HSQLDB的查询处理分析

文庆福
2011013239 thssvince@163.com
清华大学软件学院11班

2014年 5月 30日

目录

1 实验简介

1.1 实验背景

查询处理一般分为两个过程，即查询语句的解析和数据集的获得。前一阶段将用户输入的 SQL 语句解析为中间结构，后一阶段则根据该中间结构来筛选、组织符合要求的数据。

HSQLDB 查询处理的主要结构如下图所示。接收到 SQL 语句命令后，Database 会创建一个 Scanner 进行语句的分词操作。如果是“INSERT”“UPDATE”“DELETE”或“SELECT”，则创建 ParserCommand 对象进行进一步解析；对于其他命令，Database 将直接进行处理。对于“SELECT”命令，ParserCommand 会创建相应的 QuerySpecification 对象，用以表达一个具体的 SELECT 结构。QuerySpecification 对象包含一个或者多个 RangeVariable，两组分别表示选择条件和目标属性的 Expression，以及下层的 QueryExpression 实例。

本次实验将对 HSQLDB 的查询处理机制进行分析，主要内容包括：

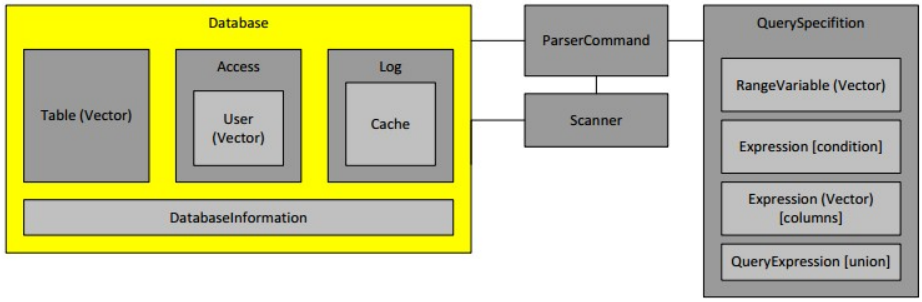

图 1: HSQLDB 查询处理结构

- SQL 语句解析的过程。
- 数据获取的过程。

1.2 实验环境

- 操作系统：Windows 8 企业版
- JDK: OpenJDK 7（64-Bit）
- HSQLDB: 2.3.1
- IDE: Eclipse Standard(Kepler Service Release 1)

2 HSQLDB 查询处理流程概述

HSQLDB 处理 SQL 语句大致分为两个部分——编译和运行。当用户从客户端发送一个 SQL 查询请求之后，服务器端首先对其进行编译，分析生成 Statement，然后利用 Statement 作为参数，调用相应的 execute 方法，执行查询，最后得到查询结果。

首先，HSQLDB 的服务器程序首先在 ServerConnection 类的 run 方法处等待用户请求。当收到请求后，经过一些对消息类型（msgType）的判断，对于查询语句 msgType = ResultConstants.EXECDIRECT，通过 run→receiveResult→ execute，最终跳转到 executeDirectStatement 方法进行处理。

```
1   public Result executeDirectStatement(Result cmd) {
2
3       String         sql = cmd.getMainString();
4       HsqlArrayList  list;
5       int            maxRows = cmd.getUpdateCount();
6
7       if (maxRows == -1) {
8           sessionContext.currentMaxRows = 0;
9       } else if (sessionMaxRows == 0) {
10          sessionContext.currentMaxRows = maxRows;
11      } else {
12          sessionContext.currentMaxRows = sessionMaxRows;
13          sessionMaxRows                = 0;
14      }
15
16      try {
17          list = parser.compileStatements(sql, cmd);
18      } catch (Throwable e) {
19          return Result.newErrorResult(e);
20      }
21
22      Result result = null;
23
24      for (int i = 0; i < list.size(); i++) {
25          Statement cs = (Statement) list.get(i);
26
27          cs.setGeneratedColumnInfo(cmd.getGeneratedResultType(),
28                                    cmd.getGeneratedResultMetaData());
29
30          result = executeCompiledStatement(cs, ValuePool.emptyObjectArray,
31                                            cmd.queryTimeout);
32
33          if (result.mode == ResultConstants.ERROR) {
```

```
34                  break;
35              }
36          }
37
38          return result;
39      }
```

