.id = 10 retrieve (CIRCLE.all) where CIRCLE.id = 40
orange	retrieve (LINE.all) where LINE.id = 17 retrieve (POLYGON.all) where POLYGON.id = 10

Figure 1. Example of an OBJECT relation.

Time varying Data

- Supports users to save and access historical data and versions
- Historical data can be accessed by indicating the desired time as shown

retrieve (E.all)
from E in EMP[‘‘7 January 1985’’]

Unwanted historical data can be deleted using DISCARD command specifying the cutoff point

discard EMP before ‘‘1 week’’

Time varying Data

- 3 level Memory hierarchy
- Main memory
- Secondary memory
- Tertiary memory
- Data location not needed for querying
- Supports version management

Ex: newversion EMPTEST from EMP

merge EMPTEST into EMP

Iteration Queries, Alerters, Triggers, and Rules

- an asterisk (‘‘*’’) used with a command for repetitive exceution

  retrieve* into SUBORDINATES(E.name, E.mgr) from E in EMP, S in SUBORDINATES
  where E.name=‘‘Jones’’ or E.mgr=S.name

- Alerters and triggers are specified by adding the keyword ‘‘always’’ to a query

  delete always DEPT
  where count(EMP.name by DEPT.dname

  where EMP.dept = DEPT.dname) = 0

defines a trigger that will delete DEPT records for departments with no employees.

- “Demand” keyword to introduced to define the priority of the rule

replace EMP (desk = ‘‘steel’’) where EMP.age < 40
replace EMP (desk = ‘‘wood’’ where EMP.age >= 40
replace demand EMP (desk = ‘‘wood’’) where EMP.name = ‘‘hotshot’’

replace demand EMP (desk = ‘‘steel’’) where EMP.name = ‘‘bigshot’’

PROGRAMMING LANGUAGE INTERFACE

- HITCHING POST is the programming language interface introduced
- Objectives
- Design a mechanism to make the development of browsing style applications simple
- All Programs that need to access the database could be written with the interface
- Allow application developer the flexibility to tune the performance of his program
- PORTALS

The portal can be thought of as a query plan in execution in the DBMS process and a buffer containing fetched data in the application process.

- Compilation and Fast-Path

improve the performance of the query using a system catalogue named “CODE” in which application programs can store queries that are to be compiled.

SYSTEM ARCHITECTURE

Process structure

- 2 types of process structures available
- Process per user module
- Server model

- POSTGRES uses process per user model because of limited programming resources

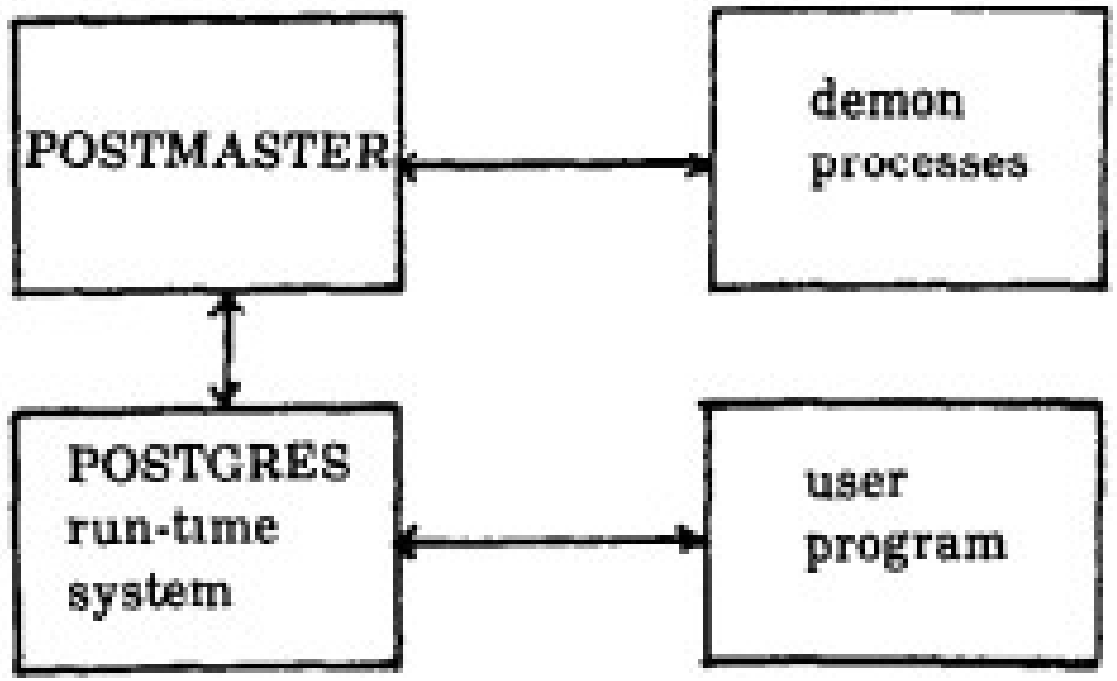


Figure 3 POSTGRES process structure

Query Processing

- Support for New types
- Support for procedural data
- Alerters , triggers and inference

