在我们的实现中，查询模块的主要功能主要有两点，一是通过对sql语句的解析，得到输入文本中对应的sql语义和操作。二是根据我们解析得到的sql语句，执行对应的操作，与服务端其它部分对接，返回查询和操作结果。我们将详细介绍这两个子模块的设计思路。

语句解析

在语句解析上，我们主要使用antlr的visitor模块，遍历实际语法树，对不同的节点采取对应的操作。对于每一条sql中合法的statement，我们将visitor解析得到的元数据封装为一个`BaseStatement`的子类，例如一个`SelectStatement`对象封装了一条具体的查询语句，其包括查询的列名，where条件，涉及的表等内容。

为了更好地结构化sql statement的数据，我们还设计了一些辅助类。`Comparer`代表一个布尔表达式中的元素，可能是常数，也可能是一个列名。`Condition`代表一个具体的条件，可能对应where子句，也可能对应join condition。`TableQuery`代表一个from子句的子项，例如单独的一张表或是多张表的连接。`ValueEntry`代表insert操作中的一组值。

语句执行

在语句执行上，为了解耦各个模块的功能，我们在查询模块并不直接负责语句的执行，而是通过`Statement`对象封装执行该语句所需的所有数据，并提供`exec`接口，该接口负责执行当前语句，并返回一个`SQLEvalResult`。这一设计下，我们的语句执行可以与之后的事务并发和控制相结合，实现更多样的执行过程。

附加功能

目前，我们实现的附加功能包括:1.where条件逻辑运算符的支持(and/or) 2.三张表以上的join 3. 其他类型的join。其实现思路如下:

1.对于where条件逻辑运算符，我们通过`Condition`类，用二叉树组织一条语句的所有条件，一个`Condition`对象的左右子节点分别代表两条子条件，并根据该对象的`logic_op`，即逻辑运算符类型，在查询时，递归地求左右子节点的逻辑值，并进行合并。

2. 对于三张表以上的join，我们同样通过`TableQuery`类，用二叉树结构组织待连接的表，其两个子节点分别代表左表和右表(或是连接的中间体)，通过不同的join类型，实现不同的连接操作

3. 对于不同类型的join,我们通过`TableQuery`类的递归执行流程，通过join type的判断和分支，可以很自然的根据其原理实现。目前我们支持natural,left/right/full outer,inner这几种join类型。

虽然目前我们并没有实现所有的进阶要求，但我们计划持续更新代码，争取实现更多的查询功能。