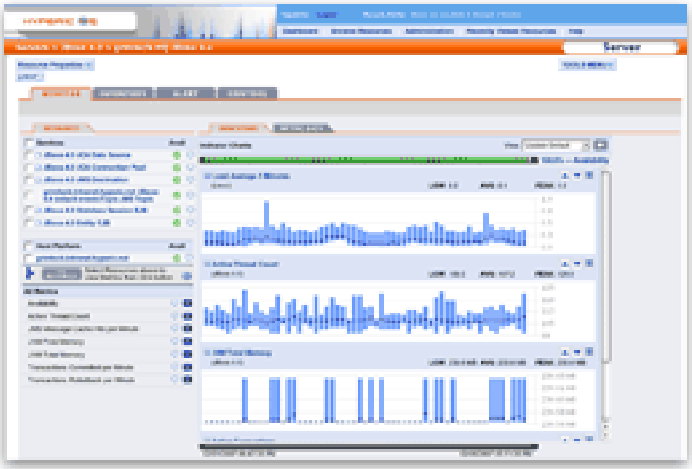
Scaling Java and PostgreSQL to Great Heights

Charles Lee

Hyperic

What is Hyperic HQ?



Hyperic HQ is the industry's only comprehensive product that provides cross-stack visibility for software in production, whether it's open source, commercial, or a hybrid.

Translation: HQ collects and transactionally read and write a lot of data

Just How Much Data?

Scenario: IT infrastructure of 100 servers (medium size deployment)

- 100 Platforms
- 700 Servers
- 7000 Services
- 150,000 metrics enabled (20 metrics per resource)
- 15,000 metric data points per minute (average)
- 21,600,000 metric data rows per day

Distribution of HQ

- All inclusive installer package (JRE, JBoss, HQ, embedded database)
- PostgreSQL embedded - easy license, cross-platform support, enterprise performance
- Works with PostgreSQL, Oracle, or MySQL backends

Application Performance Bottleneck

It’s the Database

- Dependent on hardware performance (disk, CPU, memory, etc)
- I/O
- Network latency (remote database)
- Slow queries

Breaking Through the Bottleneck

Case Study: Hi5.com (top 15 visited website)

Using Hyperic 2.6

- Upgrade I/O (DAS/NAS/SAN)
- Upgrade H/W (64-bit multi-processors, increased RAM, etc)
- Upgrade to PostgreSQL 8.2.4
- Upgrade to 64-bit JRE 6

Anything else we can do?

