**SQL查询语句自动生成的用户手册**

1. 软件安装

## 运行环境

* 运行环境：WINDOWS 或LINUX, JAVA8及以下
* 数据库：MYSQL

## 安装步骤

### 创建数据库

执行Universe.sql创建Universe数据库。（Universe.sql由数据库物理视图生成）

数据库的逻辑视图：



数据库的物理视图

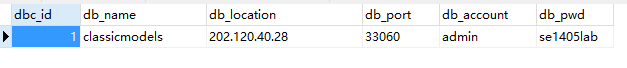


### 设计Universe

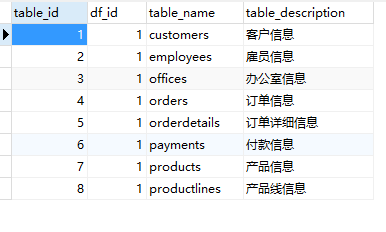
根据业务数据库及业务需求来设计Universe，将设计结果的数据存入数据库中。

以下为部分表的示意图。

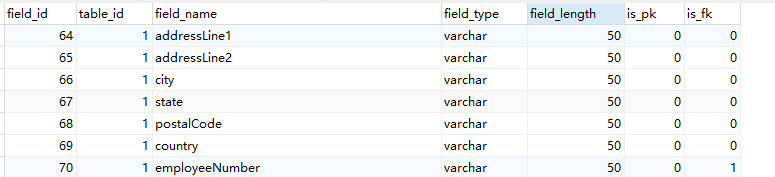
DB\_CONNECTION：



DATA\_TABLE：



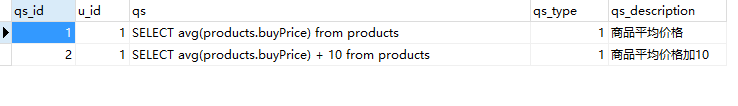
TABLE\_FIELD：



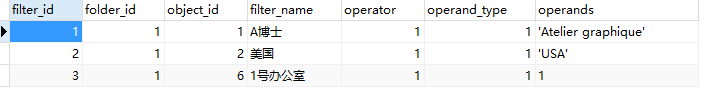
OBJECT：



QUERY\_STATEMENT：



FILTER：



FOLDER:



UNIVERSE



DATA\_FOUNDATION



### 运行jar包

可在命令行中运行*java –jar JAR\_NAME --DATA\_SOURCE\_URL --DATA\_SOURCE\_USERNAME --DATA\_SOURCE\_PWD语句启动程序。*

其中JAR\_NAME为jar包的名称，例如sqlgeneration-1.0-SNAPSHOT.jar。

DATA\_SOURCE\_URL为Universe数据库的地址及配置参数，例如spring.datasource.url=jdbc:mysql://202.120.40.28:33060/sql\_generation\_t1?characterEncoding=utf8&useSSL=true。

DATA\_SOURCE\_USERNAME为Universe数据库的登陆账号名称。例如spring.datasource.username=admin

DATA\_SOURCE\_PWD为Universe数据库的登陆账号密码。例如：

spring.datasource.password= xxxxxx

正确示例：java -jar E:\workspace\SQLGeneration\target\sqlgeneration-1.0-SNAPSHOT.jar --spring.datasource.url=jdbc:mysql://202.120.40.28:33060/sql\_generation\_t1?characterEncoding=utf8&useSSL=true --spring.datasource.username=admin --spring.datasource.password=xxxxxx

1. API使用

## API接口说明

本工具以RESTful Web Service来提供服务。

### setObjects

HTTP请求：POST

HTTP header：Content-Type: application/json

接口说明：设置需要查询的对象集合

请求参数：

|  |  |  |  |
| --- | --- | --- | --- |
| **参数** | **必选** | **类型和范围** | **说明** |
| objects | True | List<int> | 需要查看的对象ID集合 |

返回格式：json

返回字段：

|  |  |  |  |
| --- | --- | --- | --- |
| **返回字段** | **必选** | **类型和范围** | **说明** |
| status | True | int | 状态码 |
| info | False | String | 详细的状态信息 |

