

**FOLLOW US:**  
TWITTER.COM/REDHATSUMMIT

**TWEET ABOUT US:**  
ADD #SUMMIT AND/OR #JBOSSWORLD TO THE END  
OF YOUR EVENT-RELATED TWEET

# The Tao of Teiid

Steve Hawkins

Principal Software Engineer,  
Red Hat

Sept. 4, 2009



# What is Teiid?

- Teiid is an open source solution for scalable information integration through a relational abstraction.
- Teiid focuses on:
  - Real-time integration performance
  - Feature-full integration via SQL/Procedures/XQuery
  - Providing JDBC access
- Teiid enables:
  - Data Services / SOA
  - Legacy / JPA integration

# Overview

- Background
- Architecture
- Internals
- Wrap-up
- Q & A

# Where did Teiid come from?

- Project lineage is from MetaMatrix starting in ~1999.
  - Teiid - <http://www.jboss.org/teiid>
  - Teiid Designer - <http://www.jboss.org/teiiddesigner>
  - DNA - <http://www.jboss.org/dna/>
- MetaMatrix was the leader in Enterprise Information Integration (EII) – hence **Teiid**.
- Red Hat acquired MetaMatrix in 2007.
- Last major MetaMatrix product release, 5.5.3 - 10/2008

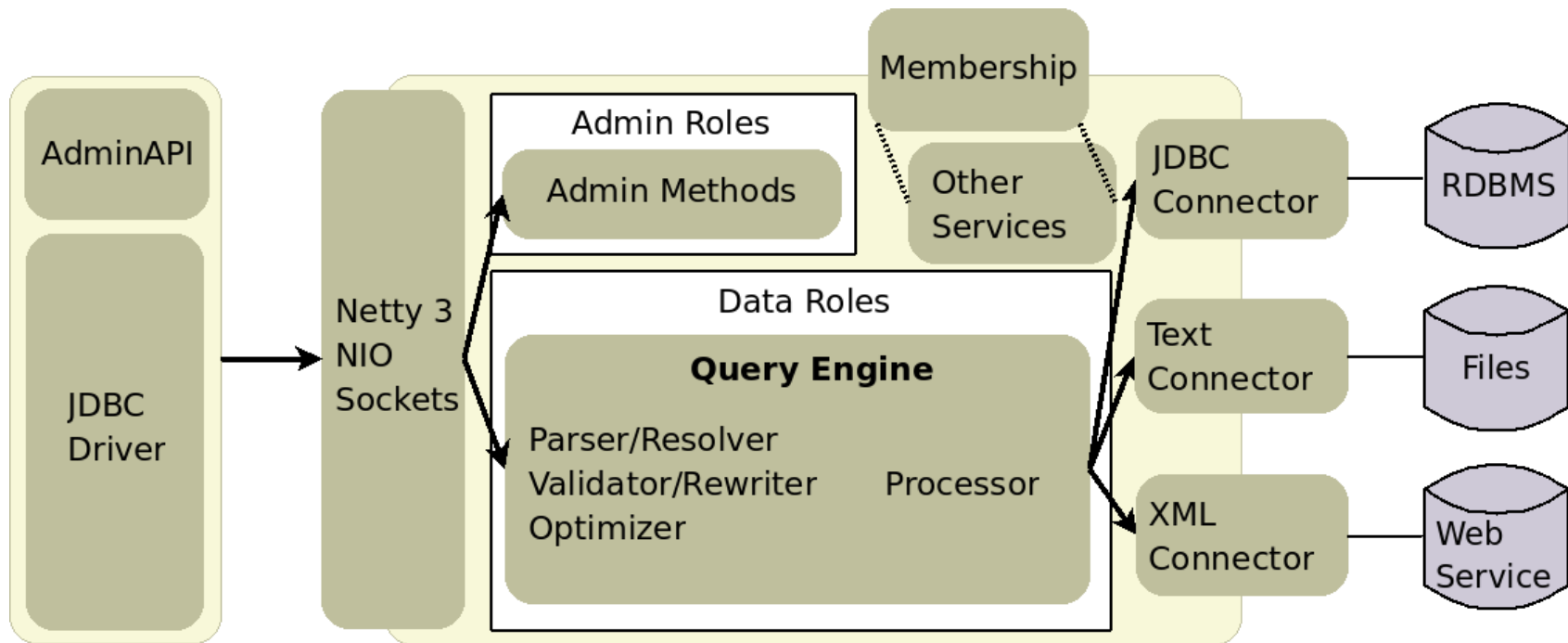
# Project Status

- Open source 2/2009 – heavily refactored from 5.5 line
- 6.0 Initial release 3/2009
- 6.1 Teiid / Teiid Designer release 6/2009
- 6.2 Coming Soon! Embedded/server deployments, Designer-less usage, AdminShell, and much more.
- Anticipate a platform release combining Teiid and other JBoss technologies next year.



# Architecture

# Architecture



- SEDA - Connector bindings, socket transport, query engine, admin methods all have queues/thread pools
