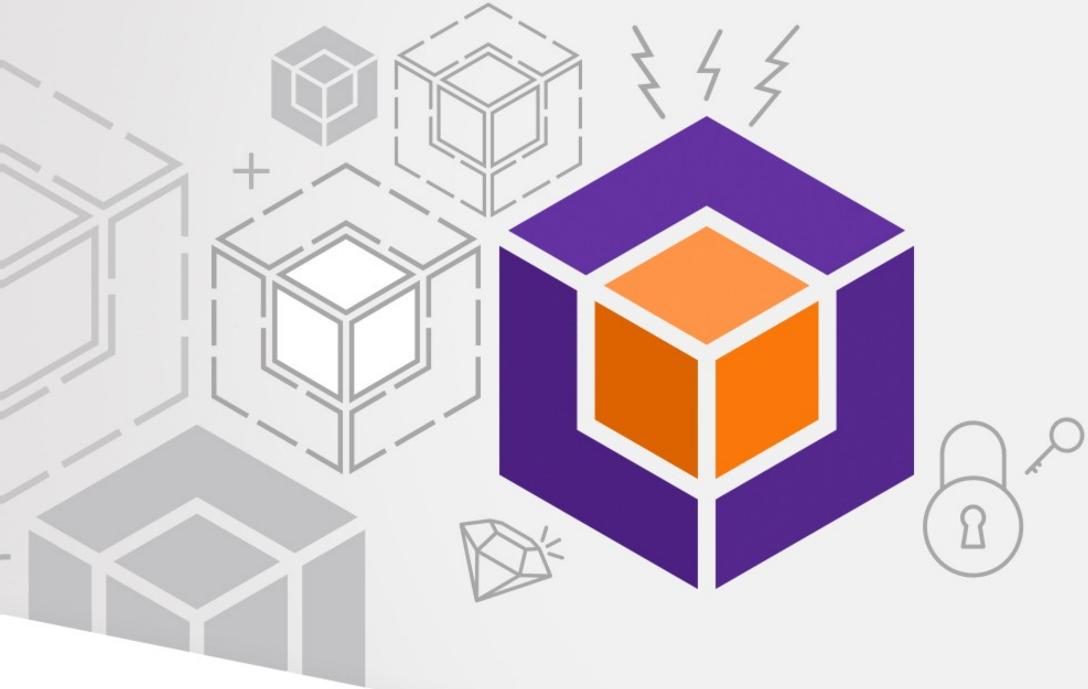
YottaDB

April 28, 2019



# When NoSQL isn't enough, SQL is too much

Charles Hathaway

# Outline

- Why NoSQL?
- Why SQL?
- How to SQL?

# What is SQL?

YottaDB

- Structured Query Language
  - What data to get, not how to get it
  - Ubiquitous for databases until NoSQL hit the streets in 2010's
  - Uses a relational schema; stores data in tables, tables can be related to each other, as indicated by a 'key' on the database

# What is SQL?

- Databases which rely on SQL as the primary means for accessing their data
  - Oracle
  - MySQL
  - Postgres
  - MS-SQL
  - Most SQL implementations are more-or-less uniform, with a standards committee

# What is SQL?

YottaDB

- Key statement types
  - SELECT – Get data from the database
  - INSERT – Put data into the database
  - DELETE – Remove data from the database
  - UPDATE – Update data in the database
- The most interesting is SELECT

# What is SQL?

- SELECT statements look like:
  - SELECT <select list> FROM <source> WHERE <condition> GROUP BY <columns to group by> HAVING <conditions of the group by> ORDER BY <columns to order by>
  - <source> can be a:
    - Table/View
    - SELECT statement
    - JOIN statement

# What is SQL?

- Joins are how we relate tables to each other
  - The most simple join is a “CROSS JOIN” which:
    - Creates a new table of size M x N, M is the number of rows in the first table and N is the number of rows in the second table
  - Most other joins are CROSS JOINs with conditions
    - INNER JOIN, NATURAL JOIN

# What is SQL?

- Joins are how we relate tables to each other
  - OUTER JOINs are the other type of join
    - LEFT, RIGHT, and FULL OUTER JOINs
    - Ways of fetching all data from one table, and empty data sets from another table
    - Often uses set operations (UNION, INTERSECT, DIFFERENCE) to construct resulting tables

# What is SQL?

- WHERE <condition>
  - Conditions on the rows we select from the database
  - Boolean expressions, which can contain arithmetic and functions
  - Can contain subqueries (additional SELECT statements)

# What is SQL?

- Summary
  - You specify what you want, and where you want it from
  - The SQL engine decides how to fetch the data
    - !! This is a hard problem!