### setFilter

HTTP请求：POST

HTTP header：Content-Type: application/json

接口说明：设置单个过滤条件，在结果对象集中，选中某个对象，选择操作符（例如大于），然后选择操作数（例如类型的值或其它对象）。

当需要设置多个过滤条件时，则继续选中该对象，选择操作符和操作数，构成新的过滤条件，多个条件间是“AND”关系。

请求参数：

|  |  |  |  |
| --- | --- | --- | --- |
| **参数** | **必选** | **类型和范围** | **说明** |
| object | True | int | 过滤条件涉及的对象ID |
| operator | True | int | 过滤条件操作符 |
| operandType | True | int | 过滤条件操作数类型 |
| operand | True | String | 过滤条件操作数（数值，字符串，子查询等等） |

返回格式：json

返回字段：

|  |  |  |  |
| --- | --- | --- | --- |
| **返回字段** | **必选** | **类型和范围** | **说明** |
| status | True | int | 状态码 |
| info | False | String | 详细的状态信息 |

### setPredefinedFilter

HTTP请求：POST

HTTP header：Content-Type: application/json

接口说明：设置创建好的单个预定义过滤条件

当需要设置多个预定义过滤条件时，则多次选中预定义的过滤条件，这些条件间是“AND”关系。

请求参数：

|  |  |  |  |
| --- | --- | --- | --- |
| **参数** | **必选** | **类型和范围** | **说明** |
| filter | True | Int | 需要调用的预定义过滤条件ID |

返回格式：json

返回字段：

|  |  |  |  |
| --- | --- | --- | --- |
| **返回字段** | **必选** | **类型和范围** | **说明** |
| status | True | int | 状态码 |
| info | False | String | 详细的状态信息 |

### setOrder

HTTP请求：POST

HTTP header：Content-Type: application/json

接口说明：设置单个排序标准，选择需排序的对象和排序标准。

当需要设置多个排序标准时，则继续选择排序的对象和排序标准，多个排序标准之间依据设置的先后进行排序。

请求参数：

|  |  |  |  |
| --- | --- | --- | --- |
| **参数** | **必选** | **类型和范围** | **说明** |
| object | True | int | 需要排序的对象ID |
| order | True | int | 排序标准 |

返回格式：json

返回字段：

|  |  |  |  |
| --- | --- | --- | --- |
| **返回字段** | **必选** | **类型和范围** | **说明** |
| status | True | int | 状态码 |
| info | False | String | 详细的状态信息 |

### setDistinctRecords

HTTP请求：POST

HTTP header：Content-Type: application/json

接口说明：设置返回结果中是否要去掉重复记录。

请求参数：

|  |  |  |  |
| --- | --- | --- | --- |
| **参数** | **必选** | **类型和范围** | **说明** |
| distinct | True | Bool | 是否去掉返回结果中的重复记录（默认为False） |

返回格式：json

返回字段：

|  |  |  |  |
| --- | --- | --- | --- |
| **返回字段** | **必选** | **类型和范围** | **说明** |
| status | True | int | 状态码 |
| info | False | String | 详细的状态信息 |

### setReturnNumber

HTTP请求：POST

HTTP header：Content-Type: application/json

接口说明：设置返回的记录数。

请求参数：

|  |  |  |  |
| --- | --- | --- | --- |
| **参数** | **必选** | **类型和范围** | **说明** |
| returnNumber | True | int | 设置返回的记录数量（若returnNumber小于0，则返回所有记录） |

返回格式：json

返回字段：

|  |  |  |  |
| --- | --- | --- | --- |
| **返回字段** | **必选** | **类型和范围** | **说明** |
| status | True | int | 状态码 |
| info | False | String | 详细的状态信息 |

### generateSQL

HTTP请求：GET

HTTP header：无

接口说明：根据之前设置的各项条件，生成SQL查询语句。

请求参数：无

返回格式：json

返回字段：

|  |  |  |  |
| --- | --- | --- | --- |
| **返回字段** | **必选** | **类型和范围** | **说明** |
| status | True | int | 状态码 |
| SQL | False | String | 生成的SQL语句 |
| info | False | String | 详细的状态信息 |

