Welcome to OSCON 2012

MySQL Cluster and NoSQL

• If you want to run a node in a **multi-node cluster**:
  • Pick a seat where there is a network cable
  • Plug in
  • Turn off your firewall (*or at least allow ports 1186 & 2186*)
  • Point your browser to [http://uplink/](http://uplink/)

• **OR**, to run a **one-node cluster** contained entirely on your own laptop, sit away from the network cables and use the conference wireless.
MySQL Cluster and NoSQL

John David Duncan
Craig L Russell
MySQL Cluster Engineering
Oracle Corp
Everyone needs the tutorial files!

- From USB stick
- From uplink
- From github
  - oscon-2012-mysql-nosql.tar
  - or
  - oscon-2012-mysql-nosql.zip
  - or
  - oscon-2012-mysql-nosql-MSDOS.zip


git clone

git://github.com/jdduncan/oscon-2012-mysql-nosql.git
Platform Census Time
MySQL Cluster and NoSQL

- There are 3 handouts
  - README for Java / Twitter development exercise
  - MySQL pathname reference
  - Memcache reference card (2-sided)

- If you still need MySQL Cluster 7.2.6 and Cache::Memcached, they are on the USB sticks – go get ‘em

- If you want to be in a multi-node cluster with other people, find a network cable
Big Picture Agenda

• First half: deploy cluster
  – Some lectures
  – Some lab - run a cluster

• Second half: develop apps
  – Some lectures - Memcached & Java
  – Some lab - you write the code
Presenter's "Baggage"

J.D. Duncan

• Former Systems Administrator, web developer, MySQL DBA
• Joined MySQL AB in 2004

Image used under a Creative Commons license from Flickr user Digiart2001
Presenter's "Baggage"

Craig L. Russell

- Systems Architect focused on Enterprise Java
- Specification Lead for Java Data Objects
- Architect of Container-Managed Persistence
- Likes SQL well enough
- Likes well-designed APIs even more

*Image used under a Creative Commons license from Flickr user Digiart2001*
1:30 - 3:00 P.M  Agenda

- Seating & Introductions  (10 min.)
- MySQL Cluster Essentials  (JD - 25 min.)
- Hands-on MySQL Cluster setup  (35 min.)
- API Overviews  (20 min.)
  - ClusterJ (Craig)
  - Memcache (JD)
  - Node.js Connector Project
MySQL Cluster Essentials
What is a database?

• It’s a computer system that manages data ... in a useful way.
How do database systems make themselves useful?

- Persistent storage
  - in “the system of record”

- Indexed retrieval
  - the database gets big, but retrieval time remains small

- Central enforcement of data integrity
  - Every Social Security Number has 9 digits
  - Every order has a billing address

- Maintain an illusion of consistency
  - isolating a transaction other concurrent changes
How is the data structured?

• **Key : Value**
  - Hash table (or memcache):
    • **lookups** only
    • no **scans**
  - Ordered Index (or binary tree):
    • **Lookups** and **scans** both.

• **Key : Structured Value**
  - redis: common data structures used in application code
  - mongodb: structured documents, possible secondary indexes
  - dynamo: schema-free tables

• **Relational (Tables with a schema)**
Document Databases

• Examples: MongoDB, CouchDB
• Expected lookup:
  – “Fetch this patient’s record” ✓
• Unexpected lookup:
  – “Who took Accutane after 1990?” X
Relational Databases

- In a truly normalized relational schema, no table is more central than any other.
- “Fetch this patient’s record” ✓
- “Did more of our patients get the flu in 2008 or in 2009?” ✓
- “Which doctor prescribes the most prednisone?” ✓
But what about schema changes?
But what about schema changes?

They need to be online operations.
In the last few minutes we actually covered a lot of important theory.

• But here’s one more bit: **consistency vs. availability.**
• Look at MySQL 4.0, *circa* 2000
What happens when a link goes down?

- CAP (Eric Brewer): You have to choose. You can’t have both. Commercial databases usually chose **C**, but MySQL replication chose **A**.
Now We Are Ready

MySQL Cluster
NDB Cluster
Ericsson, 1997

• Hardware Design
  – Inexpensive x86 hardware
  – "Shared nothing" clustering

• Software Design
  – 99.999% availability
  – Guaranteed consistency
  – Parallel execution
  – Commit at speed of network (rather than disk)
NDB Cluster: Where does it fit?