# What is NoSQL?

- Databases that don't require SQL statements are called NoSQL databases
  - YottaDB
  - MongoDB
  - Redis

# What is NoSQL?

- Often requires the users to think about how to fetch the data, with unique API's for each system
- Often has unique structure to how data is stored; hierarchical, JSON/BSON, graphs
- Designing the schema for a NoSQL database has a very different process than designing for a SQL database; a lot more thought about how

# What is NoSQL?

- YottaDB is a NoSQL database
  - Data is stored as a hierarchy of key-value data
  - i.e., [“people”, “sanchez”, “rick”, “alive”]=”?”
  - Very good for data which has hierarchy
  - Where in SQL you would have a related table, you instead represent it as a “subscript” of the parent key

# Why NoSQL?

YottaDB

- Performance.
  - NoSQL has much less overhead in simple operations
  - Programmers have total control over execution, and therefore total control to utilize “meta data”
  - i.e., I know that there are many Rick Sanchez’s, so searching ones alive might be hard. Therefore, I maintain a cross reference of known alive people

# Why NoSQL?

- Generally, less-strict schema definitions
  - A blessing and a curse
  - Allows for easily adding items to the schema
  - Storage is different than storing tables consisting of rows; in some cases, this can save space

# Why NoSQL?

YottaDB

- In YottaDB's case, extremely fast transaction processing
  - With less overhead, committing "sets" of operations is more straightforward
  - Less overhead in tracking touched data

# Why SQL?

YottaDB

- Separation of concerns
  - Let the database people focus on making the database fast; let the application developers focus on making their application work
- Consistency of interfaces
  - As mentioned, SQL has a fairly regular syntax across vendors, with some small exceptions

# Why SQL?

- Tooling
  - Lots of tools connect to SQL engines and understand how to parse the results
    - Business Intelligence
    - Data Warehousing
  - Adapters for every major programming language on the market
  - Well-defined language API's

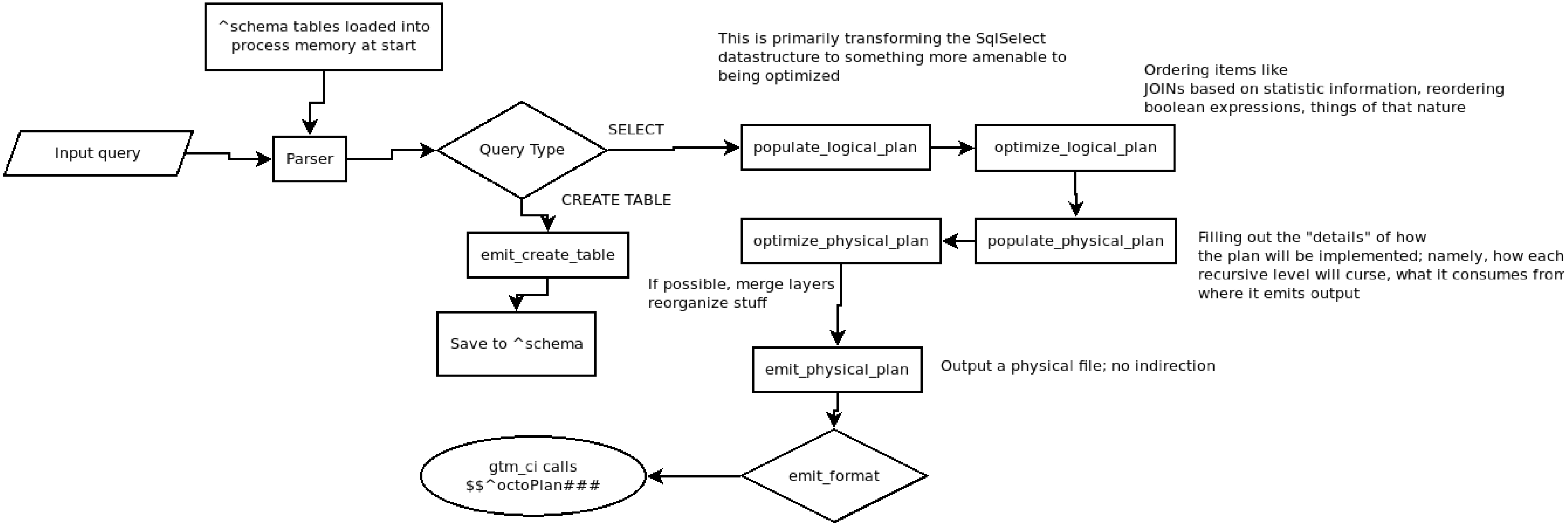
# SQL for NoSQL?