第 17 行调用 org.hsqldb.ParserCommand 类的 compileStatements 方法得到了一个 list，事实上，list 中存放的是经过解析后的 Statement 对象数组（因为一次请求可能包括多个 SQL 语句）；第 30 行调用 executeCompiledStatement 方法分别处理每一个 Statement。最后，该方法返回最后一条 SQL 语句执行的结果。

3 SQL 语句的解析

SQL 语句的解析其实就是从 SQL 语句到 statement 的过程。整个过程所涉及的类及其关系如下： 从上图中可以看到，语法的解析过程是有DQL、DML 和 DDL 三种不同

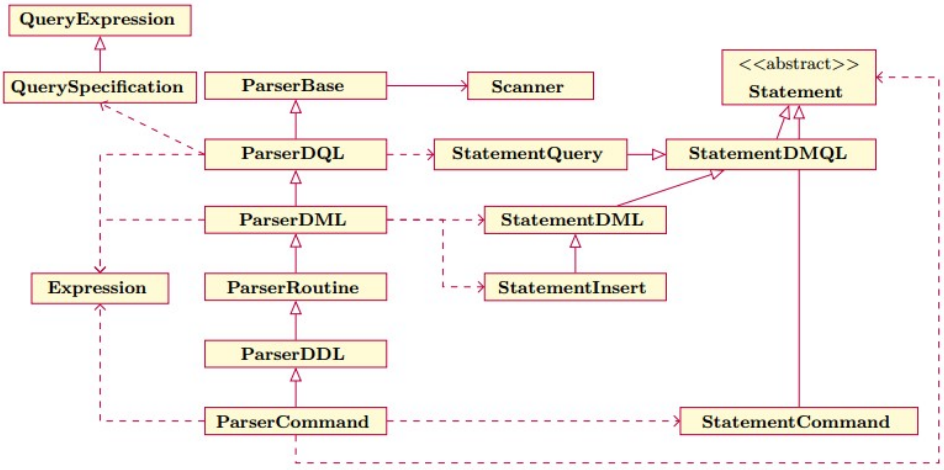

图 2: SQL 解析

类型的解析器，分别对应到数据库中的数据查询语言、数据操纵语言以及数据定义语言。不同类型的 SQL 语句都会对应到一个自己的 Expression 和 Statement。

在 org.hsqldb.ParserCommand 类的 compilePart 方法中，我们可以看到如下代码：

```
1  switch (token.tokenType) {
2
3      // DQL
4      case Tokens.WITH :
5      case Tokens.OPENBRACKET :
6      case Tokens.SELECT :
```

```
7       case Tokens.TABLE : {
8           cs = compileCursorSpecification(RangeGroup.emptyArray, props,
9                                           false);
10
11          break;
12      }
13      case Tokens.VALUES : {
14      ....
```

通过阅读这部分的代码我们可以发现，HSQLDB 对于 SQL 语句的分类如下：

DQL: WITH OPENBRACKET SELECT TABLE VALUES

DML: INSERT UPDATE MERGE DELETE TRUNCATE

DDL: CREATE ALTER DROP GRANT REVOKE COMMENT

PROCEDURE: CALL

diagnostic: GET START COMMIT ROLLBACK SAVEPOINT RELEASE

SESSION: SET

HSQL SESSION: LOCK CONNECT DISCONNECT

HSQL COMMAND: SCRIPT SHUTDOWN BACKUP CHECKPOINT EXPLAIN DECLARE

从之前的类图，再结合编译原理课程的知识，我们知道，SQL 语句的解析实际上是利用语法分析器，生成语法树的过程，这个语法树用 Expression 和 QueryExpression 来组织，并保存在 Statement 及其子类中。

根据上述内容及本次实验的目的，本节将重点分析 DQL 的解析，并对 DML 和 DDL 做简要说明。

3.1 SQL 语句预处理

SQL 语句的解析从 ParserCommand 类的 compileStatements 开始。首先对解析环境进行初始化。

reset(sql);

reset 方法最终会调用 org.hsqldb.Scanner 类的 reset 方法，利用 SQL 语句来初始化词法分析器，同时置当前的 token 为 Tokens.X_STARTPARSE，表示解析的开始。

初始化完成之后，开始对每一句 SQL 语句进行编译，并将得到的 Statement 结果加

入到 list 中。

在 compilePart 方法中，首先设置 parser 指向该 SQL 语句的起点， 然后调用 read 方法读 取一个 token。在大多数 SQL 语句中，一条语句的第一个词代表了该语句的类型和采取的动 作，后面会根据读取的这个词进一步选取后续分析的方法。