• Consistent or Available?
  – Availability is important. NDB is designed to remain running if one node in a node group fails.
  – But **consistency** is even more important. If *all* nodes in a node group fail, the database will shut itself down.

• Relational? **Yes.**
  – Hashed primary key for data distribution
  – Secondary Unique indexes
  – Ordered indexes for scans
  – Online schema changes.

• SQL? **No.**
  – C++ NDB API for object-relational application development.
2003: MySQL + NDB Cluster = MySQL Cluster

• MySQL:
  – built for cheap linux boxes
  – free, popular, revolutionary, fast, easy to use
  – proven ability to run with different back-end data stores, e.g. MyISAM and InnoDB.
  – MySQL replication has *asynchronous, best-effort* design that prioritizes availability

• NDB:
  – built for cheap linux boxes
  – high performance and availability
  – had no support for SQL, or ODBC, or JDBC
The Basics

- **Management server**
  - read the one central configuration file
  - allow other nodes to join
  - distribute the configuration to them
  - get the cluster up & running

- **Data Node**
  - ndbd (single-threaded) or ndbmttd (multi-threaded)
  - Stores data and indexes
  - Manage all transactions and operations for API nodes

- **API Node**
  - Join the cluster as a member node with node id
  - *e.g.* mysql, memcached, JVM running ClusterJ application

- **Client**
  - MySQL or Memcache client; not a member of the cluster
Big Picture

API Nodes

Node Group 1

Node Group 2

NDB Data Nodes
NOT a shared-disk architecture
“Shared-Nothing” Architecture
Synchronous data replication

- Data changes take place on all the servers in coordination
- The "two-phase commit" (2PC) is the most commonly used protocol
  - The changes are only committed after they’ve been successfully executed on all servers
  - TC (Transaction Coordinator) manages 2PC
  - LQH (Local Query Handler) at each data node manages local transaction
Heartbeat Circle

Nodes organized in a logical circle

Heartbeat messages sent to next node in circle
Heartbeat Failure Example

Node 1 fails to send heartbeats three consecutive times

It is declared as dead and removed from the cluster
Network Partitioning

- The servers are organised in a network
- What happens if the servers lose contact to each other?

"Split brain"

- Neither side can continue as this would lead to inconsistency
Using an Arbitrator

• In order to handle an even number of servers an arbitrator is added

• When a majority is needed by a set of servers, the arbitrator “votes” for that set, making it a majority

• The arbitrator must not vote for several different subsets and the arbitrator must be known to all servers

• By using an arbitrator the two server system becomes more fault tolerant than a single server system
Who Uses MySQL Cluster?

Until 2010: Telco Networks & Network Equipment Makers

- Alcatel-Lucent
- Ericsson
- Motorola
- Juniper Networks
- Italtel
- Utstarcom
- Nokia
Who Uses MySQL Cluster?
Since 2010: Web & Online gaming

- Session management
- Shopping carts
- Game play
- Internal site monitoring
MySQL Cluster Today

ACID Compliant Relational Database
  SQL & NoSQL interfaces

Write-Scalable & Real-Time
  Distributed, auto-partitioning (sharding), multi-master

99.999% Availability
  Shared-nothing, integrated clustering & sub-second recovery, local & geographic replication, on-line operations

Low TCO
  Open-source, management & monitoring tools, scale-out on commodity hardware
Online Maintenance

• Online schema changes
  – ALTER TABLE ...

• Online backup
  – START BACKUP ...

• Online storage addition
  – e.g. from 4 data nodes to 6
The long road to SQL JOIN performance

• Initial integration (2004)
  – Move the data to the query
  – ... one row at a time
  – ... one loop iteration in a nested-loop join = 1 network trip

• 2005 - 2009
  – Improve efficiency ...
  – within the existing framework of the MySQL optimizer
  – e.g. Batch Key Access, push-down filters

• MySQL Cluster 7.2 (2011)
  – Adaptive Query Localization
  – Algorithmic changes to optimizer
  – Push the majority of the query to the data nodes
Results: Adaptive Query Localization in MySQL Cluster 7.2