排序规则：

|  |  |
| --- | --- |
| **顺序名称** | **对应值** |
| 上升顺序 | 1 |
| 下降顺序 | 2 |

过滤条件操作符：

|  |  |
| --- | --- |
| **操作符** | **对应值** |
| 等于 | 1 |
| 不同 | 2 |
| 大于 | 3 |
| 大于或等于 | 4 |
| 小于 | 5 |
| 小于或等于 | 6 |
| 之间 | 7 |
| 不包括 | 8 |
| 在列表里 | 9 |
| 不在表里 | 10 |
| 空 | 11 |
| 非空 | 12 |
| 匹配模式 | 13 |
| 不匹配模式 | 14 |
| 满足两个 | 15 |

操作数类型：

|  |  |
| --- | --- |
| **操作数** | **对应值** |
| 类型的值 | 1 |
| 从对象值列表里选择的值 | 2 |
| 由其他查询返回的值 | 3 |
| 由其他查询返回的所有值。 | 4 |
| 计算的结果（总和，最小值，最大值，平均值或数量） | 5 |

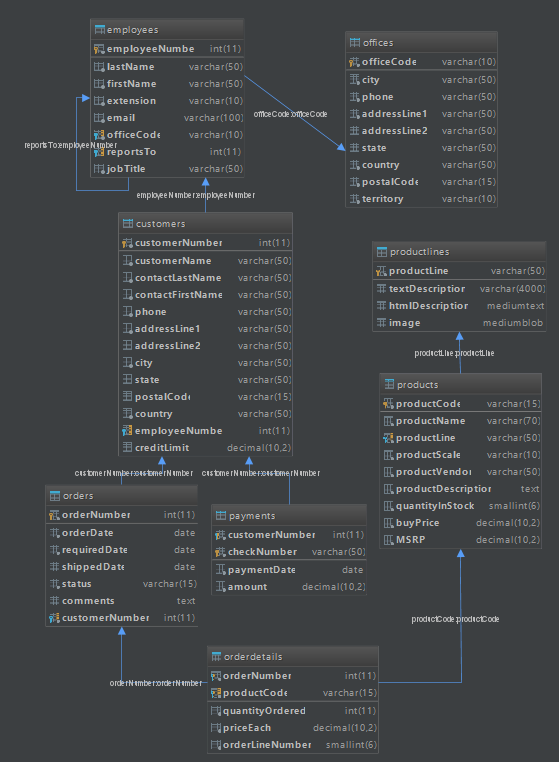
## API调用示例

假设服务器部署在地址1.1.1.1，端口为8080。

### 业务数据库与Universe

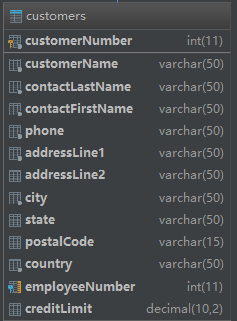
1. 业务数据库

本示例的业务数据库如下



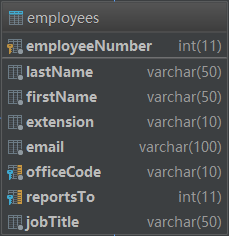
以下示例涉及到的表包括：

customers：



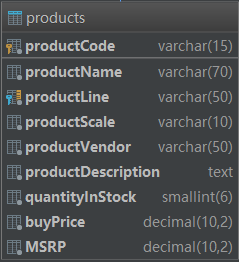
其中customerName字段为主键，employeeNumber为引用自表employees的外键

employees:



其中employeeNumber字段为主键，officeCode为引用自表offices的外键，reportsTo为引用自表employees的外键。

products：



其中productCode为主键，productLine为引用自表productlines的外键。

1. Universe

Universe的设计见[1.2.2](#_设计Universe)节。

### 查询所有美国国籍的顾客姓名和电话号码

步骤：

1. 设置查看的对象：顾客，对应universe内id为1的属性对象，在数据库内对应表customers的customerName字段；电话号码，对应universe内id为8的属性对象，在数据库内对应表customers的phone字段。

HTTP方法：POST

API路径: 1.1.1.1:8080/setObjects [全文检查]

HTTP Header: Content-Type: application/json

HTTP Body: Json字符串格式

{

“objects”:[1,8]

}

返回结果：

{

“status”:0,

“info”:”正常”

