[SQL4ES介绍 2](#_Toc332)

[使用方法 2](#_Toc28973)

[通过maven库方式 2](#_Toc21280)

[通过导入需要的jar包依赖 2](#_Toc10097)

[使用步骤 2](#_Toc20271)

[加载JDBC驱动程序   2](#_Toc28760)

[创建数据库的连接    3](#_Toc28770)

[创建一个Statement 3](#_Toc32675)

[执行SQL语句    3](#_Toc13811)

[处理结果    4](#_Toc28729)

[关闭JDBC对象     4](#_Toc19830)

[完整示例 4](#_Toc26238)

[支持语法 5](#_Toc19037)

[SELECT 5](#_Toc3666)

[文本匹配，搜索和评分 6](#_Toc15976)

[通过\_id获取文档 7](#_Toc6705)

[通过\_search查询 7](#_Toc25321)

[通过\_child、\_parent执行父子关系查询 8](#_Toc21500)

[聚合 8](#_Toc16572)

[函数 9](#_Toc5503)

[CREATE AND DROP 11](#_Toc30049)

[CREATE TABLE (index.)type ([field] "[field definition]" (, [field2])...) WITH (property="value" (, property2=...) ) 11](#_Toc28076)

[CREATE TABLE (index.)type AS SELECT ... 12](#_Toc26021)

[CREATE VIEW [alias] AS SELECT \* FROM index1 (, [index2])... (WHERE [condition]) 12](#_Toc8479)

[DROP TABEL [index] / DROP VIEW [alias] 12](#_Toc29359)

[INSERT AND DELETE 13](#_Toc26484)

[INSERT INTO (index.)type ([field1], [field2]...) VALUES ([value1], [value2], ...), ([value1], ...), … 13](#_Toc11384)

[INSERT INTO (index.)type SELECT … 13](#_Toc5219)

[DELETE FROM type (WHERE [condition]) 14](#_Toc5360)

[UPDATE 14](#_Toc4813)

[UPDATE index.type SET field1=value, fiedl2='value', "doc.field"=value WHERE condition 14](#_Toc24927)

[SHOW COLUMNS 14](#_Toc2329)

[配置 15](#_Toc21449)

[注意事项 15](#_Toc31123)

[不支持的语法 16](#_Toc21586)

SQL4ES使用说明文档

# SQL4ES介绍

sql4es，即 Sql-for-Elasticsearch，是 Elasticsearch 的 JDBC 驱动程序，是一个用于Elasticsearch 2.x,5.x,6.x版本的能实现大多数JDBC4.1接口的驱动程序: Connection, Statement, PreparedStatment, ResultSet, Batch, DatabaseMetaData 和ResultSetMetadata。

# 使用方法

## 通过maven库方式

1.添加新的maven源

<!-- 增加新的maven源 -->  
<repositories>  
 <repository>  
 <id>public</id>  
 <name>Public Repositories</name>  
 <url>C:\Users\HTH-DEV\AppData\Roaming\Tencent\QQ\Temp\%W@GJ$ACOF(TYDYECOKVDYB.pnghttp://code.bonc.com.cn/nexus/content/groups/public/</url>  
 </repository>  
</repositories>

2.加入sql4es依赖。

<dependency>

<groupId>com.bonc.usdp</groupId>

<artifactId>bonc-sql4es</artifactId>

<version>5.5.2-v1.2-SNAPSHOT</version>

</dependency>

## 通过导入需要的jar包依赖

导入se/pkg/dependency下所有的jar包。

选中项目右键Build Path->Configure Build Path,在library下单击Add External JARs,将se/pkg/dependency下的jar包选中并导入。

## 使用**步骤**

### 加载JDBC驱动程序

在连接数据库之前，首先要加载想要连接的数据库的驱动到JVM（Java虚拟机），这通过java.lang.Class类的静态方法forName(String  className)实现。

try{

Class.forName("com.bonc.usdp.sql4es.jdbc.ESDriver");

}catch(ClassNotFoundException e){

System.out.println("找不到驱动程序类 ，加载驱动失败！");

e.printStackTrace();

}

### 创建数据库**的**连接

要连接数据库，需要向java.sql.DriverManager请求并获得Connection对象，该对象就代表一个数据库的连接。

使用DriverManager的getConnection(String url)方法传入指定的欲连接的数据库的路径。

驱动程序需要具有以下格式的URL： jdbc:sql4es://host:port/index?params.

