**Schemasync工具使用教程**

**环境以及工具准备**

**Ubuntu16.04**

新建一个Ubuntu16的服务器，或者容器即可

**python2.7**

1. sudo apt-get update
2. sudo apt-get install python2.7
3. sudo apt update
4. sudo apt install python-pip
5. pip install --upgrade pip

**schemasync工具**

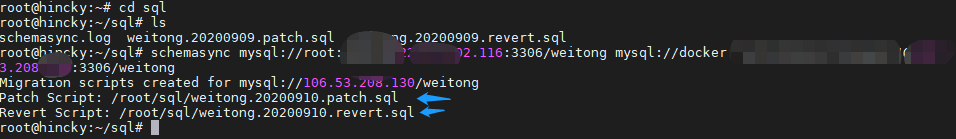
<https://github.com/mmatuson/SchemaSync>

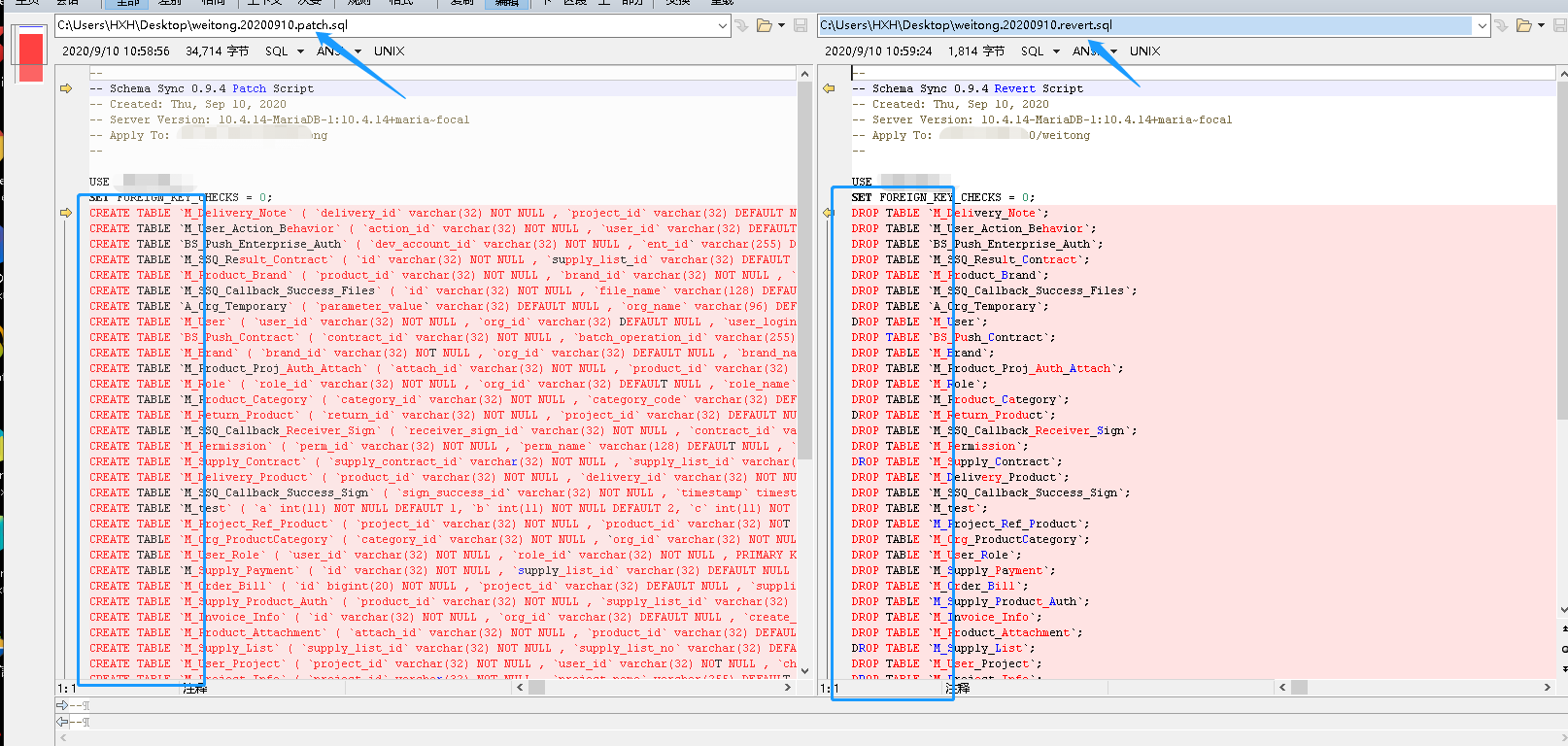
克隆或者直接下载到本地。并进行解压

**具体使用**

Ubuntu环境下

1. cd schemasync
2. python setup.py install
3. apt-get install python-mysqldb
5. mkdir sql
6. -- 打开这个sql目录，是因为在哪执行schemasync，对应文件就在哪生成
7. cd sql
8. -- 同步数据库A的表结构，到数据库B
9. ~~schemasync mysql://root:admin@122.51.202.116:3306/weitong~~ ~~mysql://docker:"uF%zX6baX2q^n&Ua@vj"@106.53.208.130:3306/weitong~~





