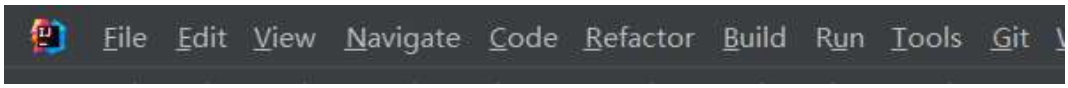
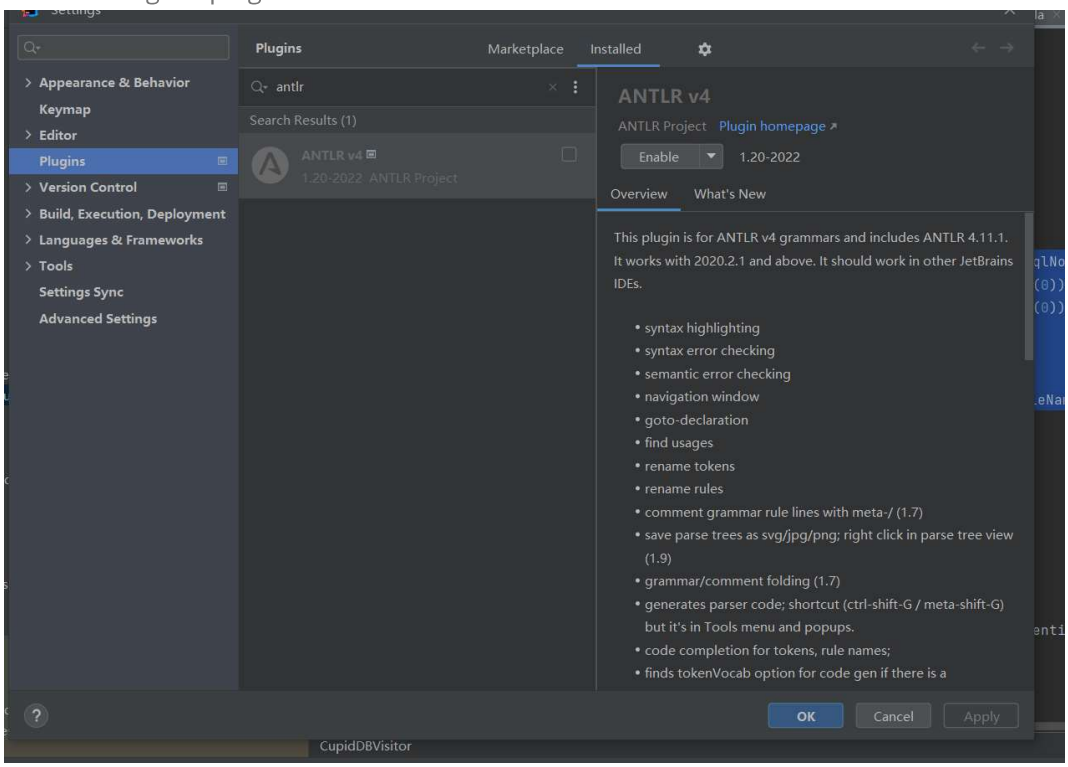
sql：create table like 的实现样例

1. 首先 完成sql语句的文法定义： cupid-db-calcite/src/main/antlr4/org/urbcomp/cupid/db/parser/parser/CupidDBSql.g4

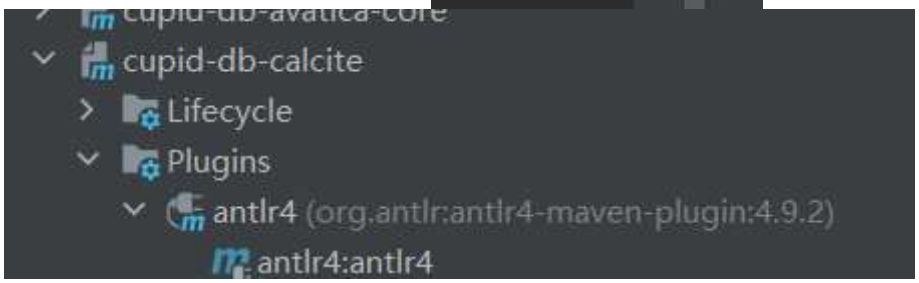
```
stmt :
        createDatabaseStmt
      | createTableStmt
      | createTableLikeStmt
```

```
createTableLikeStmt: T_CREATE T_TABLE (T_IF T_NOT T_EXISTS) ? table_name T_LIKE table_name;
```