Before: 48.68 sec

```sql
mysql> SELECT COUNT(*)
    FROM residents, postcodes, towns
WHERE residents.postcode=postcodes.postcode
    AND postcodes.town=towns.town
    AND towns.county="Berkshire";

+----------+
| COUNT(*)  |
+----------+
|    40001  |
+----------+
```

• After: 2.02 sec
Tour
Tour Stop 1: Config File

- A single central file for the whole cluster
- Loaded by the management server on --initial
- And then stored in cache for future management server restarts
- Sections:
  - [MGM]
  - [DB DEFAULT]
  - [NDBD]
  - [API]
Tour Stop: Management Server

[MGM]
NodeId=1
datadir=/Users/jdd/sandbox/ndb
hostname=localhost

% ndb_mgmd -f config-file-path
Tour Stop: Data Nodes

[DB DEFAULT]
NoOfReplicas=1
TimeBetweenGlobalCheckpoints=12000
HeartbeatIntervalDbApi=15000
DataMemory=180M
IndexMemory=60M
Tour Stop: Data Nodes

[NDBD]
NodeId=2

% ndbd -c mgm_host:mgm_port
Tour Stop: API Nodes

[API]
[API]
[API]
[API]
[API]
[API]
[API]
[API]
Tour Stop: API Nodes

- mysql, memcached, java, etc.
  - Java API nodes connect to clusters based on Java connection properties
  - mysql connects to cluster based on my.cnf file
  - memcached connects based on command-line options
Tour Stop: Management & Monitoring

• From the mysql server
  – SQL for table creation, etc.
  – `ndbinfo` schema
    • `e.g. SELECT * FROM ndbinfo.memoryusage;`
  – `SHOW ENGINE NDB STATUS;`

• Management Clients
  – `ndb_mgm`

• Toolsets
  – From Oracle: MySQL Cluster Manager
  – Third party tools: severalnines.com
MySQL Cluster Hands-On Lab

• **Solo Clusters**
  - Your config file is here:
  - `run-cluster/ndb/solo-cluster.ini`

• **Multi-Node Clusters**
  - [http://uplink/](http://uplink/)
  - 1 management server per cluster
  - 2, 4, or 6 data nodes per cluster
  - Others can run API nodes
  - MGM and NDB nodes register with the web app
  - Then the web app generates a config file for the management server and you (mgm server person) download it and save it in `run-cluster/ndb/`
MySQL Cluster Hands-On Lab

- Open 4 or more terminal windows

- Set some pathnames
  - Check your handout!
  - `export PATH=/usr/local/mysql/bin:$PATH`
  - (or whatever is correct for your platform)
  - `cd into oscon-2012-mysql-nosql`
Start your management servers!

• **Wireless:**
  • `cd run-cluster/ndb`
  • `ndb_mgmd --initial -f solo-cluster.ini --configdir=.`

• **Wired (MGM person only!):**
  • `save cluster.ini from uplink`
  • `cd run-cluster/ndb`
  • `ndb_mgmd --initial -f cluster.ini --configdir=.`
What’s going on with this cluster?

- Start a management client!
- (And leave it running in its window)
- Wireless:
  - `ndb_mgm -c localhost:1186`

- Wired:
  - Where is your management server?
  - `http://uplink` should tell you
  - `ndb_mgm -c box-xx:1186`
Start your data nodes!

- You are still in run-cluster/ndb
- And you remember your connectstring ...
- So:

  ndbd -c <connectstring>
What’s going on with this cluster?

```
ndb_mgm> all status
Node 2: starting (Last completed phase 2) ...
```

```
ndb_mgm> show
Cluster Configuration
```

---
Configure a MySQL server

- **Create a my.cnf file**
  - `cd run-cluster/sql`
  - `./create-config.sh`
  - `ls (you should see my.cnf)`
- **Wireless**: leave it as-is
- **Wired**: edit the connectstring
Here’s what my own my.cnf looks like

[mysqld]
ndbcluster
user=jdd
datadir=/Users/jdd/.../run-cluster/sql
ndb-connectstring=localhost:1186
Configure a MySQL server

cd oscon-2012-mysql-nosql
  . pathnames.sh
cd run-cluster/sql
$MYSQL_INSTALL_DIR/scripts/mysql_install_db
  --datadir=. \
  --basedir=$MYSQL_INSTALL_DIR
Configure a MySQL server

Installing MySQL system tables...  OK
Filling help tables...   OK
...