这时即可看到生成的patch.sql和revert.sql

patch.sql 是参照于A，数据库B所要进行的动作。对比A少了就加东西，反之则删东西

revert.sql 是用于数据库B执行了patch.sql脚本后，用了回滚到未执行patch.sql前的状态

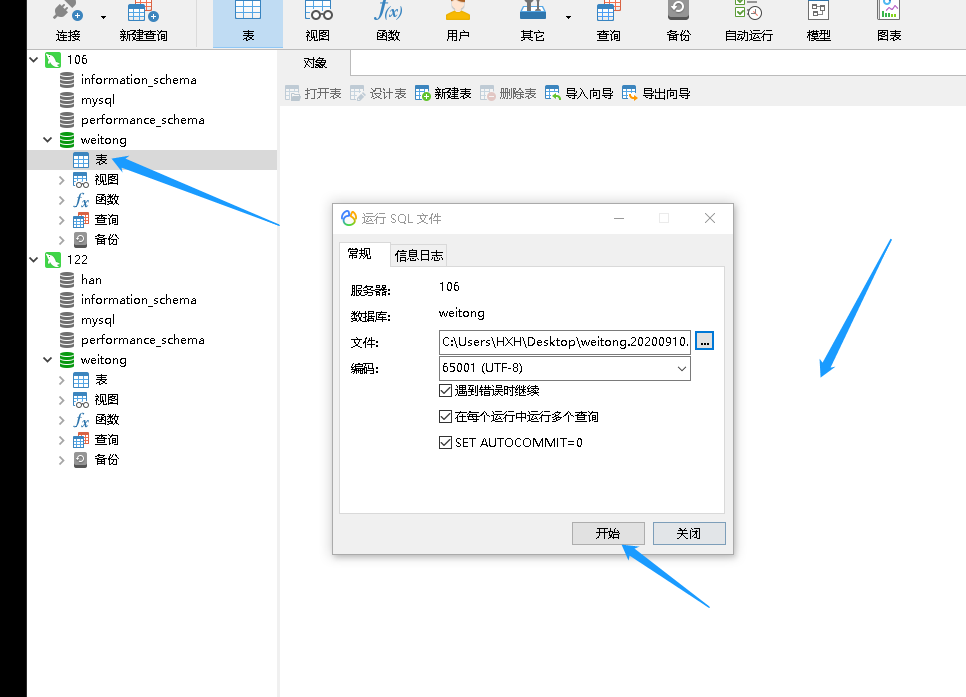
schemasync.log 则是用来记录操作和生成文件的日志

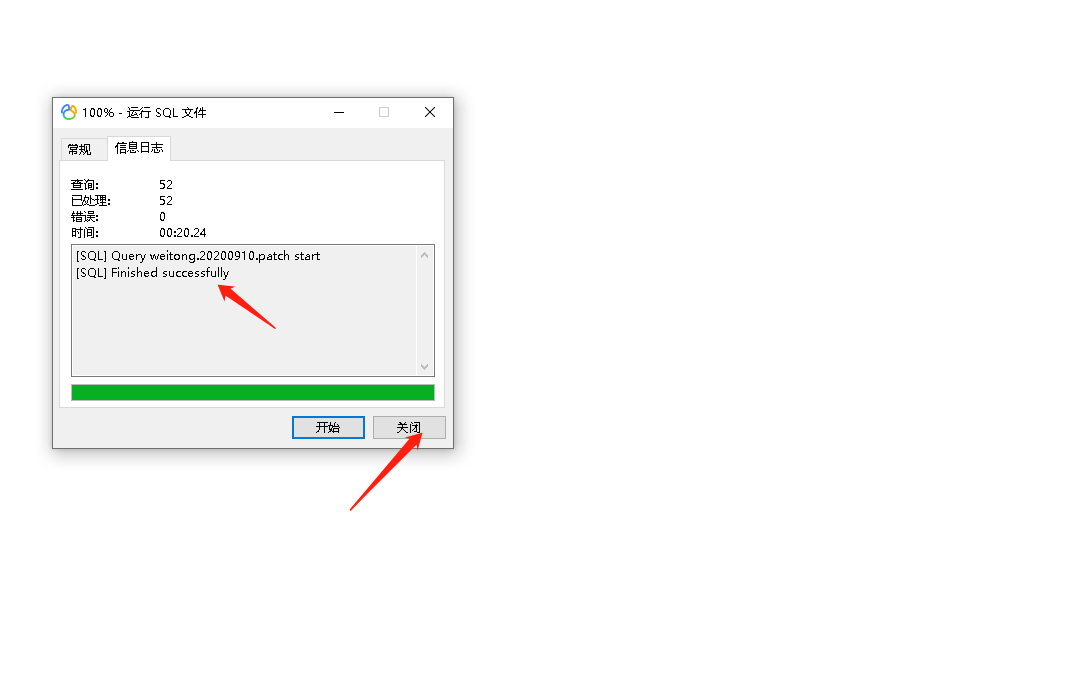
**同步测试**

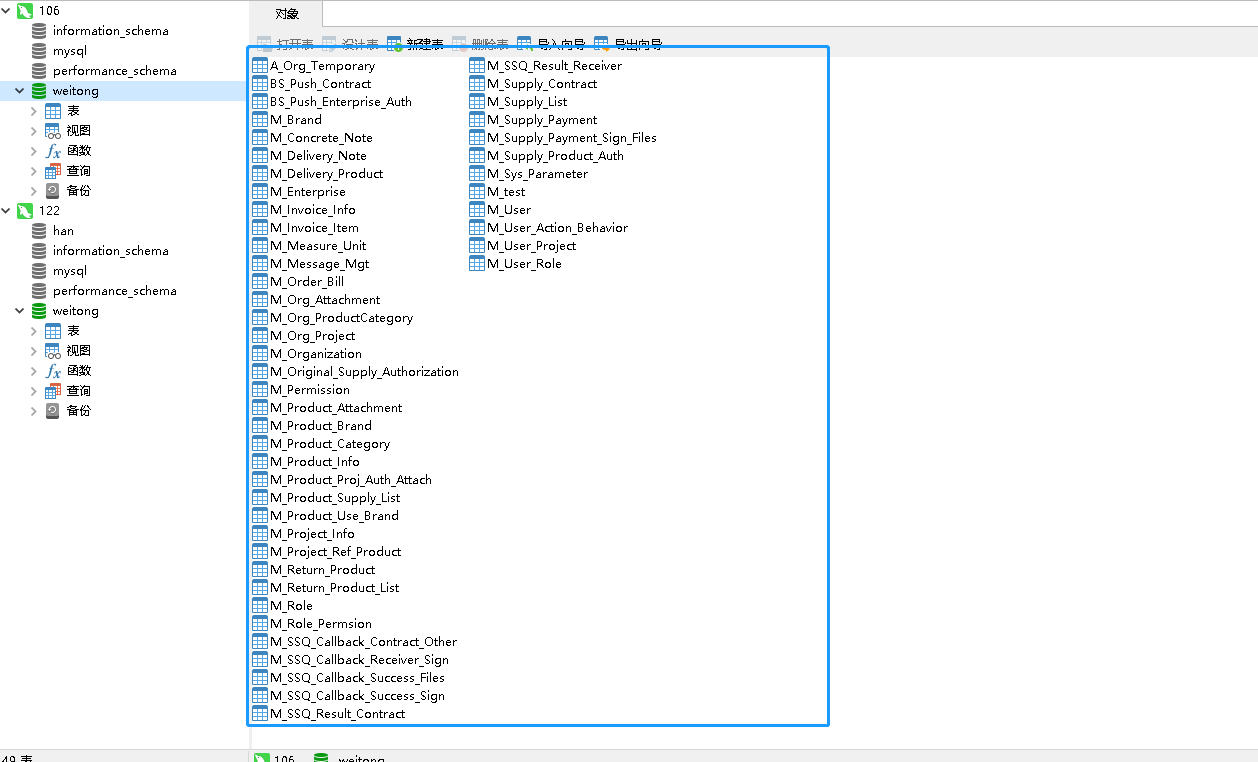
**表结构同步**

打开任意sql可视工具，笔者这里用的是Navicat

打开要执行sql的数据库位置，执行sql

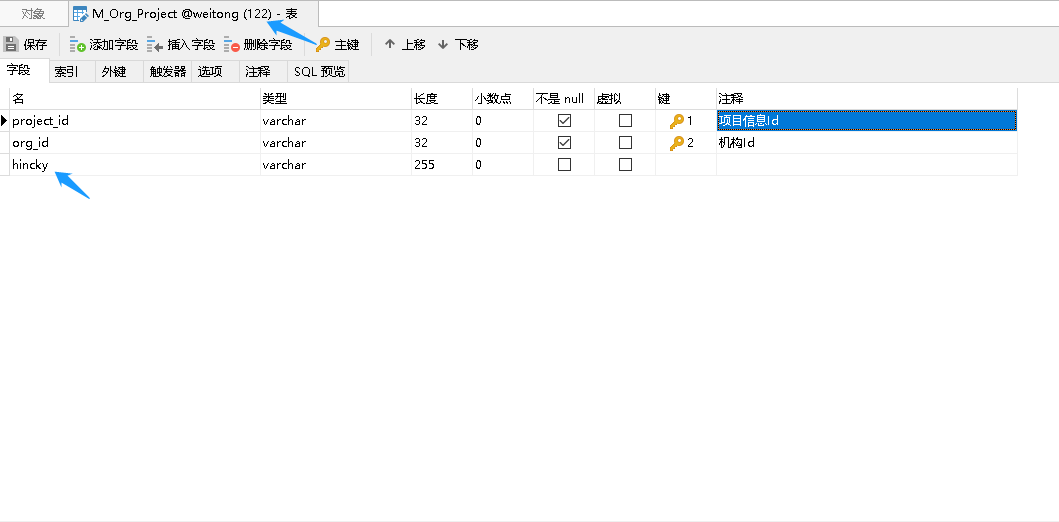




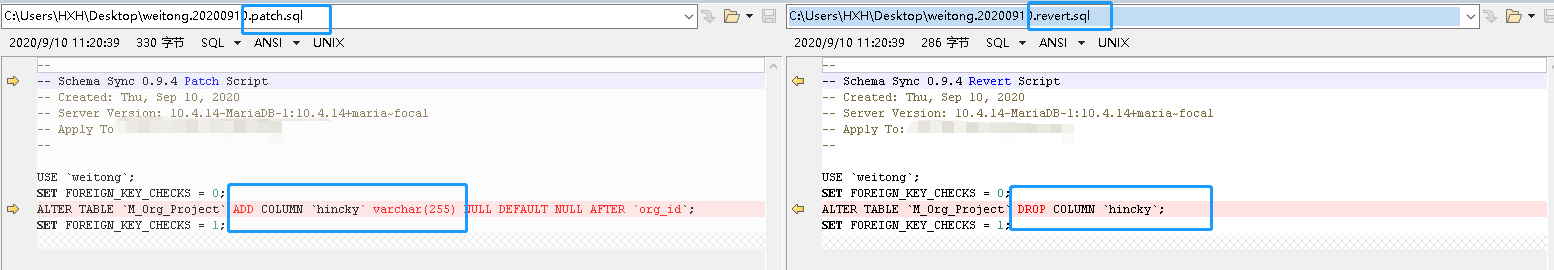