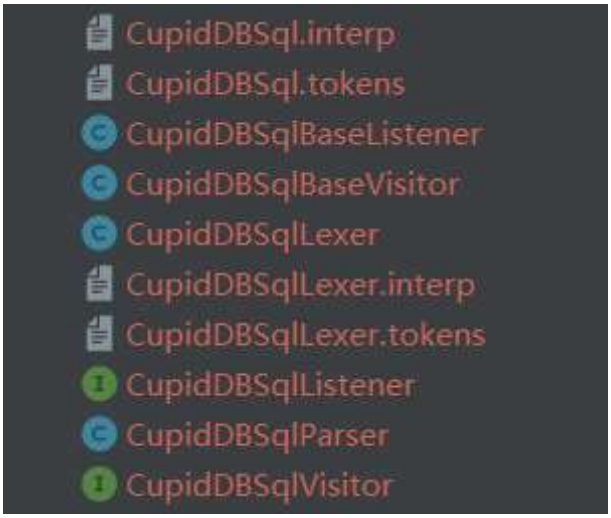
2. 通过antlr工具生成我们需要的工具类：

> 方法一：安装插件： 直接右键 g4文件 可以看到 gennerate选项
>
> 安装插件的方法： idea 左上角
>
> 
>
>
> file -> settings -> plugins
>
> 
>
>
> 安装了插件之后 重启idea 右键 g4文件： configure可以配置 生成 文件的位置 Generate是生成我们需要的工具类

方法二： 打开idea右面的maven页面



建议使用方法一 idea的插件 ， 方法二， 好像需要完全其他模块的打包才能运行

> 工具类生成之后 ，
>
> 1. 这是生成的代码：
>
> 
>
>
> 2. listener不需要， 直接删除， 其他的直接覆盖原文件 。注意： 记得在生成文件加上包名 eg:package org...

3. 实现createTableLike： 目录： cupid-db-calcite/src/main/java/org/urbcomp/cupid/db/parser/ddl/SqlCupidCreateTableLike.java

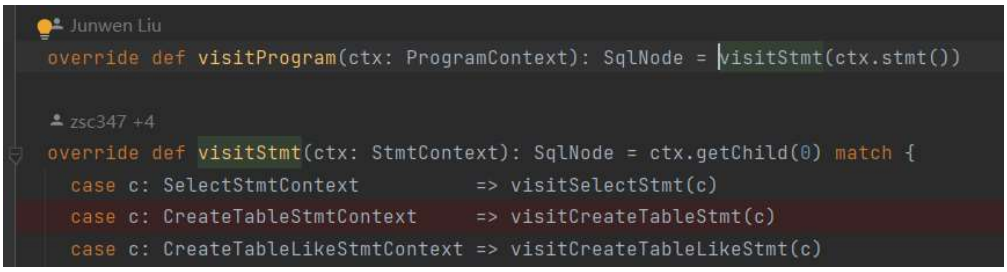
``

```
package org.urbcomp.cupid.db.parser.ddl;

import org.apache.calcite.sql.*;
import org.apache.calcite.sql.parser.SqlParserPos;
import org.apache.calcite.util.ImmutableNullableList;

import java.util.List;

public class SqlCupidCreateTableLike extends SqlCreate {

    public final SqlIdentifier targetTableName;

    public final SqlIdentifier sourceTableName;

    private static final SqlOperator OPERATOR = new SqlSpecialOperator(
        "CREATE TABLE",
        SqlKind.CREATE_TABLE
    );

    public SqlCupidCreateTableLike(
        SqlParserPos pos,
        boolean replace,
        boolean ifNotExists,
        SqlIdentifier targetTableName,
        SqlIdentifier sourceTableName
    ) {
        super(OPERATOR, pos, replace, ifNotExists);
        this.targetTableName = targetTableName;
        this.sourceTableName = sourceTableName;
    }
```

```
    /** Creates a SqlCreateTableLike. */

    public List<SqlNode> getOperandList() {
        return ImmutableNullableList.of(targetTableName, sourceTableName);
    }

    @Override
    public void unparse(SqlWriter writer, int leftPrec, int rightPrec) {
        writer.keyword("CREATE");
        writer.keyword("TABLE");
        if (ifNotExists) {
            writer.keyword("IF NOT EXISTS");
        }
        targetTableName.unparse(writer, leftPrec, rightPrec);
        writer.keyword("LIKE");
        sourceTableName.unparse(writer, leftPrec, rightPrec);

    }
}
```