Support for New types

- Existing access methods must be usable for new data types
- New access methods must be definable
- Table driven query optimizer
- When user defines new operators, optimizer will have the necessary information in catalogs to process them

Support for procedural data

performance tactic utilizes Pre-computing and caching the result of procedural data

2 steps for pre-computation

Compiling an access plan

Executing the access plan

	R	W	I
R	ok	no	ok
W	no	no	*
I	ok	no	ok

Support new kind of lock called I Lock

I Locks must be persistent, of fine granularity and escalatable.

Alerters , triggers

- Triggers and alerters use “always” command
- This command executes until the end of triggers or alerters
- Uses T-lock on all the objects which the “always” command read or write

	R	W	I	T
R	ok	no	ok	ok
W	no	no	*	#
I	ok	no	ok	ok
T	ok	no	ok	ok

Inference

Inferencing will be supported by “Demand” command

Implemented in the same way as always

Uses D lock on the objects

	R	W	I	T	D
R	ok	no	ok	ok	&
W	no	no	*	#	no
I	ok	no	ok	ok	ok
T	ok	no	ok	ok	ok
D	ok	no	*	#	ok

Storage Systems

Data format

Update and access rules

The POSTGRES Log

Transaction Management

Access methods

Vacuuming the disk

Version Management

Data format

- Every tuple has the following fields

  IID : immutable id of this tuple
  tmin : the timestamp at which this tuple becomes valid
  BXID : the transaction identifier that assigned tmin
  tmax : the timestamp at which this tuple ceases to be valid

  EXID : the transaction identifier that assigned tmax
  v-IID : the immutable id of a tuple in this or some other version
  descriptor : descriptor on the front of a tuple

Update and Access Rules

- During Insertion of tuple , tmin is marked with timestamp of insert transaction and its identity is recorded in BXID
- And during deletion of tuple, tmax is marked with the timestamp of delete transaction and its identity is recorded in EXID
- The update to a tuple is done by insertion followed by deletion as explained above

The POSTGRES Log

- Every new transaction has a new XACTID
- on committing a transaction, data is flushed out from the memory to a stable storage
- Then a single bit is written into the POSTGRES log

Transaction Management

- Soft crash – when a crash occurs and disk-based database is stable , it just rolls back to the previous commit.
- Usually these are instantaneous
- Hard Crash- when a crash occurs and disk is not stable , mirroring data base files on magnetic disk is provided
- “object locking” concept is introduced
- When two users attempt to access an object , the first executor will place a lock hence allowing consistency

Access Methods

- Supports collection of user defined indexes.
- Each index of this type is a pair of indexes one for the magnetic disk records
- Another for Archival records

Vacuuming the Disk

- In committed transactions , any record with BXID and EXID can be written on to optical disk or any long term repository

- In case of Aborted transactions , any records with BXID and EXID are simply flushed out.

- The vacuum demon takes care of these functionalities

Version Management

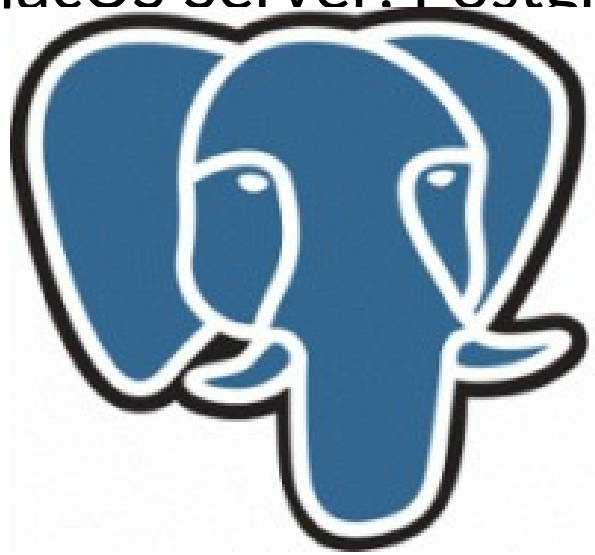
- Versions are implemented by allocating differential file for each separate version
- Any updates or delete operations are added to this file
- Indexes will be build on versions to refer the base relations on which these versions are created
- To merge the version changes into parent version following steps are performed
- if it is an insert, then insert record into older version
- if it is a delete, then delete the record in the older version
- if it is a replace, then do an insert and a delete

Summary

- POSTGRES proposes to support complex data objects, user-defined types, alerting and triggers, and time-varying data.
- In addition, Postgres has been continually evolving and is one of the leading open-source relational databases in the world.
- The future contribution of this paper has been immense.