  - Each connector binding operates independently
- Other services include JBoss Transactions JTA, BufferManager, sessioning, etc.



# Connector API

- Simplified object form of JDBC with concepts of JCA.
  - Pooling, caching, some security handled by the runtime.
  - Queries are resolved objects not just a string.
- Extended metadata (ConnectorCapabilities) directs the optimizer source query formation.
- In addition to out of the box offerings, our JDBC Connector is easily extended.
- Can be thought of as a JDBC toolkit.

# Other Extension Points

- Logging (Log4j), specific contexts for audit and commands
- MembershipDomains – handle authentication/group assignment. Provide File and LDAP by default.
- User defined functions – Implementation method in Java, currently only defined through Designer.
- Scripting through AdminShell

# Internals

# Teiid Internals

- Integration Features
- Planning
- Processing
- Transactions

# Integration Features

- Access Patterns – criteria requirements on pushdown queries
- Pushdown – decompose user query into source queries
  - Remove unused select clause items
  - Decompose aggregates over joins/unions
- Dependent Joins (can use hints) – feed equi-join values from one side of the join to the other
- Optional Join (can use hints) – removes an unused join child
- Multi-source connector bindings – allows for multiple homogeneous schemas to be used through the same model.
- Copy Criteria – uses criteria transitivity to minimize join tuples.