Now start the mysqld

```shell
mysqld_safe --defaults-file=./my.cnf
```

```shell
mysqld_safe Starting mysqld daemon with databases from ...
```
MySQL Client

$ mysql -u root

Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 2
### MySQL Client

```sql
mysql> select * From ndbinfo.nodes;
```

<table>
<thead>
<tr>
<th>node_id</th>
<th>uptime</th>
<th>status</th>
<th>start_phase</th>
<th>config_generation</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>3453</td>
<td>STARTED</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

1 row in set (0.00 sec)
Create some tables (from the shell)

- We need to run two SQL scripts:
  - One called `ndb_memcache_metadata.sql` in `$MYSQL_HOME/share/memcache-api/`
  - Plus `java/labs/schema/create.sql` from `oscon-2012-mysql-nosql`

% mysql -u root < filename
MySQL Client

mysql> use test

mysql> show tables ;

+----------------+
| Tables_in_test |
+----------------+
| hashtags       |
| tweets         |
+----------------+
Make a backup!

ndb_mgm> start backup

Waiting for completed, this may take several minutes
Node 2: Backup 1 started from node 1
Node 2: Backup 1 started from node 1 completed
  StartGCP: 351  StopGCP: 354
#Records: 2058  #LogRecords: 0
Data: 51728 bytes  Log: 0 bytes
Things you might do if we had more time

- Set up cluster-to-cluster replication
  - uses a designated mysql server in each cluster
- Online alter table operations
  - ALTER ONLINE TABLE x ADD i int NULL;
  - (note that some ALTER operations can be done online and some others cannot)
- Rolling Restart
- Expand a cluster online (add nodes)
API Overviews

- ClusterJ (Craig, 10 min.)
- Memcache (JD, 5 min.)
- Node.JS project (JD & Craig, 10 min.)
- Break time
ClusterJ Overview

Craig L Russell
Architect, Oracle Corp.
ClusterJ

- High Performance, Light Weight, Easy to Use Direct Connection
  - In the style of Hibernate / JPA / JDO
  - Insert, delete, find by key, update, simple query

- Shared Data storage with:
  - MySQL server
  - Native C++ applications
  - Other ClusterJ, memcached applications

- Domain Object Model DataMapper pattern
  - Data is represented as domain objects
  - Domain objects are separate from business logic
  - Domain objects are mapped to database tables
ClusterJ as NoSQL Store

• Key/value Store
  – Arbitrary text storage (String <=> CLOB)
  – Arbitrary data storage (byte[] <=> BLOB)

• Document Store
  – JSON, XML, etc.

• Session Store
  – key = session id
  – value = serialized session state

• Back end for REST plugin
Domain Object Model Mapping

- Tables map to Persistent Interfaces / Classes
- Columns map to Persistent Properties
  - column names default to property name
- Rows map to Persistent Instances
- Annotations on Interfaces / Classes customize mappings
- User chooses which to write:
  - User interface (ClusterJ then generates implementation class)
  - Persistent class (ClusterJ provides base implementation class)
@PersistenceCapable(table="tweets")
public interface Tweets {

    String getId();
    void setId(String id);

    String getAuthor();
    void setAuthor(String value);

    Date getTime_stamp();
    void setTime_stamp(Date value);

    String getTweet();
    void setTweet(String value);
}

Copyright © 2012 Oracle Corp.
ClusterJ – Dynamic Object

public class Tweets
extends DynamicObject {

public String table() {
    return "tweets";

    String getId() {
        return (String)get(0);
    }
    void setId(String value) {
        set(0, value);
    }

    ...