Read 方法调用 Scanner 的 scanNext 方法得到一个 token， 如果有词法错误，这里会报告 错误，如果没有错误，则将该 token 加到 Statement 中。词法分析的过程详细分析可以参考 编译原理相关内容，与 SQL 语言相关性不太大，这里不做详细分析了。

3.2 Expression 类的分析

Expression 是 SQL 语句解析中很重要的一个类，它和它的子类保存了 SQL 语句的各种信息，形成一个树状结构。 Expression 类是各个 Expression* 的父类。opType

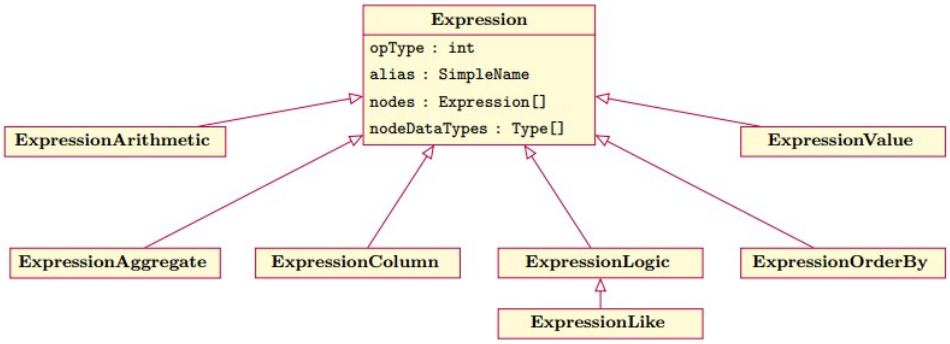

图 3: Expression 及其子类

字段标明了这个表达式 的类别，类别定义在 org.hsqldb.OpTypes 接口中，包括 ORDER_BY、LIMIT、AND 等各种“运算”及 VALUE、COLUMN 等各种值类型。alias 是表达式的别名，用于 as 的处理。nodeDataTypes 和 nodes 分别记录了子表达式的类型和子表达式，以 此来维护一棵表达式树。

此 外，Expression 类 还 通 过 OrderedIntHashSet 维 护 了 6 个 表 达 式 集： aggregateFunctionSet、columnExpressionSet、subqueryExpressionSet、 subqueryAggregateExpressionSet、functionExpressionSet 和 emptyExpressionSet，用于表达式的分类处理。

Expression 的子类根据类名，用于保存不同的表达式，上图列举了其中的一部分子类。例如，ExpressionValue 类用于保存常量，其 valueData 域保存了常 量值；ExpressionOrderBy 类保存了排序表达式，子结点即为排序的关键字。

3.3 DQL 语句解析

HSQLDB 中的 DQL 语句的语法如下：

```
SELECT [ LIMIT n m] [ DISTINCT ]
{ selectExpression | table .* | * } [, ... ]
[ INTO [CACHED|TEMP|TEXT] newTable]
FROM tableList
[ WHERE Expression ]
[ ORDER BY selectExpression [{ ASC | DESC }] [, ...] ]
[ GROUP BY Expression [, ...] ]
[ UNION [ ALL ] selectStatement ]
```

HSQLDB 对 DQL 语句的解析是在 org.hsqldb.ParserDQL 类中进行的。在上 一节的分析中提到，org.hsqldb.ParserCommand 类用来处理传入的 SQL 语句，对 于 DQL 语句，该类的 compileStatements 方法最终会调用 org.hsqldb.ParserDQL 类的 compileCursorSpecification 方法进行解析，并返回一个 StatementQuery 对象。

3.3.1 QueryExpression 与 QuerySpecification

对于 DQL 语句，解析后的表达式树会保存在 StatementQuery 对象的 queryExpression 字段中，queryExpression 实际上是一个 QuerySpecification 对象。

QueryExpression 类保存了用于集合运算的信息，包括两个表达式 leftQueryExpression 和 rightQueryExpression，运算类型 unionType，列映射信息 unionColumnMap。

QuerySpecification 类保存了查询所需的各表达式，包括各列信息（exprColumns 的一部分）、where 子句对应的查询条件 queryCondition、having 子句对应的 条件 havingCondition，以及一些非常重要的下标信息 index*。