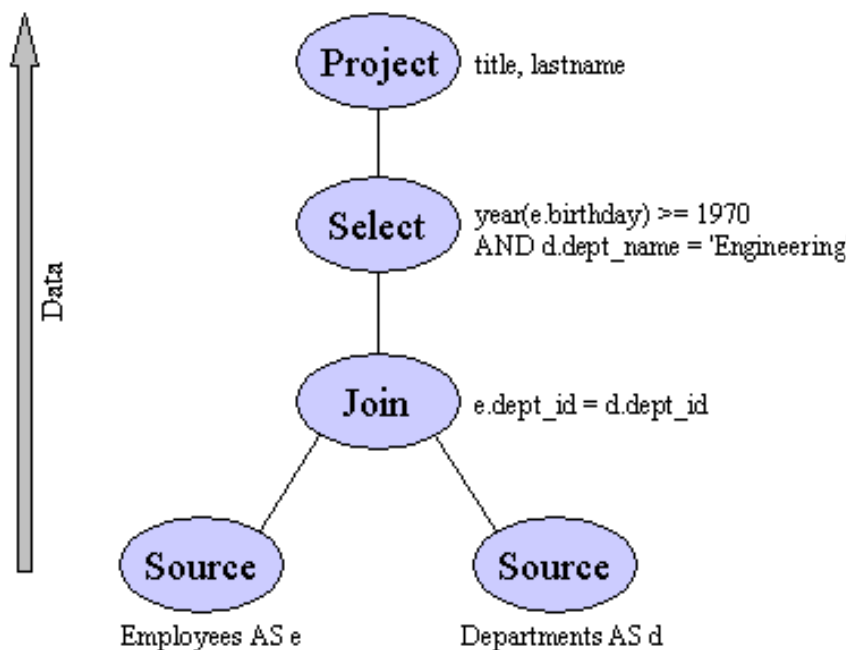
# Planning

- Distinct phases: parse, resolve, validation, rewrite, optimization, process plan creation.
- Rewrite canonicalizes and simplifies.
- The optimization phase follows with rules/hints/costing
  - Procedures/XQuery not formally optimized
  - Non-federated optimization is similar to mature RDBMS
- Optimizer plan structure is a flexible tree - distinct from the command form and processing plans.
- Planning is typically quick and deterministic – prepared plans are recommended



# Visualizing a Plan

```
select e.title, e.lastname from Employees as e JOIN  
Departments as d ON e.dept_id = d.dept_id where  
year(e.birthday) >= 1970 and d.dept_name = 'Engineering'
```



Project(groups=[e] ...)  
= Select(groups=[e,d] ...)  
= Join(groups=[e,d] ...)  
= Source(groups=[e] ...)   
Source(groups=[d] ...)

# Understanding Planning

- Initial canonical plans follow the logical SQL processing flow:
  - from/where/group by/having/select/order by/limit/into
- Each node corresponds to a logical SQL operation
- Canonical relational plans not performant for federated queries – optimization is necessary
- Processing plans and intermediate plans can be shown in the log/obtained by the client.

```
select * from ... option debug - with DETAIL logging
```

# Plan Rules

- Initial sequence driven by query form - some rules trigger others
- Move/create/delete/modify nodes toward more optimal form
  - RemoveVirtual – Removes inline views or nested transformations
  - RaiseAccess – Ensures access nodes are raised meaning more will be executed by the connector
  - PushSelectCriteria – Moves criteria toward tuple origin
  - CollapseSource – Takes plan nodes below an Access node and creates a query (not the final query sent to the source, which will get translated by the connector)
  - RulePlanSorts – Combines sort processing operations
  - ... many others ...
- Many rules correspond directly to federated optimizations – CopyCriteria, AggregatePushdown, RemoveOptionalJoins, etc.

# Example Rule Application

```
SetOperation(groups=[], props={USE_ALL=true, SET_OPERATION=UNION})
  Project(groups=[BQT1.SmallA], props={PROJECT_COLS=[IntKey]})
    Access(groups=[BQT1.SmallA], props={MODEL_ID=Model(BQT1)})
      Source(groups=[BQT1.SmallA], props={NESTED_COMMAND=null})
    Project(groups=[BQT1.SmallA AS SmallA__1], props={PROJECT_COLS=[SmallA__1.IntNum]})
      Access(groups=[BQT1.SmallA AS SmallA__1], props={MODEL_ID=Model(BQT1)})
        Source(groups=[BQT1.SmallA AS SmallA__1], props={NESTED_COMMAND=null})
```

=====

**EXECUTING RaiseAccess**

AFTER:

```
Access(groups=[], props={MODEL_ID=Model(BQT1)})
  SetOperation(groups=[], props={USE_ALL=true, SET_OPERATION=UNION})
    Project(groups=[BQT1.SmallA], props={PROJECT_COLS=[IntKey]})
      Source(groups=[BQT1.SmallA], props={NESTED_COMMAND=null})
    Project(groups=[BQT1.SmallA AS SmallA__1], props={PROJECT_COLS=[SmallA__1.IntNum]})
      Source(groups=[BQT1.SmallA AS SmallA__1], props={NESTED_COMMAND=null})
```

# Join Planning

- The most complicated parts of the optimizer
- It is not exhaustive, but does consider ordering (left linear), satisfying access patterns, and algorithm ([Partitioned] Merge / Nested Loop)
- Ordering/algorithm is only important for federated joins. Once a join is pushed, it's declarative to the source
- Merge joins have dependent variants, which can have large impact on performance – especially an unnecessary dependent join (see `make_not_dep`)

# Use of Costing

- Specified as attributes at the table and column level - will have a runtime interface soon
- Mostly based on cardinality with a simplistic cost model of execution
- Assign costs to different join ordering and implementations to pick the best one
- Using small, or inappropriate values, could lead to unexpected performance
- See plan info “Estimated Node Cardinality”, “Estimated Independent/Dependent ...”, etc. for values used in planning.