    // other fields and behavior
}
Numeric Column Mapping

- Java boolean, Boolean
  - BIT(1)
- Java byte, Byte
  - BIT(1) to BIT(8)
  - TINYINT
- Java short, Short
  - BIT(1) to BIT(16)
  - SMALLINT
  - YEAR
- Java int, Integer
  - BIT(1) to BIT(32)
  - INT

- Java long, Long
  - BIT(1) to BIT(64)
  - BIGINT
  - BIGUNSIGNED
- Java float, Float
  - FLOAT
- Java double, Double
  - DOUBLE
- Java BigDecimal
  - NUMERIC
  - DECIMAL
- Java BigInteger
  - NUMERIC
  - DECIMAL
Date Column Mapping

- Java util Date
  - DATETIME
  - TIMESTAMP
  - TIME
  - DATE
- Java sql Date
  - DATE
- Java sql Time
  - TIME
- Java sql Timestamp
  - DATETIME
  - TIMESTAMP
Variable Size Column Mapping

• Java String
  – CHAR
  – VARCHAR
  – TEXT
• Java byte[ ]
  – BINARY
  – VARBINARY
  – BLOB
ClusterJ Features

- Character Set Translation (all MySQL charsets)
- Automatic detection of primary keys, indexes
- Compound Primary Keys
- Ordered (btree) indexes
- Unique (hash) indexes
- Automatic use of partition key
- Multi-threaded applications
ClusterJ Interfaces

- SessionFactory
  - Instance per connection to cluster

- Session
  - Instance per "user"

- Transaction
  - Instance per Session

- Query
  - Multiple instances per Session
ClusterJ User View

SessionFactory

Configuration Properties

Session and Transaction

Domain Object

Domain Object

Domain Object

Domain Object

Domain Object

Domain Object
ClusterJ Limitations

- No Relationships
  - primitive types only
- No Multi-table inheritance
  - single table per persistent interface or class
- No joins in queries
  - column comparisons and boolean operators
- No Table creation
  - user needs to create tables and indexes
- No Lazy Loading
  - entire record is loaded at one time, including LOBs
ClusterJ Requirements

• Java 1.5 or 1.6
  – Compile-time jar:
    • clusterj-api
  – Run-time jar:
    • clusterj
    – Java Native Library
      • ndbclient
• MySQL Cluster 7.1 or higher
  – includes jars and native libraries as above
NDB Memcache API Overview
Memcached Overview:
Two levels of hashing

httpd

PHP/Perl

Memcache

friends:12389
memcache key

hash key to pick server

memcached

hash key to find data

memcached

memcached
Expected Latency & Throughput

- **httpd**: 1,000s of operations/sec. 
  ~ 2 ms round trip

- **PHP/Perl**: 10,000s of operations/sec. 
  ~ 200 µs round trip

- **memcached**: 

- **mysql**: 1,000s of operations/sec. 
  ~ 2 ms round trip

- **MySQL Slave**
Goals

• Access stored data directly from memcache clients

  – *Memcached perspective:*
    • MySQL Cluster is a write-scalable, replicated data store
      – with *reliable* in-memory storage,
      – plus *on-disk* storage when data is too big for memory.

  – *MySQL Cluster perspective:*
    • memcache is a high performance API
      – providing *easy access* to in-memory data,
      – plus an extra layer of caching when data is on disk.
Memcache Protocol & Commands

see handout!
Anatomy of a Memcache key

user:1248

the prefix

the database key
A key-prefix mapping

Memcache key prefix → Cluster → Container → Cache Policy
A memcache server role
Mynode Preview

Craig L Russell
Architect, Oracle Corp.
The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle’s products remains at the sole discretion of Oracle.
Mynode: Node.js module for MySQL

- Advance Planning – availability not announced
- Domain Object Model DataMapper pattern
  - Domain objects are mapped to database tables
- High Performance, Light Weight, Easy to Use Direct Connection
  - Asynchronous API
  - Insert, delete, find by key, update, simple query
- Shared Data storage with:
  - MySQL server
  - Native C++ applications
  - Other Mynode, ClusterJ, memcached applications
Domain Object Model Mapping

• Tables map to Persistent Classes
• Columns map to Persistent Properties
  – column names default to property name
• Rows map to Persistent Instances
• Annotations customize mappings
  – column names
  – table names
  – null value handling
Mynode – Mapped Class

function Tweets(author, tweet) {
    this.id = randomUUID();
    this.author = author;
    this.tweet = tweet;
    this.time_stamp = new Date();
};
var tweet = new Tweet('clr', 'having a great #oscon');

insertTweet(tweet, function()...);

function insertTweet(tweet, callback) {

    session.insert('tweets', tweet,
        function(err) {
            if (err) throw ...
            callback();
        }
    );

