链滴

sql4es 示例

作者：flowaters

原文链接：https://ld246.com/article/1502277406358

来源网站：链滴

背景

sql4es(Sql-for-Elasticsearch)，是一个Elasticsearch(ES)的JDBC驱动。

特点

sql4es 支持JDBC的标准接口：Connection, Statement, PreparedStatment, ResultSet, Batch and ataBase- / ResultSetMetadata。

通过sql4es，可以将ES当做数据库来使用。

目前在 sqlWorkbench/J 和 Squirrel 上经过了测试。

特点

ES的SQL特征

1. 事务操作，不支持
2. UPDATE操作，成本比较高
3. INSERT nested对象，支持
4. LIMIT offset, number中的offset，不支持
5. 父子文档，不支持
6. Count (Distinct...)，是估算

之后的改进

目前最新的sql4es版本4.1，仅支持ES 2.0 ~ 2.4，还不支持 ES 5.x。

支持的SQL语法

下列内容来自官方github文档

- SELECT: fetches documents (with or without scoring) or aggregations from elasticsearch
  - COUNT (DISTINCT ...), MIN, MAX, SUM, AVG
  - DISTINCT
  - WHERE (=, >, >=, <, <=, <>, IN, LIKE, AND, OR, IS NULL, IS NOT NULL, NOT [condition])
  - GROUP BY
  - HAVING
  - ORDER BY
  - LIMIT (without offset, offsets are not supported by sql4es)
- CREATE TABLE (AS) creates an index/type and optionally indexes the result of a query into it
- CREATE VIEW (AS): creates an alias, optionally with a filter
- DROP TABLE/VIEW removes an index or alias

- INSERT INTO (VALUES | SELECT): inserts documents into an index/type; either provided valu s or results of a query. Possible to UPDATE documents using INSERT by specifying existing do ument _id's
- UPDATE: executed as an elasticsearch Upsert
- DELETE FROM (WHERE): removes documents
- USE: selects an index as the driver's active one (used to interpret queries)
- EXPLAIN SELECT: returns the Elasticsearch query performed for a SELECT statement
- Table aliases like SELECT ... FROM table1 as T1, table2 t2...
    - Table aliases are parsed but not used during query execution

demo示例

从github上下载其源码，跑其中test中的单测

注：单测中mock了一个server，所以不必启动es server.

实际使用

```
// register the driver and get a connection for index 'myidx'
Class.forName("nl.anchormen.sql4es.jdbc.ESDriver");
Connection con = DriverManager.getConnection("jdbc:sql4es://localhost:9300/myidx?cluster.
ame=your-cluster-name");
Statement st = con.createStatement();
// execute a query on mytype within myidx
ResultSet rs = st.executeQuery("SELECT * FROM mytype WHERE something >= 42");
ResultSetMetaData rsmd = rs.getMetaData();
int nrCols = rsmd.getColumnCount();
// get other column information like type
while(rs.next()){
    for(int i=1; i<=nrCols; i++){
        System.out.println(rs.getObject(i));
    }
}
rs.close();
con.close();
```

更多示例

本来不想摘过来的，无奈官网的太精彩了。

概念

- Database = Index
- Table = Type
- Record = document
- Column = Field

- View = Alias

SELECT

```
/* basic syntax */
SELECT [field (AS alias)] FROM [types] WHERE [condition] GROUP BY [fields] HAVING [conditi
n] ORDER BY [field (ASC|DESC)] LIMIT [number]
```

具体示例

```
/* the following wil explode any nested objects into a lateral view */
SELECT * from mytype

SELECT _id as id, myInt, myString FROM mytype WHERE myInt >= 3 OR (myString IN ('hello','h
','bye') AND myInt <= 3)

/* If nestedDoc contains 2 fields the result will be exploded to [myInt,nestedDoc.field1, neste
Doc.field2] */
SELECT myInt, nestedDoc FROM mytype WHERE myInt > 3 AND myString <> 'bye'

/* If the array contains 3 objects the resultset will contain 3 rows, despite the LIMIT used! */
SELECT array_of_nested_objects FROM mytype LIMIT 1
```

Tables/Types

```
/* fetch some data from type */
SELECT DISTINCT field, count(1) FROM type, query_cache
/* exactly the same as above but now also hitting the query cache */
SELECT DISTINCT field, count(1) FROM type
```

Text matching, search and scoring

```
/* term query */
SELECT _score, myString FROM mytype WHERE myString = 'hello' OR myString = 'there'
/* Same as above */
SELECT _score, myString FROM mytype WHERE myString IN ('hello', 'there')
/* use of NOT; find all documents which do not contain 'hello' or 'there' */
SELECT _score, myString FROM mytype WHERE NOT myString IN ('hello', 'there')

/* check for NULL values (missing fields) */
SELECT myInt FROM mytype WHERE myString NOT NULL
SELECT myInt FROM mytype WHERE myString IS NULL

/* phrase query */
SELECT _score, highlight(myString), myString FROM mytype WHERE myString = 'hello there'
/* wildcard query */
SELECT _score, myString FROM mytype WHERE myString = 'hel%'
/* a search for exactly the same as the first two */
SELECT _score, highlight(mystirng) FROM mytype WHERE _search = 'myString:(hello OR there)
```

