LogicBlox Smart database for next-generation applications

Making Databases Smarter with Optimization

April 11, 2014

MOTIVATION

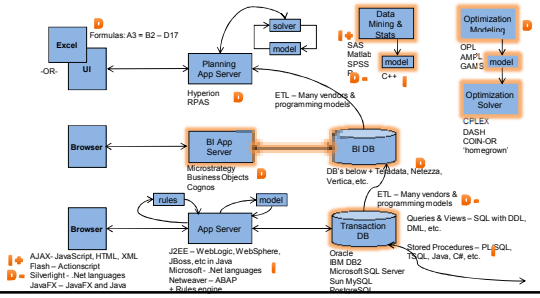
Enterprise Software: what and why

You are the Supermarket Emperor

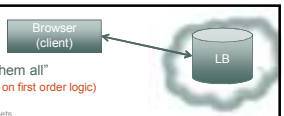
- A national store chain might have
  - A couple of hundred stores
  - Five large regional distribution centers [warehouses]
  - Hundreds of suppliers
- You are a large consumer of *enterprise software*
  - *Databases, reporting software, OLAP, …*
- You are a huge producer of data
  - Point of sale, inventories, purchase orders, shipping orders,…
- You really want useful, actionable information
  - *How many packs of frozen fish croquettes should I ship to the Olympia Emporium on Thursday, given that I have ordered 400 from the supplier 4 weeks ago (they will arrive to my Tacoma distribution center on Monday), assuming that I have some idea about how many packs of croquettes I am likely to sell there daily over the next 6 weeks?*
- So you commission some business intelligence software
  - … and quickly end up with

The Hairball

9+ Tech Stacks with 12+ different programming languages = HAIRBALL



Is there a better way? LB?



- Radical simplification: "one platform to rule them all"
  - A single expressive declarative language (based on first order logic)
    - Record keeping, querying
    - User interfaces: logical relations between models of widgets
    - Machine learning: the database "stores" complex objects such as nearest neighbors, probability densities, classification functions as if they were any other data and takes care of interacting with sophisticated machine learning algorithms
    - Optimization/mathematical programming
  - **May not do everything superbly but does everything well (enough)**
  - Investment in the underlying engine and platform
    - Support for transactions: atomicity, consistency, isolation, durability
    - View maintenance: changes to data are propagated to calculations (views) that depend on them
    - Very sophisticated query optimization, non-blocking concurrency, parallelism and out of core computation
  - On demand "platform as a service" automatically provisioning machines on the cloud
- Non-technical environment can be greatly simplified
- And if that is not good enough for the application at hand, there's always the hairball for someone else
  - In the enterprise software area it's often been quite good enough

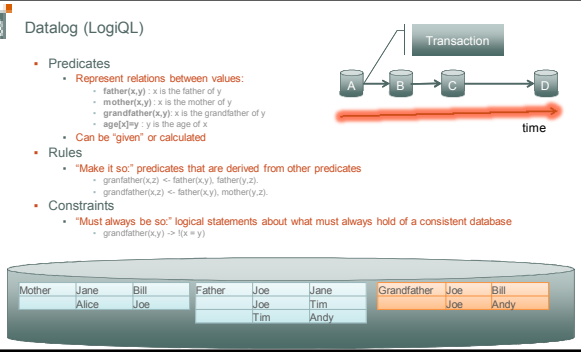
THE REST OF THIS TALK

- A Four Slide Introduction to LogiQL Databases
- A quick crash course in Mathematical Programming
- A Real-world Example of Retail Supply Chain Network
- Concluding remarks

A Four Slide Introduction to LogiQL

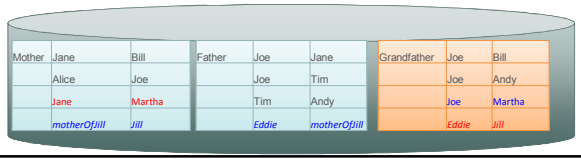
Datalog (LogiQL)

- Predicates
  - Represent relations between values:
    - **father(x,y)** : x is the father of y
    - **mother(x,y)** : x is the mother of y
    - **grandfather(x,y)**: x is the grandfather of y
    - **age[x]=y** : y is the age of x
  - Can be "given" or calculated
- Rules
  - "Make it so:" predicates that are derived from other predicates
    - grantfather(x,z) <- father(x,y), father(y,z).
    - grandfather(x,z) <- father(x,y), mother(y,z).
- Constraints
  - "Must always be so:" logical statements about what must always hold of a consistent database
    - grandfather(x,y) -> !(x = y)



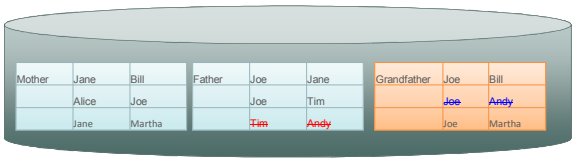
Rules are incrementally maintained by the database

- Insertion: adding a fact that Jane is Martha's mother, results in insertion and update to the 'Grandfather' predicate asserting that Joe is Martha's Grandfather
  - Rules are evaluated minimally – only re-computing additions / removals to predicates
  - Atomicity, consistency, isolation and durability of transactions is automatically maintained
- Rules can talk about state/changes (not that important for our talk)
  - +mother(m, y), +father(x,m)
    <- +grandfather(x,y), !has_parent(y), m="motherOf"+y.
    has_parent(x) <- mother(_,x) ; father(_,x).