};
Mynode – Find by Primary Key

```javascript
function getTweet(id, callback) {

    session.find('tweets', id,
        function(err, tweet) {
            if (err) throw ...
            callback(tweet);
        }
    );
}
```
Mynode Features

- Character Set Translation (all MySQL charsets)
- Automatic detection of primary keys, indexes
- Compound Primary Keys
- Automatic use of partition key
- Multi-threaded applications
Mynode Classes

- **Mynode**
  - SessionFactory factory
  - Establishes connection to database

- **SessionFactory**
  - Instance per connection to database

- **Session**
  - Instance per "user"
Mynode Requirements

- JavaScript
- Node.js
  - Run-time module:
    - mynode
  - Java Native Library
    - ndbclient
- MySQL Cluster 7.2 or higher
One more thing before the break ...

Heartbeat messages sent to next node in circle
You want to shut down?

MySQL Servers:
  mysqladmin -u root shutdown

Management and Data Nodes:
  ndb_mgm> shutdown
MySQL Cluster and NoSQL

Mid-Session Break
3:30 - 5:00 P.M  Agenda

– SQL DDL Lecture
– Restart your clusters
– Labs
  – Java Insert
  – Time for Coding
  – Memcache Insert
  – Time for Coding
  – Java Query
  – Time for Coding
– Wrap-up with Q&A
Data Definition, SQL, and NDB Indexes
Data Definition: Distribution Keys

• Every table has a distribution key. The MD5 hash of the distribution key determines which fragment a row belongs to.

• When an API node knows the distribution key of an interesting row, it can take advantage of this by choosing the primary fragment as TC. (ClusterJ, MySQL, and Memcached all do this automatically).
Data Definition: **Ordered Indexes**

- An ordered index is an *in-memory T-Tree index*

- It is partitioned in exactly the same way as the main table.
  - *i.e.:* each data node has its own ordered index covering its own fragments of the table

- This in-memory T-Tree contains direct pointers to the in-memory rows of the main table.

- The index is not stored on disk between restarts; it is rebuilt during node startup.
Data Definition: The SQL primary key

- By default, MySQL Cluster does two things with the primary key of a table:
  - Uses it as the distribution key
  - Creates an ordered index named PRIMARY.
- This means a table’s primary key can be used both for lookups and for scans.
- You can override the default behavior:
  - `PARTITION BY HASH (col1, col2)` to get a different distribution key
  - `PRIMARY KEY USING HASH` to skip building the ordered index
Data Definition: Unique Indexes

- A unique index is a **secondary hash index**.
- Used to enforce a unique constraint.
- Used for lookups, but not scans.
- Implemented as a hidden table:
  - **key**: the unique index =>
  - **value**: the primary key of the main table.
- Because it’s really an independent table, it has its own independent distribution across fragments.
Data Definition: Indexes

- Data rows can be stored either in-memory, or on disk (in a tablespace).
- Data stored in memory is still durable, because the data nodes *checkpoint* it to disk.
- Indexes, however, are always in memory. Never on disk.
- Therefore, a cluster must have enough memory for all its indexes.
CREATE TABLE tweets (
    id varchar(36) primary key,
    tweet varchar(140),
    time_stamp timestamp,
    author varchar(15),
    index(time_stamp),
    index(author))
ENGINE=ndb;
CREATE TABLE hashtags (  
    hashtag varchar(20),  
    tweet_id varchar(36),  
    time_stamp timestamp,  
    author varchar(15),  
    primary key(hashtag, tweet_id))  
ENGINE=ndb;
Restart clusters!

ndb_mgmd --configdir=.

ndbd -c <connectstring>

ndb_mgm -c <connectstring>
ClusterJ API

Craig L Russell
Architect, Oracle Corp.
ClusterJ – Generated Class

@PersistenceCapable(table="tweets")
public interface Tweets {

    String getId();
    void setId(String id);

    String getAuthor();
    void setAuthor(String value);

    Date getTime_stamp();
    void setTime_stamp(Date value);

    String getTweet();
    void setTweet(String value);
}

Copyright © 2012 Oracle Corp.
public class Tweets
extends DynamicObject {

    public String table() {
        return "tweets";
    }

    String getId() {
        return (String)get(0);
    }

    void setId(String value) {
        set(0, value);
    }

    ...