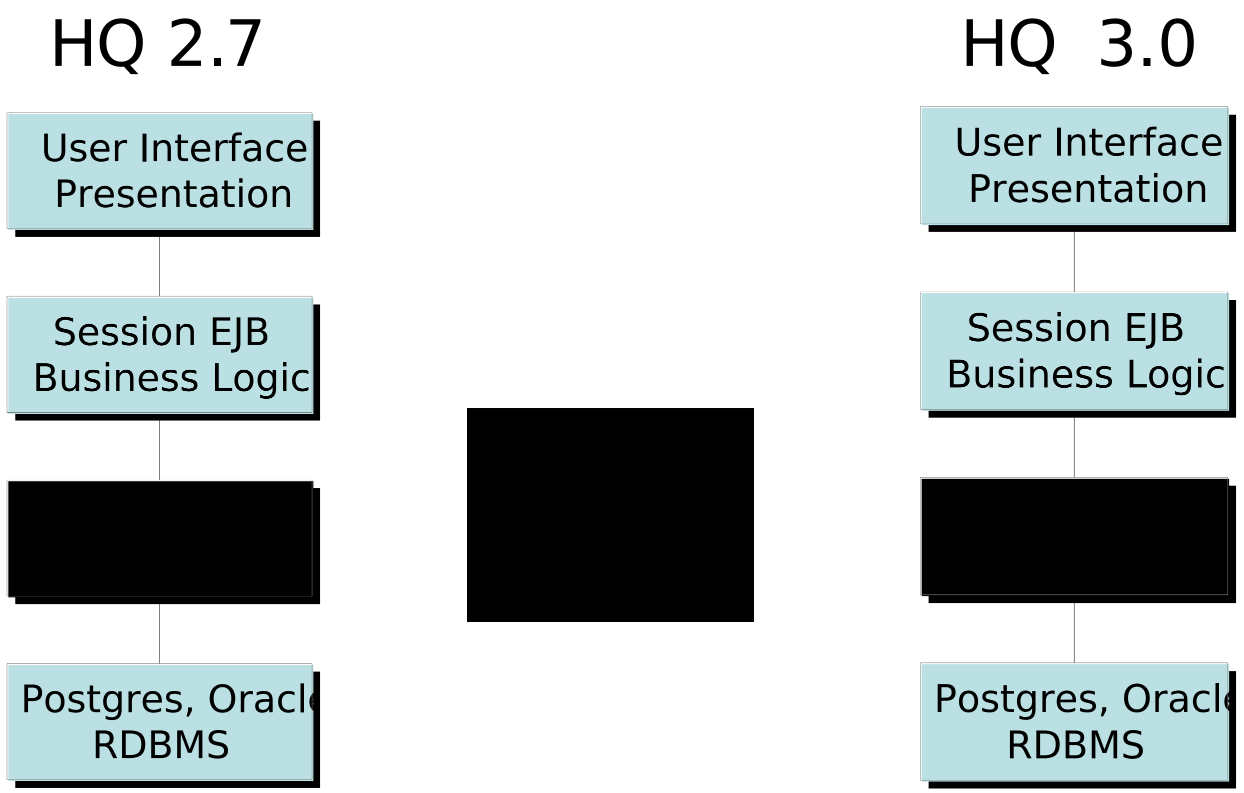
Data Access Object (or DAO)

The Data Access Object (or DAO) pattern definition from Sun:

- separates a data resource's client interface from its data access mechanisms
- adapts a specific data resource's access API to a generic client interface

The DAO pattern allows data access mechanisms to change independently of the code that uses the data.

The Hyperic HQ Stack



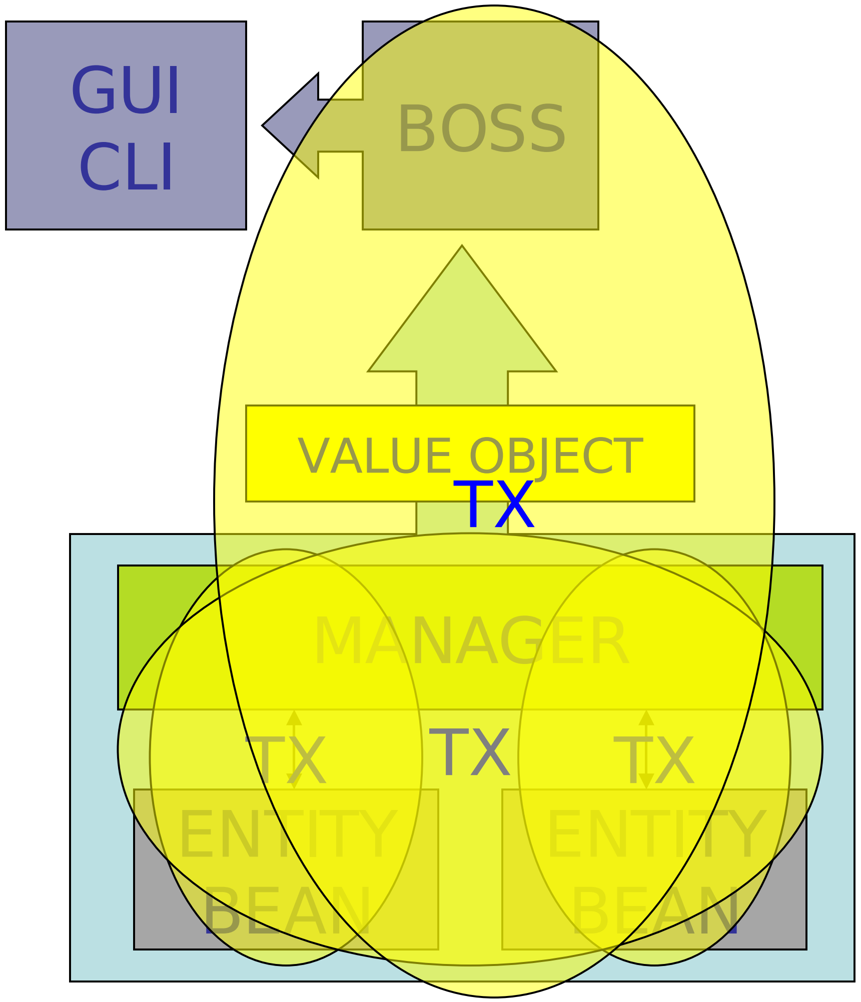
What’s the Problem (with EJB2)?

