面向数据库模式变更的代码演化推荐方法

张武能[1,2] 李宏伟[1,2,3] 沈立炜[1,2] 赵文耘[1,2]

(复旦大学计算机科学技术学院 上海 201203)[1] (上海市数据科学重点实验室(复旦大学) 上海 201203)[2]

(江西师范大学计算机信息工程学院 南昌 330022)[3]

摘 要 许多软件依赖数据库来存储信息。数据库模式的变更可能导致程序代码中与数据库相关的SQL语句代码不能正常执行,因而找出一种能够直接定位到需要修改的SQL语句代码并推荐出这些代码可能的修改方案的方法是十分必要的。提出的面向数据库模式变更的代码演化推荐方法首先自动检测出软件系统数据库模式发生的变更,随后采用程序切片技术得出与数据库操作相关的程序切片;确定受到数据库模式变更影响的程序切片后,利用源程序转换流程图算法将程序切片转化为程序流程图;根据程序流程图的分支条件得出SQL语句所有可能的特定执行路径;最后采用图映射的方法对每条路径的SQL语句进行变更语句推荐,推荐出新数据库模式下可执行的SQL语句。为了验证该方法的可行性,实现了一个用于自动检测数据库模式变更并能推荐出SQL语句演化后代码的插件工具。

关键词 数据库模式,演化,SQL语句,程序切片,程序流程图

中图法分类号 TP311 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2016.2.046

Method of Code Evolution Recommendation for Database Schema Change

ZHANG Wu-neng[1,2] LI Hong-wei[1,2,3] SHEN Li-wei[1,2] ZHAO Wen-yun[1,2]

(School of Computer Science, Fudan University, Shanghai 201203, China)[1]

(Shanghai Key Laboratory of Data Science, Fudan University, Shanghai 201203, China)[2]

(School of Computer Information and Engineering, Jiangxi Normal University, Nanchang 330022, China)[3]

Abstract Many software store information relies on database, while the database schema change will lead to some source code SQL statements associated with the database cannot be performed properly. Thus finding a way to locate the SQL statement code which needs modifying and to recommend possible changes to these programs is necessary. We proposed a code evolution recommendation method for database schema changes. Firstly, the method detects software system database schema change, and uses the program slicing technique to get source code fragments related to the operation of the database. Then it determines the program slicing impacted by the database schema change, and uses algorithm for generating flowchart from source program to get program flowcharts. SQL statements may have some distinct execution paths according to program flow branches condition. Finally, it uses figure mapping method for each SQL statement to recommend executable SQL statements under the new database schema. In order to verify the feasibility of the method, the article implemented a plug-in tool used to detect the database schema changes, and to recommend executable SQL statement under new database schema.

Keywords Database schema, Evolution, SQL statements, Program slicing, Program flow chart

1 引言

当前,大部分的软件系统都离不开数据库的支持。伴随着错误的修正、技术的革新,这些软件系统在其生命周期中都会涉及大量的维护与演化。数据库模式的演化是其中常见的类型。数据库模式主要是指数据库中表的结构、表之间的关联关系、表所包含的属性、属性的类型等,而模式变更是指数据库添加或删除表、表间关联关系的变更、表内属性增加删除以及属性的重命名、属性类型变更等。在数据库操作相关代码的逻辑较为复杂的情况下,针对数据库模式的变更,系统的开发与维护人员往往需要花费大量的时间与工作量寻找与数据库模式变更相关的程序源代码,并对其进行相应的修改。一方面,开发人员手工地定位与数据库模式变更相关的代码往往会出现定位出错或遗漏的情况;另一方面,开发人员也难以快速、准确地对具有丰富逻辑结构的代码进行修改。

以上的对软件维护带来的挑战可归纳为3个研究问题:

到稿日期:2015-01-20 返修日期:2015-05-03 本文受国家"863"高技术研究发展计划项目(2013AA01A605),国家自然科学基金项目(61402113)资助。

张武能(1989—),男,硕士生,主要研究方向为软件工程,E-mail:12210240047@fudan.edu.cn;**李宏伟** 男,博士,讲师,主要研究方向为软件工程;**沈立炜** 男,博士,副教授,主要研究方向为领域工程、软件产品线与自适应系统;**赵文耘** 男,教授,博士生导师,主要研究方向为软件工程、电子商务。

(1)如何定位源代码中与数据库模式变更相关的代码?

(2)对于存在非顺序结构的 SQL 语句(例如 if/else 等条件分支)而言,如何确定 SQL 语句的特定执行路径,从而辅助软件维护人员理解 SQL 语句的执行逻辑?

(3)找出 SQL 语句的一组执行路径后,如何向开发人员推荐变更代码?

针对这些研究问题,本文提出了一种面向数据库模式变更的代码演化推荐方法来进行修改代码的推荐。该方法首先自动检测出软件系统数据库模式发生的变更,采用程序切片技术得出与数据库操作相关的程序切片;确定受到数据库模式变更影响的程序切片后,利用源程序转换流程图算法将程序切片转化为程序流程图;根据程序流程图的分支条件得出 SQL 语句所有可能的执行路径;最后采用图映射的方法对每条路径的 SQL 语句进行变更语句推荐,推荐出新数据库模式下可执行的 SQL 语句。为了验证方法的可行性,本文实现了一个用于自动检测数据库模式变更,并能推荐出 SQL 语句演化后代码的插件工具。

本文第 2 节介绍了一个考试系统的数据库模式变更场景,基于该场景描述了人工的软件维护工作中存在的问题;第 3 节详细介绍了本文所提出的面向数据库模式变更的代码演化推荐方法;第 4 节介绍了面向数据库模式变更的代码演化推荐方法实现,并用实验验证了方法的可行性;第 5 节介绍了目前面向数据库模式变更的代码演化推荐的相关工作;最后讨论了本文方法以及研究展望。