- host: 其中一个主机的主机名或ip（必需）

- port: 可选用于传输客户端的端口号（默认为9300）

- index: 在驱动程序中设置活动的可选索引。大多数语句（如SELECT，DELETE和INSERT）都需要一个活动索引（参见下面的USE [index / alias]语句）。然而，可以创建没有活动索引的新索引，类型和别名。

- params: 用于影响驱动程序内部的可选参数集（指定其他主机，在单个请求中获取的最大文档数量等）。如果您的群集名称不是“elasticsearch”，则应该在URL中指定群集名称（请参见下面的示例）。有关所有驱动程序特定参数的说明，请参阅本文件的“配置”部分。

Connection con = DriverManager.getConnection("jdbc:sql4es://localhost:9300/test?cluster.name=elasticsearch&result.nested.lateral=false");

### 创建一个Statement

要执行SQL语句，必须获得java.sql.Statement实例，Statement实例分为以下3 种类型：

执行静态SQL语句。通常通过Statement实例实现。

执行动态SQL语句。通常通过PreparedStatement实例实现。

执行数据库存储过程。通常通过CallableStatement实例实现。

Statement st = con.createStatement();

PreparedStatement pstmt = con.prepareStatement(sql);

### 执行SQL语句

Statement接口提供了三种执行SQL语句的方法：executeQuery 、executeUpdate 和execute

1、ResultSet executeQuery(String sqlString)：执行查询数据库的SQL语句，返回一个结果集（ResultSet）对象。

2、int executeUpdate(String sqlString)：用于执行INSERT、UPDATE或 DELETE语句以及SQL DDL语句，如：CREATE TABLE和DROP TABLE等

3、execute(sqlString):用于执行返回多个结果集、多个更新计数或二者组合的语句。

ResultSet rs = stmt.executeQuery("SELECT \* FROM ...");

int rows = stmt.executeUpdate("INSERT INTO ...");

boolean flag = stmt.execute(String sql);

### 处理结果

两种情况：

1、执行更新返回的是本次操作影响到的记录数。

2、执行查询返回的结果是一个ResultSet对象。

ResultSet包含符合SQL语句中条件的所有行，并且它通过一套get方法提供了对这些行中数据的访问。

使用结果集（ResultSet）对象的访问方法获取数据：

while(rs.next()){

String name = rs.getString("name");

  String pass = rs.getString(1); // 此方法比较高效

}

### 关闭JDBC对象

操作完成以后要把所有使用的JDBC对象全都关闭，以释放JDBC资源，关闭顺序和声明顺序相反：

1、关闭记录集

2、关闭声明

3、关闭连接对象

 // 关闭记录集

try{

rs.close();

stmt.close();

conn.close();

}catch(SQLException e){

 e.printStackTrace();

}

## 完整示例

try {

Class.forName("com.bonc.usdp.sql4es.jdbc.ESDriver");

Connection con = DriverManager.getConnection("jdbc:sql4es://localhost:9300/test1?cluster.name=elasticsearch&result.nested.lateral=false");

Statement st = con.createStatement();

String sql = "SELECT \* from jobseekers where skill = ?";

PreparedStatement st = con.prepareStatement(sql);

st.setString(1, "天安门广场");

ResultSetMetaData rm = rs.getMetaData();

ResultSet rs = st.executeQuery();

int count = 0;

while(rs.next()){

System.out.println(rs.getObject("skill"));

count++;