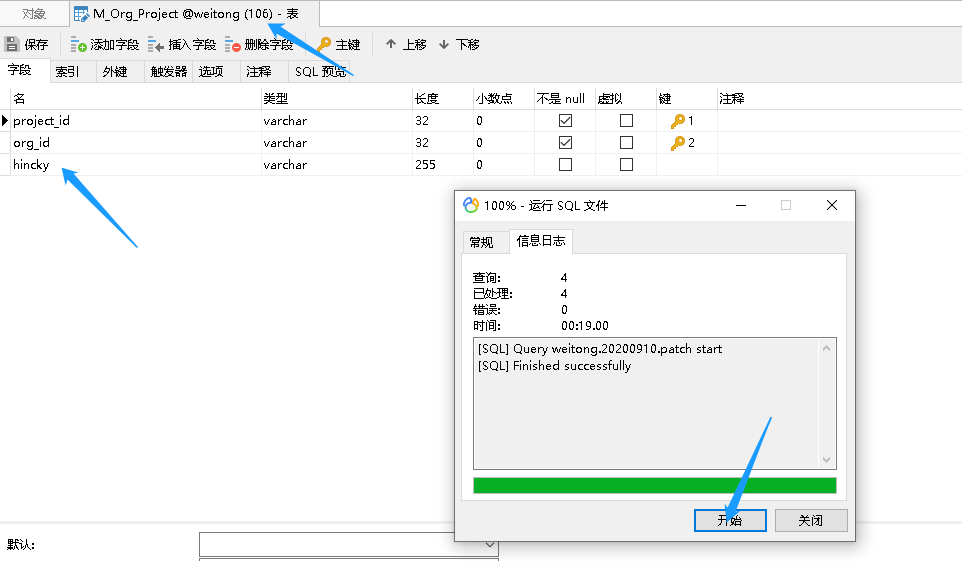
**字段添加同步**

源数据库添加字段hincky



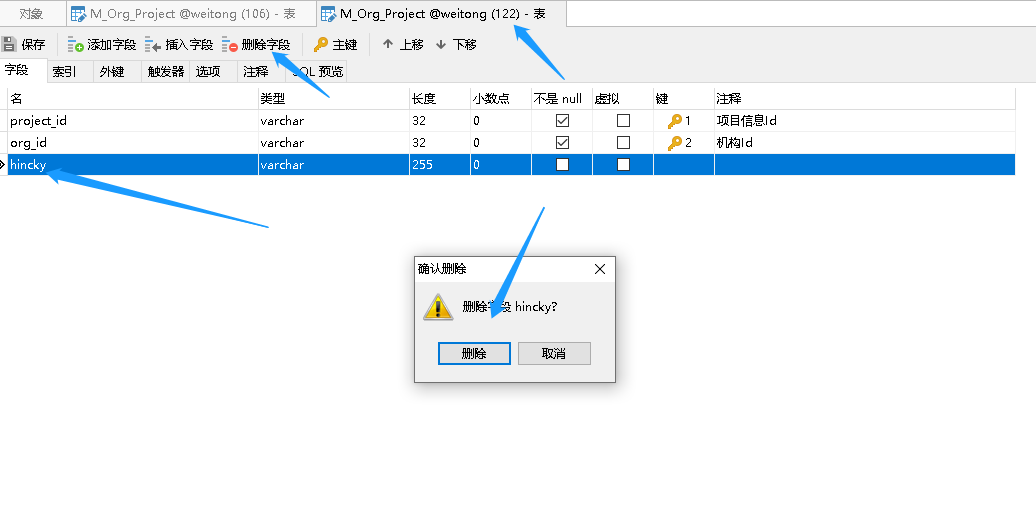
服务器执行schemasync命令并在Navicat目标数据库执行sql脚本



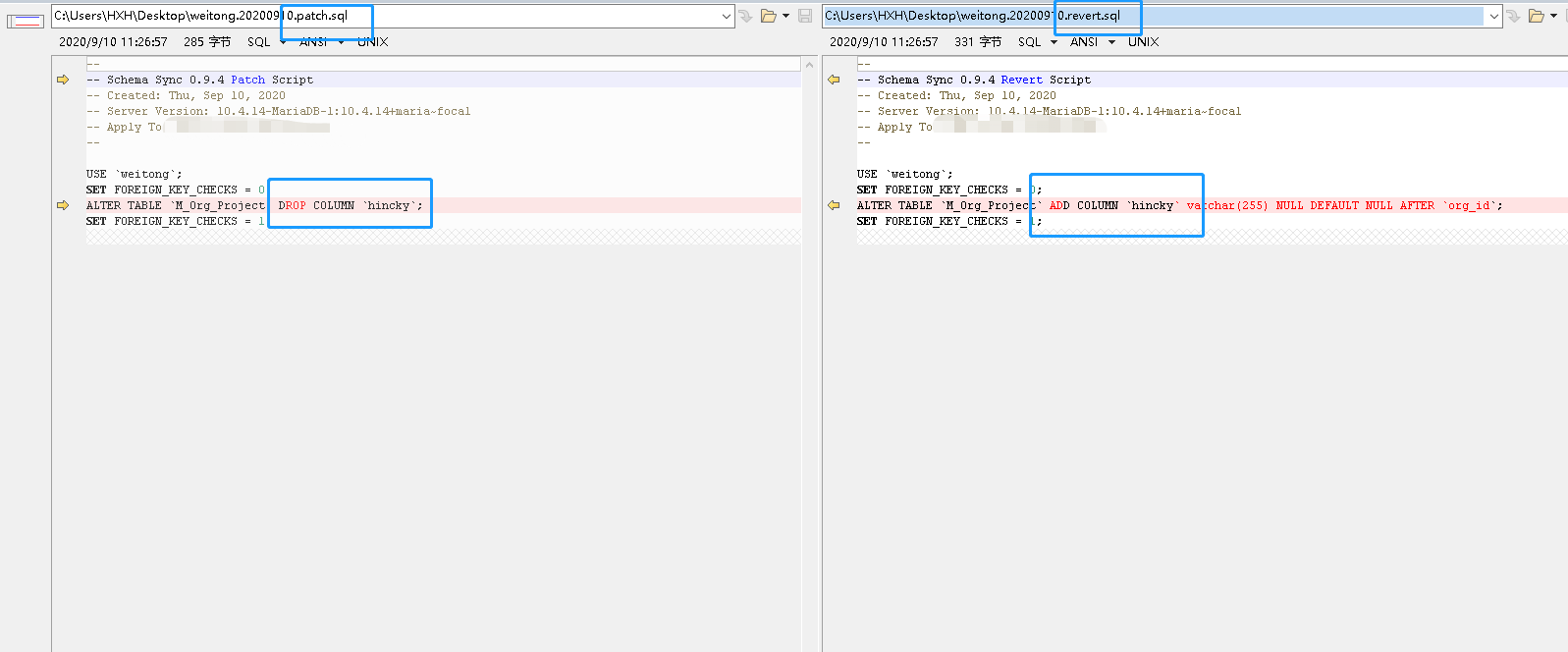


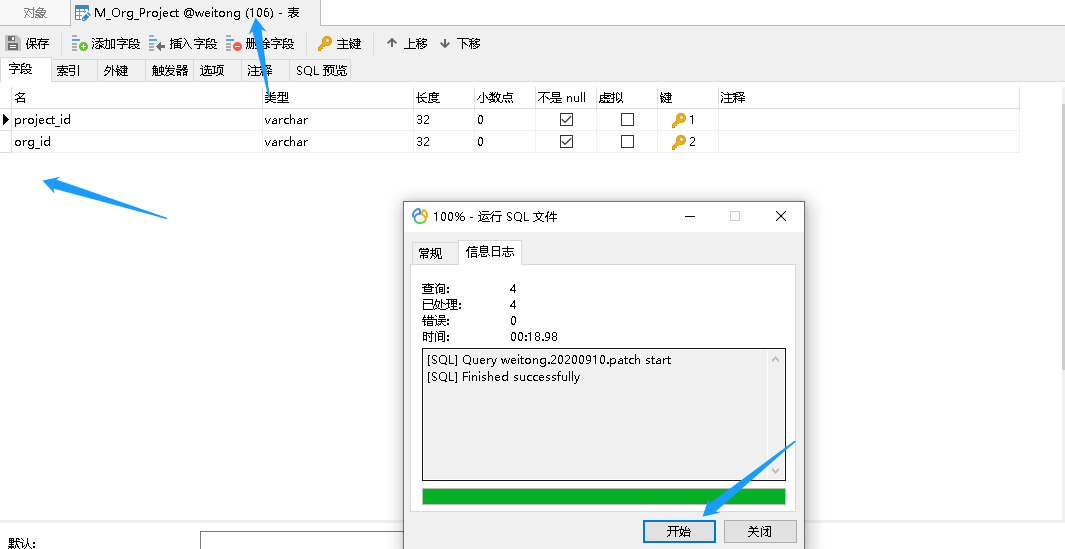
**删除字段同步**

源数据库删除字段hincky



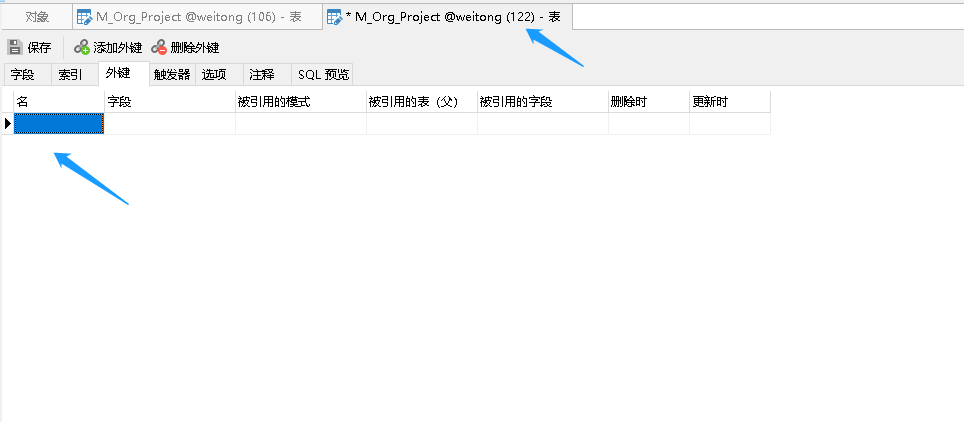
再执行schemasync命令并在Navicat目标数据库执行sql



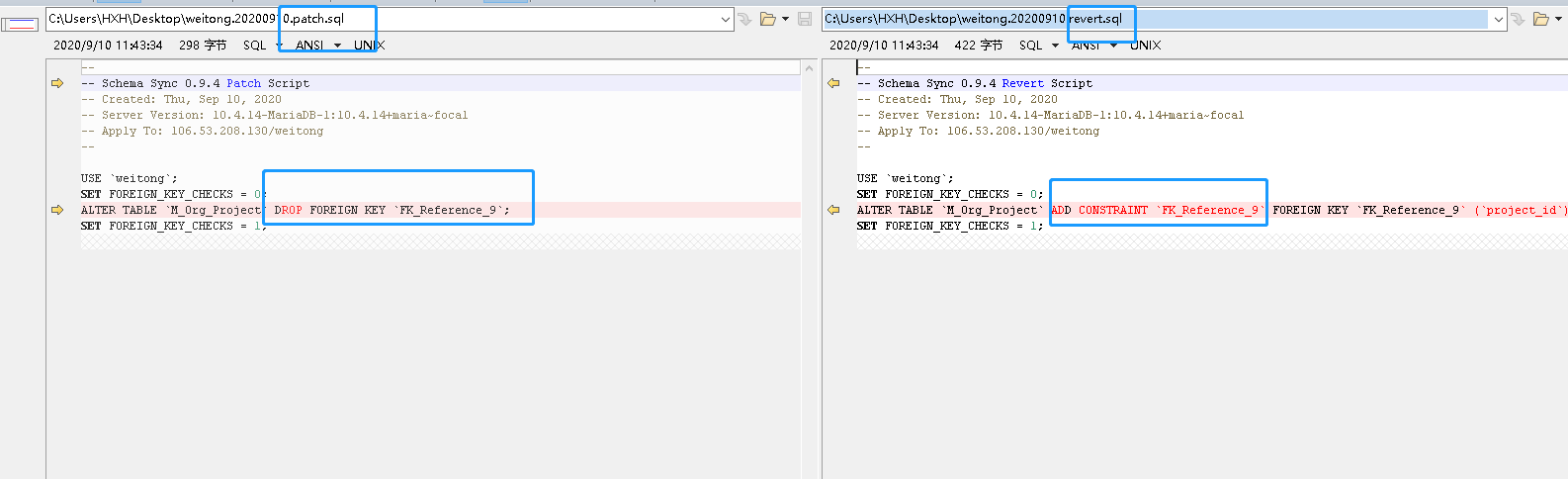


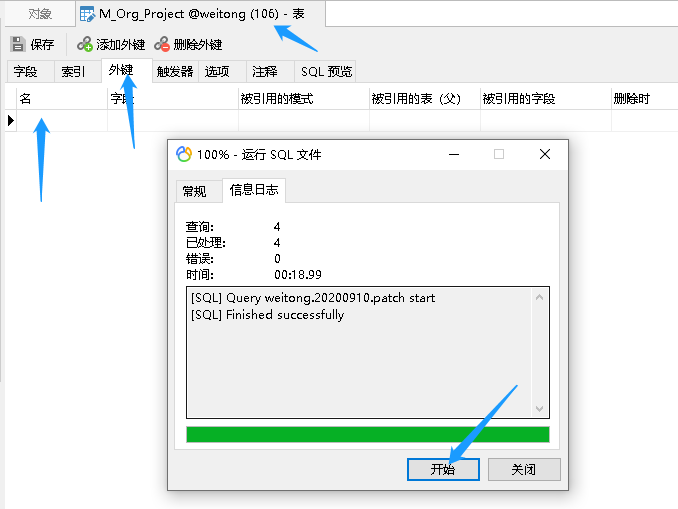