2 动机示例

我们以一个考试系统的数据库模式变更为例来说明软件演化过程中存在的挑战,模式变更前后的映射关联如图 1 所示。初始时,系统使用表 student_info 存储学生某个科目的考试成绩,针对该表的数据库操作代码如图 2 所示。在维护阶段,为了降低数据库的数据冗余,数据表 student_info 将被分解为表 student 和 subject。

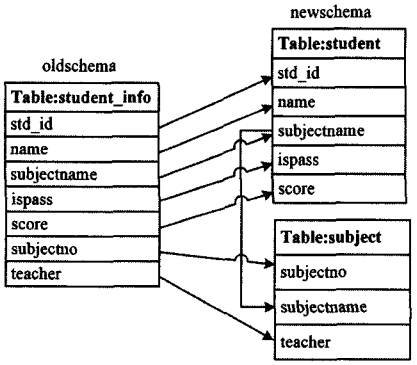

图 1 模式变更前后的映射关联图

```
11. public void GetInformation(String subjectname, String teacher,
    String subjectno,String notinsub){
12.     Connection conn=null;
13.     ResultSet rs=null;
14.     int conditionNum=0;
15.     String sql="select s. std_id, s. name, s. score, s. ispass, s.
        teacher, s. subjectno from student_info s where 1=1";
16.     if(!subjectname. equals("math")){
17.       sql=sql+" and s. subjectname'''+subjectname+''''';
18.     }
19.     else{
20.       sql = sql + " and s. subjectname like %''' subjectname
          +''''';
21.     }
22.       conditionNum++;
23.       if(!teacher. equals("#")){
24.         sql=sql. concat(" and s. teacher='''')+teacher+''''';
25.         conditionNum++;
26.         if(! subjectno. equals("#")){
27.         sql=sql+" and s. subjectno="+subjectno;
28.         conditionNum++;
29.         }
30.       }
31.     if(!notinsub. equals("#")){
32.       sql=sql. concat(" order by ")+ notinsub +"asc";
33.       conditionNum++;
34.     }
35.     try {
36.           conn=DBUtils. getConnection();
37.         PreparedStatement ps=conn. prepareStatement(sql);
38.         System. out. println(sql);
39.         rs=ps. executeQuery(sql);
41.         while (rs. next()) {
42.             System. out. print(rs. getString("std_id")+" ");
43.             System. out. print(rs. getString("name"));
44.             System. out. println();
45.           }
46.       }
47.       catch (Exception e){
48.       e. printStackTrace();
49.       }
50.     finally
51.     {
52.         DBUtils. close(rs);
53.         DBUtils. close(conn);
54.     }
55. }
```

图 2 代码变更片段示例

student_info 表分解为表 student 和表 subject 后,图 2 所示的代码片段不能正常执行,要保证代码正常执行,需要变更第 15,17,20,24,27,32 行代码中涉及到的表名和字段名为 newschema模式对应的新表名和字段名。

由于系统本身比较复杂,且数据库系统规模相对较大,数据库模式的设计不可能一蹴而就,系统的数据库模式可能不断演化。如果没有自动检测与数据库模式变更相关代码的技术支持,对于图 2 所述的代码修改过程,就需要系统开发人员手动查找并修改与数据库相关的代码,手动查找代码过程缓慢、繁琐,代码查找不全面,且存在 if/else 等条件分支结构的 SQL 语句拼接代码,不便于开发人员理解 SQL 语句特定执行路径的条件语义,修改过程困难,从而增加了开发人员的工作量。

3 面向数据库模式变更的代码演化推荐

本节介绍面向数据库模式变更的代码演化推荐方法流程、数据库模式变更的自动检测算法、数据库模式相关的程序源代码切片、程序切片至程序流程图的转化算法，以及程序源代码演化推荐算法。

3.1 方法流程

本文提出的面向数据库模式变更的代码演化推荐方法主要基于这样一种场景：某次数据库模式变更后，对程序源代码进行修改维护时，维护人员需要知道数据库模式变更后，哪些代码与数据库模式变更相关，并能得到一些代码修改建议。

因此方法的实施内容是：比较最新的数据库模式和前一个版本的数据库模式得出变更信息（或者直接获得数据库模式变更脚本）。对程序源代码进行程序切片，获得与数据库相关的程序切片，然后根据数据库模式变更信息定位到受影响的代码，将每个完整的 SQL 语句程序切片转换为程序流程图，对于存在分支的流程图，找出所有分支组合条件下 SQL 语句的特定执行路径。针对每一条 SQL 语句的特定执行路径，利用图映射的方法进行 SQL 语句代码演化推荐，推荐出可能的 SQL 语句修改方案。

面向数据库模式变更的代码演化推荐方法的流程如图 3 所示。当数据库模式发生变更后，或者维护软件的 BUG 时，或者需要为软件新增功能时，软件维护人员需要启动这个流程以维护相关的 SQL 语句。下文给出了几个主要步骤的细节。

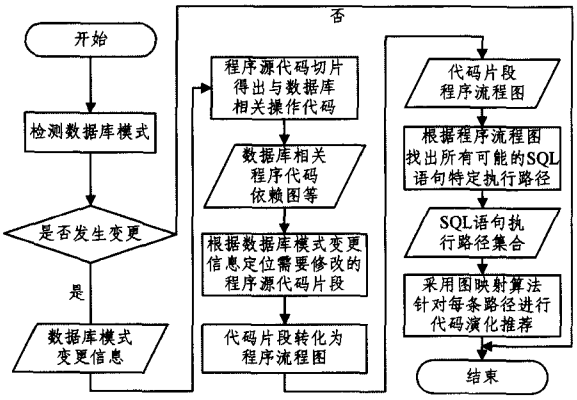

图 3 面向数据库模式变更的代码演化推荐方法流程

3.2 数据库模式变更的自动检测

自动检测数据库模式的目的是获得数据库模式的变更信息。本文作者根据文献[1,2]的结论和自身的软件开发经验，调查并总结了几种常见的数据库模式变更方式，如表 1 所列。SQL 查询语句不能体现表的主键、外键语义，因此本文后续的方法中不考虑涉及到主键、外键的数据库模式变更。