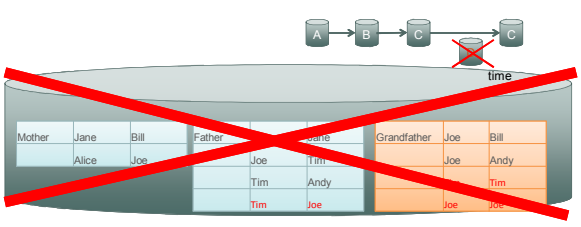
Deletion

- Removal of a fact describing the relation between Tim and Andy from predicate Father, causes the database to remove the relationship between Joe and Andy in the predicate Grandfather



Constraints

- Asserting that both Tim is Joe's father and Joe Tim's violates the constraint that no one can be his own grandfather: the system aborts the transaction and the database is restored to the original state prior to the addition of the violating facts.



Mathematical Programming (AKA Optimization)

Mathematical (Linear) Programming

- A potter can make either plates or cups with different profits from each ($1.07 and $1.05 respectively). Each plate takes 0.24 hours to make and each cup 0.2 hours. The total time available to the potter is 11 hours. How much should she make given that she has already existing orders of 4 plates and 8 cups.
- Maximize value of the *objective function* *o(x,y)* = 1.07 * x + 1.05 * y
- Subject to the conditions that:
  - 0.24 * x + 0.2 * y <= 11
  - x >= 4
  - y >= 8
  - x,y integer

Profit	Plate	1.07
	Cup	1.05

Time	Plate	0.24
	Cup	0.2

AvailableTime	11

Order	Plate	4
	Cup	8

Make	Plate	x
	Cups	y

Make	Plate	5
	Cups	49
TotalProfit	56.8	
TotalHours	11	

Mathematical (Linear) Programming

- You can pretty quickly get more elaborate
- Maximize value of the *objective function* *o(x,y)* = 1.07 * x + 1.05 * y
- Subject to the conditions that:
  - 0.24 * x + 0.2 * y <= 40
  - x,y integer
  - x1 + x2 + x3 + x4 + x5 = x
  - y1 + y2 + y3 + y4 + y4 = y
  - 0.24 * x2 + 0.2 * y2 <= 4 (dentist appointment)

Profit	Plate	1.07
	Cup	1.05

Time	Plate	0.24
	Cup	0.2

AvailableTime	11

Order	Plate	4
	Cup	8

Make	*(Product)*	*(Day)*	*(Amount)*
	Plate	1	x1
	Cups	1	y1
	… …	… …	… …

Make	Plate	
	Cups	
TotalProfit		
TotalHours	40	

Mathematical Programming

- All about maximizing (minimizing) value some function subject to constraints
  - Discrete vs continuous (integer, binary, continuous)
  - Forms of functions: linear, quadratic, non-linear and so on
  - Different classes have different algorithmic difficulty
- Constraints are expressed as inequalities featuring variables ranging over numeric quantities (e.g., *x* and *y* above)
- Objective function is an expression involving some subset of the variables
- Efficient and effective solvers exist for solving mixed integer (linear constraints, linear objective, and some variables integers)
  - Solve problems with (many many) millions of constraints and variables
  - Usually highly specialized (commercial) software
- Interfaced through **modeling languages**
  - How do we write families of related mathematical programming problems
    - For example, can we generalize to multiple products (plates, cups, etc.) and write single problem that depends on input values such as profits per product etc.

The Main Idea

LB Databases can store variables, not just values

Same Problem in LogicBlox

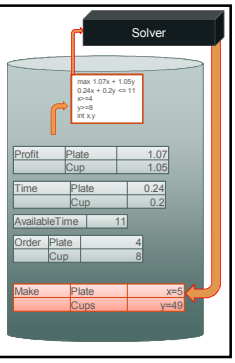
- Rules:

objective[] += profit[p] * make[p].
total_time[] += time[p] * make[p].

- Constraints

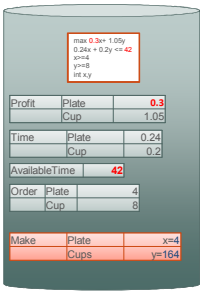
make[p] =v -> v >= order[p].
total_time[]=v
-> v <= available_time[].

