The F# Type Provider for Hadoop

# Overview

One aim is to be able to go through as much of a standard Hive tutorial using F# alone, e.g.

<https://cwiki.apache.org/Hive/tutorial.html>

# Tutorial

## Establishing a Data Context

You use the type provider to establish a data connection:

#r "Samples.Hadoop.TypeProviders.dll"

open Samples.Hadoop

open Samples.Hadoop.Hive

open Microsoft.FSharp.Linq.NullableOperators

type Data = HiveTypeProvider<Server="192.168.233.208",UserName=user,Password=pwd>

let ctxt = Data.GetDataContext()

## Creating Tables

You use ctxt.DataContext.ExecuteCommand to create tables

ctxt.DataContext.ExecuteCommand("""CREATE [EXTERNAL] TABLE [IF NOT EXISTS] [db\_name.]table\_name

[(col\_name data\_type [COMMENT col\_comment], ...)]

[COMMENT table\_comment]

[PARTITIONED BY (col\_name data\_type [COMMENT col\_comment], ...)]

[CLUSTERED BY (col\_name, col\_name, ...) [SORTED BY (col\_name [ASC|DESC], ...)] INTO num\_buckets BUCKETS]

[SKEWED BY (col\_name, col\_name, ...) ON ([(col\_value, col\_value, ...), ...|col\_value, col\_value, ...]) (Note: only available starting with 0.10.0)]

[

[ROW FORMAT row\_format] [STORED AS file\_format]

| STORED BY 'storage.handler.class.name' [WITH SERDEPROPERTIES (...)] (Note: only available starting with 0.6.0)

]

[LOCATION hdfs\_path]

[TBLPROPERTIES (property\_name=property\_value, ...)] (Note: only available starting with 0.6.0)

[AS select\_statement] """

For example:

ctxt.DataContext.ExecuteCommand("

CREATE TABLE page\_view(viewTime INT, userid BIGINT,

page\_url STRING, referrer\_url STRING,

ip STRING COMMENT 'IP Address of the User')

COMMENT 'This is the page view table'

PARTITIONED BY(dt STRING, country STRING)

CLUSTERED BY(userid) SORTED BY(viewTime) INTO 32 BUCKETS

ROW FORMAT DELIMITED

FIELDS TERMINATED BY '1'

COLLECTION ITEMS TERMINATED BY '2'

MAP KEYS TERMINATED BY '3'

STORED AS SEQUENCEFILE;

")

## Selecting from Tables

## Filtering Tables

## Writing Intermediate Tables

## Writing Rows and Partitions to Tables

## Sampling

Samples can be drawn from bucketed tables using “sampleBucket *bucketNumber numberOfBuckets*”

hiveQuery {

for i in ctxt.iris\_partitioned do

sampleBucket 3 16

select i.petallength

}

## Timeouts

Timeouts for queries can be specified.[[1]](#footnote-1)

hiveQuery { for x in ctxt.abalone do

where (x.length < 0.6)

timeout 1

select (x.length, x.rings) }

## Aggregates

hiveQuery { for x in ctxt.abalone do averageBy x.length }

## Group By

hiveQuery {

for x in ctxt.abalone do

groupBy x.gender into g

let a = hiveQuery { for x in g do averageBy (x.height \* x.diameter) }

let b = hiveQuery { for x in g do averageBy x.shellweight }

select (g.Key,a,b) }

giving the Hive query

SELECT \*,AVG(height \* diameter),AVG(shellweight) FROM abalone GROUP BY gender

## Sort By

## Units of Measure

Columns are annotated with units of measure by adding “(unit=*unit*)” to the comment description of a column.

## Required Columns

Columns are annotated as “required” (non-Nullable) by adding “(required)” to the comment description of the column.

# Technical Features

## Static Parameters

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Static Parameter** | **Description** | **Notes** | **Default** | **Status** | **Test?** |
| Server | The Hive head server | **tryfsharp** is special |  |  | Yes |
| Port | The port for the Hive service |  | 10000 |  | Yes |
| UserName |  | Not clear if Hive cares |  |  | Yes |
| Password |  | Not clear if Hive cares |  |  | Yes |
| Authentication | Integer | Becomes AUTHENTICATION=n in connection string. “-1” means no AUTHENTICATION added |  |  |  |
| UseUnitAnnotations |  |  | false |  |  |
| UseRequiredAnnotations |  |  | true |  |  |
| DefaultQueryTimeout |  |  | 20000 ms |  |  |
| DefaultMetadataTimeout |  |  | 5000 ms |  |  |
| ??? |  | Some Hive connection strings seem to use AUTH\_DATA=… |  | NYI |  |
| ExtraTables | Proposal: a set of CREATE TABLES commands and other scripts assumed to have been executed, to give strong types of random extra tables (especially partitioned tables)?? |  |  | NYI |  |

## Multiple Databases

The provider only supported the “default” database.