通过访问数据字典获取数据库模式变更前后的信息；利用这些信息构造出数据库模式名为根节点的一个树状图，该根节点的子节点由表构成、表的子节点由该表的键和属性构成、属性的子节点由该属性的类型信息构成；并以此数据结构将变更前后的信息存储于 xml 文件（当前版本的数据库模式信息文件 CurrentDataBaseSchema. xml、数据库模式变更之前最近一个版本的数据库模式描述文件 OldDataBaseSchema. xml）。将这两个 xml 文件的内容对应到两个图中，通过图遍历算法比较得出前一个文件相对于后一个文件的不同点，并将变更前和变更后对应的节点建立关联关系，即数据库模式变更信息。

存储单个表变更信息使用数据元组 CR(oldtablename, operating, oldattribute, newattribute, attributetab lename)。其中 oldtablename 表示数据库模式变更相关的表；operating 表示数据库模式变更方式；operating 值为 tablechange 时表示表重命名、增加表或者删除表；operating 值为 attributechange 时表示表内属性的变更，若 operating 值为“#”，则表示表没有发生变更；oldattribute 表示原表中的属性名；newattribute 对应原属性变更之后的属性名；attributetable 表示变更后的新属性名所属表。图 1 中的变更信息如表 2 所列。

表 1 常见的数据库模式变更方式

数据库模式变更方式	Operating 值	说明
AddTable	tablechange	新增加表
DeleteTable		删除表
RenameTable		重命名表名
AddAtrribute	attributechange	增加属性
DeleteAtrribute		删除属性
RenameAtrribute		重命名属性名
ChangeAtrribute		属性类型变更
AddPrimaryKey	primarykeychange	增加主键
DeletePrimaryKey		删除主键
ChangePrimaryKey		变更主键
AddForigenKey	forigenkeychange	增加外键
DeleteForigenKey		删除外键
ChangeForigenKey		变更外键

表 2 图 1 变更记录 CR 元组值

变量名	变量值
oldtablename	student_info
operating	tablechange
oldattribute	std_id, name, subjectname, ispass, score, subjectno, teacher
newattribute	std_id, name, subjectname, ispass, score, subjectno, teacher
attributetable	student, student, student, student, student, subject, subject, subject

3.3 数据库模式相关的程序源代码切片

程序切片方法主要用于找出程序中与某个语句相关的所有语句。本文采用了文献[3]所展示的程序切片算法进行程序切片，切片准则为包含关键字 select、insert、update、delete 等的代码语句。考虑到程序切片的效率，本文以每个类为单位进行程序切片，即切片的范围限定于类内；不涉及类与类之间的切片，方法中的程序切片主要是用于找出和数据库操作相关的所有语句。

程序切片的获取需要通过程序依赖图（Program Dependence Graph，PDG）（仅考虑过程内依赖关系）或者系统依赖图（System Dependence Graph，SDG）（既考虑过程内依赖，同时又考虑过程间依赖关系），构造 PDG 和 SDG 可采用文献[4]所提出的用虚节点栈构造控制依赖图的算法实现，以类中的每个方法为粒度，得到每个方法内部的程序依赖图，同时考虑每个类中方法间的依赖图。对于图 2 所示的代码片段，语句 String sql=“select s. std_id, s. name, s. score, s. ispass, s. teacher, s. subjectno from student_info s where 1=1”即为切片准则节点，此处命名为节点 SCN。根据程序切片准则〈15, select…from〉[5]，从程序的系统依赖图中即可得到图 2 所示代码的程序切片 A，如图 4 所示。

```
12.  Connection conn=null;
13.  ResultSet rs=null;
15.  String sql="select s. std_id, s. name, s. score, s. ispass, s. tea-
     cher, s. subjectno from student_info s where 1=1";
16.  if(!subjectname. equals("math")){
17.    sql=sql+" and s. subjectname'''+subjectname+''''';
18. }
19.  else{
20.     sql = sql +" and s. subjectname like %''' + subjectname
       +''''';
21.  }
23.  if(!teacher. equals("#")){
24.    sql=sql. concat(" and s. teacher='''')+teacher+''''';
26.    if(!subjectno. equals("#")){
27.    sql=sql+" and s. subjectno="+subjectno;
29.    }
30.  }
31.  if(!notinsub. equals("#")){
32.    sql=sql. concat(" order by ")+ notinsub +"asc";
34.  }
35.  try {
36.       conn=DBUtils. getConnection();
37.         PreparedStatement ps=conn. prepareStatement(sql);
38.         System. out. println(sql);
39.         rs=ps. executeQuery(sql);
41.       while (rs. next()) {
42.            System. out. print(rs. getString("std_id")+" ");
43.            System. out. print(rs. getString("name"));
45.       }
46.  }
50.  finally
51.  {
52.       DBUtils. close(rs);
53.       DBUtils. close(conn);
54.  }
55. }
```

图 4　程序切片 A

得到数据库模式变更记录信息，并通过程序切片得出与数据库相关的程序切片之后，本文方法通过变更之前的表名，即以 TableName 的值作为键值，查找变更记录，以确定该表是否发生了变更。若该表发生了变更，定位到与该表相关的程序切片。在方法实现过程中，使用节点数据结构 CodeNode 表示程序切片，它存储的信息包含代码语句、该节点代码语句所在文件的代码行、语句中是否包括 SQL 语句的标志 flag 以及 SQL 语句中涉及到的表名等信息。程序切片 A 中，SCN 节点值的代码行为 15，flag 为 true，涉及到的表名是 student_info。将找出的程序切片通过代码行映射到源代码中，即可定位出需要修改的源代码片段。

