





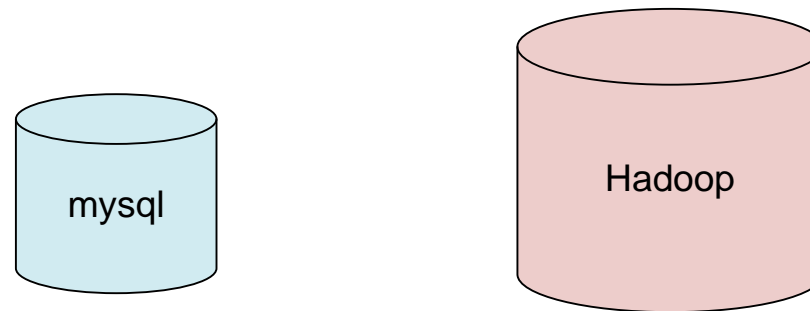
# sqoop

## Automatic database import

Aaron Kimball  
Cloudera Inc.  
June 18, 2009

# The problem

- Structured data already captured in databases should be used with unstructured data in Hadoop
- Tedious “glue” code necessary to wrap database records for consumption in Hadoop



*Where's the bridge?*

# DBInputFormat

- Connects to JDBC interface
  - Selects records out of tables, arbitrary queries
  - Provides interface to use arbitrary input queries, tables, databases
  - Records written to *DBWritable*, provided as value to Mapper
-

# DBWritable

- You define a class to hold a row from the database
    - Must be able to read from JDBC *ResultSet* into fields
    - Must be able to write to JDBC *PreparedStatement*
  - Should also implement regular *Writable*
-

# DBWritable Example

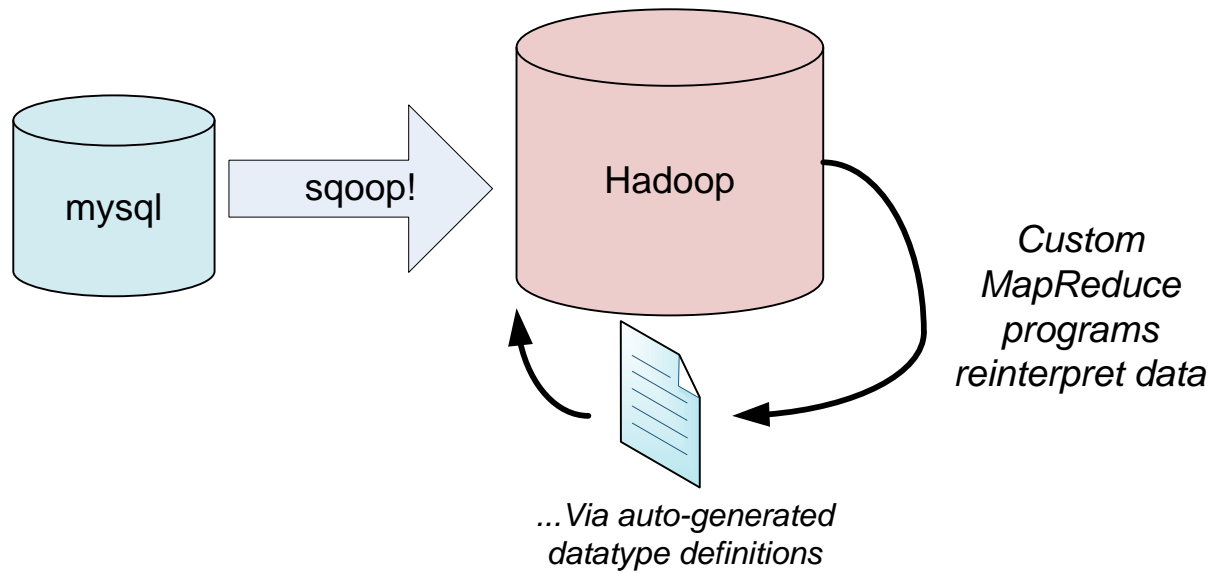
```
1. class MyRecord implements Writable, DBWritable {
2.     long pkey;
3.     long val;
4.     public void readFields(DataInput in) throws
5.         IOException {
6.         this.pkey = in.readLong();
7.         this.val = in.readLong();
8.     }
9.     public void readFields(ResultSet resultSet)
10.        throws SQLException {
11.        this.pkey = resultSet.getLong(1);
12.        this.val = resultSet.getLong(2);
13.    }
14. }
```

---

# A Direct Type Mapping

JDBC Type	Java Type
CHAR	String
VARCHAR	String
LONGVARCHAR	String
NUMERIC	java.math.BigDecimal
DECIMAL	java.math.BigDecimal
BIT	boolean
TINYINT	byte
SMALLINT	short
INTEGER	int
BIGINT	long
REAL	float
FLOAT	double
DOUBLE	double
BINARY	byte[]
VARBINARY	byte[]
LONGVARBINARY	byte[]
DATE	java.sql.Date
TIME	java.sql.Time
TIMESTAMP	java.sql.Timestamp