**删除外键同步**

源数据库删除外键



再执行schemasync命令并在Navicat目标数据库执行sql





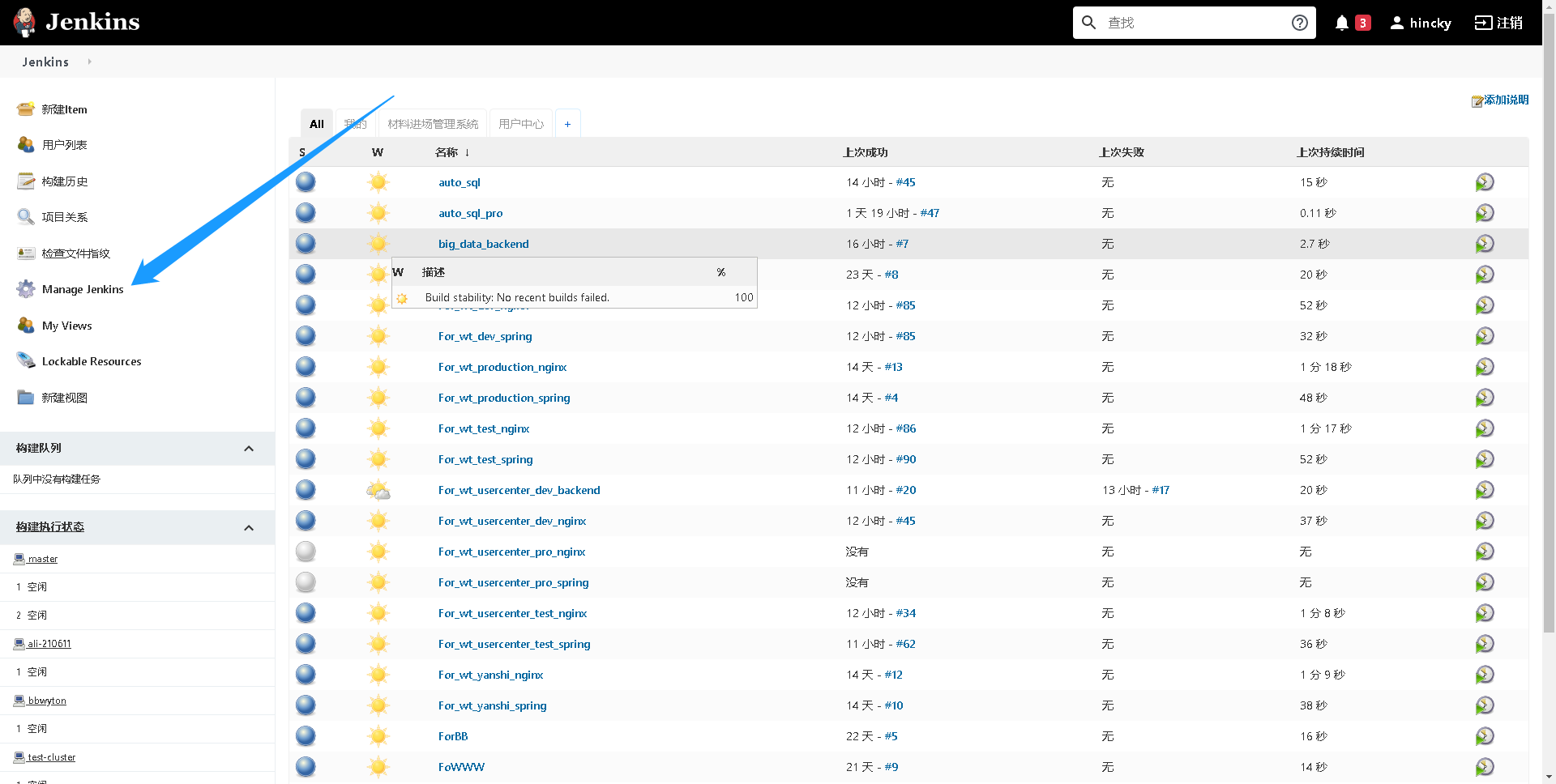
**sql自动化**

1. **编写拆除目标数据库外键和约束的py文件**

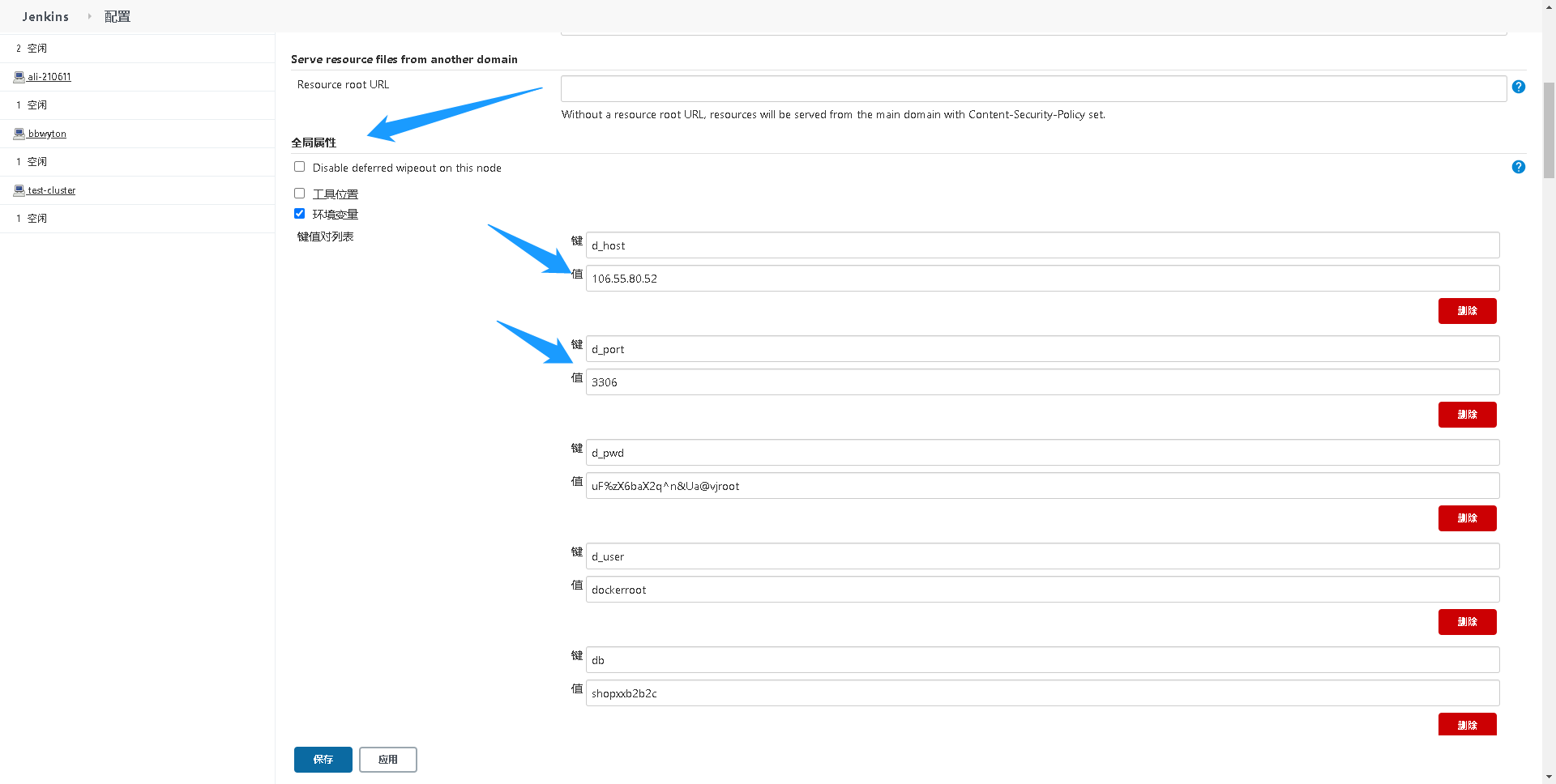
autosql.py

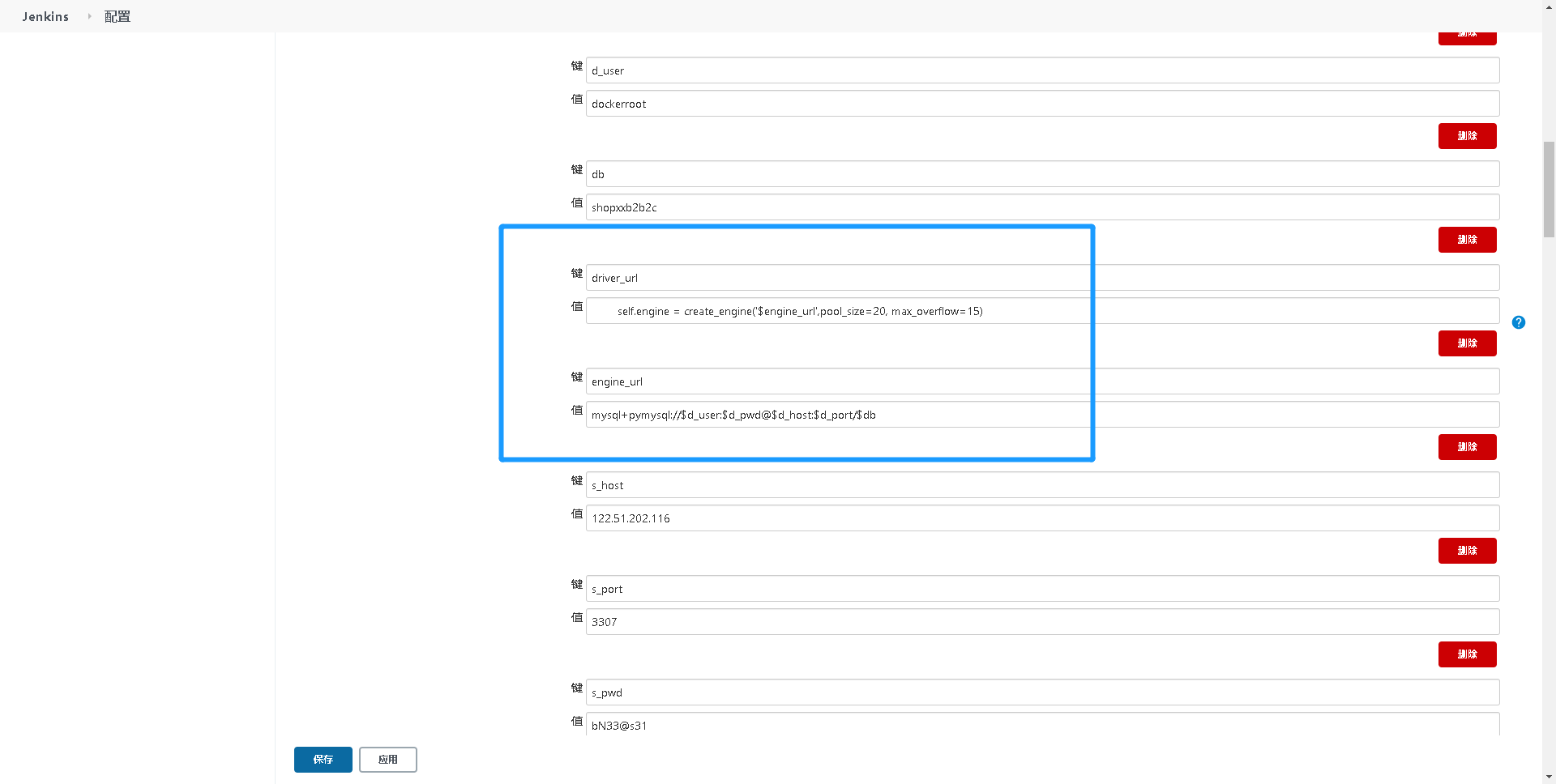
1. # encoding: utf-8
2. from sqlalchemy import create\_engine
3. from sqlalchemy.orm import sessionmaker, scoped\_session
4. from sqlalchemy.ext.declarative import declarative\_base
5. import os
7. table\_key = []
8. class tb\_:
9. def \_\_init\_\_(self):
10. #数据库名字
11. self.db\_name= '';
12. #表名字
13. self.table\_name = '';
14. #单元名字
15. self.column\_name = '';
16. #类型
17. self.type = '';
18. #限制名
19. self.constraint\_name = '';
20. #update rule
21. self.update\_rule = '';
23. class Sql\_Auto():
25. def \_\_init\_\_(self):
26. self.engine = create\_engine('mysql+pymysql://docker:uF%zX6baX2q^n&Ua@vj@106.53.208.130:3306/weitong',pool\_size=20, max\_overflow=15)
27. self.DBSession = sessionmaker(bind=self.engine)
28. #-----------------------获取所有表-------------------------------
29. def get\_all\_table(self):
30. tables = self.engine.table\_names()
31. return tables
32. #---------------------------------------------------------
33. def loop\_all\_table(self):
34. self.delete\_all\_keys()
35. tables = self.get\_all\_table()
36. for itr in tables:
37. self.delete\_foreign\_key(itr)