QuerySpecification 类会把查询的信息、group by 子句的列名、having 子句的条件、order by 子句的属性以及聚集（aggregate）函数全部放在 exprColumns 中，然后通过一组 index* 变量来进行区分。

3.3.2 语法解析

org.hsqldb.ParserDQL 类是一个完整的语法解析器，通过浏览其代码，我们 可以知道，这个解析器使用的是自顶向下的语法解析方法，这与 SQL 语句（特别 是 DQL 语句）的良好结构是密不可分的，鉴于此，我们也采用自顶向下的方法进 行分析。

之前提到过，解析是从 compileCursorSpecification 方法开始的。

```
StatementQuery compileCursorSpecification(RangeGroup[] rangeGroups,
            int props, boolean isRoutine) {

        OrderedHashSet  colNames        = null;
        QueryExpression queryExpression = XreadQueryExpression();

```

```
7           if (token.tokenType == Tokens.FOR) {
8               read();
9
10              if (token.tokenType == Tokens.READ
11                      || token.tokenType == Tokens.FETCH) {
12                  read();
13                  readThis(Tokens.ONLY);
14
15                  props = ResultProperties.addUpdatable(props, false);
16              } else {
17                  readThis(Tokens.UPDATE);
18
19                  props = ResultProperties.addUpdatable(props, true);
20
21                  if (token.tokenType == Tokens.OF) {
22                      readThis(Tokens.OF);
23
24                      colNames = new OrderedHashSet();
25
26                      readColumnNameList(colNames, null, false);
27                  }
28              }
29          }
30
31          if (ResultProperties.isUpdatable(props)) {
32              queryExpression.isUpdatable = true;
33          }
34
35          queryExpression.setReturningResult();
36          queryExpression.resolve(session, rangeGroups, null);
37
38          StatementQuery cs = isRoutine
39                              ? new StatementCursor(session, queryExpression,
40                                  compileContext)
41                              : new StatementQuery(session, queryExpression,
42                                  compileContext);
43
44          return cs;
45      }
```

3.3.3 构建表达式树

上面代码中的 XreadQueryExpression 方法就是构建表达式树的。

```
QueryExpression queryExpression = XreadQueryExpressionBody();
SortAndSlice    sortAndSlice    = XreadOrderByExpression();
```

其中的 XreadQueryExpressionBody 方法解析 select 语句的主体，而 XreadOrderByExpression 方法解析排序部分。

在 XreadQueryExpressionBody 解析主体时，又分为两部分，第一部分处理单个的查询操作，第二部分循环检查 token 是否为 Tokens.INTERSECT、Tokens.UNION、Tokens.EXCEPT 或 Tokens.MINUS_EXCEPT，如果是，则调用 XreadSetOperation 方法处理集合操作。事实上，由于这些运算的优先级不同，Tokens.INTERSECT 操作是在更内层的方法判断的。

其结构层次如下：

最后， select 语句的主体部分是在的主体部分在 XreadQuerySpecification 方法中被解

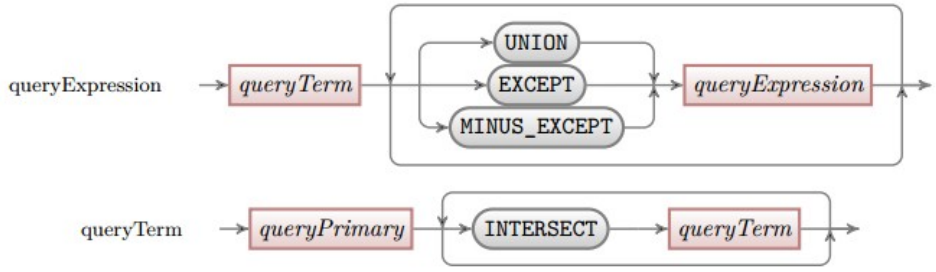

图 4: queryExpression 结构

析。

```
QuerySpecification XreadQuerySpecification() {

    QuerySpecification select = XreadSelect();

    if (!select.isValueList) {
        XreadTableExpression(select);
    }

    return select;
}
```

对于 ”FROM” 或 ”INTO” 这两个词，是在 XreadSelect 方法中被解析的。

```
QuerySpecification XreadSelect() {

    QuerySpecification select = new QuerySpecification(compileContext);

    readThis(Tokens.SELECT);