- Bad transaction handling, pessimistic locking
- N+1 database problem
- EJBQL - a poor query language
- Home grown caches

Why Migrate to Hibernate?

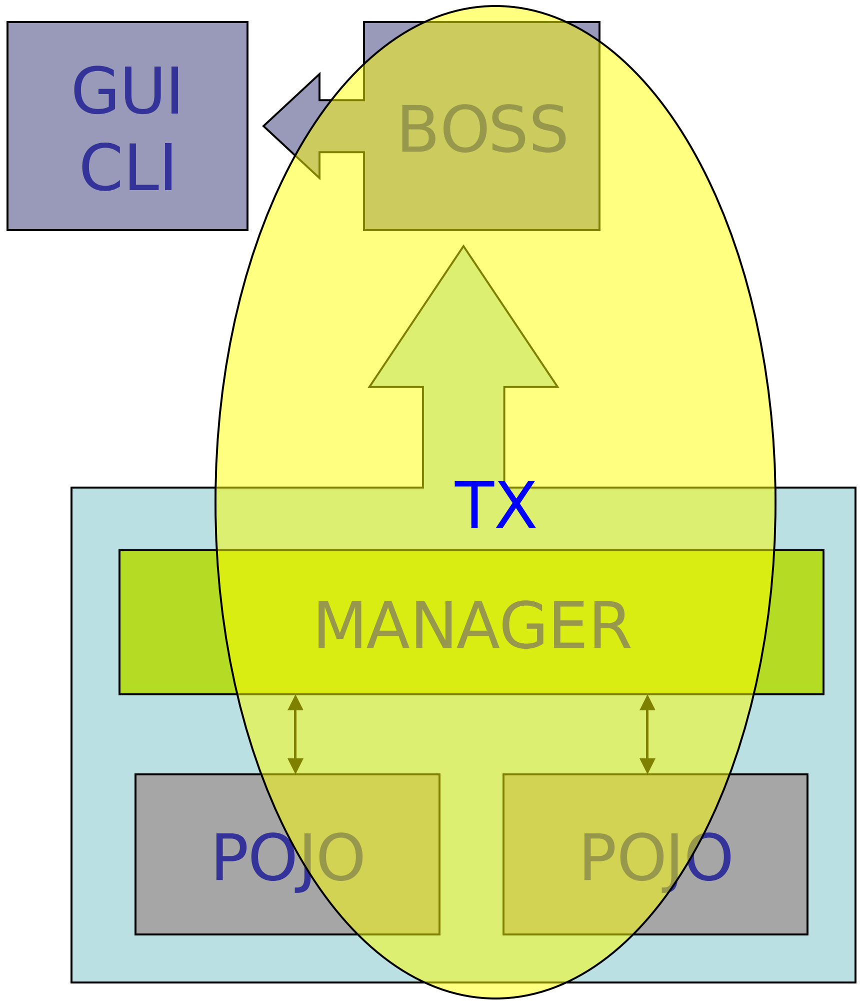
- Straightforward transaction demarcation
- Lazy fetching
- HQL and Criteria based queries - more fully featured
- Secondary cache integration
- Popular framework