- Critics of NoSQL predicted we would be writing SQL engines for our NoSQL databases
  - They were right
  - Consider projects like nosqlbooster, Data Virtuality, rediSQL
  - And they're cocky about it (http://www.redbook.io/all-chapters.html)
    - "Declarative queries have returned as the primary interface to big data, and there are efforts underway in essentially all the projects to start building at least a 1980's-era optimizer"

# SQL for NoSQL?

- YottaDB wants SQL access too!
  - Customers have interest in the tools
  - We have some unique things we can use which are in-line with recent literature in the area
- Octo
  - SQL engine for accessing YottaDB datastores
  - Not-yet-released; early alpha state
  - Written in C, open to contributions

# How to SQL


^schema tables loaded into process memory at start
Input query
Parser
Query Type
SELECT
CREATE TABLE
emit_create_table
Save to ^schema
This is primarily transforming the SqlSelect datastructure to something more amenable to being optimized
populate_logical_plan
optimize_logical_plan
Ordering items like JOINs based on statistic information, reordering boolean expressions, things of that nature
optimize_physical_plan
populate_physical_plan
Filling out the "details" of how the plan will be implemented; namely, how each recursive level will curse, what it consumes from where it emits output
If possible, merge layers reorganize stuff
emit_physical_plan
Output a physical file; no indirection
emit_format
gtm_ci calls
$$^octoPlan###

# Octo design

- 3 main phases
  - Parse expressions (we use a YACC/Bison parser)
    - Once the expression is normalized, see if we can reuse a previously generated execution plan
  - Initial optimization pass
    - Resolve tables, columns, and order loops
  - Physical planning
    - Generate data structures that pretty much map to our compiled routines

# Octo design

- Running the query
  - Each SQL query gets transformed to a series of M programs
  - YottaDB knows how to compile M programs to object code, performance is very reasonable
  - Currently, everything is executed in a single process; near term, we will add in 'JOB's to allow parallel evaluation where possible

# Thinking about queries

- YottaDB stores data as a hierarchy
  - ^people("sanchez", "rick")="alive"
  - How do we represent this as a relational schema?
- CREATE TABLE people (firstName VARCHAR KEY NUM "0", lastName VARCHAR KEY NUM "1", alive VARCHAR);

# Thinking about queries

- YottaDB stores data as a hierarchy
  - ^people(“sanchez”, “rick”)=”alive”
  - How do we represent this as a relational schema?
- 3 basic operations we must perform
  - Fetching data
  - Iterating ordered data
  - Storing data

# Thinking about queries

- CREATE TABLE people (firstName VARCHAR KEY NUM “0”, lastName VARCHAR KEY NUM “1”, alive VARCHAR);
- How do we query against this table?
  - YottaDB provides ways to fetch data (get), set data (set), and iterate over nodes (subscript_next)
  - Some data is part of the “key” component here

# Thinking about queries

- CREATE TABLE people (firstName VARCHAR KEY NUM “0”, lastName VARCHAR KEY NUM “1”, alive VARCHAR);