Get document by _id

```
SELECT * FROM mytype WHERE _id = 'whatever_id'
SELECT * FROM mytype WHERE _id = 'whatever_id' AND myInt > 3
SELECT * FROM mytype WHERE _id = 'whatever_id' OR _id = 'another_ID' /* WRONG */
SELECT * FROM mytype WHERE _id IN ('whatever_id', 'another_ID') /* CORRECT */
```

Aggregation

```
/* Aggregates on a boolean and returns the sum of an int field in desc order */
SELECT myBool, sum(myInt) as summy FROM mytype GROUP BY myBool ORDER BY summy
ESC

/* This is the same as above */
SELECT DISTINCT myBool, sum(myInt) as summy ROM mytype ORDER BY summy DESC

/* Aggregates on a boolean and returns the sum of an int field only if it is larger than 1000 */
SELECT myBool, sum(myInt) as summy ROM mytype GROUP BY myBool HAVING sum(myInt)
> 1000

/* Gets the average of myInt in two different ways... */
SELECT myBool, sum(myInt)/count(1) as average, avg(myInt) FROM mytype GROUP BY myBoo


/* Calculates the percentage of growth of the myInt value acros increasing dates */
SELECT myDate, sum(myInt)/sum(myInt)[-1]*100 FROM mytype GROUP BY myDate ORDER BY
myDate ASC

/* aggregation on all documents without a DISTINCT or GROUP BY */
SELECT count(*), SUM(myInt) from mytype

/* the following will NOT WORK, a DISTINCT or GROUP BY on mytext is required */
SELECT mytext, count(*), SUM(myInt) from mytype
```

注：

1. aggregation中的limit会被忽略
2. fields间的运算是由presto框架实现的
3. having算子目前是由presto框架实现的
4. aggregated结果的排序，目前是由presto框架实现的

EXPLAIN

```
EXPLAIN [SELECT statement]
```

USE

```
USE [index / alias]
```

CREATE

支持多种CREATE方式

直接CREATE

CREATE TABLE (index.)type ([field] "[field definition]" (, [field2])...) WITH (property="value" (, p
operty2=...))

示例

```
/* creates a mapping for mytype within newindex with a template to store any strings without
analysis */
CREATE TABLE index.mytype (
    myInt "type:integer",
      myDate "type:date, format:yyyy-MM-dd"
      myString "type:string, index:analyzed, analyzer:dutch"
) WITH (
  dynamic_templates="[{
 default_mapping: {
 match: *,
 match_mapping_type: string,
 mapping: {type: string, index: not_analyzed }
 }
 }]"
)
```

通过SELECT来CREATE

CREATE TABLE (index.)type AS SELECT ...

示例

```
/*Create another index with a type mapping based on the mapping created before*/
CREATE TABLE index.mytype AS SELECT myDate as date, myString as text FROM anyType

/* create a type with a (possibly expensive to calculate) aggregation result */
CREATE TABLE index.myagg AS SELECT myField, count(1) AS count, sum(myInt) AS sum from
nyType GROUP BY myField ORDER BY count DESC
```

创建视图

CREATE VIEW [alias] AS SELECT * FROM index1 (, [index2])... (WHERE [condition])

删除表

DROP TABEL [index] / DROP VIEW [alias]

示例

```
/*Create an elasticsearch alias which includes two indexes with their types */
CREATE VIEW newalias AS SELECT * FROM newindex, newindex2

/* Same as above but with a filter*/
CREATE VIEW newalias AS SELECT * FROM newindex, newindex2 WHERE myInt > 99
```

```
/*Use the alias so it can be queried*/
USE newalias

/* removes myindex and remove newalias */
DROP TABLE myindexindex
DROP VIEW newalias
```

插入数据

```
INSERT INTO (index.)type ([field1], [field2]...) VALUES ([value1], [value2], ...), ([value1], ...), ...

INSERT INTO (index.)type SELECT ...
```

示例

```
/* Insert two documents into the mytype mapping */
INSERT INTO mytype (myInt, myDouble, myString) VALUES (1, 1.0, 'hi there'), (2, 2.0, 'hello!')

/* insert a single document, using quotes around nested object fields */
INSERT INTO mytype (myInt, myDouble, "nestedObject.myString") VALUES (3, 3.0, 'bye, bye')

/* update or insert a document with specified _id */
INSERT INTO mytype (_id, myInt, myDouble) VALUES ('some_document_id', 4, 4.0)

/* copy records from anotherindex.mytype to myindex.mytype that meet a certain condition *

USE anotherindex
INSERT INTO myindex.mytype SELECT * from newtype WHERE myInt < 3
```

按条件删除数据

```
DELETE FROM type (WHERE [condition])
```

示例

```
/* delete documents that meet a certain condition*/
DELETE FROM mytype WHERE myInt == 3

/*delete all documents from mytype*/
DELETE FROM mytype
```

更新数据

```
UPDATE index.type SET field1=value, fiedl2='value', "doc.field"=value WHERE condition
```

配置项

见官方文档

FAQ

使用中，遇到了java安全策略的问题。测试环境解决方式是：

生成文件 ~/.java.policy

内容如下

```
/* AUTOMATICALLY GENERATED ON Wed Aug 09 17:32:33 CST 2017*/
/* DO NOT EDIT */

grant {
  permission java.lang.RuntimePermission "accessClassInPackage.sun.misc";
  permission java.io.FilePermission "<<ALL FILES>>", "read,write";
  permission java.lang.reflect.ReflectPermission "suppressAccessChecks";
};
```

关于java安全策略，具体见下一篇.