EJB2 Entity Beans and Transactions



- **Entity Beans are proxy objects**
- **Use Value objects to travel through transaction boundaries**
- **Often lock pessimistically causing transaction deadlocks**

Hibernate POJOs and Transactions



- **POJOs set up to lazy load collections**
- **POJOs travel through transaction boundaries**
- **Hibernate Sessions use optimistic locking because of session cache**

N + 1 Database Problem

Database lookups to retrieve object data

- **Issue query to retrieve collection of primary keys (PKs)**
- **Issue individual query to retrieve object data per PK in collection**

Total number of queries performed: N (rows) + 1 (for PKs)

N + 1 Database Relieve

- **EJB2 - none**
  - Problematic in HQ 2.7
- **Hibernate has several solutions:**
  - Lazy fetching
  - Explicit outer-join declaration for associations
  - HQL supports explicit outer join fetch (“left join fetch”)
  - Multi-level caching support

Secondary Level Cache

Caching reduces unnecessary roundtrips to the database

- **EJB2 has no support for second-level cache**
- **Hibernate supports multi-level cache with pluggable architecture**
  - Database state can stay in memory
  - Caches both POJOs and queries
  - Use fast and proven caches: EHCache, JBoss TreeCache, etc

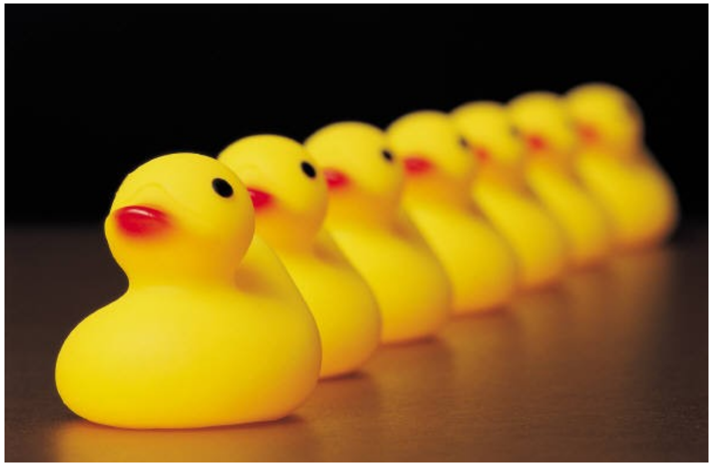
The Query Languages

- **EJBQL**
  - Lacks trivial functions like “ORDER BY”
- **JBossQL**
  - Some additional functions like “ORDER BY”, “LCASE”, etc.
- **Declared SQL**
  - Direct SQL-like queries declared as EJB finder methods, but not database-specific
- **HQL (and Criteria API)**
  - Close to SQL
  - Object oriented
  - Hibernate allows query results to be paged