# Sqoop = SQL-to-Hadoop



# Sqoop: Features

- JDBC-based interface (MySQL, Oracle, PostgreSQL, etc...)
  - Automatic datatype generation
    - Reads column info from table and generates Java classes
    - Can be used in further MapReduce processing passes
  - Uses MapReduce to read tables from database
    - Can select individual table (or subset of columns)
    - Can read all tables in database
  - Supports most JDBC standard types and null values
-

# Example input

```
mysql> use corp;
```

```
Database changed
```

```
mysql> describe employees;
```

Field	Type	Null	Key	Default	Extra
id	int(11)	NO	PRI	NULL	auto_increment
firstname	varchar(32)	YES		NULL	
lastname	varchar(32)	YES		NULL	
jobtitle	varchar(64)	YES		NULL	
start_date	date	YES		NULL	
dept_id	int(11)	YES		NULL	

# Loading into HDFS

```
$ sqoop --connect jdbc:mysql://db.foo.com/corp \  
    --table employees
```

- Imports “employees” table into HDFS directory
- Generates employees.java for your use

# Auto-generated class

```
public class employees {  
    public Integer get_id();  
    public String get_firstname();  
    public String get_lastname();  
    public String get_jobtitle();  
    public java.sql.Date get_start_date();  
    public Integer get_dept_id();  
    // and serialization methods for Hadoop  
}
```

# Additional Options

- Multiple data representations supported
    - TextFile – ubiquitous; easy import into Hive
    - SequenceFile – supports compression, higher performance
  - Supports local and remote Hadoop clusters, databases
  - Can select a subset of columns, adjust input ordering
-

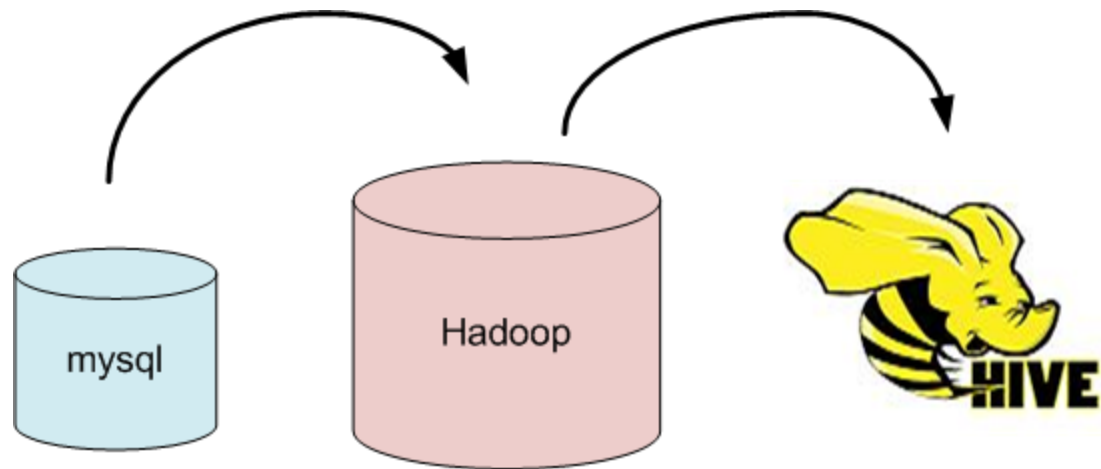
# Limitations

- JDBC implementation based on DBInputFormat
    - Requires unique sorting column (will autodetect primary key)
    - Performance is less-than-impressive
  - No mechanism to filter rows (i.e., no WHERE clauses)
  - No support for LOBs
-



# Hive Integration

- Generates CREATE TABLE / LOAD DATA INPATH script
- After data is imported to HDFS, auto-execute Hive script
- Follow-up step: Loading into partitions



# Future Directions

- WHERE clauses for incremental import (in progress)
  - Output format control (delimiters and parsing)
  - “unsqoop” to export data from Hive into database
  - More database-specific bulk export mechanisms
  - Hive: Support for partition columns, buckets, SequenceFiles
  - Integration with Avro
  - Large datatypes (CLOB, BLOB, etc.)
-

# Current Status

- Pushed to the public Hadoop JIRA for 0.21.0:
    - HADOOP-5815 (Sqoop)
    - HADOOP-5844 (mysqldump integration)
  - 95% complete: Hive integration
  - Working on bug fixes, quoting, WHERE clauses
  - Available now in Cloudera's Distribution for Hadoop
-

# Conclusions

- Most database import tasks are “turning the crank”
- Sqoop can automate a lot of this
  - Allows more efficient use of existing data sources in concert with new, unstructured data
- Available as part of Cloudera’s Distribution for Hadoop

[www.cloudera.com/hadoop-sqoop](http://www.cloudera.com/hadoop-sqoop)

[www.cloudera.com/hadoop](http://www.cloudera.com/hadoop)

[issues.apache.org/jira/browse/HADOOP-5815](https://issues.apache.org/jira/browse/HADOOP-5815)

[aaron@cloudera.com](mailto:aaron@cloudera.com)

---



(c) 2008 Cloudera, Inc. or its licensors. "Cloudera" is a registered trademark of Cloudera, Inc.. All rights reserved. 1.0