注释： 因为 在执行createTableLike 语句的时候 我们需要知道的是： 目的表名 以及 like的源表名，以及 是否有if not exists 的说明

4. 实现sqlCupidCreateTableLike 类的visitor： 因为需要通过该visitor 访问 create table like 这天sql语句生成的抽象语法树 得到我们步骤三实现的 sqlCupidCreateTableLike 类 最后传递给执行器执行

> 该visitor在calcite模块 的scala子模块下 ： cupid-db-calcite/src/main/scala/org/urbcomp/cupid/db/parser/visitor/CupidDBVisitor.scala
>
> 1. 首先 添加 createTableLikestmtContext（这是工具自动生成的）的case
>
> 
>
>
> 2. 实现 visitCreateTableLikeStmt();
>
> 该函数的功能就是通过访问抽象语法树得到 我们需要的sqlCupidCreateTableLike类 然后交给执行器执行
>
> ``

```
override def visitCreateTableLikeStmt(ctx:
CreateTableLikeStmtContext): SqlNode = {
  val targetTableName =
visitIdent(ctx.table_name(0).qident().ident().get(0))
  val sourceTableName =
visitIdent(ctx.table_name(1).qident().ident().get(0))

  val ifNotExists = null != ctx.T_EXISTS()

  new SqlCupidCreateTableLike(pos, false, ifNotExists,
targetTableName, sourceTableName)
}
```

3. 测试

   测试模块在 calcite的　test模块的CupidDBVisitorTest中

``

```
test("convert create table like statement to SqlNode") {
  val sql = CupidDBSQLSamples.CREATE_TABLE_LIKE_SAMPLE
  val parsed = driver.parseSql(sql)
  val node = parsed.asInstanceOf[SqlCupidCreateTableLike]
  assertEquals("target_table", node.targetTableName.names.get(0))

}
```

5. 实现执行器： 传入的参数为我们上面通过visitor得到的sqlCreateTableLike 类　（里面有我们执行的时候需要的两个表名 以及 是否有 if not exists的申明 ）

