The Old Remains New
Evolution for Relevance
Core YottaDB® Database Technology

- Mature, high performance, hierarchical key-value, *language-agnostic*, NoSQL database whose code base scales up to mission-critical applications like large real-time core-banking and electronic health records, and also scales down to run on platforms like the Raspberry Pi Zero, as well as *everything in-between*.

The First Database – IMS

- Created by IBM & Rockwell to manage the Bill of Materials for the Saturn V rocket used in the Apollo missions.
- Key-value database
- Ran on a mainframe
- Programmed in IBM 360 Basic Assembly Language
The First Database Machine – MUMPS

- **Massachusetts General Hospital Utility Multi-Programming System**
- Created to manage life sciences laboratory records
- Key-value database
- Ran on a PDP-7
- Operating system + database (filesystem) + programming language + programming environment
Key-Value Tuples

["Capital","Belgium","Brussels"]
["Capital","Thailand","Bangkok"]
["Capital","USA","Washington,DC"]

Hierarchical Key

Single Value

Always sorted – you never have to say you’re sorting!
Schemaless

["Capital","Belgium","Brussels"]
["Capital","Thailand","Bangkok"]
["Capital","USA","Washington,DC"]
["Population","Belgium",13670000]
["Population","Thailand",84140000]
["Population","USA",325737000]

Default order for each key:
- Empty string ("")
- Canonical numbers in numeric order
- Strings (blobs) in lexical order

Schema determined entirely by application – database assigns no meaning

Numbers and strings (blobs) can be freely intermixed in values and keys except first key
Mixed Key Sizes

["Capital",”Belgium”,”Brussels”]  
[“Capital”,”Thailand”,”Bangkok”]  
[“Capital”,”USA”,”Washington, DC”]  
[“Population”,”Belgium”,13670000]  
[“Population”,”Thailand”,84140000]  
[“Population”,”USA”,325737000]  
[“Population”,”USA”,1790,3929326]  
[“Population”,”USA”,1800,5308483]  
...  
[“Population”,”USA”,2010,308745538]
Keys, Array References, (Sub)Trees

Population("Belgium")=13670000
Population("Thailand")=84140000
Population("USA")=325737000
Population("USA",1790)=3929326
Population("USA",1800)=5308483
...
Population("USA",2010)=308745538

First key is variable name
Other keys are subscripts

Array references are a familiar programming paradigm

Any JSON structure is representable as a tree, but not vice versa
Sharing and Persistence – Database Access

• Process private, available only for lifetime of process
  - Population("Belgium")
  - Population("Thailand")
  - Population("USA")
  “local” variables

• Shared across processes, persistent beyond lifetime of any process
  - ^Population("Belgium")
  - ^Population("Thailand")
  - ^Population("USA")
  “global” variables

Spot the difference?
Simple Standard Language

- 26 Commands, e.g., SET, KILL, XECUTE
- 24 Functions, e.g., $DATA(), $ORDER()
- 19 Intrinsic Special Variables, e.g., $ETRAP, $TEST
- At discretion of implementations
  - Anything beginning with Z
  - Deviceparemeters, Jobparameters
The Real Power of MUMPS

- **Database updates**
  
  ```plaintext
  set ^Cust(1234)="King|Martin|Luther"
  set ^Cust(1234,"Birthday")=20290115
  ```

- **Database access**
  
  ```plaintext
  set AcctId=$order(^Cust(AcctId))
  set NextCust=^Cust(AcctId)
  ```

- **Seamless coupling between a simple language and a simple database**

- **It Just Works™**
Today ... 1

- MUMPS → M (ISO/IEC 11756:1999)
- World’s largest real-time core-banking and electronic medical record systems are use M databases
- One major proprietary implementation
- Two major (related) FOSS (AGPL v3) implementations
  - GT.M (upstream) → YottaDB (downstream)
- Several others: smaller, hobbyist, pedagogic, POC, ...
Today ... 2

- M language standard abandoned by major proprietary implementation
- Languages are like religions
  
  “I’ll use your language if you use my language ... but you go first”

- C is a *lingua franca* of programming languages
YottaDB vision

- The database is more important than the language
- Seamless coupling between database and other languages is achievable
Data-Centric Architecture

YOTTADB DATA-CENTRIC ARCHITECTURE

- Additional applications
- Language wrapper (Go, Perl, Python, Rust, ...)
- C application programs
- C language API
- M language subsystem
- M application programs
- Octo SQL engine
- JDBC access
- Remote client
- Server for client server access
- YottaDB network protocol (TCP/IP)

Data organization, management, and storage engine
### Native C API

#### Sample Program
- `ydb_data_s()` / `ydb_data_st()`
- `ydb_delete_s()` / `ydb_delete_st()`
- `ydb_delete_excl_s()` / `ydb_delete_excl_st()`
- `ydb_get_s()` / `ydb_get_st()`
- `ydb_incr_s()` / `ydb_incr_st()`
- `ydb_lock_s()` / `ydb_lock_st()`
- `ydb_lock_decr_s()` / `ydb_lock_decr_st()`
- `ydb_lock_incr_s()` / `ydb_lock_incr_st()`
- `ydb_node_next_s()` / `ydb_node_next_st()`
- `ydb_node_previous_s()` / `ydb_node_previous_st()`
- `ydb_set_s()` / `ydb_set_st()`
- `ydb_str2zwr_s()` / `ydb_str2zwr_st()`
- `ydb_subscript_next_s()` / `ydb_subscript_next_st()`
- `ydb_subscript_previous_s()` / `ydb_subscript_previous_st()`
- `ydb_tp_s()` / `ydb_tp_st()`
- `ydb_zwr2str_s()` / `ydb_zwr2str_st()`
Native Lua API

Thank you to University of Antwerp’s Anet for contributing the native Lua API
local ydb = require('yottadb')
ydb.set('^hello', {'Lua'}, 'Hallo Wereld')
Tools to Access & Manipulate Data

● Languages are just tools
  – C, M, Go, JavaScript, Lua, Perl, Python, Rust
  – https://docs.yottadb.com/MultiLangProgGuide/
  – SQL too – https://docs.yottadb.com/Octo/

● 100% FOSS – mostly AGPL v3

● GNU/Linux on x86_64, ARM (64- and 32-bit)
More Information

- https://yottadb.com
- https://gitlab.com/YottaDB
- https://docs.yottadb.com
- https://www.uantwerpen.be/nl/projecten/anet/
- https://gitlab.com/YottaDB/Demo/performance-comparisons (performance comparison with Redis you can try yourself)
- K.S. Bhaskar – bhaskar@yottadb.com
Thank You!

YottaDB

Thank You!

yottadb.com