3.4　程序切片至程序流程图的转换

在文献[6-8]中已有各种将代码转换为程序流程图的算法实现。文献[8]提出的方法利用了栈和二叉树数据结构处理源程序，将程序切片 A 转化为程序流程图，如图 5 所示。

程序片段转化为流程图后，为了找出 SQL 语句的所有特定执行路径，本文的方法提取出程序流程图中的程序执行语句块。该语句块是以每行代码为粒度，使用代码行标记流程图中的节点，可以进一步简化程序流程图，去除程序流程图的控制节点后，图中并不是每个节点都受到数据库模式变更的影响，而对于那些不受数据库模式变更影响的代码，并不需要添加到 SQL 拼接语句中。如程序切片 A 第 32 行代码(底纹灰色的代码行)虽然包含 SQL 语句，但使用算法检测时，通过对每个节点 SQL 语句的单词进行逐个扫描检测的方式，发现该行语句只包含 SQL 关键字，并不含有表名和字段名，即该行并不受数据库模式变更的影响，因此在找出所有可能的 SQL 语句特定执行路径之前，将行数为 32 的节点从图中去掉后得到程序流程图简化图，如图 6 所示。

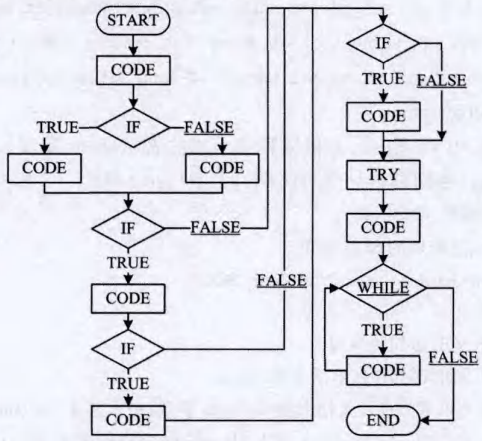

图 5　程序代码片段对应程序流程图

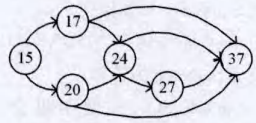

图 6　程序流程图简化图

将图 6 存储为邻接矩阵，利用图深度优先遍历算法即可得到图 6 代码片段中从第 15 行代码到第 37 行代码的所有的路径。图 6 得出的所有 6 条路径如下：

1) 15→17→37
2) 15→17→24→37
3) 15→17→24→27→37
4) 15→20→37
5) 15→20→24→37
6) 15→20→24→27→37

3.5　程序源代码演化推荐

找出所有的 SQL 语句特定执行路径后，为了便于解析 SQL 语句，对于 SQL 语句中属性值有参数的地方使用 # 号代替；针对每一条路径解析出完整的 SQL 语句，并找出执行该条路径的语义条件。可以对每一条完整的 SQL 语句进行语法分析，生成语法树，采用图文法[9]进行 SQL 语句的变更适配。由于本文主要对程序进行静态分析，为了提高效率，本文提出图映射的方法对 SQL 语句进行适配。图映射方法是指新旧数据库模式分别存储于两个图中，在比较数据库模式变更时，将原数据库模式下的表和属性字段映射到新数据库模式下的表和属性字段，即原数据库模式和新数据库模式的关联关系，表与表之间可能存在一对多、一对一、多对多 3 种关系，属性字段间则限定为一对一关系。进行代码变更时，找

到原数据库模式下表和属性字段所对应的新表名和属性字段，并根据新表、新属性字段之间的关系重新构造可正常执行的 SQL 语句，最终将适配后的 SQL 语句推荐给维护人员。本算法具体描述如图 7 所示。对于图 1 中 oldshema 模式下的查询语句为：

select s. std_id, s. name, s. subjectname, s. ispass, s. core, s. subjectno, s. score, s. teacher form student_info s where s. name＝＃

当模式 oldshema 变为模式 newshema 时，表 student_info 分解为表 student 和表 subject。此时该 SQL 语句可通过算法变更为：

select stt. std_id, stt. name, stt. subjectname, stt. ispass, stt. core, sut. subjectno, stt. score, sut. teacher form student stt, subject sut where stt. name＝＃ and stt. subjectname＝sut. subjectname

其中 stt 和 sut 是根据算法生成的表 student 和表 subject 的别名(相同别名加序号区别，如 stt1、stt2 等)。

```
Input:完整 SQL 语句
Output:适配后的 SQL 语句
Method GenRecommandSQLCode(SQL)
//预处理
  统计 SQL 语句的单词数;
  统计 SQL 语句中所有的表和别名;
  将每个单词节点写入相应的 SqlNode 节点，记入链表 nodelink;(含节点标志位 flag,名称 name,别名 aliasname,节点所属表 table)
//采用 flag 标记每个节点 SqlNode。
  For 每个单词节点 SqlNode∈SQL 语句
    Case SqlNode. name∈符号集合(",","(",")","="等符号)
      flag 置为 0 表示 SqlNode 为符号;
    Case SqlNode. name∈SQL 关键字集合("select","from","where","and"等关键字)
      flag 置为 1 表示 SqlNode 为 SQL 语句关键字;
    Case SqlNode. name∈数据库模式中的某个表
      flag 置为 2 表示 SqlNode 为某个表 T;
    Case SqlNode. name∈某个表 T 的属性集合
      flag 置为 3 表示 SqlNode 为某个表 T 的 A 属性，置 SqlNode. table 值为 T;
    Case SqlNode. name 为符号"(",且下一个节点为"select"
    //处理嵌套 SQL 语句
      获取该嵌套的语句 sqls;
      GenRecommandSQLCode(sqls);
    Case 其它情况
      flag 置为 4 表示 node 为其它单词;
    End Case
  End For
//处理数据库模式变更后的 SQL 语句
定义集合 tableset;
  For 每个单词节点 SqlNode // select,insert,update 语句
    If SqlNode. name 不是 SQL 关键字 from
      将 SqlNode. table 加入到 tableset;
    else if SqlNode. name 是 SQL 关键字 from
      根据新旧数据库模式的图映射对应关系，重新生成 from 子句;
      利用模式变更后表间外键关联关系，重新生成 where 条件子句;
      将新生成节点插入到 where 之后;
      处理后续节点;
      清空 tableset 集合;
    End If
  else if SqlNode. name 为某个表 T 的属性 A
    根据新旧数据库模式的图映射对应关系，变更 SqlNode. name 值、SqlNode. aliasname 值和 SqlNode. table 值，其中 SqlNode. aliasname 值根据指定算法生成;
    End If
  End If
End For
//生成 SQL 语句
遍历由 nodelink 链表生成变更之后的 SQL 语句;
End
```