   执行器类 在 db-core模块的executor子模块中

   新建 我们的 执行器类 CreateTableLikeExecutor 类：

``

```
package org.urbcomp.cupid.db.executor

import org.geotools.data.DataStoreFinder
import org.geotools.feature.simple.SimpleFeatureTypeBuilder
import org.urbcomp.cupid.db.executor.utils.ExecutorUtil
import org.urbcomp.cupid.db.infra.{BaseExecutor, MetadataResult}
import org.urbcomp.cupid.db.metadata.MetadataAccessUtil
import org.urbcomp.cupid.db.metadata.entity.{Field, Index, Table}
import org.urbcomp.cupid.db.parser.ddl.SqlCupidCreateTableLike
import org.urbcomp.cupid.db.transformer.{
RoadSegmentAndGeomesaTransformer,
TrajectoryAndFeatureTransformer
}
import org.urbcomp.cupid.db.util.{DataTypeUtils, MetadataUtil}

import scala.collection.JavaConverters._

case class CreateTableLikeExecutor(n: SqlCupidCreateTableLike) extends
BaseExecutor {
```

```
override def execute[Int]():  MetadataResult[Int] = {

 val targetTable = n.targetTableName
 val (userName, dbName, tableName) =
ExecutorUtil.getUserNameDbNameAndTableName(targetTable)
 val db = MetadataAccessUtil.getDatabase(userName, dbName)
 val existedTargetTable = MetadataAccessUtil.getTable(db.getId,
tableName)
 if (existedTargetTable != null) {
   if (n.ifNotExists) {
     return MetadataResult.buildDDLResult(0)
   } else {
     throw new IllegalArgumentException("table already exist " +
tableName)
   }
 }

 val sourceTable = n.sourceTableName
 val (sourceUserName, sourceDbName, sourceTableName) =
   ExecutorUtil.getUserNameDbNameAndTableName(sourceTable)
 val sourceDb = MetadataAccessUtil.getDatabase(sourceUserName,
sourceDbName)
 val existedSourceTable = MetadataAccessUtil.getTable(sourceDb.getId,
sourceTableName)
 if (existedSourceTable == null)
   throw new IllegalArgumentException("sourceTable not exist " +
tableName)

 var affectedRows = 0L
 MetadataAccessUtil.withRollback(
   _ => {
     affectedRows =
       MetadataAccessUtil.insertTable(new Table(0L /* unused */,
db.getId, tableName, "hbase"))
     val createdTable = MetadataAccessUtil.getTable(db.getId,
tableName)
     val tableId = createdTable.getId

     val sfb = new SimpleFeatureTypeBuilder
     val schemaName = MetadataUtil.makeSchemaName(tableId)
     sfb.setName(schemaName)

     val params = ExecutorUtil.getDataStoreParams(userName, dbName)
     val dataStore = DataStoreFinder.getDataStore(params)
     val schema = dataStore.getSchema(schemaName)
     if (schema != null) {
       throw new IllegalStateException("schema already exist " +
schemaName)
     }

     //copy col
     val fieldMap = collection.mutable.Map[String, Field]()
     MetadataAccessUtil
       .getFields(sourceUserName, sourceDbName, sourceTableName)
```

```
        .forEach(field => {
          val sourceDataType = field.getType
          val sourceClassType = DataTypeUtils.getClass(sourceDataType)
          if (DataTypeUtils.isGeometry(sourceDataType)) {
            sfb.add(field.getName, sourceClassType, 4326)
          } else {
            sfb.add(field.getName, sourceClassType)
          }

          val sourceField =
            new Field(0, tableId, field.getName, field.getType,
field.getIsPrimary)
          MetadataAccessUtil.insertField(sourceField)
          fieldMap.put(field.getName, field)
        })

      //copy index
      val indexes = MetadataAccessUtil
        .getIndexes(sourceUserName, sourceDbName, sourceTableName)
        .asScala
        .toList
        .toArray

      if (indexes != null) {
        checkIndexNames(indexes)
        indexes.foreach(index => {
          val tarIndex = new Index(
            tableId,
            index.getIndexType,
            index.getName,
            index.getFieldsIdList,
            index.getIndexProperties
          )
          MetadataAccessUtil.insertIndex(tarIndex)
        })

      }

      var sft = sfb.buildFeatureType()
      sft = new TrajectoryAndFeatureTransformer().getGeoMesaSFT(sft)
      sft = new RoadSegmentAndGeomesaTransformer().getGeoMesaSFT(sft)

      // allow mixed geometry types for support cupid-db type `Geometry`
      sft.getUserData.put("geomesa.mixed.geometries",
java.lang.Boolean.TRUE)

      val geomesaIndexDecl = indexes
        .map(idx => {

s"${idx.getIndexType}:${idx.getFieldsIdList.split(",").mkString(":")}"
        })
        .mkString(",")
      sft.getUserData.put("geomesa.indices.enabled", geomesaIndexDecl)

      dataStore.createSchema(sft)
```

```
    },
    classOf[Exception]
  )

  MetadataResult.buildDDLResult(affectedRows.toInt)
}

private def formatName(colName: String, order: Int): String = {
  s"$colName${if (order == 1) "" else s"_$order"}"
}

/**
    * Check if index names duplicate
    * For index name not explicitly defined, use ${columnName}_${order}
with minimum order satisfy name not duplicate
    */
  private def checkIndexNames(indexes: Array[Index]): Unit = {
    val names = collection.mutable.Set[String]()
    indexes.foreach(idx => {
      if (idx.getName == null) {
        val colName = idx.getFieldsIdList.split(",")(0)
        var order = 1
        while (names.contains(formatName(colName, order))) {
          order += 1
        }
        idx.setName(formatName(colName, order))
      }
      if (names.contains(idx.getName)) {
        throw new IllegalArgumentException(s"Duplicate index name
${idx.getName}")
      }
      names.add(idx.getName)
    })
  }

}
```

6. 最后 下面的两个地方需要新增我们创建的类的一个判断

```
cupid-db-core/src/main/java/org/apache/calcite/prepare/CalcitePrepareImpl.java   Viewed
@@ -708,7 +708,8 @@ <T> CalciteSignature<T> prepare2_(
                    || sqlNode instanceof SqlDropIndex
                    || sqlNode instanceof SqlDropSchema
                    || sqlNode instanceof SqlTruncateTable
-                   || sqlNode instanceof SqlRenameTable) {
                    return
    startDBExecutorFactory.convertExecutor(sqlNode).execute();
                }

708                 || sqlNode instanceof SqlDropIndex
709                 || sqlNode instanceof SqlDropSchema
710                 || sqlNode instanceof SqlTruncateTable
711 +               || sqlNode instanceof SqlRenameTable
712 +               || sqlNode instanceof SqlCupidCreateTableLike) {
713                 return
    startDBExecutorFactory.convertExecutor(sqlNode).execute();
714             }
715
```