- And we declare the predicate **make** as a special kind of predicate
  - The system assumes that any value that exists in the second column of make is an unknown (variable) which must obey the constraints
  - The system will "fill in" the optimal values for the variables such that the objective is function is maximized and the constraints satisfied



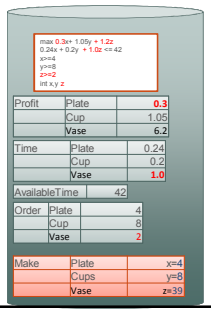
Change some values stored in predicates

- The system will
  - Automatically re-synthesize a new system of inequalities
  - Invoke the solver to obtain new solution values
  - Update the predicate **Make**
- So that
  - At the end of the transaction that changed the values in red
  - The original semantics is maintained: the predicate Make still contains the optimal solution to the constraints that maximizes the objective function
- Benefits
  - Interoperation with solver 100% hidden from programmer
  - Support for multiple solver back ends
  - Built-in optimization of the instance part of the platform
  - Maintenance ensures that all rules, values etc. that depend on the predicate **Make** get recalculated to obtain consistency



And automagically get an updated solution

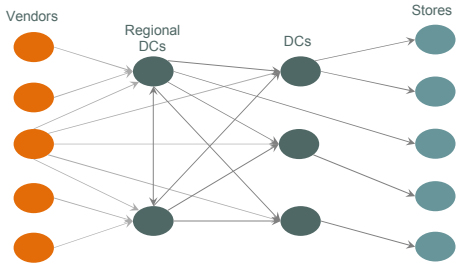
- We can even add more facts. Suppose we the potter can also make vases. We simply insert the relevant facts into the database
- Invariant
  - At any given time (i.e., at the end of any transaction) the database is guaranteed to contain a set of optimal values for the second column of the predicate Make or, if a solution cannot be found, the transaction is aborted, and the database restored to its previous state



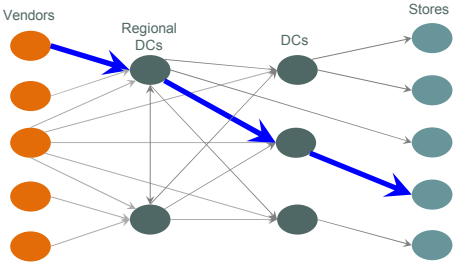
RETAIL SUPPLY CHAIN NETWORK

A Real-world Example

RETAIL SUPPLY CHAIN NETWORK



FOCUS ON ONE SKU-VENDOR-STORE PATH



REPLENISHMENT SUPPLY CHAIN NETWORK

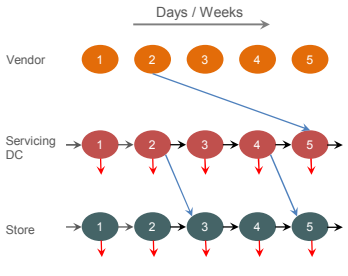
Network

- Each node represents SKU-location-time period
- Each arc represents merchandise flowing:
  - Red: demand
  - Blue: replenishment order
  - Black: inventory

Unit costs & prices

- Each unit flowing on an arc has a cost or revenue:
  - Red: unit price
  - Blue: procurement, transportation, handling
  - Black: holding

Our objective is to find flows that meet demand, maximize profit, and satisfy business constraints

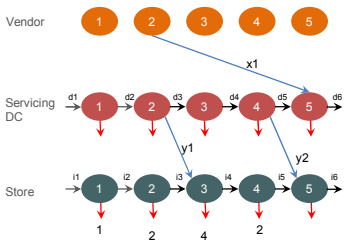


Modeling Network Flows

- What comes in must go out

profit =
2.50 * (1+2+4+2) – (1.05 * x1 + 0.05 * y1 + 0.05 + y2)
- 0.005(i1 + … + i6) – 0.002 * (d1 + … + d6)

d1 = d2
d2 = d3+y1
d3=d4
d4=d5+y2
d5+x1=d6
i1=i2+1
i2=i3+2
i3+y1=i4+4
i4=i5+2
i5+y2=i6



OBJECTIVE FUNCTION

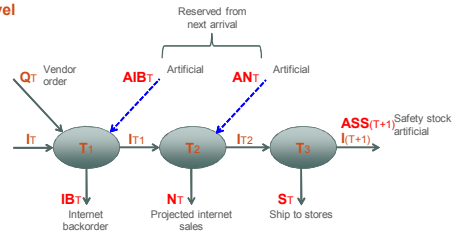
- Maximize profit
- Revenue
  - Units sold (forecasting, seasonality, promotions)
  - Unit selling price (typically varies)
  - Lost sales = (shortages – backorders) * unit selling price
  - Transforming a portion of shortages into backorders
  - Loss of goodwill due to shortages
- Cost
  - Procurement cost = quantities bought * unit purchasing price (incorporate quantity discounts and special vendor agreements)
  - Transportation cost = shipped quantities * unit shipping cost (can incorporate shipping in full containers or trailers)
  - DC and store inventory holding cost
  - Warehouse handling costs
  - Capital cost
  - Penalty costs
  - Many more