# Processing

- A relational processing plan is composed of discrete operations organized as a tree – very similar to the optimizer form:

AccessNode – Source Query/Procedure

GroupingNode – Grouping operations and aggregate calculation

JoinNode – Joins the left and right tuple sources together

LimitNode – Honors limits and offset

ProjectNode – Converts tuples (select clause)

SelectNode – Applies selection (where clause) criteria

SortNode – Sorts incoming tuples

...

- Procedure plans are composed of instructions.
- Tuples are processed in batches. The BufferManager is set to a specific memory limit; excess batches are written to disk.
- Processing algorithms are sort based, variants chosen during planning and processing.

# Example Process Plan

- `select * from System.DataTypeElements`

ProjectNode(1) [dt.Name AS DataTypeName, c.NAME, ...]

JoinNode(2) [PARTITIONED SORT JOIN (SORT/**SORT\_DISTINCT**)] [INNER JOIN]

\ criteria=[c.PARENT\_UUID=dt.UID]

**AccessNode(3)** SELECT c.PARENT\_UUID, ... FROM SystemPhysical.COLUMNS AS c

ProjectNode(4) [dt.NAME, dt.IS\_BUILTIN AS IsStandard, ...]

JoinNode(5) [MERGE JOIN (SORT/SORT)] [LEFT OUTER JOIN]

\ criteria=[dt.UUID=a.ANNOTATED\_UUID]

**AccessNode(6)** SELECT dt.UUID, dt.NAME, ... FROM SystemPhysical.DATATYPES AS dt

**AccessNode(7)** SELECT a.ANNOTATED\_UUID, ... FROM SystemPhysical.ANNOTATIONS AS a

- Shows decomposition into 3 source queries.
- Also the optimizer has combined a distinct operation into JoinNode(2) loading of the right child.

# Handling Load

- Memory Usage – the BufferManager acts as a memory manager for batches (with passivation) to ensure that memory will not be exhausted.
- Non-blocking source queries – rather than waiting for source query results processor thread detach from the plan and pick up a plan that has work.
- Time slicing – plans produce batches for a time slice before re-queuing and allowing their thread to do other work (preemptive control only between batches)
- Caching – ResultSets at the connector and user query level can be reused on a session or vdb basis

# Transactions

- Three scopes
  - Global (through XAResource)
  - Local (autocommit = false)
  - Command (autocommit = true)
- All scopes are handled by JBoss Transactions JTA
- Command scope behavior is handled through `txnAutoWrap={ON|OFF|OPTIMISTIC|PESSIMISTIC}`
- Isolation level is set on a per connector basis.

# Wrap Up

# Performance

- Raw (cpu-intensive) overhead is typically sub-millisecond per prepared user query.
- Integration performance – check the processing plan. We'll usually have the best form.
- Consider using UDFs (Java) for reusable subroutines rather than stored procedures.
- Client result sets can be scroll insensitive and backed by the BufferManager.



# Differences with traditional Java DBs

- Flexible planning architecture
- Geared to high-performance integration processing – task specific queues and thread pools, advanced buffer management, batching, etc.
- Lack of DDL support
- Loose constraint handling
  - pk/fk, unique, and type constraints are in metadata, but are not enforced at runtime.
- Temp tables backed by BufferManager rather than a relational/indexed storage engine.

# Future Releases

- We'll look even more like a database - direct usage of DDL for metadata.
- More features around materialization, data locality, and caching.
- Continued integration with other JBoss projects.
- More design-time integration with Eclipse DTP  
<http://www.eclipse.org/datatools/>

**QUESTIONS?**

**TELL US WHAT YOU THINK:  
[REDHAT.COM/JBOSSWORLD-SURVEY](https://redhat.com/jboss-world-survey)**