图 7　SQL 语句变更算法描述

为了保证挖掘数据库模式信息和推荐 SQL 变更语句的可行性与准确性，本文的方法给出了一些约束，对于维护的系统：(1)数据库操作的定义使用标准的 SQL 语句，不能使用 Hibemate 等框架定义的简化的 SQL 语句；(2)系统中不能出现方法之间的循环调用以避免在方法之间的依赖关系分析中出现死循环的现象，如 A 调用 B，B 调用 C，C 调用 A；(3)SQL 语句关键字、表名、属性字段不能参数化实现，如语句 SELECT attributeparam FROM tableparam 中属性名和表名均参数化，本文方法会把 tableparam 当做表名，得出错误结果。设定约束后，方法流程的具体算法描述如图 8 所示。

```
input:系统路径 Path,数据库用户名 User 和密码 Pass
output:数据库模式变更相关的程序切片,适配后的 SQL 语句集合
//获取模式变更信息
Method compareOldschemaAndNewschema(Path,User,Pass)
begin
  获取 newschema 模式信息存于树 NewTree,并存储于 currentDataBaseSchema. xml 文件;
  读取 OldDataBaseSchema. xml 内容存储于树 OldTree;
  compare OldTree and NewTree,生成变更记录 CR 元组集合 CRSet;
  备份 CurrentDataBaseSchema. xml;
  CurrentDataBaseSchema. xml 重命名为 OldDataBaseSchema. xml;
end;
Method programSlicing()// 程序切片
begin
  设置代码块数据结构 codeblock,它由多个 CodeNode 组成;(含节点所属代码和 line,是否涉及到表名标志位 flag,涉及到的表名 table)
  查找源代码中包含 select,insert,update,delete 等关键词的类方法，生成方法集合 methodSet;
  对 methodset 中的方法构建程序依赖图，并根据"select…from"等准则进行程序切片;
  将得出的程序切片存入 codeblockSet;
  end;//变更后的 SQL 语句推荐给维护人员
  Method showResult(codeblockSet)
  begin
  定义存储结果用的 resultSet;
  for 每个 codeblock∈codeblockSet
    begin
    采用源代码转化程序流程图算法将 codeblock 转化为程序流程图 F;
    去除 F 中的控制节点和与数据库变更不相关节点得到 SF;
    图深度优先遍历算法求得 SF 中路径集合 pathSet;
    for 每条 path∈pathSet
      begin
```

```
    解析 path 得到完整 SQL 语句,并根据 SQL 语句行数反向推导
    执行此条 SQL 语句的条件语义;
    调用 GenRecommandSQLCode(SQL)产生新的 SQL 语句 New-
    SQL;
    将 NewSQL 加入到 resultSet;
      end
end;
展示生成的 resultSet 集合。
end;
```

图 8　本文方法流程具体算法描述

4　工具实现与实验

为了验证本文方法推荐 SQL 语句代码是否具有可行性,实现了一个 Eclipse 插件。该插件将软件项目源码所在目录和该项目所操作的数据库用户名和密码作为输入。通过该插件可以自动得出该项目的数据库模式发生了哪些变更,定位出受数据库模式变更影响的代码片段,并将对应代码片段演化后的 SQL 语句推荐展示给系统维护人员。整个工具的架构如图 9 所示,图中主要包含数据库模式变更信息提取模块(Extracting schema change information)、程序切片模块(Program Slicing)、源代码转化为程序流程图模块(Transforming code to program flow chart)和变更 SQL 语句模块(Changing SQL statement)。

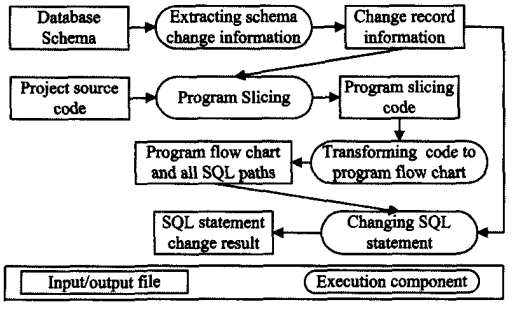

图 9　工具架构

实验设置:实验对象为一个学生考试成绩管理系统,该系统的数据库模式发生了 4 次变更。针对每一个数据库模式的变更,面向数据库模式变更的代码演化推荐方法捕捉每一次数据库模式所发生的变更,并推荐出可能的 SQL 语句演化代码。初始时数据库模式为 schema1,表名及属性如图 1 中数据库模式 oldschema 所示。