40. #except primary key
41. def delete\_foreign\_key(self, table\_name):
42. show\_key = 'SHOW keys FROM ' + table\_name;
43. session = self.DBSession()
44. cursor = session.execute(show\_key)
45. ret = cursor.fetchall()
46. for itr in ret:
47. if itr[1] == 1:
48. drop\_index = 'alter table ' + itr[0] + ' drop index ' + itr[2]
49. drop\_key = 'alter table ' + itr[0] + ' drop foreign key ' + itr[2]
50. try:
51. session.execute(drop\_index)
52. session.execute(drop\_key)
53. tmp = {itr[0]:itr[2]}
54. table\_key.append(tmp)
55. except BaseException:
56. pass
57. #print(drop\_key)
58. #print(drop\_index)
60. if itr[1] != 'PRIMARY' and itr[1] != 0 and itr[1] != 1: ## for more debuging
61. drop\_index = 'alter table ' + itr[0] + ' drop index ' + itr[1]
62. drop\_key = 'alter table ' + itr[0] + ' drop foreign key ' + itr[1]
63. try:
64. session.execute(drop\_key)
65. session.execute(drop\_index)
66. tmp = {itr[0]:itr[1]}
67. table\_key.append(tmp)
68. except BaseException:
69. print(drop\_key)
70. print(drop\_index)
72. #---------------------------------------------------
73. def delete\_all\_keys(self):
74. session = self.DBSession()
75. cursor = session.execute('SELECT O.TABLE\_NAME, O.CONSTRAINT\_NAME FROM ( SELECT K.CONSTRAINT\_SCHEMA, K.CONSTRAINT\_NAME, K.TABLE\_SCHEMA, K.TABLE\_NAME, K.COLUMN\_NAME, K.REFERENCED\_TABLE\_SCHEMA, K.REFERENCED\_TABLE\_NAME, K.REFERENCED\_COLUMN\_NAME, R.UPDATE\_RULE, R.DELETE\_RULE, R.UNIQUE\_CONSTRAINT\_NAME FROM information\_schema.KEY\_COLUMN\_USAGE K LEFT JOIN information\_schema.REFERENTIAL\_CONSTRAINTS R ON K.CONSTRAINT\_NAME = R.CONSTRAINT\_NAME) AS O INNER JOIN Information\_schema.TABLE\_CONSTRAINTS T ON O.Table\_Name = T.TABLE\_NAME AND T.CONSTRAINT\_NAME = O.CONSTRAINT\_NAME')
76. result = cursor.fetchall()
77. tk = []
78. print(result)
79. for itr in result:
80. if itr[1] != 'PRIMARY':
81. drop\_key = 'alter table ' + itr[0] + ' drop foreign key ' + itr[1]
82. drop\_index = 'alter table ' + itr[0] + ' drop index ' + itr[1]
83. try:
84. cursor = session.execute(drop\_key)
85. cursor = session.execute(drop\_index)
86. #except BaseException:
87. except Exception as e:
88. print(str(e))
89. #print('error',str(e))
91. sqlauto = Sql\_Auto().loop\_all\_table()
92. #sqlauto = Sql\_Auto()
93. #sqlauto.loop\_all\_table()
94. #print(table\_key)
96. #restore\_all\_key
97. #rt = sqlauto.get\_all\_keys()
98. #print(rt[-1].constraint\_name)
99. #print(rt[-1].update\_rule)
100. **设置jenkins中的环境变量**









1. **jenkins中的shell脚本**
2. backup\_dir="backup"
3. if [ ! -d "$backup\_dir" ];then
4. mkdir $backup\_dir
5. fi
6. revertbackup\_dir="revert\_backup"
7. if [ ! -d "$revertbackup\_dir" ];then
8. mkdir $revertbackup\_dir
9. fi
11. #同步之前，要用下面这个autosql.py断掉目标数据库的所有外键
12. cp /usr/autosql.py $WORKSPACE
13. echo $driver\_url
14. sed -i "25 a\\$driver\_url" autosql.py
15. sed -i "27d" autosql.py
16. python autosql.py
18. sync\_date=`date +"%Y%m%d"`
20. schemasync mysql://${s\_user}:${s\_pwd}@${s\_host}:${s\_port}/${db} mysql://${d\_user}:${d\_pwd}@${d\_host}:${d\_port}/${db}
21. pwd
23. patch\_file="${db}.${sync\_date}.patch.sql"
24. revert\_file="${db}.${sync\_date}.revert.sql"
25. if [ -f $patch\_file ]
26. then
27. mysql -h ${d\_host} -P ${d\_port} -u${d\_user} -p"${d\_pwd}" -D$db < "${patch\_file}"
28. cp ${patch\_file} backup/
29. rm ${patch\_file}
30. cp ${revert\_file} revert\_backup/
31. rm ${revert\_file}
32. else
33. echo "结构一致，不需要Migration操作"
34. fi
35. **sqlalchemy创建数据库**

<https://blog.csdn.net/qq_36306369/article/details/89240837?utm_medium=distribute.pc_relevant.none-task-blog-title-3&spm=1001.2101.3001.4242>

**createdatabase.py**

1. # encoding: utf-8
2. import sqlalchemy
4. with sqlalchemy.create\_engine(
5. 'mysql+pymysql://dockerroot:uF%zX6baX2q^n&Ua@vjroot@106.55.80.52:3306/',
6. isolation\_level='AUTOCOMMIT').connect() as connection:
7. connection.execute('CREATE DATABASE IF NOT EXISTS shopxxb2b2c DEFAULT CHARSET utf8 COLLATE utf8\_general\_ci;')
9. **将步骤4的py文件合并到步骤1中**

然后重新编写插入环境变量的语句。

注意由于shell脚本写死了autosql.py文件是从/usr复制的，所以编写sed命令是的文件行号是参考/usr路径下的autosql.py，而非jenkins工作目录复制过来的autosql.py，不然会被覆盖，执行时出错