}

rs.close();

st.close();

con.close();

} catch (ClassNotFoundException | SQLException e) {

e.printStackTrace();

}

}

# 支持语法

简单来说，sql4es驱动程序将SQL语句转换为它们的Elasticsearch语意，并将结果解析成ResultSet实现。支持以下sql语句：

- SELECT: 从elasticsearch获取文档（有无评分均可）或聚合

\* 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 offset,size

- CREATE TABLE (AS) 创建一个索引/类型，并可选地将查询的结果索引到其中

- CREATE VIEW (AS): ：创建别名，可选地带有过滤器

- DROP TABLE/VIEW 删除索引或别名

- INSERT INTO (VALUES | SELECT): 将文档插入索引/类型; 提供查询的值或结果。可以通过指定现有文档\_id来使用INSERT UPDATE文档

- UPDATE: 作为elasticsearch的Upsert执行

- DELETE FROM (WHERE):删除文档

- USE: 选择索引作为驱动程序的活动索引 (用于interpret queries)

- EXPLAIN SELECT: 返回为SELECT语句执行的Elasticsearch查询

- SHOW COLUMNS:返回查询的表结构

## SELECT

/\* basic syntax \*/

SELECT [field (AS alias)] FROM [types] WHERE [condition] GROUP BY [fields] HAVING [condition] ORDER BY [field (ASC|DESC)] LIMIT [[offset,]size]

- fields (AS alias): 定义要从elasticsearch检索并放在ResultSet中的字段。可以使用\*来表示应该检索所有字段（包括\_id和\_type）。字段可以通过他们的名字来表示，嵌套字段可以使用点状表示法使用它们的层次结构名称来表示，如：nesteddoc.otherdoc.field。使用一个星号将简单地获取文档中存在的所有字段，包括嵌套字段。可以指定对象的根以获取其所有字段。像SELECT nesteddoc FROM type这样的查询将获取nesteddoc中存在的所有字段。因此，如果nesteddoc有数百个字段，它可能返回数百列。

- types: 执行查询的类型。这只能是当前处于活动状态的索引或别名中的类型（也可参见'use'语句）。

- condition: 标准SQL条件使用=，>，> =，<，<=，<>，IN和LIKE运算符。Sql4es不支持NOT运算符，而可以使用'<>'。使用AND和OR组合条件。

- limit: 仅适用于非聚合查询。不支持使用偏移量！（不支持分页）

#### 示例

/\* 以下结果会将nested对象解析成侧视图 \*/

SELECT \* from mytype

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

/\* 如果nestedDoc包含两个字段，结果会解析成[myInt,nestedDoc.field1, nestedDoc.field2] \*/

SELECT myInt, nestedDoc FROM mytype WHERE myInt > 3 AND myString <> 'bye'

/\* 如果数组包括3个对象，ResultSet将包含3条记录，即使使用了LIMIT! \*/

SELECT array\_of\_nested\_objects FROM mytype LIMIT 1

/\* 从类型查询\*/

SELECT DISTINCT field, count(1) FROM type, query\_cache

/\* 与上述完全相同，但现在也从缓存查询 \*/

SELECT DISTINCT field, count(1) FROM type

### 文本匹配，搜索和评分

默认情况下，查询将作为过滤器执行，这意味着elasticsearch不会对结果进行评分，并以任意顺序返回。将“\_score”添加为所选列之一，以便更改此行为并请求评分。默认情况下，结果返回按\_score DESC排序（可以更改为ORDER BY \_score ASC）。在另一个字段上排序将禁用评分！另外，可以通过分别指定\_id和\_type来获取文档的id和类型。

是否删除记录？

Sql4es在文本字段的搜索和匹配之间没有区别。行为完全取决于对正在查询/搜索的文本字段执行的analysis（如果有）。在底层，使用几个简单的规则来确定应该使用什么类型的查询：

- 单个词放入TermQuery (example: myString = 'hello')