    // other fields and behavior
}
Numeric Column Mapping

- Java boolean, Boolean
  - BIT(1)
- Java byte, Byte
  - BIT(1) to BIT(8)
  - TINYINT
- Java short, Short
  - BIT(1) to BIT(16)
  - SMALLINT
  - YEAR
- Java int, Integer
  - BIT(1) to BIT(32)
  - INT
- Java long, Long
  - BIT(1) to BIT(64)
  - BIGINT
  - BIGUNSIGNED
- Java float, Float
  - FLOAT
- Java double, Double
  - DOUBLE
- Java BigDecimal
  - NUMERIC
  - DECIMAL
- Java BigInteger
  - NUMERIC
  - DECIMAL
Date Column Mapping

- Java util Date
  - DATETIME
  - TIMESTAMP
  - TIME
  - DATE
- Java sql Date
  - DATE
- Java sql Time
  - TIME
- Java sql Timestamp
  - DATETIME
  - TIMESTAMP
Variable Size Column Mapping

- Java String
  - CHAR
  - VARCHAR
  - TEXT
- Java byte[ ]
  - BINARY
  - VARBINARY
  - BLOB
ClusterJ Interfaces

- SessionFactory
  - Instance per connection to cluster
- Session
  - Instance per "user"
- Transaction
  - Instance per Session
- Query
  - Multiple instances per Session
SessionFactory

• Configured by properties
  – com.mysql.clusterj.connectstring (the only really important property)
  – com.mysql.clusterj.connect.retries
  – com.mysql.clusterj.connect.delay
  – com.mysql.clusterj.connect.timeout.before
  – com.mysql.clusterj.connect.timeout.after
  – com.mysql.clusterj.max.transactions

• One SessionFactory per cluster per JVM
  – Connection pooling (multiple TCP connections per SessionFactory)
Example

SessionFactory sessionFactory;
Properties props = new Properties();

void initProps() {
props.put(PROPERTY_CLUSTER_CONNECTSTRING,
    "localhost:1186");
props.put(PROPERTY_CLUSTER_DATABASE,
    "personnel");
}

public init(Properties props) {
    initProps();
    sessionFactory = ClusterJHelper
        .getSessionFactory(props);
}
Session

- Persistent Instance Factory
- Lifecycle Management
  - persist() // record must not already exist
  - remove() // record must already exist
- Update Management
  - update() // no need to read first
  - write() // record will be overwritten if already exists
- Find by ID
- Query Factory
Example Insert

```java
Session session;
void getSession() {
    session = sessionFactory.getSession();
}
Tweets createTweet(String author, String message) {
    Tweets tweet =
        session.newInstance(Tweets.class);
    tweet.setId(UUID.randomUUID().toString);
    tweet.setAuthor(author);
    tweet.setTweet(message);
    tweet.setTime_stamp(new Date());
    session.persist(tweet);
    return tweet;
}
```
Example Find

Tweets findTweet(String id) {
    Tweets tweet =
        session.find(Tweets.class, id);
    return tweet;
}

Transaction

• Optionally used to delimit transactions
  – begin()
  – commit()
  – rollback()

• Optionally mark a transaction for rollback
  – setRollbackOnly()
  – getRollbackOnly()

• If not used, every operation is in its own transaction
Example

Transaction transaction;

void getTransaction() {
    transaction = session.getTransaction();
}

void createTweets() {
    getTransaction();
    transaction.begin();
    createTweet("amos", ">#oscon rocks");
    createTweet("barb", "sw33t tutorial");
    createTweet("chuck", "lets #partay");
    createTweet("dave", "just one mo hour");
    transaction.commit();
}
ClusterJ Exercises

- Create schema for tweets, hashtags
  - create.sql in labs/schema
- Write domain models for Tweets, Hashtags
  - skeletons in labs/model
- Write Insert.java program
  - skeleton in labs/test/Insert.java
- Compile-time jar: clusterj-api-7.2.6.jar
- Run-time jar: clusterj-7.2.6.jar
- Java Native Library: ndbclient
ClusterJ Exercises