  SELECT * FROM people
      FOR firstName in people
          FOR lastName in people(firstName)
              yield (firstName, lastName,
  people[firstName][lastName])

# Thinking about queries – Optimizations (equi join)

- SELECT * FROM people WHERE lastName = “Sanchez”
  lastName = “Sanchez”
  FOR firstName in people(lastName)
  yield (lastName, firstName, people[lastName][firstName])

# Thinking about queries – Optimizations (equi join)

- What about if we condition our query on something that isn't a key?
- SELECT * FROM people WHERE alive = "true"
- Option A: order over every row in the database, and only select those where alive = "true"
- Option B: construct a cross reference, and order over that instead

# Thinking about queries – Optimizations (equi join)

- SELECT * FROM people WHERE alive = “true”
- Cross reference looks like xref(“<table name>”,”<xref key>”,.. keys for table)
- i.e. xref(“people”,”true”,”sanchez”,”rick”)
  FOR fn in xref(“people”, “true”)
      FOR ln in xref(“people”, “true”, fn)
          yield

# Good Stuff - JOINs

- This is where we do the relational bit of relation databases
- Let's create a new table
  - CREATE TABLE morty(id INTEGER PRIMARY KEY, rickLastName VARCHAR, rickFirstName VARCHAR, alive VARCHAR)
  - rickLastName and rickFirstName are keys from the people table

# Good Stuff - JOINs

- Let's fetch the rick-morty pair
  - SELECT * FROM people p1 CROSS JOIN mortys m1
  - Render as:

```
FOR fn in people
    FOR ln in people(fn)
        FOR id in morty
            yield (...)
```

# Good Stuff - JOINs

- Let's fetch the rick-morty pair
  - SELECT * FROM people p1 CROSS JOIN mortys m1
    WHERE p1.firstName = m1.firstName
    AND p1.lastName = m1.lastName
  - Render as:
    FOR fn in people FOR ln in people(fn)
    mln = ln; mfn = fn;
    yield

# Good Stuff – SET operations

- UNION, INTERSECT, and EXCEPT
  - UNION is easy; just combine the output of two statements and remove duplicates
  - In YottaDB, we can maintain an output key and only add something to it if it doesn't exist
  - To get ordering correct, we can use a cross reference
    - i.e. xref("temp table","1",<ln>,<fn>)
      output("ln","fn")=1

# Good Stuff – SET operations

- UNION, INTERSECT, and EXCEPT
  - INTERSECT isn't too hard either; iterate the first table, populating output, iterate second table, copy each found element to output2, yield from output2

# Good Stuff – SET operations

- UNION, INTERSECT, and EXCEPT
  - EXCEPT can be simulated by deleting each value from output found in the second table

# Good Stuff – OUTER JOIN

- LEFT JOIN and RIGHT JOIN are OUTER JOINs
  - Consider LEFT JOIN; select all elements from table 1 and their corresponding elements in table 2, or if no such elements exist, a bunch of nulls
  - This can be done using SET operations!
  - (table1 INTERSECT table2) UNION (table1 EXCEPT (table1 INTERSECT table2))

# Optimizing the good stuff

YottaDB

- INNER JOIN, NATURAL JOIN are basically CROSS JOINs with conditions
  - We already saw an equijoin optimization
  - We can simply apply those to get JOINs that have no more cost than joining a single table
  - How do we handle conjunctions (AND)s?
  - How do we handle disjunction statements (OR)s?

# Optimizing the good stuff

- INNER JOIN, NATURAL JOIN are basically CROSS JOINs with conditions
  - We already saw an equijoin optimization
  - We can simply apply those to get JOINs that have no more cost than joining a single table
  - How do we handle conjunctions (AND)s?
  - How do we handle disjunction statements (OR)s?

# The Hard Problem

- Optimizing SQL queries is NP-hard
- We use heuristics to try and limit the scope of our search
  - Conveniently, storing the metadata for these heuristics looks almost identical to the way we store cross indexes
  - We are still working on this

# Section Title

Subtitle

*Thank You!*

yottadb.com