- 多个单词放入 MatchPhraseQuery (example: myString = 'hello there')

- 通配符的存在（％和\_）将触发使用 WildcardQuery (% 代替\* 、 \_ 代替 ?)。(Examples: mystring = '%something' is the same as mystring LIKE '%something')

- IN（...）的使用将被放在一个 TermsQuery中

此外，可以执行ES支持的所有功能的常规搜索。搜索是通过在虚构字段“\_search”上执行匹配来完成的（见下面的例子）。可以使用高亮功能为任何文本字段请求高亮，如：SELECT highlight(field), ...可通过全局配置设置片段大小和数字。

#### 示例

/\* term query \*/

SELECT \_score, myString FROM mytype WHERE myString = 'hello' OR myString = 'there'

/\* 同上 \*/

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')

/\* 检查NULL值（缺少字段） \*/

SELECT myInt FROM mytype WHERE myString NOT NULL

SELECT myInt FROM mytype WHERE myString IS NULL

/\* 短语查询 \*/

SELECT \_score, highlight(myString), myString FROM mytype WHERE myString = 'hello there'

/\* 通配符查询 \*/

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)'

### 通过\_id获取文档

可以通过在\_id字段上指定'='或IN来执行文档ID的搜索。可以将\_id与其他字段的匹配组合，但是应该始终使用IN完成匹配多个\_id。

### 通过\_search查询

在\_search字段上通过'='执行对文档的queryString查询，通常格式为：\_search='type:value AND age:[1 TO 10} OR content:value'。ES官方网站有更多关于queryString的语法说明。

### 通过\_child、\_parent执行父子关系查询

我们支持了新的父子关系查询，在有父子关系的两个type之间执行hasChild和hasParent查询。在\_child、\_parent字段上指定子表或者父表的表名：\_child.<childType>、\_parent.<parentType>。'='之后的值将会返回queryStringQuery的查询结果，queryString查询语法参照\_search。

//在父表my\_parent中查询子表my\_child满足age为0到50的条件的记录

SELECT \* FROM my\_parent WHERE \_child.my\_child = 'age:{0 TO 50}';

//在子表my\_child中查询父表my\_parent满足下列条件的记录

SELECT \* FROM my\_child WHERE \_parent.my\_parent = 'number: 10';

关于父子关系查询的注释:

- 父子关系必须在创建index时在mapping里定义，也就是说子表必须有\_parent字段。详见ES官方网站父子关系说明。

- 必须声明正确的父子表关系。\_child之后的值必须为所查询的表的子表，\_parent之后的值必须为所查询表的父表。

#### 示例

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' /\* 错误 \*/

SELECT \* FROM mytype WHERE \_id IN ('whatever\_id', 'another\_ID') /\* 正确 \*/

### 聚合

Sql4es在检测到DISTINCT，GROUP BY 或者聚合函数（MIN，MAX，SUM，AVG或者COUNT）而且没有普通字段时，会返回一个聚合请求。聚合请求不会返回任何搜索结果。

Sql4es支持一些基本的算术函数：\*，/，+，- 和％，也可以在诸如AVG(field)/100和SUM(field)/count(1)之类的计算中组合来自结果集的不同字段。请注意，在当前实现中，一旦从Elasticsearch获取数据，这些计算就在驱动程序内执行。可以使用括号 \*[offset]\*来引用函数中其他行中的值。例如，SUM(volume)/ SUM(volume)[-1]将将行X的列的和与行X-1中的值进行比较。如果无法计算值，例如上例中的行号0，将获得值Float.NaN。

#### 示例

/\* 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 DESC

/\* 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 myBool

/\* 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

### 函数

#### HIGHLIGHT

Sql4es支持在SELECT 条件里增加高亮查询函数。需要指定高亮的字段名和高亮的前缀与后缀（如果需要更改）。默认的高亮标记前缀为：<em>，后缀为：</em>。格式为：SELECT HIGHLIGHT(fieldname(,preTag,postTag)) FROM type WHERE fieldname=’value’;