表 3 列出了 4 次数据库模式变更的信息。工具实现的结果在展示中分为两个部分:第一个部分是定位到需要修改的代码;第二部分给出了 SQL 语句演化推荐的代码。

表 3　数据库模式变更类型及说明

变更类型	变更操作说明
表字段增加	schema2 下表 student_info 增加字段 subjectno 和 teacher
表分解	schema3 下表 student_info 分解为表 student 和表 subject
表重命名	schema4 下表 student 重命名为表 student_info
表字段名变更	schema5 下表 subject 的属性字段 teacher 变更为 sub_teacher,其中涉及到复杂 SQL 语句的变更

在 schema2 模式下运行工具时,SQL 语句推荐结果如图 10 所示。当变更数据库模式为 schema3 时,SQL 语句推荐结果如图 11 所示,工具代码演化推荐的效果图如图 12 所示。表 student_info 分解为表 student 和表 subject,实验中找到了 6 条需要演化的 SQL 特定执行路径,输出结果明确指明了每条 SQL 语句执行的特定条件,以及执行该条路径的推荐代码。后续实验中,schema4 下表 student 重命名为表 student_info,schema5 下表 subject 的属性字段 teacher 变更为 sub_teacher,均可得出明确的演化推荐代码。因此,本文的方法在一定程度上能有效定位需要修改的代码,将代码片段转化为程序流程图之后,根据执行条件推荐代码,有助于维护人员理解程序的逻辑结构,提高软件开发维护效率。

```
第 1 个代码块
64:0 String sql="insert into student_info(std_id,name,subjectname,
ispass,score) values (?,?,?,?,?)";
66:1 conn=DBUtils.getConnection();
**************
insert into student_info ( std_id, name, subjectname, ispass, score)
values ( ?,?,?,?,? )
路径 1 变更后 SQL 代码推荐如下
insert into student_info (std_id, name, ispass, score, teacher, subject-
no, subjectname ) values ( ?,?,?,?,?,?,? )
```

图 10　schema2 模式 SQL 语句推荐结果

```
第 1 个代码块
14:0 String sql="select s.std_id,s.name,s.score,s.ispass,s.tea-
cher,s.subjectno from student_info s where 1=1";
15:1 if(!subjectname.equals("math")){
16:2 sql=sql+" and s.subjectname='''+subjectname+''''';
17:3 }
18:4 else{
19:5 sql=sql+" and s.subjectname like %'''subjectname+''''';
20:6 }
21:7 if(!teacher.equals("#")){
22:8 sql=sql.concat(" and s.teacher='''+teacher+''''';
23:9 if(! subjectno.equals("#")){
24:10 sql=sql+" and s.subjectno="+subjectno;
25:11 }
26:12}
27:13 if(!notinsub.equals("#")){
28:14 sql=sql.concat(" and s.ispass not in('N')");
29:15 }
31:16 conn=DBUtils.getConnection();
**************
路径 1 变更后 SQL 代码推荐如下
路径执行条件:!subjectname.equals("math")=true AND ! teacher.
equals("#")=true AND !subjectno.equals("#") =true
select stt.std_id,stt.name,stt.score,stt.ispass,stt.teacher,sut.sub-
jectno from student stt,subject sut where stt.subjectname=sut.sub-
jectname and 1=1 and stt.subjectname=# and stt.teacher=# and
sut.subjectno=#
路径 2 变更后 SQL 代码推荐如下
路径执行条件:!subjectname.equals("math")=true AND !teacher.
equals("#")=true
select stt.std_id,stt.name,stt.score,stt.ispass,stt.teacher,sut.sub-
jectno from student stt,subject sut where stt.subjectname=sut.sub-
jectname and 1=1 and stt.subjectname=# and stt.teacher=#
路径 3 变更后 SQL 代码推荐如下
路径执行条件:!subjectname.equals("math")=true
select stt.std_id,stt.name,stt.score,stt.ispass,stt.teacher,sut.sub-
jectno from student stt,subject sut where stt.subjectname=sut.sub-
jectname and 1=1 and stt.subjectname=#
路径 4 变更后 SQL 代码推荐如下
```

路径执行条件：!subjectname. equals("math")=false AND !teacher. equals("#")=true AND !subjectno. equals("#")=true

select stt. std_id, stt. name, stt. score, stt. ispass, stt. teacher, sut. subjectno from student stt, subject sut where stt. subjectname=sut. subjectname and 1=1 and stt. subjectname like %# and stt. teacher=# and sut. subjectno=#

路径 5 变更后 SQL 代码推荐如下

路径执行条件：!subjectname. equals("math")=false AND !teacher. equals("#")=true

select stt. std_id, stt. name, stt. score, stt. ispass, stt. teacher, sut. subjectno from student stt, subject sut where stt. subjectname=sut. subjectname and 1=1 and stt. subjectname like %# and stt. teacher=#

路径 6 变更后 SQL 代码推荐如下

路径执行条件：!subjectname. equals("math")=false

select stt. std_id, stt. name, stt. score, stt. ispass, stt. teacher, sut. subjectno from student stt, subject sut where stt. subjectname=sut. subjectname and 1=1 and stt. subjectname like % #

图 11　schema3 模式 SQL 语句推荐结果

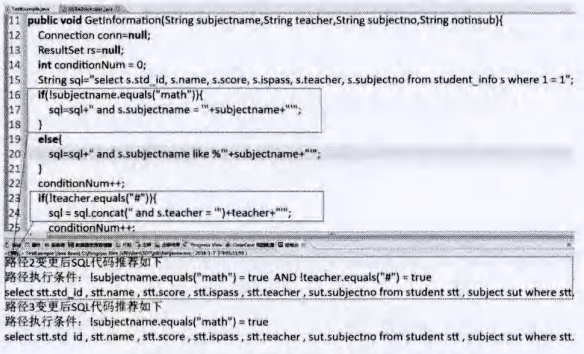

图 12　schema3 下代码演化推荐效果图

5　相关工作

1979 年 M. Weiser 首次提出了程序切片的思想[6]。通过构造程序依赖图(PDG)，并给定某个程序的切片准则，即可通过图的可达性算法[10]求出与切片相对应的依赖图。方法间调用后，方法内的依赖和方法间的调用构成了完整的系统依赖图(SDG)[10]。