## Column Names

The provider currently always used the lowercase column names returned by

DESCRIBE extended tableName

The results of “Detailed Table Information” are also parsed to detect the partition keys, bucket keys and sort keys.

## Supported Column Types

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Hive Column Type** | **F# Type** | **Notes** | **Status** | **Test?** |
| FLOAT | float32  Nullable<float32>  float32<unit>  Nullable<float32<unit> |  |  |  |
| DOUBLE | double  Nullable<double>  double<unit>  Nullable<double<unit> |  |  | Yes |
| TINYINT | sbyte  Nullable<sbyte> |  |  |  |
| SMALLINT | int16  Nullable<int16> |  |  |  |
| INT | int  Nullable<int> |  |  |  |
| BIGINT | int64  Nullable<int64> |  |  |  |
| BOOLEAN | bool  Nullable<bool> |  |  |  |
| STRING | string |  |  |  |
| BINARY | byte[] |  | NYI |  |
| TIMESTAMP | System.DateTimeUtc (?) |  | NYI |  |
| DECIMAL | System.Decimal |  |  |  |

The following are not yet implemented or mapped:

* Record columns in tables
* Array columns in tables
* Maps columns in tables
* Unions in columns in tables

union\_type

: UNIONTYPE < data\_type, data\_type, ... >

## Partition Keys for Partitioned Tables

Some columns are known as being partition keys. These are identified by are the Hive table descriptions returned by

DESCRIBE extended tableName

e.g.

partitionKeys:[FieldSchema(name:dt, type:string, comment:null), FieldSchema(name:country, type:string, comment:null)]

## Unit and Required Annotations in Hive Metadata

Unit annotations are added in Hive table descriptions returned by

DESCRIBE extended tableName

e.g.

FieldSchema(name:visceraweight, type:double, comment:gut weight (after bleeding) **(g, required)**)

The annotations allowed are

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Annotation** | **Notes** | **Default** | **Status** | **Test?** |
|  |  |  |  |  |
| (unit=*unit*) |  |  |  |  |
| (unit=*unit*, required) |  |  |  |  |
| (unit=required) |  |  |  | Yes |

*Unit* can be of the form

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Annotation** | **Notes** | **Default** | **Status** | **Test?** |
| Canonical abbreviations reported by Freebase | Excludes those using /, ^, \* |  |  |  |
| l and ml |  |  |  |  |
| Unique non-canonical abbreviations reported by Freebase |  |  | NYI |  |
| (unit / unit) \* … \* (unit / unit) |  |  |  |  |
| unit^n |  |  | NYI |  |

Precedences: ^ binds tightest, then \*, then /

## Transformation of data according to unit annotations

Units are translated to SI units. At runtime, data must also be scaled when queries are executed. This happens when

* Selecting a column of unit U
* Selecting an average, minimum, maximum, etc. for an expression of type U

In this case,

* the unit U is statically it is transformed to SI unit SU
* at runtime the scale/shift adjustment for U 🡪 SU should be applied (e.g. cm 🡪 m, or cm^2 🡪 m^2)

Likewise, data used in the SI unit system must be translated to the non-SI unit system of the target tables. This happens when a constant is used as part of a query expression, e.g. in a “where” clause. In this case we must apply the reverse SU 🡪 U.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Transform** | **Notes** | **Default** | **Status** | **Test?** |
| Selecting columns |  |  |  | Yes |
| Selecting average |  |  |  | Yes |
| Selecting min, max |  |  |  | Yes |
| Selecting sum |  |  |  | Yes |
| Column comparison in ‘where’ |  |  |  | Yes |

## The Provided Type

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Method** | **Notes** | **Default** | **Status** | **Test?** |
| GetDataContext |  |  |  | Yes |

The GetDataContext method has the following optional arguments, which default to their static values. [

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Method** | **Notes** | **Default** | **Status** | **Test?** |
| server |  |  |  |  |
| port |  |  |  |  |
| auth |  |  |  |  |
| password |  |  |  |  |
| queryTimeout |  |  |  |  |
| metadataTimeout |  |  |  |  |

## The Strongly Typed Data Context Object (returned by GetDataContext)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Properties** | **Notes** | **Default** | **Status** | **Test?** |
| DataContext |  |  |  | Yes |
| One property for each Hive table |  |  |  | Yes |