##### 示例

SELECT HIGHLIGHT(tableName) FROM mytype WHERE tableName = 'file';

SELECT HIGHLIGHT(tableName,"<red>","</red>") FROM mytype WHERE tableName = 'file';

#### SUBSTRING

Sql4es支持了Elasticsearch本身并不支持的SUBSTRING函数，使用了Elasticsearch提供的Groovy脚本实现，需要在Elasticsearch服务的config/script目录下拷贝substring.groovy。

SUBSTRING函数提供了对结果分割需要字符的操作。例如：Tomlikeeating ->SUBSTRING(field,8,6)->eating。格式为：SELECT SUBSTRING(fieldname,startIndex,subLength) FROM mytype。

##### 示例

SELECT SUBSTRING(field,1,2) FROM mytype;

#### MATCH

Sql4es提供了Elasticsearch独有的MATCH查询。需要了解Elasticsearch对字段分词查询的逻辑。MATCH查询支持2种参数，是否为match\_phrase查询（phrase查询的slop为多少）和权重值boost。

match查询指的是将查询条件先进行分词再与倒排索引进行匹配，match\_phrase查询是指被分词的查询结果相隔slop距离之内的会被匹配到。例如：查询条件为“中国用户”，数据库中的数据为“中国北京用户量”，可以被match查询匹配到，但match\_phrase查询只有在设置slop>1的情况下才会被匹配到（中国北京用户量被分词为[中国,北京,用户,用户量]，中国与用户两词之间距离为1）。

##### 示例

SELECT \* FROM mytype WHERE MATCH(fieldName) = 'tom';

SELECT \* FROM mytype WHERE MATCH(fieldname,phrase) = 'likeeating'

/\*查询slop为2的match\_phrase \*/

SELECT \* FROM mytype WHERE MATCH(fieldname,phrase:2) = 'likeeating';

/\*slop为2且boost权重值为1.5的match\_phrase\*/

SELECT \* FROM mytype WHERE MATCH(fieldname,phrase:2,1.5) = 'likeeating';

/\*权重值为2的match查询\*/

SELECT \* FROM mytype WHERE MATCH(fieldname,2) = 'likeeating';

#### BOOST

Sql4es支持类似于MATCH函数提供的BOOST权重值的设置。格式为:BOOST(fieldname,boost)

##### 示例

/\*查询fieldName为20的记录，并且设置这个查询的权重为2\*/

SELECT \* FROM mytype WHERE BOOST(fieldname,2) = 20;

#### GEO

Sql4es支持地理位置检索。封装了Elasticsearch中对地理位置检索的Geo Bounding Box Query，Geo Distance Query，Geo Polygon Query。地理位置的值以数组形式提交给Sql4es，格式为[lon,lat]。

##### GEO(fieldname,box) = “[left,top],[bottom,right]”

Geo Box是用来查询存在于矩形区域内的点的查询。其中需要注意的是这个矩形必须是以经线和纬线为一边的矩形。左上的位置和右下的位置限定这个矩形的形状。

###### 示例

SELECT gisInfo FROM mytype WHERE GEO(gisInfo,box)= '[114,32],[116,36]'

##### GEO(fieldname,circle,radius) =”[lon,lat]”

GEO Distance 是用来查询与指定坐标点相聚radius半径以内的点的记录的查询。radius为需要指定的半径，单位可以为m(米)，km(千米)。

###### 示例

SELECT gisInfo FROM mytype WHERE GEO(gisInfo,circle,2km)= '[114,32]'