在软件演化方面，目前的工作通常是将数据库模式变更和系统演化分开研究。Scott W. Ambler 和 Pramod J. Sadalage[11]详细介绍了怎样处理变更数据库模式以及重构数据库模式。一旦数据库模式变更，应用系统访问数据库的源代码也要做出相应的调整。

目前的一些代码变更影响分析方法主要是依据字符串搜索和简单的模式匹配，效果更好的则是采用数据流分析和字符串匹配相结合的方法进行分析。Andy Maule 和 Wolfgang Emmerich[12]研究了面向对象应用软件数据库模式变更的影响，除了采用简单的字符串匹配算法，其利用程序切片的方法得到了与数据库相关的程序片段，但他们没有针对数据库模式变化给出并推荐适应变更的新 SQL 语句的生成方法。

CA Curino 等人开发了 PRISM workbench 工具[13-16]来收集系统数据库模式演化的大量历史信息，并开发了一套数据库模式描述运算符 SMO(Schema Modification Operators)来描述数据库模式的持续演化历史。DBA 可以利用该工具预测这些变更可能造成的系统影响，修改并优化原有的查询语句使之能正确工作在新的模式版本下。

Maxime Gobert 等人开发了一个可用于分析数据库模式演化历史并使之可视化的工具[17]，并用该工具对一个复杂的信息系统进行了大规模的逆向工程案例研究[19]，结果指出数据库模式的演化历史信息有助于软件的维护与再工程。Martin Chytil 等人则研究了关系型数据库模式变更后 SQL 语句的适配问题。他们提出了一种基于图的查询模型方法，能使 SQL 语句适配新的数据库模式[18]，但该方法不支持 UNION、INTERSECT 等语句，且并没有涉及到程序代码的演化问题。

上述方法都能从某个方面较好地解决数据库模式变更和代码同步演化的问题，但是所花代价较大，而在数据库模式变更后，对 SQL 语句相关代码进行变更推荐的研究鲜有人关注。

6　结论与展望

本文提出的方法在数据库模式变更之后，能够较为准确地自动获取数据库模式的变更信息。对于程序流程图存在分支的 SQL 语句执行路径，面向数据库模式变更的代码演化推荐方法能够较为准确地利用相关语义的提示，执行 SQL 语句特定执行路径，并推荐出该条路径下可能的 SQL 演化语句，让软件维护人员能够更好地理解数据库模式变更之后可能需要修改的程序源代码。

尽管如此，本文提出的面向数据库模式变更的代码演化推荐方法还需要解决一些问题。例如 SQL 语句参数化的问题，还需要在静态分析方法的基础上引入更多技术来处理。另外，由于静态分析本身的缺陷，对于十分复杂的 SQL 语句并不能完全准确地推荐出可能的 SQL 适配语句。

结束语　本文后续的工作将瞄准如何提高推荐代码的准确性，以及如何处理复杂的 SQL 语句。通过初步的研究，可通过收集人工校验或者对 SQL 相关代码的训练来获得经验和知识，构造知识库用于对最终结果进行指导，以此提高结果准确度。对于复杂的 SQL 语句，可采用的潜在方法是将复杂 SQL 语句分解为多个简单的语句或者采用 SQL 优化技术化简 SQL 语句。如关系数据库的 DBMS 中提供的优化处理技术，将 SQL 语句转换为等价的关系代数表达式，获得语法树表示形式，然后通过等价规则，将投影选择移到树叶实现优化等，或者引入代码克隆来识别可能的、潜在的与数据库相关的代码片段等技术。

参 考 文 献

[1] Lin D Y, Neamtiu I. Collateral evolution of applications and databases[C]// Proceedings of the joint international and annual ERCIM workshops on Principles of software evolution (IWPSE) and software evolution (Evol) workshops. ACM, 2009: 31-40

[2] Qiu D, Li B, Su Z. An empirical analysis of the co-evolution of schema and code in database applications[C]// Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering. ACM, 2013: 125-135

[3] Field J, Ramalingam G, Tip F. Parametric program slicing[C]// Proceedings of the 22nd ACM SIGPLAN-SIGACT symposium on Principles of programming languages. ACM, 1995: 379-392

[4] Han Zhe, Chen Shi-hong. Analysis on follow region and algorithm for construction program dependence graph[C]// China

National Computer Conference. 2009:499-507(in Chinese)
韩喆,陈世鸿. 跳转语句跟随域分析与程序依赖图构造算法[C]// 中国计算机大会. 2009:499-507

[5] Weiser M D. Program slices: formal, psychological, and practical investigations of an automatic program abstraction method[D]. Ann Arbor, MI: University of Michigan, 1979

[6] Shan Yong-ming. An automatic method for generation control flow graph from source program[J]. Journal of Chinese Computer Systems, 1996, 17(10): 45-49(in Chinese)
单永明. 一种源程序到控制流图的自动生成方法[J]. 小型微型计算机系统, 1996, 17(10): 45-49

[7] Wang Wen-yong, Wang Xue-dong, Xiang Yu, et al. Research on automatic construction of program flow graph for assembley embedded software[J]. Computer Science, 2005, 32(2): 173-175(in Chinese)
汪文勇,王学东,向渝,等. 汇编嵌入式软件程序流程图自动生成的研究[J]. 计算机科学, 2005, 32(2): 173-175