40 cupid-db-core/src/main/scala/org/urbcomp/cupid/db/executor/CupidDBExecutorFactory.scala Viewed

```
40      override def convertExecutor(node: SqlNode): BaseExecutor = node match {
41  -     case n: SqlShowCreateTable  => ShowCreateTableExecutor(n)
42  -     case n: SqlUpdate           => UpdateExecutor(n)
43  -     case n: SqlInsert           => InsertExecutor(n)
44  -     case n: SqlDelete           => DeleteExecutor(n)
45  -     case n: SqlCreateDatabase   => CreateDatabaseExecutor(n)
46  -     case n: SqlCupidCreateTable => CreateTableExecutor(n)
47  -     case _: SqlShowDatabases    => ShowDatabaseExecutor()
48  -     case _: SqlShowStatus       => ShowStatusExecutor()
49  -     case n: SqlUseDatabase      => UseDbExecutor(n)
50  -     case n: SqlCreateUser       => CreateUserExecutor(n)
51  -     case n: SqlShowTables       => ShowTablesExecutor(n)
52  -     case n: SqlShowIndex        => ShowIndexExecutor(n)
53  -     case n: SqlDropTable        => DropTableExecutor(n)
54  -     case n: SqlDropIndex        => DropIndexExecutor(n)
55  -     case n: SqlDescribeTable    => DescribeTableExecutor(n)
56  -     case n: SqlDropSchema       => DropDatabaseExecutor(n)
57  -     case n: SqlTruncateTable    => TruncateTableExecutor(n)
58  -     case n: SqlRenameTable      => RenameTableExecutor(n)
59  -     case _                      => throw new IllegalStateException("Not
    Support SQL")
60      }
61    }
```

```
41      override def convertExecutor(node: SqlNode): BaseExecutor = node match {
42  +     case n: SqlShowCreateTable      => ShowCreateTableExecutor(n)
43  +     case n: SqlUpdate               => UpdateExecutor(n)
44  +     case n: SqlInsert               => InsertExecutor(n)
45  +     case n: SqlDelete               => DeleteExecutor(n)
46  +     case n: SqlCreateDatabase       => CreateDatabaseExecutor(n)
47  +     case n: SqlCupidCreateTable     => CreateTableExecutor(n)
48  +     case _: SqlShowDatabases        => ShowDatabaseExecutor()
49  +     case _: SqlShowStatus           => ShowStatusExecutor()
50  +     case n: SqlUseDatabase          => UseDbExecutor(n)
51  +     case n: SqlCreateUser           => CreateUserExecutor(n)
52  +     case n: SqlShowTables           => ShowTablesExecutor(n)
53  +     case n: SqlShowIndex            => ShowIndexExecutor(n)
54  +     case n: SqlDropTable            => DropTableExecutor(n)
55  +     case n: SqlDropIndex            => DropIndexExecutor(n)
56  +     case n: SqlDescribeTable        => DescribeTableExecutor(n)
57  +     case n: SqlDropSchema           => DropDatabaseExecutor(n)
58  +     case n: SqlTruncateTable        => TruncateTableExecutor(n)
59  +     case n: SqlRenameTable          => RenameTableExecutor(n)
60  +     case n: SqlCupidCreateTableLike => CreateTableLikeExecutor(n)
61  +     case _                          => throw new IllegalStateException("Not
    Support SQL")
62      }
63    }
```

测试 ：

在测试模块中 新建一个测试：

``

```
package org.urbcomp.cupid.db

import org.junit.Assert.assertEquals

class CreateTest extends AbstractCalciteSparkFunctionTest {

  test("create table like") {

    val stmt = connect.createStatement()
    stmt.executeUpdate("drop table if exists sourceTable")
    stmt.executeUpdate(
      "create table sourceTable(age INTEGER, name String, st Point, SPATIAL INDEX
indexName(st))"
    )
    stmt.executeUpdate("drop   table if exists target1Table")
    stmt.executeUpdate("create table target1table like sourceTable")

    //test col name
    val rsOfCol = stmt.executeQuery("select  * from target1Table")
    rsOfCol.next()
    assertEquals(1, rsOfCol.findColumn("age"))
    assertEquals(2, rsOfCol.findColumn("name"))
    assertEquals(3, rsOfCol.findColumn("st"))

    //test index
    val rsOfIndex = stmt.executeQuery("show index from target1Table")
    rsOfIndex.next()
    assertEquals("target1table", rsOfIndex.getString(1))
    assertEquals("indexName", rsOfIndex.getString(2))
    assertEquals("z2", rsOfIndex.getString(3))
```

```
    assertEquals("st", rsOfIndex.getString(4))

  }

}
```