}

1. 设置过滤条件：顾客国籍 = 美国，其中顾客国籍，对应universe内id为2的属性对象，在数据库对应表customers的country字段

HTTP方法：POST

API路径: 1.1.1.1:8080/setFilter

HTTP Header: Content-Type: application/json

HTTP Body: Json字符串格式

{

"object":2, //过滤条件对象

"operator":1, //过滤条件操作符，1对应“等于”

"operandType":1, //过滤条件操作数类型，1对应“数值”

"operand":"USA" //过滤条件操作数

}

返回结果

{

"status":0,

"info":"正常"

}

1. 生成SQL语句

HTTP方法：GET

API路径: 1.1.1.1:8080/generateSQL

HTTP Header: 无

返回结果：

{

"status":0,

"info":"正常",

"SQL":"SELECT customers.customerName, customers.phone FROM customer WHERE customers.country = 'USA'",

}

### 查询办公室1所有雇员的相关顾客的姓名，不返回重复结果，升序排列，返回前100条结果

步骤：

1. 设置对象：顾客，对应universe内id为1的属性对象，在数据库内对应表customers的customerName字段

HTTP方法：POST

API路径: 1.1.1.1:8080/setObjects

HTTP Header: Content-Type: application/json

HTTP Body: Json字符串格式

{

"objects":[1]

}

返回结果

{

"status":0,

"info":"正常"

}

1. 设置预过滤条件：预过滤条件1号办公室，对应universe内id为3的预过滤条件，即表employees的officeCode = 1

HTTP方法：POST

API路径: 1.1.1.1:8080/setPredefinedFilter

HTTP Header: Content-Type: application/json

HTTP Body: Json字符串格式

{

"filter":3

}

返回结果

{

"status":0,

"info":"正常"

}

1. 设置排序规则：按顾客姓名升序排列

HTTP方法：POST

API路径: 1.1.1.1:8080/setOrder

HTTP Header: Content-Type: application/json

HTTP Body: Json字符串格式

{

"object":1,

"order":1

}

返回结果

{

"status":0,

"info":"正常"

}

1. 设置不返回重复结果

HTTP方法：POST

API路径: 1.1.1.1:8080/setDistinctRecords

HTTP Header: Content-Type: application/json

HTTP Body: Json字符串格式

{

"distinct":true

}

返回结果

{

"status":0,

"info":"正常"

}

1. 设置返回记录数量：100条

HTTP方法：POST

API路径: 1.1.1.1:8080/setReturnNumber

HTTP Header: Content-Type: application/json

HTTP Body: Json字符串格式

{

"returnNumber":100

}

返回结果

{

"status":0,

"info":"正常"

}

1. 生成SQL语句

HTTP方法：GET

API路径: 1.1.1.1:8080/generateSQL

HTTP Header: 无

返回结果：

{

"status":0,

"info":"正常",

"SQL":"SELECT DISTINCT customers.customerName FROM customers JOIN employees WHERE employees.officeCode = 1 ORDER BY customers.customerName LIMIT 0, 100"

}

### 查询所有价格在平均价格和平均价格高十元之间的产品价格

步骤：

1. 设置对象：产品价格，对应universe内id为7的属性对象，在数据库内对应表products的buyPrice字段

HTTP方法：POST

API路径: 1.1.1.1:8080/setObjects

HTTP Header: Content-Type: application/json

HTTP Body: Json字符串格式

{

"objects":[7]

}

返回结果

{

"status":0,

"info":"正常"

}

1. 设置过滤条件：过滤条件为buyPrice between 商品均价 and 商品均价+10，这里的操作数类型为3，即子查询，对应universe中id为1和2的两个查询

HTTP方法：POST

API路径: 1.1.1.1:8080/setFilter

HTTP Header: Content-Type: application/json

HTTP Body: Json字符串格式

{

"object":7,

"operator":7,

"operandType":3,

"operand":[1,2]

}

返回结果

{

"status":0,

"info":"正常"

}

1. 生成SQL语句

HTTP方法：GET

API路径: 1.1.1.1:8080/generateSQL

HTTP Header: 无

返回结果：

{

"status":0,

"info":"正常",

"SQL":"SELECT products.buyPrice FROM products WHERE products.buyPrice BETWEEN (SELECT avg(products.buyPrice) from products) AND (SELECT avg(products.buyPrice) + 10 from products) "

}