[8] Mou Zhan-sheng, Zhang Hong-jun, Hu Li-fang. Research and Realization on the Automatic-converting Algorithm from Source Program to Flowchart[J]. Computer Development & Applications, 2008, 21(7): 28-30(in Chinese)
牟占生,张红军,胡丽芳. 源程序到流程图自动转换算法的研究与实现[J]. 电脑开发与应用, 2008, 21(7): 28-30

[9] Song Y, Peng X, Xing Z, et al. Automatic adaptation of software applications to database evolution by graph differencing and AOP-based dynamic patching[C] // 2012 IEEE 36th Annual Computer Software and Applications Conference (COMPSAC). IEEE, 2012: 111-118

[10] Ottenstein K J, Ottenstein L M. The program dependence graph in a software development environment[J]. ACM Sigplan Notices, 1984, 19(5): 177-184

[11] Ambler S W, Sadalage P J. Refactoring databases: Evolutionary database design[M]. Pearson Education, 2006

[12] Maule A, Emmerich W, Rosenblum D S. Impact analysis of database schema changes[C]// Proceedings of the 30th international conference on Software engineering. ACM, 2008: 451-460

[13] Curino C A, Moon H J, Zaniolo C. Graceful database schema evolution: the prism workbench[J]. Proceedings of the VLDB Endowment, 2008, 1(1): 761-772

[14] Curino C A, Moon H J, Ham M, et al. The PRISM Workwench: Database Schema Evolution without Tears[C]// IEEE 25th International Conference on Data Engineering, 2009(ICDE'09). IEEE, 2009: 1523-1526

[15] Curino C, Moon H J, Zaniolo C. Automating Database Schema Evolution in Information System Upgrades[C]// Proceedings of International Workshop on Hot Topics in Software Upgrades. 2009: 1-5

[16] Curino C, Moon H J, Deutsch A, et al. Automating the database schema evolution process[J]. The VLDB Journal, 2013, 22(1): 73-98

[17] Gobert M, Maes J, Cleve A, et al. Understanding Schema Evolution as a Basis for Database Reengineering[C]// 2013 29th IEEE International Conference on Software Maintenance (ICSM). 2013: 472-475

[18] Chytil M, Polák M, Nečcaský M, et al. Evolution of a Relational Schema and Its Impact on SQL Queries[J]. Studies in Computational Intelligence, 2014, 511(2): 5-15

[19] Cleve A, Gobert M, Meurice L, et al. Understanding database schema evolution: A case study[J]. Science of Computer Programming, 2015, 97: 113-121

(上接第 191 页)

[4] Li Wei, Chen Bin, Wei Ji-bo, et al. Secure Communications via Sending Artifical Noise by the Receiver: Ergodic Secure Region Analysis[J]. Journal of Signal Processing, 2012, 28(9): 1314-1320(in Chinese)
李为,陈彬,魏急波,等. 基于接收机人工噪声的物理层安全技术及保密区域分析[J]. 信号处理, 2012, 28(9): 1314-1320

[5] Li Xiang-yu, Jin Liang, Huang Kai-zhi, et al. A Physical Layer Security Transmission Mechanism of Relay System Based on Joint Channel Characteristics[J]. Chinese Journal of Computers, 2012, 35(7): 1439-1406(in Chinese)
李翔宇,金梁,黄开枝,等. 基于联合信道特征的中继物理层安全传输机制[J]. 计算机学报, 2012, 35(7): 1439-1406

[6] Xiao L, Greenstein L, Mandayam N, et al. Using the physical layer for wireless authentication in time-variant channels[J]. IEEE Trans. Wireless Commun., 2008, 7(7): 2571-2579

[7] Hou Wei-kun, Wang Xian-bin, Jean Y C, et al. Physical layer authentication for mobile systems with time-varying carrier frequency offsets[J]. IEEE Trans. Commun., 2014, 62(5): 1658-1667

[8] Dai Qiao, Song Hua-wei, Jin Liang, et al. Physical-layer Authentication and Key Distribution Mechanism Based on Equivalent Channel[J]. Science China, 2014, 44(12): 1580-1592(in Chinese)
戴峤,宋华伟,金梁,等. 基于等效信道的物理层认证和密钥分发机制[J]. 中国科学, 2014, 44(12): 1580-1592

[9] Zeng K, Govindan K, Mohapatra P. Non-cryptographic authentication and identification in wireless networks[J]. IEEE Trans. Wireless Commun., 2010, 17(5): 56-62

[10] Liu Y, Ning P. Enhanced wireless channel authentication using time-synched link signature, 2012 [C] // IEEE International Conference on Computer Communications (INFOCOM'12). Orlando, FL, 2012: 2636-2640

[11] Wu Xiao-fu, Yang Zhen. Physical-layer authentication for multicarrier transmission[J]. IEEE Commun. Lett., 2014, 19(1): 74-77

[12] Ma Ting, Ren Meng-yin, Wen Hong, et al. Bi-Directional Physical Layer-Assist Authentication in Smart-Meter System[J]. China Information Security, 2014(11): 85-87(in Chinese)
马婷,任梦吟,文红,等. 智能电表系统中的双向物理层辅助认证技术[J]. 信息安全与通信保密, 2014(11): 85-87

[13] Shan Dan, Zeng Kai, Xiang Wei-dong, et al. PHY-CRAM: Physical layer challenge-response authentication mechanism for wireless networks[J]. IEEE J. Sel. Areas Commun., 2013, 31(9): 1817-1827

[14] Tugnait J K. Wireless user authentication via comparison of power spectral densities[J]. IEEE J. Sel. Areas Commun., 2013, 31(9): 1791-1802

[15] 3GPP. Technical specification group radio access networks-deployment aspects: TR25. 943[R]. 2009

[16] Yahong R. Improved Models for the Generation of Multiple Uncorrelated Rayleigh Fading Waveforms[J]. IEEE Commun. Lett., 2002, 6(6): 256-258