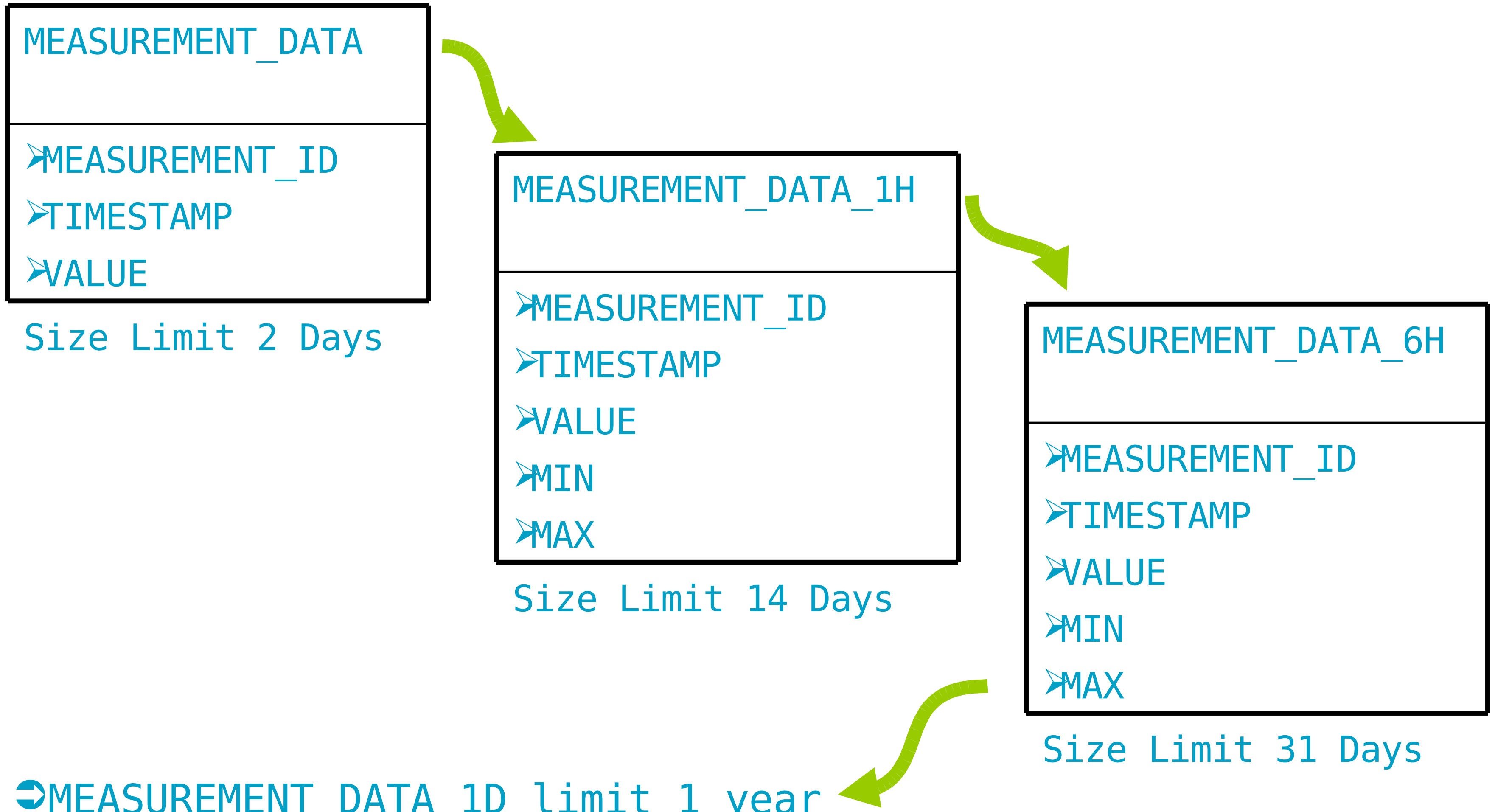
Data Consolidation

Inspired by RRDtool - an open source Round Robin Database to store and display time series data

- **Data compression runs hourly**
- **Table storing all collected data points (most activity) capped at 2 days worth**
- **Lower resolution tables track min, avg, and max**



Limited Table Growth



➲MEASUREMENT_DATA_1D limit 1 year

Performance Comparison

H5.com - using external database server

		HQ 2.6	HQ 3.0
Application Server	Load	6	1
	Total JVM Memory	4.5 GB	1.5 GB
Database Server	Load	5	3
	TCP Inbound	268	124

All Systems Go.

As A Result

- HQ’s performance improved dramatically
- Hi5.com can downgrade hardware
- Continue to rely on PostgreSQL
- Adding <u>hundreds</u> more boxes to production environment under HQ’s management

Questions and Comments

Charles Lee

clee@hyperic.com