##### GEO(fieldname, polygon)=“[point1](,[point2]…[pointn])”

GEO Polygon 提供了查询在多边形内的点的记录。格式与GEO Box查询相同，不再做赘述。

## CREATE AND DROP

Sql4es支持创建索引，类型（创建表）和别名（创建视图）。这些语句需要了解ES工作原理，如映射，类型定义和别名。

### CREATE TABLE (index.)type ([field] "[field definition]" (, [field2])...) WITH (property="value" (, property2=...) )

这将在当前活动的索引或使用点符号指定的索引中为[type]创建映射。每当使用点号时，默认第一个点之前的部分指的是索引。 如果指定的索引已经存在，则只将该类型添加到此索引。

字段定义是将json定义放在映射中而不引用json元素！字符串类型可以定义如下：CREATE TABLE mytype (stringField "type:text, index:analyzed, analyzer:index\_ansj")。任何映射元素（如模板）都可以使用WITH子句设置（参见下面的示例）。所有这些json部分将被正确引用，并将它们混合成一个映射请求。

#### 示例

/\*为newindex中的mytype创建映射，并使用模板存储任何不经过分析的字符串 \*/

CREATE TABLE index.mytype (

myInt "type:integer",

myDate "type:date, format:yyyy-MM-dd"

myString "type:text, index:analyzed, analyzer:index\_ansj"

) WITH (

dynamic\_templates="[{

default\_mapping: {

match: \*,

match\_mapping\_type: keyword,

mapping: {type: keyword, index: not\_analyzed }

}

}]"

)

可以使用CREATE TABLE index.type（\_id“type：keyword”）创建一个空索引。\_id字段被省略，因为它是一个标准ES字段。

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

根据SELECT语句的结果创建新的索引/类型。新的字段名称取自SELECT，可以使用列别名来影响字段名称。例如CREATE TABLE myaverage AS SELECT avg（somefield）AS average将导致当前活动索引中的一个新类型myaverage，单个Double字段称为“average”。

请注意，驱动器通过两步过程执行。首先执行查询，其次创建索引，并将结果（批量）写入新类型。

#### 示例

/\*创建基于之前创建的映射与类型映射另一个index\*/

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 anyType GROUP BY myField ORDER BY count DESC

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

创建一个包含指定索引的新ES别名，或将索引添加到现有别名。可选的WHERE子句在指定的索引别名对象上添加一个过滤器。

有关别名的信息，请参阅[elasticsearch文档](https://www.elastic.co/guide/en/elasticsearch/reference/current/indices-aliases.html)

### DROP TABEL [index] / DROP VIEW [alias]

删除指定的索引或别名

#### 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

## INSERT AND DELETE

介绍通过sql插入和删除数据

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

在索引中添加一个或多个文档到指定的类型。必须定义字段，值的数量必须与定义的字段数匹配。可以在单个INSERT语句中添加多个文档。可以在insert语句中指定\_id字段。在这种情况下，它将强制ES来插入指定的文档ID。 如果\_id已经存在，则插入作为一个UPDATE。

插入嵌套对象是不可能的，因为它们不能在SQL语言中指定。

### INSERT INTO (index.)type SELECT …

将SELECT语句中的所有结果添加到索引中指定的类型。要从结果中取出要插入的字段名称（即可以使用列别名）。请注意，类似于“CREATE TABLE .. AS SELECT”，结果将被拉入驱动程序，然后索引（使用Bulk）。

#### 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

/\*插入格式为nested对象的数据\*/

INSERT INTO document (user,\"fieldDescribe.\_nested\") VALUES('root',' [{\"fieldCode\": \"DAY\_ID\",\"fieldName\": \"日账期\",\"fieldTypeName\": \"字符串\",\"comments\": \"日账期:YYYYMMDD\"},{\"fieldCode\": \"KPI\_CODE\",\"fieldName\": \"指标编码\",\"fieldTypeName\": \"字符串\",\"comments\": \"指标编码\"}]')";

INSERT INTO document ("gisinfo.\_geo") VALUES ('[108.3,31]')

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

从符合条件的指定类型中删除所有文档。如果未指定WHERE子句，则将删除所有文档。由于Elasticsearch只能根据它们的\_id删除文档，这意味着这个语句是分两步执行的。首先从满足条件的文档中收集所有\_id，其次使用批量API删除这些文档。