```

```
7           if (token.tokenType == Tokens.TOP || token.tokenType == Tokens.LIMIT) {
8               SortAndSlice sortAndSlice = XreadTopOrLimit();
9
10              if (sortAndSlice != null) {
11                  select.addSortAndSlice(sortAndSlice);
12              }
13          }
14
15          if (token.tokenType == Tokens.DISTINCT) {
16              select.isDistinctSelect = true;
17
18              read();
19          } else if (token.tokenType == Tokens.ALL) {
20              read();
21          }
22
23          while (true) {
24              Expression e = XreadValueExpression();
25
26              if (token.tokenType == Tokens.AS) {
27                  read();
28                  checkIsNonCoreReservedIdentifier();
29              }
30
31              if (isNonCoreReservedIdentifier()) {
32                  e.setAlias(HsqlNameManager.getSimpleName(token.tokenString,
33                          isDelimitedIdentifier()));
34                  read();
35              }
36
37              select.addSelectColumnExpression(e);
38
39              if (token.tokenType == Tokens.FROM) {
40                  break;
41              }
42
43              if (token.tokenType == Tokens.INTO) {
44                  break;
45              }
46
47              if (readIfThis(Tokens.COMMA)) {
48                  continue;
49              }
50
```

```
            ......
        }

        return select;
    }
```

解析完这个之后，接下来将使用 XreadTableExpression 方法解析表名及之后的部分。这又分为两个阶段，第一阶段读取 FROM 子句，第二阶段读取 WHERE、GROUP BY 和 HAVING 子句。

```
void XreadTableExpression(QuerySpecification select) {
    XreadFromClause(select);
    readWhereGroupHaving(select);
}
```

在构建表达式树后，compileCursorSpecification 方法还需要调用 QueryExpression 类的 resolve 方法对表达式树进行解析。

```
    public void resolve(Session session, RangeGroup[] rangeGroups,
                        Type[] targetTypes) {

        resolveReferences(session, rangeGroups);

        if (unresolvedExpressions != null) {
            for (int i = 0; i < unresolvedExpressions.size(); i++) {
                Expression e = (Expression) unresolvedExpressions.get(i);
                HsqlList list = e.resolveColumnReferences(session,
                    RangeGroup.emptyGroup, rangeGroups, null);

                ExpressionColumn.checkColumnsResolved(list);
            }
        }

        resolveTypesPartOne(session);

        if (targetTypes != null) {
            for (int i = 0;
                    i < unionColumnTypes.length && i < targetTypes.length;
                    i++) {
                if (unionColumnTypes[i] == null) {
                    unionColumnTypes[i] = targetTypes[i];
                }
            }
        }

```

```
28          resolveTypesPartTwo(session);
29          resolveTypesPartThree(session);
30      }
```

3.4 DML、DDL 语句简析

DML 和 DDL 语句的解析过程与 DQL 语句的解析是一样的，都是按照自顶向下的方法完成的，分别在 org.hsqldb.ParserDML 类和 org.hsqldb.ParserDDL 类中完成。DML 语句解析后生成 StatementDMQL 对象，DDL 语句解析后生成 StatementSchema 对象。StatementDMQL 类记录了插入、更新操作所需的列映射和表达式信息。StatementSchema 类记录了各参数，然后再单独处理。由于这两类语句非本次实验的重点，就不再详细分析了。

4 数据的获取过程

之前，我们已经提到过的 executeCompiledStatement 方法首先会针对事务（Transaction）进行处理，真正的查询操作是在调用 execute 方法执行的。 execute 方法首先会根据 Statement 的一些字段处理权限，对于有非临时目标表的操作，如果当前用户只有只读（ReadOnly）权限，则产生异常。否则，处理子查询，然后调用 getResult 方法继续执行。经过若干次调用，结果集最终在getSingleResult 方法处理。

buildResult 方法创建结果集，首先初始化 navigator 和 result，使它们和 querySpecification 建立关联。然后设置并发处理的权限，有 updatable 和 read only 两种。接下来给 rangeVariables 中的每一个建立一个 rangeIterator。

```
1  for (int i = 0; i < rangeVariables.length; i++) {
2          rangeIterators[i] = rangeVariables[i].getIterator(session);
3      }
```