- Create schema for tweets, hashtags
  - create.sql in labs/schema
- Write domain models for Tweets, Hashtags
  - skeletons in labs/model
- Write Insert.java program
  - skeleton in labs/test/Insert.java
- Compile-time jar: clusterj-api-7.2.6.jar
- Run-time jar: clusterj-7.2.6.jar
- Java Native Library: ndbclient
Memcache Insert
A key-prefix mapping
Memcache Config

- Generate a containers record for the tweets table
  - value columns in the containers record are separated by commas

- Generate a key-prefix mapping for role 0

- Start a memcached server

  cd run-cluster/ndb
  ./start-memcached.sh
Start Memcached

$ memcached -E /usr/local/mysql/lib/ndb_engine.so -e connectstring="localhost:1186"

15-Jul-2012 23:51:46 PDT NDB Memcache 5.5.22-ndb-7.2.6 started [NDB 7.2.6; MySQL 5.5.22]
Contacting primary management server (localhost:1186) ...
Connected to "localhost:1186" as node id 4.
Retrieved 4 key prefixes for server role "default_role".
The default behavior is that:
  GET uses NDB only
  SET uses NDB only
  DELETE uses NDB only.
The 3 explicitly defined key prefixes are "b:" (demo_table_large), "mc:" () and "t:" (demo_table_tabs)
Server started with 4 threads.
Priming the pump ...
Connected to "localhost:1186" as node id 5.
Scheduler: using 2 connections to cluster 0
Scheduler: starting for 1 cluster; c0,f0,g1,t1 done [0.656 sec].
Memcache Exercises

• In perl/labs/
• 1. config.sql: create metadata for app.
• 2. tweets.pl: insert tweets using memcached
• 3. counter.sql: revise app to count tweets per user
Memcache tips

- `perldoc Cache::Memcached`
- The file in labs/ already connects to a server for you.
- In an actual Memcache command, the 3 fields of the value should be separated by tab characters ("\t")
ClusterJ Query

Craig L Russell
Architect, Oracle Corp.
ClusterJ Query

- Builder pattern
- Similar to JPA criteria query
- Predicates compare column values to parameters:
  - equal, lessEqual, greaterEqual, lessThan, greaterThan, between, in
  - comparison with null
- Combine predicates using binary and unary operators:
  - or, and, not
- Execution is optimized to use indexes
  - primary or unique key lookup
  - ordered scan for complete or partial keys
  - table scan if no index can be used
ClusterJ Query Interfaces

- **Query**
  - the "executable" part
  - Parameters and limits are bound to this instance

- **QueryBuilder**
  - used to build queries

- **QueryDomainType**
  - represents the user's domain type for queries

- **PredicateOperand**
  - represents either a query parameter or a query property

- **Predicate**
  - represents a comparison between a property and a parameter
  - can be combined using "and", "or", and "not" operators
ClusterJ Query Optimizer

- Optimizer looks for indexes
- PRIMARY key all columns equal
- PRIMARY key leading columns equal
- Unique (hash) key equal
- Ordered (btree) key equal, greater, less
- Table scan if no indexes are usable
- All terms used for filter
- After parameters are bound, ask for query plan:
  - Query.explain()
  - Keys: Query.SCAN_TYPE, Query.INDEX_USED
Query Example

```java
QueryDomainType<Tweets> qdtTweets = 
    builder.createQueryDefinition(Tweets.class) ;

Predicate eqAuthor = qdtTweets.get("author")
    .eq(qemp.param("pauthor"));
Predicate gtTime_stamp = qdtTweets.get("time_stamp")
    .greaterEqual(qemp.param("ptime_stamp"));

qdtTweets.where(eqAuthor.and(gtTime_stamp));

Query query = session.createQuery(qdtTweets) ;

query.setParameter("pauthor", getAuthor());
query.setParameter("ptime_stamp", getTime());

List<Tweets> results = query.getResultList() ;
```
ClusterJ Query Exercises

• Write Find.java program
  – Skeleton in labs/test/Find.java
  – Query by author
  – Query by hashtag
  – Query by date range
  – Query by combining above criteria
ClusterJ Query Extra Credit

- Write Delete.java program
  - skeleton in labs/test/Delete.java
  - Delete by author
  - Delete by hashtag
  - Delete by date range
  - Delete by combining above criteria
Final Questions & Answers