**总结**

连上阿里云服务器，服务器上宿主机已经装好了mysql

现在要测试拆掉外键前，连接的数据库是否能判断有误数据库并自动作创建的任务

确认了宿主机的mysql里面没有需要同步的数据库名（即没有对应的数据库）

<https://cloud.tencent.com/developer/ask/74646>

按照上面的方法，进行数据库存在的判断，若没有数据库就开始进行创建数据库。

之后再执行拆外键和同步工作，sql自动化完成。

sed命令https://www.cnblogs.com/tureno/articles/6677942.html

<https://blog.csdn.net/qq_37674858/article/details/80066264>

**文本，脚本最终版**

**autosql.py**

1. # encoding: utf-8
2. from sqlalchemy import create\_engine
3. from sqlalchemy.orm import sessionmaker, scoped\_session
4. from sqlalchemy.ext.declarative import declarative\_base
5. import os
6. import sqlalchemy
7. table\_key = []
8. class tb\_:
9. def \_\_init\_\_(self):
10. #数据库名字
11. self.db\_name= '';
12. #表名字
13. self.table\_name = '';
14. #单元名字
15. self.column\_name = '';
16. #类型
17. self.type = '';
18. #限制名
19. self.constraint\_name = '';
20. #update rule
21. self.update\_rule = '';
23. class Sql\_Auto():
25. def \_\_init\_\_(self):
26. self.engine = create\_engine('mysql+pymysql://docker:uF%zX6baX2q^n&Ua@vj@106.53.208.130:3306/weitong',pool\_size=20, max\_overflow=15)
27. self.DBSession = sessionmaker(bind=self.engine)
28. #-----------------------获取所有表-------------------------------
29. def get\_all\_table(self):
30. tables = self.engine.table\_names()
31. return tables
32. #---------------------------------------------------------
33. def loop\_all\_table(self):
34. self.delete\_all\_keys()
35. tables = self.get\_all\_table()
36. for itr in tables:
37. self.delete\_foreign\_key(itr)

40. #except primary key
41. def delete\_foreign\_key(self, table\_name):
42. show\_key = 'SHOW keys FROM ' + table\_name;
43. session = self.DBSession()
44. cursor = session.execute(show\_key)
45. ret = cursor.fetchall()
46. for itr in ret:
47. if itr[1] == 1:
48. drop\_index = 'alter table ' + itr[0] + ' drop index ' + itr[2]
49. drop\_key = 'alter table ' + itr[0] + ' drop foreign key ' + itr[2]
50. try:
51. session.execute(drop\_index)
52. session.execute(drop\_key)
53. tmp = {itr[0]:itr[2]}
54. table\_key.append(tmp)
55. except BaseException:
56. pass
57. #print(drop\_key)
58. #print(drop\_index)
60. if itr[1] != 'PRIMARY' and itr[1] != 0 and itr[1] != 1: ## for more debuging
61. drop\_index = 'alter table ' + itr[0] + ' drop index ' + itr[1]
62. drop\_key = 'alter table ' + itr[0] + ' drop foreign key ' + itr[1]
63. try:
64. session.execute(drop\_key)
65. session.execute(drop\_index)
66. tmp = {itr[0]:itr[1]}
67. table\_key.append(tmp)
68. except BaseException:
69. print(drop\_key)
70. print(drop\_index)
72. #---------------------------------------------------
73. def delete\_all\_keys(self):
74. session = self.DBSession()
75. cursor = session.execute('SELECT O.TABLE\_NAME, O.CONSTRAINT\_NAME FROM ( SELECT K.CONSTRAINT\_SCHEMA, K.CONSTRAINT\_NAME, K.TABLE\_SCHEMA, K.TABLE\_NAME, K.COLUMN\_NAME, K.REFERENCED\_TABLE\_SCHEMA, K.REFERENCED\_TABLE\_NAME, K.REFERENCED\_COLUMN\_NAME, R.UPDATE\_RULE, R.DELETE\_RULE, R.UNIQUE\_CONSTRAINT\_NAME FROM information\_schema.KEY\_COLUMN\_USAGE K LEFT JOIN information\_schema.REFERENTIAL\_CONSTRAINTS R ON K.CONSTRAINT\_NAME = R.CONSTRAINT\_NAME) AS O INNER JOIN Information\_schema.TABLE\_CONSTRAINTS T ON O.Table\_Name = T.TABLE\_NAME AND T.CONSTRAINT\_NAME = O.CONSTRAINT\_NAME')
76. result = cursor.fetchall()
77. tk = []
78. print(result)
79. for itr in result:
80. if itr[1] != 'PRIMARY':
81. drop\_key = 'alter table ' + itr[0] + ' drop foreign key ' + itr[1]
82. drop\_index = 'alter table ' + itr[0] + ' drop index ' + itr[1]
83. try:
84. cursor = session.execute(drop\_key)
85. cursor = session.execute(drop\_index)
86. #except BaseException:
87. except Exception as e:
88. print(str(e))
89. #print('error',str(e))

92. with sqlalchemy.create\_engine(
93. 'mysql+pymysql://dockerroot:uF%zX6baX2q^n&Ua@vjroot@106.55.80.52:3306/',
94. isolation\_level='AUTOCOMMIT').connect() as connection:
95. connection.execute('CREATE DATABASE IF NOT EXISTS shopxxb2b2c DEFAULT CHARSET utf8 COLLATE utf8\_general\_ci;')



100. sqlauto = Sql\_Auto().loop\_all\_table()
102. #sqlauto.loop\_all\_table()^M
103. #print(table\_key)^M
104. #restore\_all\_key^M
105. #rt = sqlauto.get\_all\_keys()^M
106. #print(rt[-1].constraint\_name)^M
107. #print(rt[-1].update\_rule)^M

**jenkins中的配置的shell脚本**

1. backup\_dir="backup"
2. if [ ! -d "$backup\_dir" ];then
3. mkdir $backup\_dir
4. fi
5. revertbackup\_dir="revert\_backup"
6. if [ ! -d "$revertbackup\_dir" ];then
7. mkdir $revertbackup\_dir
8. fi
10. cp /usr/autosql.py $WORKSPACE
11. #同步之前先检查目标数据库是否已经有数据库，若无就创建
12. echo $create\_url
13. echo $create\_command
14. echo $print\_message
15. sed -i "92 a\\$create\_url" autosql.py
16. sed -i "94d" autosql.py
18. sed -i "94 a\\$create\_command" autosql.py
19. sed -i "96d" autosql.py
21. sed -i "98 a\\$print\_message" autosql.py
23. #
25. #同步之前，要用下面这个autosql.py断掉目标数据库的所有外键
27. echo $driver\_url
28. sed -i "25 a\\$driver\_url" autosql.py
29. sed -i "27d" autosql.py
30. python autosql.py
32. sync\_date=`date +"%Y%m%d"`
34. schemasync mysql://${s\_user}:${s\_pwd}@${s\_host}:${s\_port}/${db} mysql://${d\_user}:${d\_pwd}@${d\_host}:${d\_port}/${db}
35. pwd
37. patch\_file="${db}.${sync\_date}.patch.sql"
38. revert\_file="${db}.${sync\_date}.revert.sql"
39. if [ -f $patch\_file ]
40. then
41. mysql -h ${d\_host} -P ${d\_port} -u${d\_user} -p"${d\_pwd}" -D$db < "${patch\_file}"
43. cp ${patch\_file} backup/
44. rm ${patch\_file}
46. cp ${revert\_file} revert\_backup/
47. rm ${revert\_file}
48. else
49. echo "结构一致，不需要Migration操作"
50. fi

53. #cat -n filename 可以查看文件的同时查看行号