## The Weakly Typed Data Context Object (returned by ctxt.DataContext)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Members** | **Description** | **Notes** | **Status** | **Test?** |
| GetTable<T>(string) | See below |  |  | Yes |
| ExecuteCommand(string, ?timeout) | Accepts a Hive query command and returns the output diagnostics/response/??? |  |  |  |
| ExecuteQuery(string, ?timeout) | Accepts a Hive query string and returns a sequence of objects. The spec for this is TBD |  | NYI |  |
| ExecuteQueryValue(string, ?timeout) | Accepts a Hive query string and returns a single value. The spec for this is TBD |  | NYI |  |

## ctxt.GetTable<T>

Behaves as follows

* T must be “HiveDataRow” or an F# record type
* Record entries must be in the same order and have identical names in F# record type and Hive metadata

Returns a HiveTable<T>

## Queries

Queries are of the form:

hiveQuery { for x in ctxt.Table do … }

The following constructs may appear in queries

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Construct** | **Description** | **Notes** | **Status** | **Test?** |
| select *expr* |  | Uses a client-side tail operation if ‘expr’ is not a table column |  |  |
| where *expr* |  | *expr* must be of an appropriate form |  |  |
| count |  |  |  |  |
| take *expr* |  | *expr* must evaluate to a constant integer |  |  |
| timeout *expr* | Adjusts the timeout associated with the query. | *expr* must evaluate to a constant integer |  |  |
| sortBy *expr*  sortByNullable *expr*  sortByDescending *expr*  sortByDescendingNullable *expr*  thenBy *expr*  thenByDescending *expr*  thenByNullable *expr*  thenByDescendingNullable *expr* |  |  | NYI |  |
| groupBy *expr* |  |  | NYI |  |
| writeTable *string-expr* | Writes to the intermediate table as a side effect |  |  |  |
| writeLocalFile *string-expr* | Writes to the local file as a side effect |  |  |  |
| writeDistributedFile *string-expr* | Writes to a DFS file as a side effect |  |  |  |
| sampleBucket *int-expr int-expr* | Draws samples from the given bucket | Table must use CLUSTERED BY |  |  |
| writeRows (table.NewRow(*column-exprs))*  insertRows (table.NewRow(*column-exprs))* | Writes the rows to a partition of the identified table as a side effect, overwriting any existing rows in that partition |  |  |  |
| skip *expr* | Allows paging. | Needs to use ‘ROW\_NUMBER’ sorting | NYI |  |

## The HiveQuery<T> type

All Supports

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Member** | **Description** | **Notes** | **Status** | **Test?** |
| q.QueryString | Returns the text of the Hive query |  |  |  |
| a.Execute(?timeout) | Executes the query and returns a sequence of objects |  |  |  |
| IEnumerable<T> | Executes the query and returns a sequence of objects | uses the default timeout |  |  |
| IQueryable<T> | Allows LINQ IQueryable compositions | uses the default timeout | NYI |  |

## The HiveTable<T> type

Extends HiveQuery<T>

Supports

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Member** | **Description** | **Notes** | **Status** | **Test?** |
| q.GetPartitions() | Returns the names of the partitions |  |  |  |
| q.WithXYZ(xyz)  *one for each named partition* | Spec to be decided. Should allow navigation of the partitions in a cube-like way |  | NYI |  |

## Partitions

Hive tables support data-cube-like partitioning using named partitions

This could be projected in a cube-like way:

q.WithXYZ(“x”).WithABC(“a”)

or through cube of optional arguments:

q.With(xyz=”x”, abc=”a”)

If enabled, some documentation notes for the partitions might be accessed using:

DESCRIBE EXTENDED page\_view PARTITION (ds='2008-08-08');

## User Defined Functions and Custom Map/Reduce Scripts

These are not supported

## Operators in Expressions in Queries

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **F# Construct** | **Hive Construct** | **Notes** | **Status** | **Test?** |
| *boolean constants* |  | need to check various forms |  |  |
| *string constants* |  | need to check various forms |  |  |
| *integer constants* |  | need to check various forms |  |  |
| *floating point constants* |  | need to check various forms |  |  |
| *relational operators*  *expr* < *expr*  *expr* <= *expr*  *expr* > *expr*  *expr* >= *expr*  *expr* = *expr*  *expr* <> *expr*  *…* |  |  |  |  |
| *arithmetic operators*  *expr* + *expr*  *expr* – *expr*  *…* |  |  |  |  |
| *nullable variations of the above* |  |  |  |  |
| a = null | A IS NULL |  | NYI |  |
| a <> null | A IS NOT NULL |  | NYI |  |
| ??? | A LIKE B |  | NYI |  |
| ??? | A RLIKE B  A REGEXP B |  | NYI |  |
| a.[n] | A[n] |  | NYI |  |
| m.[key] | M[key] |  | NYI |  |
| s.x | S.x |  | NYI |  |
| ??? | rand() |  | NYI |  |
| ??? | rand(seed) |  | NYI |  |
| ??? | regexp\_replace(A,B,C) |  | NYI |  |
| ??? | size(map) |  | NYI |  |
| ??? | size(array) |  | NYI |  |
| ??? | from\_unixtime(int unixtime) |  | NYI |  |
| ??? | to\_date(string timestamp) |  | NYI |  |
| ??? | get\_json\_object(string json\_string, string path) |  | NYI |  |
| ??? | sum(DISTINCT col) |  | NYI |  |
| ??? | avg(DISTINCT col) |  | NYI |  |
| *lots more here* |  |  |  |  |
| *some functions are NYI as indicated in the source* |  |  | NYI |  |