/\* delete documents that meet a certain condition\*/

DELETE FROM mytype WHERE myInt == 3

/\*delete all documents from mytype\*/

DELETE FROM mytype

## UPDATE

可以使用标准SQL语法更新索引/类型中的文档。请注意，嵌套对象名称必须用双引号括起来：

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

更新将分两步执行。首先获取与条件匹配的所有文档的\_id，之后使用Upsert API批量更新这些文档的指定字段。

## SHOW COLUMNS

使用show columns查询表的结构。格式为：show columns from <index>.<type>。

使用show tables 查询索引中的表的属性。格式为：show tables from <index>.

#### 示例：

/\*查询test下my\_child表的字段结构\*/

SHOW COLUMNS FROM test.my\_child

/\*查询test中存在的表的目录\*/

SHOW TABLES FROM test

# 配置

可以通过提供的URL来设置参数。所有参数都暴露于Elasticsearch，这意味着可以设置客户端参数，参见 [elasticsearch docs](https://www.elastic.co/guide/en/elasticsearch/client/javascript-api/current/configuration.html). 可以设置以下驱动程序的具体参数：

- es.hosts: 带逗号分隔列表，带有可选端口的附加主机，格式为host1（：port1），host2（：port2）...）当未指定端口时，将使用默认端口9300。

- cluster.name: 集群名，默认为elasticsearch

- searchguard.authcz.authinfo: 用户验证信息，在集群使用权限管理时使用

- fetch.size (int default 10000): 在单个请求中获取的最大结果数（10000是elasticsearch的最大值）。可以降低文件以避免记录问题，当文档获取非常大时。

- results.split (default: false): 设置它将将整个结果分成多个ResultSet对象，每个对象具有最大的\*fetch.size\*记录数。可以使用Statement.getMoreResults（）获取下一个ResultSet。默认值为\*false\* ，驱动程序将把所有结果放在一个ResultSet中。当客户端没有足够的内存将单个ResultSet中的所有结果保存下来时，应使用此设置。

- scroll.timeout.sec (int, default 10): 滚动ID保持有效的时间，可以调用“getMoreResults（）”。应根据实际情况增减。

- query.timeout.ms (int, default 10000): 查询上设置的超时值。可以根据用例进行更改。

- default.row.length (int, default 250): 为结果创建的列的初始数。只有在搜索结果被解析时触发的结果不适合（通常由数组索引超出范围异常）时才增加此属性。

- query.cache.table (string, default 'query\_cache'): 用于指示使用elasticsearch查询缓存的虚构表名。可以更改，使其更短或更方便。

- result.nested.lateral (boolean, default true): 指定嵌套结果必须分解（默认）与否。从您自己的代码处理驱动程序时可以设置为false。在这种情况下，包含嵌套对象（包装在ResultSet中）的列将具有java.sql.Types = Types.JAVA\_OBJECT，并可用作（ResultSet）rs.getObject（colNr）。

- fragment.size (int, default 100): 以字符为单位指定首选片段长度。

- fragment.number (int, default 1): 指定请求突出显示时返回的最大片段数。

- precision.threshold (int, default 3000): 指定用于[基数聚合的精度](https://www.elastic.co/guide/en/elasticsearch/reference/current/search-aggregations-metrics-cardinality-aggregation.html)

# 注意事项

Sql4es需要从一个**活动的索引/别名**启动，这意味着它只解析了这个索引的类型。例如myIndex当前处于活动状态，SELECT \* FROM sometype将只返回myindex的类型的所有结果。在索引中不存在的类型上执行SELECT将返回一个空的结果。可以通过执行USE [otherIndex]来更改活动索引。

## 不支持的语法

子查询在字段中：

SELECT something FROM type, (SELECT max(x) FROM type2) as type2

连接查询：

SELECT something FROM type1, type2 where type1.col1 = type2.col2