这里的 rangeVariable 是在编译的时候处理 FROM 子句时生成的。然后开始处理并查找满足查询条件的行。

接下来的 for 循环完成了除聚集函数、HAVING 子句以外的其他数据的获取。这里，fullJoinIndex 其实是一个代码上的 trick，正如上一实验中 HSQLDB 把 lower_bound 和 find 写在一起一样，这里利用了一些技巧，把有 RIGHT JOIN 修饰的 rangeVariable 和其他 rangeVariable 放在一起处理。对于非 RIGHT JOIN 的 rangeVariable，这里的行为类似于 DFS：通过 currentIndex 来维护深度，由于迭代器的存在，这里的DFS不需要记录状态，当 it.next()（18 行）返回 false 就说明当前层次遍历结束，此时在 24 行重置该层次的迭代器，同时跳到上一层；否则，继续递归到下一层。DFS的终点是 14 行的判

断。如果条件成立，有两种可能：一是遇到了右连接，二是递归结束。对于右连接操作，会指定 currentIndex 的值，并做一些标记，以便接下来统一处理。

整个过程就是一个在所有 rangeVariable 限定的范围内进行深度优先搜索的过程。由于有条件限制，搜索剪枝使得搜索的性能仍然能够令人满意。

4.1 索引加速查询

HSQLDB 对于可以使用索引的查询操作，用左闭右开区间的方式进行处理：在 RangeVariableConditions 类中记录了 indexCond 和 indexEndCond，首先利用索引找到第一条满足条件的记录，然后开始遍历，直到遇到第一个不满足索引条件的记录，查询结束。当然，在这个遍历过程中，非索引条件还是要用 4.1 节最后描述的方法进行判断。

indexCond 和 indexEndCond 是在解析类型的第二阶段（见 3.3）生成的，org.hsqldb.RangeVariableConditions 类的setIndexConditions 方法和 addCondition 方法共同产生了这两个 Expression[]。现在，可以看看 RangeIteratorMain 类的 next 方法：

```
1  public boolean next () {
2      while ( condIndex < conditions .length) {
3
4          if ( isBeforeFirst ) {
5              isBeforeFirst = false ;
6              initialiseIterator ();
7          }
8          boolean result = findNext ();
9          if (result) {
10             return true ;
11         }
12         reset ();
13         condIndex ++;
14     }
15     condIndex = 0;
16     return false ;
17 }
```

这里同样利用了类似 DFS 的方法，逐层遍历每一个条件。当此方法第一次被调用时，将会调用 initialiseIterator 方法初始化迭代器，否则，调用 findNext 方法向后遍历。

5 问题总结

5.1 QueryExpression、QuerySpecification及Statement的结构分别是怎样的？SQL语句是如何用这些结构表示的？

QueryExpression 是整个 SQL 语句解析的结果，其中主要包括 SortAndSlide 对象（存储 OrderBy 后的表达式和其他相关信息），RangeVariableList（存储 from 子句中的查询的表相关的表达式和信息），QueryCondition（存储 Where 子句中的查询条件），resultTable（存储查询的数据），以及一些索引和其他属性信息。

QuerySpecification 是 QueryExpression 的子类，存储单个 Select 体（select * from * Where*）的相关内容。对于单个的 Select 来说主要是比 QueryExpression 少存储 SortAndSlide 信息。

Statement 除了包括 QueryExpression 的内容外，主要还包括一些其他与该 session 相关的属性和一个 Scanner 用来扫描 SQL 语句的Token 流，以及一个 Token 对象，保存当前的 Token。

5.2 SQL语句中的查询条件是如何转化为关系代数树的？

Where 子句是一棵 boolean 值的表达式树，对于每个 boolean 值的叶子结点，两个孩子分别是 ExpressionColumn 和 ExpressionValue 类型，根结点记录了比较运算符，用这样的形式组织了一棵树，可以方便的判断条件的真假。

5.3 表的连结运算是如何运行的？条件选择运算是如何实现的？

首先进行 Join 分支条件判断，然后进行 Where 分支条件判断。对一个 equal 操作的返回值进行判断。

5.4 查询处理中是怎样利用索引来加快查询速度的？

查询处理中表的数据扫描是在 FindNext() 中进行的，其中会首先通过 getNextRow 来找到下一个行。下面是 getNextRow 的代码，从中可以看出，系统在表中查询的时候是在索引树中查询的，这样就充分利用了索引来加快查询。