# Binary/Source Compatibility

Source compatibility of compiled code using the Hive provider:

* The column names (not positions) are present in source code. So, if any of the following changes occur, for a column accessed in the source code, the code will no longer type-check when re-compiled or re-checked:
  + If an individual column names changes
  + If a column disappears
  + If a column changes its data type
  + The required/unit annotations on the column change

However, not all changes are breaking, e.g.

* If columns are re-ordered

Likewise, binary compatibility of compiled code using the Hive provider is similar:

* The column names (not positions) are persisted into compiled code. So, if any of the following changes occur, for a column accessed in the compiled code, and the column is accessed at runtime, then a runtime error occurs:
  + If an individual column names changes
  + If a column disappears
  + If a column changes its data type
  + The required/unit annotations on the column change

However, not all changes are breaking, e.g. if columns are re-ordered

## Unsupported Features

* Adding a JAR file reference is not supported
* Adding a Python mapper file reference is not supported
* Using “TRANSFORM….USING….AS” is not supported, i.e. SELECT TRANSFORM(col1,col2) USING 'mapper.py' AS (col1,col2) FROM data"
* NO: Ability to evaluate aggregations on multiple "group by" columns for the data stored in a table. Writing a table to the local directory is not supported, i.e. OVERWRITE LOCAL DIRECTORY '/tmp/reg\_3' SELECT ... FROM ...;
* Loading a table from the local directory is not supported, i.e. "LOAD DATA LOCAL INPATH './examples/foo' OVERWRITE INTO TABLE bar;"
* GROUP BY is not yet supported
* Multitable Insert is not supported, i.e.

FROM src

INSERT OVERWRITE TABLE dest1 SELECT src.\* WHERE src.key < 100

INSERT OVERWRITE TABLE dest2 SELECT

* YES: Ability to filter rows from a table using a where clause.
* YES: Ability to select certain columns from the table using a select clause.
* YES: Ability to store the results of a query into another table.
* YES: Ability to manage tables and partitions (create, drop and alter).
* NO: Ability to do equi-joins between two tables.
* NO: Ability to store the results of a query in a hadoop dfs directory.
* NO: Ability to plug in custom scripts in the language of choice for custom map/reduce jobs.

## Proxying

When running in Silverlight, we connect to a server-side proxy which then allows us to talk to the hadoop instance.

When running in Visual Studio we used to connect to a local background process using an Anonymous named pipe which plays the same role. However the Visual Studio proxy is not used for Hive commands at the moment – the devenv.exe process executes the commands directly.

## Random Notes

// TODO: Take/Skip -.> ROW\_NUMBER() OVER(ORDER BY A,B,C DESC)

// TODO: RowNumber -.> ROW\_NUMBER() OVER(ORDER BY A,B,C DESC)

//

// #if AllowMultilpeCommands

// if a then b -.> IF a BEGIN b END

// if a then b else c -.> IF a BEGIN b END ELSE BEGIN c END

//

// let var = ... -.> DECLARE @var INT <-. check me

// for var1 in e -.> DECLARE @var INT SELECT @var1 = e FROM tableName WHERE where

// let var1 = e -.> SET @var = e

//

// // fix up any order-by's

// expression = OrderByRewriter.Rewrite(this.Language, expression);

// expression = base.Translate(expression);

// // convert skip/take info into RowNumber pattern

// expression = SkipToRowNumberRewriter.Rewrite(this.Language, expression);

// // fix up any order-by's we may have changed

// expression = OrderByRewriter.Rewrite(this.Language, expression);

// expression = RedundantColumnRemover.Remove(expression);

// expression = UnusedColumnRemover.Remove(expression);

1. These are currently somewhat unreliable due to lack of cancellation in the ODBC Hive driver [↑](#footnote-ref-1)