CONSTRAINTS AND BUSINESS RULES

- Network infrastructure
  - Vendors, DCs, cross-docks, stores
  - Order placement dates
  - Lead times, truck schedules
- Shipping multiples (pack size)
- Flow conservation equations
- Service level, safety stock
- Display stock
  - Sellable, non-sellable
  - Primary, secondary
- Inventory should not fall below
  - Safety stock
  - Primary and secondary display
  - Soft constraints
- Prioritization in case of constrained supply
  - Backorders
  - Store and internet sales
  - Display stock
  - Safety stock

NODE SPLITTING AND ARTIFICIAL VARIABLES

DC level



$I_{(T+1)} + ASS_{(T+1)} >= SS_{(T+1)}$

NUMBER OF VARIABLES

- Number of inventory variables alone for a typical medium size retail replenishment problem is huge:
  - Active SKUs: 50,000
  - Number of stores: 500
  - Number of periods: 100 (assuming modeling the first 6 weeks at the daily level and the following 420 days at the weekly level)
  - Number of inventory variables:

50,000 * 500 * 100 = 2.5 billion variables
In fact a lot larger (cca 20+ variables per node)

- Use of the cloud, parallelization, and powerful dis-aggregation and aggregation techniques enable us to formulate and solve very large problems

Pragmatics: the devil, the details

- So far, all of the above can be modeled pretty easily with modern MIP solvers
- We represent the structure of the network with predicates and rules
  - Turns out to be a very convenient formalism for specifying very large set of complicated constraints
- Creating problem instances is a challenge
  - Problem instances may be too large to fit into memory
  - Luckily, we happen to have a database that can do out-of-core computation:
    - Represent the instance itself as a set of predicates coefficient[row,column]=v
    - Generate lots of LogiQL code (invisible to the user) to construct it and marshal it to the solver
    - Generate rules that rewrite the instance (decomposition, aggregation, variable elimination) to obtain smaller instances to solve
  - Incremental update of instances
    - If only a single order change update only a small portion of the instances and solver data structures
- Complex business rules are surprisingly easy to add
  - Often just extra LogiQL constraints:
    - "sum of all the fish ordered from the vendor during months which are marked as no fishing season should be 0"
    - the total cubic footage ordered must be some multiple of the volume of a shipping container
    - you should never have more than 20 tons of fish at any distribution center
- Computation time becomes very expensive
  - Amazon EC2 instance with 80G of memory a lot more expensive than one with 24G and we need hundreds or thousands of those daily

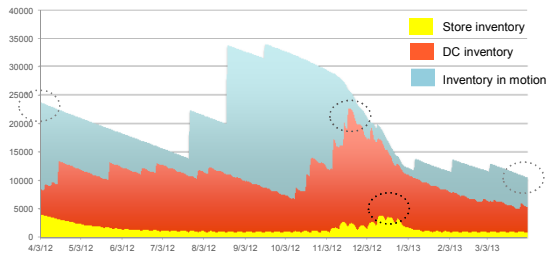
SUPPLY CHAIN NETWORK OPTIMIZATION IN PRODUCTION

Input data
SKUs, DCs, stores, vendors, historical demand, DC calendar, store calendar, ordering DCs, initial conditions

Input Parameters
Demand forecast, price, procurement cost, transportation cost, holding cost, cost of capital, fill rate, safety stock

Outputs
Orders, Reports, Charts

SYSTEM-WIDE SKU INVENTORY



Conclusions

- Who would have thought that something as so boring sounding as “enterprise software” could be so much fun to work on?!
- Fancy Booklearnin’: Do I use anything I learned along the way?
  - Mr Kunic, my elementary school teacher: 2x=4 → x = 2
  - Undergraduate Evergreen [Quite a lot]
    - Judy’s Database Class: I learned about datalog [In fact, Judy’s advisor invented it]
    - C&C I learned about 1st Order Logic, models thereof, mathematical logic and Prolog [Al Leisenring]
  - OGI: Programming Languages, Functional Programming, Type Theory
    - More about Logic [basis of Datalog]
    - Much of what I do to make the Mathematical Programming work at LB is translation from one language to another
  - Rice: Postdoc
    - Started working on implementing dynamic programming and linear algebra numerical algorithms – came in handy with optimization
    - Theorem proving (Coq) : still a dream – Can we use a proof assistant to write our constraints, and actually prove safety, coherence and convergence properties of our MP models a priori.

THANK YOU