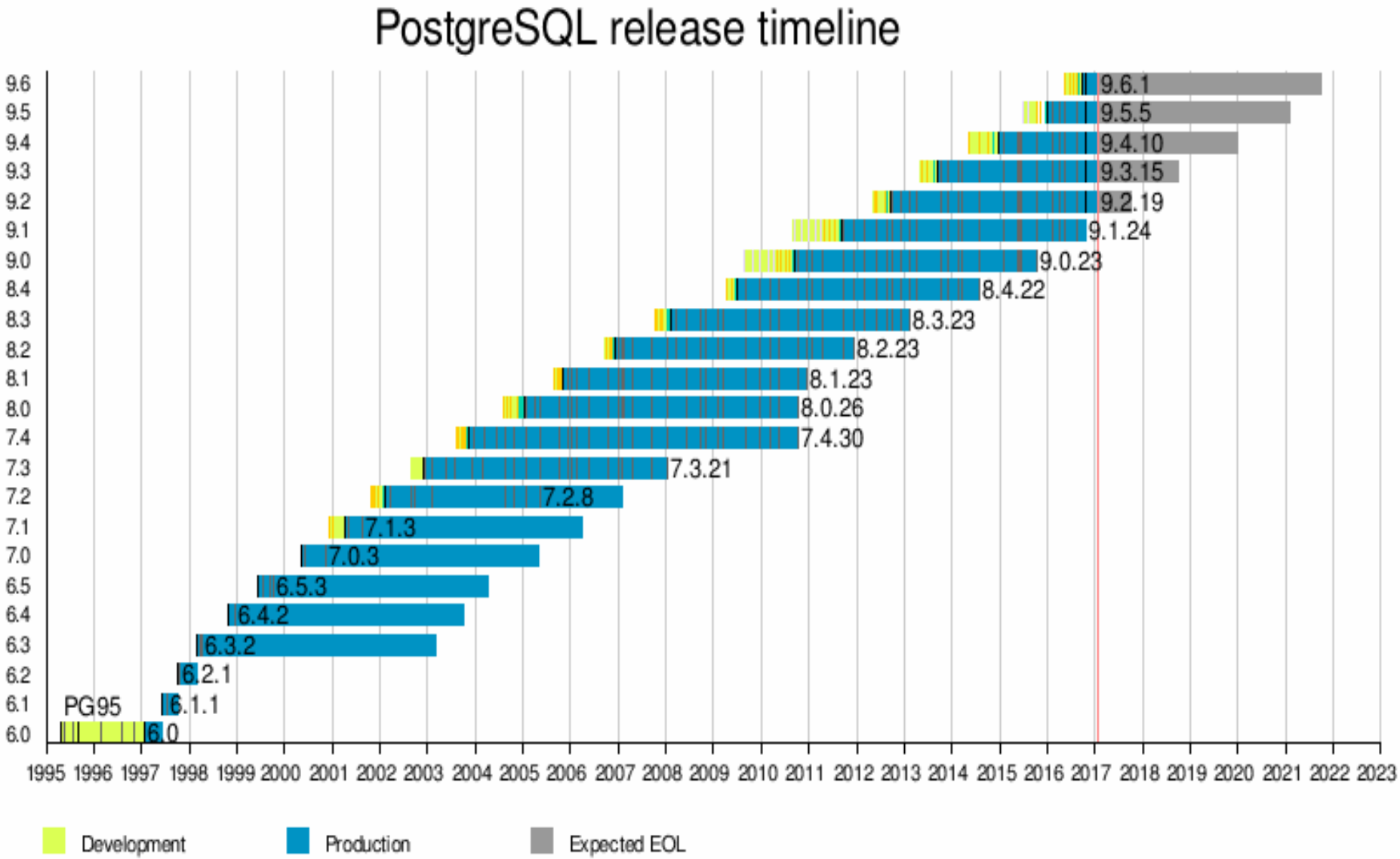
Future directions for this research

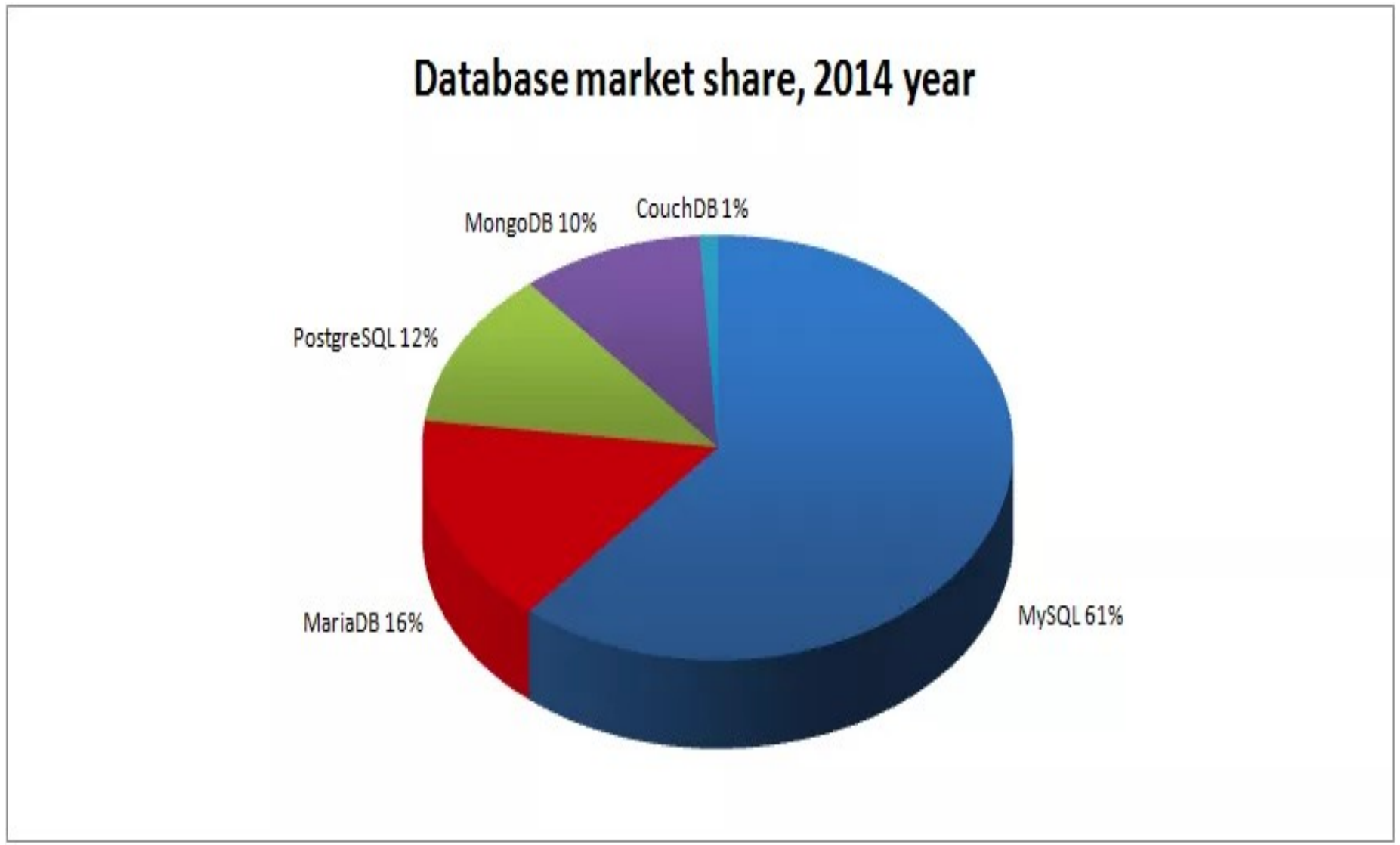
- The future directions of this paper have been realized with the open-source Postgres database.
- **Postgres ,** now **PostgreSQL** is an object-relational database
- can handle workloads ranging from small single-machine applications to large Internet-facing applications with many concurrent users
- on macOS Server, PostgreSQL is the default database
- also



PostgreSQL

the world's most advanced open source database


PostgreSQL release timeline
9.6
9.5
9.4
9.3
9.2
9.1
9.0
8.4
8.3
8.2
8.1
8.0
7.4
7.3
7.2
7.1
7.0
6.5
6.4
6.3
6.2
6.1
6.0
9.6.1
9.5.5
9.4.10
9.3.15
9.2.19
9.1.24
9.0.23
8.4.22
8.3.23
8.2.23
8.1.23
8.0.26
7.4.30
7.3.21
7.2.8
7.1.3
7.0.3
6.5.3
6.4.2
6.3.2
6.2.1
6.1.1
6.0
PG95
1995 1996 1997 1998 1999 2000 2001 2002 2003 2004 2005 2006 2007 2008 2009 2010 2011 2012 2013 2014 2015 2016 2017 2018 2019 2020 2021 2022 2023
Development
Production
Expected EOL
Updated 2017-01-29


Database market share, 2014 year
MongoDB 10%
CouchDB 1%
PostgreSQL 12%
MariaDB 16%
MySQL 61%

- Prominent organizations that use PostgreSQL as the primary database include:

My learning

- Architecting a database for extensibility gives a method for future sealing your database against change.

Q&A

References

- https://en.wikipedia.org/wiki/PostgreSQL
- https://www.postgresql.org/
- https://sookocheff.com/post/databases/the-design-of-postgres/